2022

Program-based analysis for quantitative properties

BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

PROGRAM-BASED ANALYSIS FOR QUANTITATIVE PROPERTIES

by

WEIHAO QU

B.A., Tianjin University, 2014
M.S., University at Buffalo, 2016

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2022

Approved by

First Reader

Marco Gaboardi, Ph.D.
Associate Professor of Computer Science

Second Reader

Alley Stoughton, Ph.D.
Research Professor of Computer Science

Third Reader

Assaf Kfoury, Ph.D.
Professor of Computer Science

Fourth Reader

Hongwei Xi, Ph.D.
Associate Professor of Computer Science

DEDICATION

This dissertation is dedicated to

my loving grandfather

Zhaosheng Jin

&

my dear grandmother

Dongju Gao

ACKNOWLEDGMENTS

I would like to first express my sincerest thanks to Boston University, it is a great experience to study, do research, and enjoy my life here. I am so lucky that I made my decision to transfer to Boston University during the summer of 2019. I like the seminars, like the classes, like my colleagues here, like the discussion about logic, type systems.

I have been at BU for more than two years. Before that, I stayed at Buffalo for three years. Reviewing my path to Ph.D., so many people have helped me. In particular, I would like to say thank you so much to my advisor, Professor Marco Gaboardi. Thanks for your being an amazing advisor, for your bringing me to BU, for encouraging me to attend conferences, for providing great opportunities to collaborate with great researchers, for teaching me how to become a better human being! I am so fortunate that I can work with Marco for these years.

I want to thank my committee members. Great thanks to Professor Assaf Kfoury for your kindness in being the chair of my committee, and Professor Hongwei Xi for being the committee member of my dissertation. I want to thank Professor Alley Stoughton for your patience to read my dissertations so carefully and all your suggestions are important in this dissertation. I learned a lot from you, especially in Easycrypt. I also want to thank Deepak, for your kindness to be my committee member even though you are in Germany, for your invitation to MPI-SWS during the summers of 2018 and 2019, for your great ideas on the works presented in this dissertation, and for your careful reading and suggestions on improving this dissertation!

I would like to thank all my colleagues at Boston University as well as at the University at Buffalo. Thanks to Professor Lukasz Ziarek for his help at Buffalo. Thanks to all my collaborators in these works in the dissertation.

Last, but by no means least, I want to thank my father, Peijun Qu, my mother, Hui

Jin, for your support in pursuing my Ph.D. I am also thankful to my family members, my grandfather Xiaoping Qu, grandmother Ling Zhang. Thanks to my loving Liuyi Yao, lovely cats Cream, Oreo, Ergou, Nainiu, Haowen for their company.

PROGRAM-BASED ANALYSIS FOR QUANTITATIVE PROPERTIES

WEIHAO QU

Boston University, Graduate School of Arts and Sciences, 2022

Major Professor: Marco Gaboardi, Associate Professor of Computer Science

ABSTRACT

Many properties of programs can be expressed through quantities such as power consumption, memory usage, execution time, the number of function calls, etc. Controlling these quantities can improve program reliability, security, privacy, usability, etc. For instance, the performance in terms of the execution time of two pieces of code for the same task can be used to decide which one to use. Providing tight bounds on these quantities is useful to improve software design for different potential applications. In this dissertation, I will focus on several different quantitative properties of programs that can be used to improve programs performance, along different axes.

The first domain is relational cost analysis, which aims to provide tight bounds on the difference of the costs of evaluating two programs. The cost here is abstract, can be execution time, reduction step or number of functions calls, etc. Relational cost analysis earned people's attention for its application in program optimization and side-channel attack detection. Most of the related works have focused on pure functional programs. However, these reasoning principles may not be directly applied to situations where the program uses some effect, for example, destructive updates. To increase the expressiveness, we propose a type and effect system ARel to enable precise relational cost analysis over high-order functional programs, with mutable arrays, an important imperative

feature. It can be regarded as the first step toward relational cost analysis for functional imperative programs with mutable state. Additionally, relational type and effect systems such as ARel suffer from the lack of methodology of implementation. To fill this gap, we propose a bidirectional type checking methodology to algorithmize these systems.

Another quantity is the adaptivity in adaptive data analysis, where the queries asked by data analysts may depend on the results of previous queries. If we represent the interactions with a data analyst as a program, we can find some chain of queries so that every query may rely on the previous query in the chain. The adaptivity of this program is defined to be the length of the longest chain of queries. Reasoning about adaptivity helps to choose the appropriate statistical technique to use to control the generalization error of the data analysis. We propose a program analysis algorithm to statically estimate adaptivity.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction to Quantitative Properties

When people talk about a computer program, they care about whether this program can bring something good and also something bad will not happen. The good thing happens when the program can give the users the correct answer as they expect or it can solve their problems correctly, in this case, we say that the program is *functionally correct*. Also, the bad thing will not happen if this program executes as expected and never gets stuck, for this reason, in the programming language literature this property is often referred to as *safety*. Actually, besides the functional correctness and safety property of a program, people also care about some other properties which describe the side effects of programs, such as the execution time, memory usage, and power consumption. For instance, when you are installing an app on your phone, you not only care about whether the app is successfully installed, you may also want a progress bar providing an estimation time by when the installation can be done. These properties can usually be expressed in terms of quantities, and are called quantitative properties of programs.

In this dissertation, I will present two studies on reasoning about quantitative properties, from different axes.

1.1 MOTIVATION OF REASONING ABOUT QUANTITATIVE PROPERTIES

Nowadays, along with the popularity of mobile devices, more and more programs will be executed on machines with limited resources. We have seen that games are running on smartphones and tablets, apps tracking people's health data are running on wearable devices such as smartwatches, autopilot programs are running on Unmanned Aerial Vehicles(UAV). For these programs, quantitative properties such as memory usage, power consumption, and execution time become very important considering the

resource limit of the devices that these programs are running on. In this sense, reasoning about these quantitative properties has wide potential applications.

Another application is program optimization. People can safely replace a piece of code with the optimized one when they can reason about the memory usage or execution time of these two pieces of code and show the optimized one is always better. Reasoning about these resources can be realized by resource analysis.

Another application domain is security. Reasoning about quantitative properties can help detect side-channel attacks. A side-channel attack is any attack where the adversary uses some extra information from the cryptosystem. This extra information can be the timing information, power consumption. In timing channel attacks, the adversary observes the timing information of the cryptosystem handling the private information, such as the time of the system to verify one user's id and password. Let us assume the adversary knows the user's id (public in some other places) but does not know the password. The adversary can then have a malicious attempt with the user's id and a fake password. If the timing information of this cryptosystem is observably different from the target user's timing information, the adversary can compare the two passwords and learn the extra information on how far the two passwords are away from each other. Then the adversary can modify the next attempt accordingly. Fortunately, reasoning about the execution time of the target cryptosystem can tell us the system is resilient with respect to these attacks. In particular, reasoning about the difference of the execution time of the target system in two runs, one run takes arbitrary input and another one uses the password to be protected, can show that the attacker can not learn any information by observing the execution time of the two runs when the difference that is reasoned about is very small to be observed. This application requires relational reasoning, which is one important part of this dissertation.

Also, quantitative properties can be used indirectly as some auxiliary component in other research. For example, in the data analysis community, the analysts run data analysis to study some general property of some population, such as people's age in the united states. Instead of running their data analysis on the population(big data), they choose the sample data drawn from the population because it is often impossible to run it on the population. People use generalization errors to describe the degree that the conclusion drawn from the sample data generalizes to the population. The analysts usually ask queries to interact with the sample data. A study has shown that when the queries are asked adaptively, which means the choice of one query may rely on the results of other queries, the generalization error of this data analysis becomes very large. A new technique to control the generalization error is to add mechanisms. The key point is to choose the "right" mechanism for the data analysis. Some researchers (Dwork et al., 2015c; Bassily et al., 2016) shows that choice of the mechanisms for a data analysis is related to the relation between its queries. The relation between queries can be expressed as the length of the longest chain of queries in the algorithm such that every query in the chain may rely on its previous queries' results. The length is called adaptivity in this dissertation.

1.2 METHODOLOGY

We use static analysis techniques to reason about the two main quantitative properties studied in this dissertation: the relational cost of two programs and the adaptivity of adaptive data analysis programs.

1.2.1 Relational Refinement Type and Effect Systems

We choose the relational type and effect system, along with relational refinement, to perform the relational cost analysis. The benefit of this choice is that the dynamic information of the programs, which is the relational cost in our case, can appear in the type as an effect. We can then verify the cost of a program by type checking the program with the appropriate type annotated with the cost bound. We are interested in cost in the relational setting, so the type and effect system we use is relational. To be precise, the cost in the relational setting refers to the difference of execution cost of two programs, called relative cost. The relational type and effect system usually relates two programs to one relational type, which ascribes a pair of computations. The effect then becomes the upper bound on the relative cost of the two programs. Suppose we would like to compare two closed programs t_1 and t_2 and check whether t_1 runs fast than t_2. In a relational type and effect system, we can provide a judgment as follows.

$$\vdash t_1 \ominus t_2 \lesssim D : \tau$$

The relational cost (effect) D reveals the upper bound on the difference of the cost of running the two programs t_1 and t_2, related by the relational type τ. If D is less than 0 and cost stands for the execution time, we can safely say that t_1 runs faster than t_2.

Relational refinement is another important component of our relational cost analysis, which is used to provide extra information about data structures to enable a precise cost estimation. For instance, our system relates a pair of lists, both of length N, which differ in at most α indices, the relational type $\mathsf{list}^{\alpha}[N]$. The main novelty of our system is the support of mutable arrays with destructive updates, one of the standard imperative features, which is missing in related work on this topic. Naturally, along with the sup-

port of mutable arrays, the modification of the heap where the arrays are stored before and after the computation involving array operations comes into our sight. We use relational refinement to reason about these imperative components. More details about these refinements will be explained in the later chapters along with concrete examples.

1.2.2 Dependency Graph

Another quantitative property we study is the adaptivity of adaptive data analysis programs. The adaptivity is defined to be the length of the longest chain of queries of the program in which one query may rely on its previous queries in the same chain. The adaptivity of a data analysis program is quite different from the relative cost of the two programs. To reason about cost, we can simply sum the costs of sub-programs to get the upper bound of the cost. However, the adaptivity of each sub-program of the target program does not help much in finding the adaptivity. To use a type and effect system to reason about adaptivity, this system should be able to express certain dependencies. We expect adaptivity can also be reasoned about by type and effect systems but in this work, we choose to use a dependency graph to reason about adaptivity.

There are two challenges to reason about adaptivity: what will a formal definition of the adaptivity of a data analysis program look like, and how to reason about the adaptivity. In our work, the first challenge is solved by a query-based dependency graph generated along with the evaluation of the program. Our trace-based operational semantics can generate the trace which tracks the queries asked in the evaluation of the program. The query-based dependency graph is constructed based on the queries in the trace. Every node of the graph represents a unique query asked in the program, and the directed edge between two nodes, identifying one query (one node) may depend on the other one. The adaptivity is then formally defined as the length of the longest path in the

query-based dependency graph. The second challenge is how to estimate the adaptivity of a data analysis program we have just defined. In order to upper bound the length of the longest path in the query-based dependency graph, we want to find a path in a dependency graph as well such that the path contains all the queries in that longest path in the query-based dependency graph. We develop an algorithm to build a variable-based dependency graph, in which every node represents the unique variable that is assigned in one assignment statement of the program. The direct edge between two nodes reflects both the data dependency and control flow dependency between variables. Also, we add the unit weight to the node whose variable is associated with a query result. Then the upper bound on the adaptivity of the data analysis program is estimated by the weight of the path with the highest weights in the variable-based dependency graph.

To summarize, we use two dependency graphs for reasoning about the adaptivity of a data analysis program: one query-based dependency graph to define the adaptivity, and one weighted variable-based dependency graph to estimate an upper bound on the adaptivity.

1.3 CONTRIBUTIONS

Program analysis techniques are usually applied to study quantitative properties. In comparison to those related works, this dissertation has the following contributions.

1. We propose a refinement type and effect system ARel, which provides relational cost analysis on programs with mutable arrays. It can be regarded as one step forward to the full support of imperative programs of relational cost analysis by means of type and effect systems.

2. We propose a bidirectional type checking system that implements ARel. The

methodology used can guide the implementation of (relational) type and effect system, with (relational) refinements.

3. We propose a formal definition of the adaptivity of adaptive data analysis programs and develop a graph-based program analysis algorithm that statically estimates the adaptivity.

1.4 DISSERTATION OUTLINE

This dissertation includes two main parts. One reasons about the quantitative property (cost) of programs in the relational setting. Another one aims to use the quantitative property (adaptivity) of adaptive data programs indirectly, as an auxiliary component in data analysis research.

The rest of the dissertation is divided into 5 parts.

1. Part i (ARel) is about relational cost analysis for functional-imperative programs.

   Chapter 2 provides the necessary background of relational cost analysis in Section 2.1 and shows the advantage of RelCost Çiçek et al. (2017a), a type and effect system for relational cost analysis over the traditional method described in Section 2.2, through a standard map example in Section 2.3.

   Chapter 3 discusses the limitation of RelCost that it only supports functional programs. Hence, we introduce ARel, also a type and effect system, in spirit of RelCost, but supporting one of the most common imperative features, mutable state in Section 3.1. Two examples, including one which is still the map example but in its imperative version in Section 3.3.1, serve as the introduction examples of ARel. The contributions of ARel are summarized in Section 3.2.

Chapter 4 formally presents the type system of ARel. The existence of stateful computation naturally partitions the system into two parts: the pure components like those in a standard functional language, and the impure ones which are designed for those stateful computations. Hence, the syntax in Section 4.1, the operational semantics in Section 4.2, the types in Section 4.4 as well as the typing judgments (Section 4.5) have the clear distinction between the pure and impure parts. Also, the types and typing support both the standard unary reasoning and relational reasoning, which leads to the unary and relational types and typing, respectively. The subtyping is also important in ARel and is introduced in Section 4.5.3.

Chapter 5 shows the soundness of ARel. We present the logical relation model by introducing the interpretation of types in both the unary (Section 5.1) and relational setting (Section 5.2). Then the fundamental theorem as well as the necessary auxiliary lemmas are discussed in Section 5.3. Chapter 6 gives five more functional-imperative examples to help readers to understand how ARel provides the precise relational cost analysis for many interesting problems.

Chapter 7 discusses the related work of ARel, from two perspectives. Many works that perform static cost analysis are compared with ARel in Section 7.1, the main difference is that all these mentioned works focus on unary cost, while ARel cares about relational cost. Then, the other related part is the relational verification, involving researches on relational reasoning in Section 7.2.

2. Part ii (BiARel) is about the implementation of ARel.

Chapter 8 clarifies three key points during the implementation: relational type system, relational effect and relational refinements. To handle them, we choose

bidirectional type checking, a successful technique to implement type systems. The study on bidirectional type checking is well-organized step by step, by first considering the implementation of standard relational type system by bidirectional type checking, and then adding relational refinements, and then adding relational effects, and finally including the impure component. The outline of the study is in Section 8.2. The details of this study will be presented in Chapter 9.

Chapter 9 presents five type systems and the process of developing the corresponding bidirectional type systems. It starts from a standard relational type system relSTLC (Section 9.1), then adding the relational refinements to form RelRef in Section 9.2, and then adding unary type system and connecting unary and relatioan types in RelRefU in Section 9.3. After that, the relational effect is added to RelRefU to build the type system RelCost (Çiçek et al., 2017) (a prior work ARel extends from) and its bidirectional type system (in Ezgi Cicek's thesis (Çiçek, 2018)) in Section 9.4. Finally, the addition of imperative components to RelCost gives us the bidirectional type system for ARel (Section 9.5). In this chapter, a two-step method is used to algorithmize all the systems except relSTLC.

Chapter 10 shows the implementation in real world - the bidirectional type checker. We present BiARel for ARel. We also include heuristics and constraint solving technique in the type checker BiRelCost for RelCost (see Ezgi Cicek's thesis (Çiçek, 2018)) that BiARel is inspired by, which can also be applied to BiARel. Other systems in Chapter 9 can be regarded as a simplified version of these two systems. Many examples are verified in experiments of BiARel in Section 10.2.4.

Chapter 11 discusses the literature on implementation of various combination of refinement types, effect systems, modal types and subtyping. It covers the related work on bidirectional typechecking (Section 11.1), lightweight dependent

types (Section 11.2), relational verification systems (Section 11.3) as well as subtyping elimination (Section 11.4).

3. Part iii shows the work on the program analysis algorithms to study the adaptivity of the adaptive data analysis program.

   Chapter 12 gives the introduction of adaptive data analysis in Section 12.1 and the challenges (Section 12.2) we face to obtain the adaptivity to control the generalization error of an adaptive data analysis program.

   Chapter 13 presents the details of the loop language (Section 13.1) we use to express data analysis programs, and shows the definition of adaptivity from a trace-based operational semantics in Section 13.2.

   Chapter 14 introduces the SSA version of the loop language (Section 14.2) to enable an easier analysis over adaptivity by an easier tracking of dependency relations between variables. The limitation of direct analysis over the loop language is covered in Section 14.1. The transformation from the loop language to the SSA loop language is presented in Section 14.4.

   Chapter 15 describes the program analysis algorithm ADAPT that used to estimate the adaptivity of the data analysis programs (Section 15.2, Section 15.3). The idea behind the algorithms (Section 15.1) is to construct a data control dependency graph, and add weights to the graph. The adaptivity is estimated by the weight of the path with the highest weight.

   Chapter 16 presents some more interesting examples ADAPT can analyse. It includes a variant of two round data analysis algorithm (Section 16.1), an adaptive multiple rounds data analysis algorithm (Section 16.2) and an example showing the over-approximate of our approach (Section 16.3).

Chapter 17 discusses the related works from three perspectives: Static program analysis (Section 17.1), dynamic program analysis (Section 17.2) and generalization in adaptive data analysis (Section 17.3).

4. Part iv concludes the dissertation and discusses the future directions on studying the two quantitative properties: relative cost and adaptivity.

5. Part v gives the appendices. The appendix A includes the details of ARel. The appendix B includes everything we need to develop a bidirectional type system for a relational refinement type and effect system. The appendix C gives the proof of the lemmas of our framework in adaptive data analysis.

1.5 PREVIOUS PUBLISHED MATERIAL

The contents of this dissertation are partly based on the work and writing in the following papers.

The part of ARel is based on the paper "Relational cost analysis for functional imperative programs" (Qu et al., 2019) appearing at ICFP 2019 and its journal version which will appear at Journal of Functional Programming 2022, at the link:
https://www.doi.org/10.1017/S0956796821000071.

The part of BiARel is based on the paper "Bidirectional type checking for relational properties" (Çiçek et al., 2019) at PLDI 2019, and "Relational cost analysis for functional-imperative programs" (Qu et al., 2019), and it builds on previous works describing the implementation of RelCost Çiçek (2018).

I AREL

In the next few chapters, we will see ARel, a relational type and effect system, with relational refinements. It performs the relational cost analysis over programs with mutable state (arrays), and provides precise cost upper bounds.

CHAPTER 2

Relational Cost Analysis

2.1 BACKGROUND

Cost analysis is a mature topic in which one is interested in statically estimating a tight bound on the number of resources(cost) needed to run a program. One classical cost is the execution time, which is commonly used to measure the performance of software designs. When the execution time of two pieces of code for the same computing task can be estimated through cost analysis, one can decide which one to use to optimize the total performance. Also, the vulnerability of an arbitrary cryptographic system to a side-channel attack (Timing attack) can be detected when the execution time needed for the system to verify the unknown user's password and the adversary's malicious input will be observably different, through cost analysis.

On the other hand, the cost is not limited to the execution time, with respect to cost analysis. It is abstract according to the cost model, which means it can be either the number of reduction steps in an operational semantics; or the maximal number of abstract units of memory; the number of recursive calls during the execution; sometimes even the power consumption. Naturally, the abstract cost concept enriches the application of cost analysis. For instance, when the cost model counts the power consumption of programs, another channel attack (power-analysis attack) can be detected.

An interesting observation of the aforementioned applications of cost analysis is the relational nature of the cost: we compare the execution time of one piece of code to another one when optimizing the program; the execution time of the target cryptographic system for handling both the unknown user' password as well as the adversary's attempt are needed to detect side-channel attack, etc. The cost analysis needs to talk about two

runs of programs, which refers to either the executions of two programs or the same program with varied inputs. To this end, it brings to us, relational cost analysis. It has been a promising research topic in cost analysis, whose aim is to provide an upper bound on the difference of the execution costs of two runs of programs. The difference is usually known as relative cost. To be concrete, the relative cost of a program t_1 with respect to another program t_2 in a certain cost model is: $\mathsf{cost}(t_1) - \mathsf{cost}(t_2)$, where $\mathsf{cost}(t_1)$ is the execution cost of program t_1 in the model, and similarly for $\mathsf{cost}(t_2)$.

In general, the relational cost analysis can be used in the following scenarios.

1. For the same problem, reasoning about the relative cost of its two implementations helps decide which one is more efficient. According to the cost model, it can talk about memory efficiency, time efficiency, or power efficiency, etc.

2. For a single software, a piece of code can be updated safely when the relative cost of the updated piece with respect to the original piece is bounded by a certain quantity.

3. In cryptography, a small relative cost upper bound on two runs of the same cryptographic system with two different inputs can show that the cost of the system is independent of its secret inputs. Otherwise, it is vulnerable to this kind of attack.

4. In the algorithm analysis area, reasoning about the relative cost reveals the stability of the algorithm with respect to its input changes.

2.2 TRADITIONAL RELATIONAL COST ANALYSIS

The traditional relational cost analysis uses the worst- and best-case method, relying on the classic (unary) cost analysis on the single execution of a program. Its main idea is that, the upper bound on the relative cost of a program t_1 with respect to another

program t_2 can be obtained by taking the difference of the upper bound U_1 such that $\mathsf{cost}(t_1) \leq U_1$, and the lower bound L_2 such that $L_2 \leq \mathsf{cost}(t_2)$: L_2, as $U_1 - L_2$. We can then safely conclude that: $\mathsf{cost}(t_1) - \mathsf{cost}(t_2) \leq U_1 - L_2$.

The worst- and best-case analysis takes full advantage of the mature techniques in classical cost analysis but has two inevitable downsides.

1. Sometimes it is much harder to estimate the unary upper and lower bounds of the two programs respectively, than getting the relative cost upper bound of two programs directly. For instance, when we want to know the upper bound on the relative cost of a program in a distributed system with respect to its updated version, getting the upper and lower bound on the cost of either the original or updated program becomes very hard considering the complexity of the whole system. However, direct focusing on the parts that vary in the two programs is simpler, since we know that only the corresponding diffs of the two versions contribute to the relative cost. In other words, for those parts of the program that are the same (or structurally similar), we do not even care about the upper and lower bound on their cost no matter how complex they can be. This greatly decreases the effort to perform relational cost analysis.

2. Imprecision is another drawback of the traditional indirect relational cost analysis. Simply taking the difference of the upper bound of the cost of one program and the lower bound of another program inevitably ignores the inner connection between the two programs, for example, the structural similarity. Let us consider an extreme case where one is interested in establishing an upper bound on the relative cost of a program t with respect to itself. Suppose the upper bound is U and lower bound is L so that $L \leq \mathsf{cost}(t) \leq U$, then the indirect analysis gives the coarse upper bound $U - L$, while intuitively the relative cost is 0. When we con-

sider other common cases, imprecision is also there. Take merge sort as another example, the relative cost of the same program implementing merge sort on two lists of the same list size n is $\mathcal{O}(n \cdot \log n)$ by indirect analysis. A more precise bound $\mathcal{O}(n \cdot (1 + \log k))$ can be obtained by direct relational cost analysis where k is the number of indices the two lists agree on.

2.3 RELCOST

To overcome the drawbacks of the traditional relational cost analysis, performing the direct analysis on two programs becomes necessary so that the similarity of the two programs can be taken into consideration. RelCost Çiçek et al. (2017a) adopts this idea, and performs precise relational cost analysis on high order functional programs, through a relational refinement type and effect system. This idea of using relational type and effect systems, along with relational refinements, greatly influenced our proposed system ARel. We first give a standard map example in RelCost to give the readers a first taste of the benefit of performing direct relational cost analysis with respect to the traditional one.

2.3.1 Map Example through RelCost

Consider the standard list map function that applies the mapping function map to all the elements of the list. The list is inductively defined, which can be an empty list nil or

cons(h, tl), consisting of the head element h and the tail list tl.

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{map}\ (f).\lambda l. \\
\quad \mathrm{case}\ l\ \mathrm{of} \\
\quad\quad \mid \mathrm{nil}\ \rightarrow \mathrm{nil} \\
\quad\quad \mid h :: tl \rightarrow \mathrm{cons}(f\ h,\ \mathsf{map}\ f\ tl)
\end{array}
$$

Now let us suppose that we have two lists of the same length, we would like to know that whether mapping one list will always uses more time than another by the same mapping function. Then the traditional relational cost analysis requires us to get the lower and upper bounds of the execution of the mapping function on the two lists respectively. Let us assume that the running time of the input function f is related to the list element and its upper bound is U and lower bound is L. Then we know that the cost of every mapping on the head h of the list is bounded: $L \leq \mathsf{cost}(f\ h) \leq U$. Suppose the two lists l and l' have the same size N, then we know that

$$L * N \leq \mathsf{cost}(\mathsf{map}\ f\ l) \leq U * N$$

$$L * N \leq \mathsf{cost}(\mathsf{map}\ f\ l') \leq U * N$$

The traditional one then gives us the relational bound $U * N - L * N$.

Intuitively, the bound can be better. When the two input lists have the element in common at the same index, then we know that the difference of the cost of the mapping at this index is 0, when we assume f is deterministic and its cost only relies on its argument. In this case, we are able to give a better upper bound on the relative cost of two runs of $\mathsf{map}\ f$ with varied input lists l and l', when we know that how many elements of the lists at the same indices *may differ*. RelCost tracks this information by its relational

refinement type. The two lists are related at a relational type list$[N]^{\alpha}\,\tau$ where N gives the size of the two lists and α tells us the number of elements the two lists may differ. In this way, RelCost gives a better bound $(U - L) * \alpha$. The observation that there is no relative cost on indices where the two lists agree is reflected in this bound.

CHAPTER 3

Relational Cost Analysis with Mutable State

We have seen the benefits of direct relational cost analysis in exploiting the similarity of two programs for a better relative cost upper bound. Hitherto, the literature on relational cost analysis, such as RelCost, has been limited to functional languages. However, many practical programs are stateful and use destructive updates, which are more difficult to reason about. Consequently, our goal is to develop relational cost analysis for functional languages with mutable state (i.e., for functional-imperative programs).

3.1 INTRODUCTION

In spirit of RelCost, we propose a *refinement type-and-effect* system, ARel, for relational cost analysis in a functional, higher-order language with mutable state.

The first question we must decide on is *what* kind of state to consider. One option could be to work with standard references as found in many functional languages like ML. However, from the perspective of cost analysis it is often more interesting to consider programs that operate on entire *data structures* (e.g., a sorting algorithm), not just on individual references. Remember the merge sort and map example we mentioned in Chapter 2. Consequently, we consider *mutable arrays*, the standard data structure available in almost all imperative languages. This allows us to verify more interesting examples. The prior work RelCost can also simulate the mutable arrays with its list in a state-passing style. However, it can not get the precise relative cost upper bound as ARel for some interesting examples when the position information of arrays is useful for a precise cost. RelCost can not track it but ARel can. We will see this point when we see examples of ARel.

Second, we must decide *how* to treat state in our functional language. Broadly, we

have two choices: State could be a pervasive effect as in ML, or it could be confined to a monad as in Haskell, which limits the side-effect to only those sub-computations that access the heap. In ARel, we choose the latter option since this separates the pure and impure (state-affecting) parts of the language at the level of types and reduces the complexity of our typing rules.

The primary typing judgment of ARel, $\vdash t_1 \ominus t_2 \lesssim D : \tau$,[1] states that the programs t_1 and t_2 are related at type τ, which can specify relational properties of their results and, importantly, that their relative cost (cost of t_1 minus the cost of t_2) is upper bounded by D. To reason about array-manipulating programs in the relational setting, we also need to express relations between corresponding arrays across the two runs of the two programs. For this reason, our monadic type (the type of impure computations that can access state) has a *refinement* that specifies how arrays are related across the two runs *before* and *after* a heap-accessing computation. Specifically, our monadic type has the form $\{P\}\ \exists\vec{\gamma}.\tau\ \{Q\}^{\mathrm{diff}(D)}$. This type represents a pair of computations which, when starting from arrays related by the relational precondition P, end with arrays related by the relational postcondition Q, return values related at τ, newly generated arrays referred by static names $\vec{\gamma}$, and have relative cost at most D. This design is inspired by relational Hoare logics (Nanevski et al., 2013; Benton, 2004), but there are two key differences:

1. Our pre and postconditions are *minimal*—they only specify the indices at which a pair of corresponding arrays may differ across the two runs, not full functional properties. This suffices for relational cost analysis of many programs and simplifies our metatheory and, importantly, the implementation.

2. Our monadic types carry a relative cost, and the monad's constructs combine and

[1] This judgment is inspired by Çiçek et al. (2017a) proposing a type-and-effect system for relational cost analysis of functional programs *without state*. Notice that one can use this typing judgment also to reason about *lower bounds* on the relative cost, by exchanging t_2 and t_1 and considering a negative cost $-D$.

propagate the different costs generated by the computations identified by these monadic types.

ARel supports establishing lower and upper bounds on the cost of a *single* expression, and falling back to such unary analysis in the middle of a proof of relative cost. Improving over previous type and effect systems for relational cost analysis, ARel permits combinations of these two kinds of reasoning in the definition of recursive functions. Specifically, ARel provides typing rules for the fix-point operator that allow one to *simultaneously* reason about the cost in the unary and relational setting.

To prove that our type system is sound, we develop a logical relations model of our types. This model combines unary and binary logical relations and it supports two different effects, cost and state, that are structurally dissimilar. For the state aspect, we build on step-indexed Kripke logical relations (Ahmed et al., 2009; Ahmed, 2004). Specifically, our logical relations are indexed by a "step"—a standard device for inductive proofs that counts how many steps of computation the logical relation is good for (Ahmed, 2006; Appel & McAllester, 2001). Owing to the simplicity of our pre and post-conditions, we do not need state-dependent worlds as in some other work (Neis et al., 2011; Turon et al., 2013).

To show the effectiveness of our approach, we implemented a bidirectional type-checker for ARel. Thanks to the simplified form of our pre and post-conditions, we can solve the constraints generated by the type-checker using SMT solvers. The type-checker also uses a restricted number of heuristics to address some of the nondeterminism coming from the relational reasoning, and the array operations. In order to evaluate the performance of our implementation, we consider a broad set of examples showcasing different challenges for relational cost analysis in programs manipulating arrays.

3.2 CONTRIBUTIONS

Our overarching contribution lies in extending relational cost analysis to higher-order functional-imperative programs. Our specific contributions are:

- ARel, a type system for relational cost analysis of functional-imperative programs with mutable arrays.
- A design for lightweight (relational) refinements of array-based computations.
- A soundness proof for our type system via a new step-indexed logical relation.
- An implementation of ARel, based on bidirectional type checking, which we use to type check several functional-imperative examples, details in Chapter 10.

3.3 AREL THROUGH EXAMPLES

We illustrate the key ideas behind ARel through two simple examples. The first one is the `map` example as we see in Chapter 2, but in its imperative version, replacing the list with the mutable array.

3.3.1 Inplace Map

Consider the following imperative map function, named `mapi`, taking as input a pure function f, a mutable array a, an index k and the array's length n. For all $i \in [k,n]$, the function replaces the current value in the ith cell of a with $f(a[i])$, thus performing a

destructive update.

```
fix mapi (f).λa.λk.λn.
  if k ≤ n then
    let {x} = read a k in
    let {_} = updt a k (f x) in
    mapi f a (k+1) n
  else
    return ()
```

The biggest difference between the imperative map mapi and the standard functional one map is the array-based operations. The expression (read a k) returns the element at index k in the array a, and (updt a k v) updates the index k in a to v. Our language uses a state monad to isolate all side-effects like array reads and updates, so (read a k) and (updt a k v) are actually expressions of monadic types, also called *computations*. The construct (let $\{x\} = t_1$ in t_2) is monadic sequencing, often called "bind". The construct return t is the monadic unit.

Let us now consider the problem of establishing an upper bound on the *relative* cost of two runs of mapi that use the *same* function f, but now the input lists are changed to two *different* arrays a. Intuitively, the relative cost should be upper bounded by the product of the maximum variation in the cost of the function f (across inputs) and the number of indices in the range $[k, n]$ at which the two arrays differ.

To support reasoning about two runs as in this example, ARel supports *relational types* that ascribe a pair of related values or related expressions in the two runs. Relational types are written τ. In general, when we say $x : \tau$, we mean that the variable x may be bound to two different values in the two runs, but these two values will be related by the type τ. Specifically, $x : \tau_1 \to \tau_2$ means that x can be bound to two different

functions f_1, f_2 in the two runs, satisfying the property that for any two arguments v_1, v_2 of relational type τ_1, the two expressions $f_1\ v_1, f_2\ v_2$ have relational type τ_2. Naturally, ARel also supports *unary types*, denoted A, that ascribe only one value or expression in a single run.

To establish the relative cost of mapi, we first need a way to represent that the *same* function f will be given to mapi in both runs. To this end, ARel offers the type annotation $\Box$. The type $\Box\tau$ relates expressions in two runs that are (syntactically) equal and are additionally related at the relational type τ. Note that $\Box$ is a *relational refinement*: It refines the relation defined by the underlying type τ. Specifically, the relational typing assumption $f : \Box(\tau_1 \to \tau_2)$ means that, in the two runs, f will be bound to two copies of the *same* function, say f, that given arguments v_1, v_2 related at type τ_1, give expressions $f\ v_1$ and $f\ v_2$ related at type τ_2. In our example, if the array's elements have type τ, the type of f would be $\Box(\tau \to \tau)$.

Next, we need to represent the maximum possible variation in the cost of applying f. The possible variation in the cost can be seen as an *effect*, and the cost of applying a function can be seen as the effect associated with the body of the function, in particular. Hence as is common in effect systems Nielson & Nielson (1999), we can record the possible variation in cost by means of a refinement of the function type. ARel offers a refinement of this kind. We write $\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2$ to represent two functions of relational type $\tau_1 \to \tau_2$, the relative cost of whose bodies is upper bounded by D. Accordingly, if f's cost can vary by D, its type can be further refined to $\Box(\tau \xrightarrow{\mathrm{diff}(D)} \tau)$.

Next, we need a way to specify *where* the arrays given as inputs to mapi in the two runs differ. There are various design choices for supporting this. One obvious but problematic option would be to refine the type of an array itself, to specify where the two ascribed arrays differ across two runs. However, this design quickly runs into an issue:

An update on the arrays might be different in the two runs, so it might change the arrays' *type*. This would be highly unsatisfactory since we don't expect the type of an array to change due to an update; in particular, this design would not satisfy (semantic or syntactic) type preservation.

Consequently, we use a different approach inspired by relational Hoare logics: We provide a relational refinement type $\{P\}\ \exists\vec{\gamma}.\tau^{\mathrm{diff}(D)}\ \{Q\}$ for monadic expressions that manipulate arrays. The number D is an upper bound on the the relative cost of the computations in two runs, similar to the one we have in function types, and τ is the relational type relating the pair of *pure* values returned by the computations. The *precondition* P specifies for each pair of related arrays in scope where (at which indices) these corresponding arrays are allowed to differ *before* the execution of the computations, while the *postcondition* Q specifies where these arrays may differ *after* the completion of the computations. More specifically, P and Q are lists of annotations of the form $\gamma \to \beta$, where γ is a *static name* identifying a pair of related arrays and β is a set of indices where this pair of arrays *may* differ in the two runs. In other words, at any index not in β, the corresponding arrays must be the same in the two runs. Note that even at indices in β, the corresponding values must be related at τ, but our type system includes types that do not force equality of the related values. One such type is $U(A,B)$ that only insists that the left and right values have (unary) types A and B, without requiring any other relation between them. (The existentially quantified $\vec{\gamma}$ in $\{P\}\ \exists\vec{\gamma}.\tau^{\mathrm{diff}(D)}\ \{Q\}$ is the list of static names of arrays that are allocated during the computation.)

To be more concrete, let us consider an example. If $x : \Box\tau$, in other words, x is the same in two runs, and b represents a pair of arrays associated with the static name γ, then two equal update operations $(\mathtt{updt}\ b\ 5\ x)$ can be given the relational monadic type

$\{\gamma \to \beta\}\ \exists\_.\overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \to (\beta \setminus \{5\})\}$, for any β.[2] This type means that if the corresponding arrays b in two runs differ at the set of indices β before (updt b 5 x) executes in two runs, then afterwards this pair of arrays can still differ in the indices β *except* at the index 5, which has been overwritten by the same value x (indicated by the box type $\Box\tau$). If we replace the assumption $x : \Box\tau$ with $x : \tau$, so that x may differ in the two runs, then the type of (updt b 5 x) relative to itself would be $\{\gamma \to \beta\}\ \exists\_.\overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \to (\beta \cup \{5\})\}$, indicating that the arrays may differ at index 5 after the update (even if they did not differ at that index before the update).

To make this reasoning formal, we need a way to tie the static names γ appearing in computation types to specific arrays. To this end, we refine the type of arrays to include γ. We also refine the type of arrays to track the length of the array. This doubly refined type is written $\mathsf{Array}_\gamma[N]\ \tau$—a pair of arrays of length N each, identified statically by the name γ, and carrying elements related pointwise at type τ. Finally, we refine integers very precisely: The type int$[n]$ is the *singleton type* containing only the integer n in both runs. The n in the type is a static representation of the runtime values the type ascribes.

With all these components we can now represent the relative cost of mapi that we are interested in by the judgment:

$$\vdash \mathsf{mapi} \ominus \mathsf{mapi} \lesssim 0 : \begin{array}{l} \forall D :: \mathbb{R}.\Box(\tau \overset{\mathrm{diff}(D)}{\longrightarrow} \tau) \to \forall k, n, \gamma, \beta.(k \le n) \supset \\ \mathsf{Array}_\gamma[n]\ \tau \to \mathrm{int}[k] \to \mathrm{int}[n] \to \{\gamma \to \beta\}\ \exists\_.\overset{\mathrm{diff}(|\beta \cap [k,n]| * D)}{\mathrm{unit}}\ \{\gamma \to \beta\} \end{array}$$

This typing means that mapi relates to itself in the following way. Consider two runs of mapi with the same function f of relative cost D (type $\Box(\tau \overset{\mathrm{diff}(D)}{\longrightarrow} \tau)$), two arrays of static length n, statically named γ (type $\mathsf{Array}_\gamma[n]\ \tau$), two indices, both k (type int$[k]$), and two lengths, both n (type int$[n]$). Then, the two runs return computations with the following

[2] As usual, _ represents a variable whose name is unimportant.

relational property: If the two arrays differ at most at indices β before they are passed to mapi, then they differ at most at the same positions after the computations and the relative cost of the two computations is upper bounded by $|\beta \cap [k,n]| * D$, i.e., the number of positions in the range $[k,n]$ at which the arrays may differ times D. This is exactly the expected relative cost because, at positions where the arrays are equal, f will have the same cost in the two runs (we are assuming language-level determinism here). To those readers who are familiar with effect system, when the input arguments are instantiated, for example, $\vdash \mathsf{mapi}\ f\ a\ k\ n \ominus \mathsf{mapi}\ f\ a\ k\ n \lesssim D : \{\gamma \to \beta\}\ \exists\_.\mathrm{unit}^{\mathrm{diff}(D')}\ \{\gamma \to \beta\}$. We can think of D as the relative cost of computation not related to our mutable arrays, while D' reveals the relative cost bounds for those involving arrays. Note that the variables D, k, n, γ and β are universally quantified in the type above. Also, note how γ links the input array to the β in the pre and postcondition of the computation type. Notice we omit the use of expression such $\Lambda.t$ and $t[\,]$ in the mapi function, which is used for the introduction and elimination of index variables k, n, β in the type. These expressions will be discussed in the later chapter.

Consider now a slightly different situation where *different* functions f may be passed to mapi in the two runs. Suppose that the relative cost of the bodies of the two fs passed is upper bounded by D, i.e., f has the type $\tau \xrightarrow{\mathrm{diff}(D)} \tau$ (without the prefix $\Box$). In this case, the relative cost of the two runs of mapi can only be upper bounded by $|[k,n]| * D$, since even at indices where the arrays agree, the cost of applying the two different fs may differ by as much as D. Moreover, the final arrays may differ in all positions in the range $[k,n]$. This is formalized in the following, second relational type for mapi.

$$\vdash \mathsf{mapi} \ominus \mathsf{mapi} \lesssim 0 : \begin{array}{l} \forall D :: \mathbb{R}.(\tau \xrightarrow{\mathrm{diff}(D)} \tau) \to \forall k, n, \gamma, \beta.(k \le n) \supset \\ \mathsf{Array}_\gamma[n]\ \tau \to \mathrm{int}[k] \to \mathrm{int}[n] \to \{\gamma \to \beta\}\ \exists\_.\mathrm{unit}^{\mathrm{diff}((n-k)*D)}\ \{\gamma \to \beta \cup [k,n]\} \end{array}$$

3.3.2 BooleanOr

Next, we describe how high-level reasoning about relative cost is internalized in the typing. ARel supports two kinds of typing modes: *relational typing* as shown in the imperative map example above, and *unary typing* which supports traditional (unary) min- and max-cost analysis for a single run of a program. We will introduce these modes formally in Chapter 4 but here we want to show with the following example how they can be meaningfully combined via an extended fix rule r-fix-ext in Figure 4.6.

```
fix BoolOr (a). λk.λn.
  if k < n then
    let {x} = read a k in
    if x then
      return true
    else
      BoolOr a (k + 1) n
  else
    return false
```

This function, given as input an array of booleans a, an index k and the array's length n tells whether there exists an element in a with index $\geq k$ and value *true*.

Given two arbitrary arrays a in two runs, a simple upper bound on the relative cost of BoolOr is $(n-k) * c$ where c is the cost of one iteration. This is because in one run we can find an element with value $true$ AT position k, and so the computation can return immediately, while in the other run, we may not find any such element, and would need to visit every element of the array with its index greater than k. This kind of high-level reasoning corresponds to a worst-case, best-case analysis of the two individual runs.

ARel supports this kind of reasoning by supporting worst-case, best-case (unary) cost analysis in unary mode, and by means of a rule r-switch, presented formally in Figure 4.7. allows us to derive a relational typing from two unary typings, with relative cost equal to the difference between the max and the min costs of the unary typings.

However, this kind of reasoning does not account for the case where the two input arrays have a meaningful relation, e.g., they may be equal in some positions. In such cases, a better upper bound on the relative cost would be expressed in terms of the first index i (if any) where the two arrays differ. That is, we could have the upper bound $(n - i) * c$. Showing this upper bound formally is more involved. We first need to proceed by case analysis on whether the element x we are reading at each step is the same in the two runs or not. Case analysis in ARel is provided by the rule r-split, presented in Figure 4.6. Using this rule we can consider the two cases separately in typing the subexpression $\mathsf{if}\ x\ \mathsf{then}\ (\mathsf{return}\ true)\ \mathsf{else}\ \mathsf{BoolOr}\ a\ (k+1)\ n$.

If x is the same in the two runs, there is no difference in cost because we either return $true$ in both runs, or we perform the recursive call in both runs. In case the two x's differ, we must switch to the unary analysis of the two individual runs since in one run we will return immediately while in the other we will make a recursive call, so there is no way to continue reasoning relationally. Hence, in order to derive the required upper bound on the overall relative cost, we need to have information about the *unary* type of BoolOr. However, since we started by trying to type the body of BoolOr relationally, the standard fixpoint rule only allows us to assume its *relational* type.

One solution to this impasse is to automatically transform relational types of variables in context to unary types when switching from relational to unary reasoning. This approach was adopted by Çiçek et al. (2017a) for analyzing pure functional programs but it provides only trivial lower and upper bounds (0 and ∞) on the costs of function

variables in the context during the unary analysis. In our example here, this approach yields the trivial upper bound ∞, which is not what we want.

To allow for more precise analysis, ARel includes a new rule r-fix-ext which we introduce formally in Chapter 4. This rule allows us to assume the result of a *unary* typing of two recursive functions, when typing their bodies *relationally*. With this rule, we can use the (assumed) relational type of BoolOr and its unary type in typing the subexpression BoolOr $a\,(k+1)\,n$. Hence, we can conclude the inductive step and assign the precise relative cost $(n-i)*c$ to BoolOr.

CHAPTER 4

Type System of ARel

In this section, we present the syntax, semantics, and the type system of ARel. We can think about ARel as composed of two parts, a *pure* part, inspired by Çiçek et al. (2017a), which allows one to reason about the difference in the execution costs of two pure functional programs and an *impure* (imperative) part, which allows one to reason about the array operations.

4.1 SYNTAX

We summarize ARel's syntax in Figure 4.1. The term language underlying ARel is a simply typed λ-calculus with recursion and constructs for mutable arrays.

4.1.1 Pure Constructs

Most of the pure constructs (the ones in black in Figure 4.1) are similar to the ones one can find in a pure standard functional language. We have variables x, natural numbers n and real numbers r, unit $()$, lambda abstraction $\lambda x.t$, and recursion $\mathsf{fix}\, f(x).t$, and application $t_1\, t_2$. There are also the introduction and elimination constructs for product, sum, and inductive list. Additionally, ARel has some constructs to deal with type level information: the constructs $\Lambda.t$ and $t\,[]$, $\mathsf{pack}\, t$ and $\mathsf{unpack}\, t_1\, \mathsf{as}\, x\, \mathsf{in}\, t_2$ corresponding to the introduction and elimination of universal and existential types over index terms, respectively, and a term construct $\mathsf{celim}\, t$ which is used to eliminate type level constraints. We will discuss the latter constructs further when we introduce types.

Index terms	$I, L,$	::=	$i \mid b \mid n \mid r \mid I_1 + I_2 \mid I_1 - I_2 \mid I_1 \cdot I_2 \mid \lceil I \rceil \mid \lfloor I \rfloor \mid log_2 I \mid$
	$U, D,$		$I_1^{I_2} \mid \mathsf{min}(I_1, I_2) \mid \mathsf{max}(I_1, I_2) \mid \frac{I_1}{I_2} \mid \sum_{i=I_1}^{I_n} I \mid$
	α, β		$\{I_i\}_{i \in K} \mid \beta \cup \beta \mid \beta \setminus \beta \mid \beta \cap \beta$
Terms	t	::=	$x \mid n \mid r \mid () \mid \lambda x.t \mid \mathsf{fix}\, f(x).t \mid t_1\ t_2 \mid \langle t_1, t_2 \rangle \mid \pi_1(t) \mid$
			$\pi_2(t) \mid \mathsf{case}\ (t, x.t_1, y.t_2) \mid \mathsf{inl}\ t \mid \mathsf{inr}\ t \mid \mathsf{nil} \mid \mathsf{cons}(t_1, t_2) \mid$
			$\Lambda.t \mid t\ [] \mid (\ \mathsf{case}\ t\ \mathsf{of\ nil} \rightarrow t_1 \mid h :: tl \rightarrow t_2) \mid \mathsf{pack}\ t \mid$
			$\mathsf{celim}\ t \mid \mathsf{unpack}\ t_1\ \mathsf{as}\ x\ \mathsf{in}\ t_2 \mid \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \mid$
			$\mathsf{return}\ t \mid \mathsf{alloc}\ t_1\ t_2 \mid \mathsf{updt}\ t_1\ t_2\ t_3 \mid \mathsf{read}\ t_1\ t_2 \mid \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2$
Values	v	::=	$n \mid l \mid r \mid () \mid \lambda x.t \mid \mathsf{fix}\, f(x).t \mid \langle v_1, v_2 \rangle \mid \mathsf{inl}\ v \mid \mathsf{inr}\ v \mid$
			$\mathsf{nil} \mid \Lambda.t \mid \mathsf{pack}\ v \mid \mathsf{return}\ t \mid \mathsf{alloc}\ t_1\ t_2 \mid \mathsf{updt}\ t_1\ t_2\ t_3 \mid$
			$\mathsf{read}\ t_1\ t_2 \mid \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2$
Unary types	A	::=	$\mathsf{unit} \mid \mathsf{int} \mid \forall i{::}S.\ A \overset{\mathsf{exec}(L,U)}{\mid} \exists i :: S.\ A \mid A \overset{\mathsf{exec}(L,U)}{\longrightarrow} A \mid \mathsf{list}[I]\ A \mid$
			$A_1 \times A_2 \mid A_1 + A_2 \mid C \& A \mid C \supset A \mid \mathsf{int}[I]$
			$\mid \mathsf{Array}_\gamma[I]\ A \mid \{P\}\ \overset{\mathsf{exec}(L,U)}{\exists \vec{\gamma}.A}\ \{Q\}$
Relat. types	τ	::=	$\mathsf{unit} \mid \mathsf{int} \mid \forall i{::}S.\ \tau \overset{\mathsf{diff}(D)}{\mid} \exists i{::}S.\ \tau \mid \tau \overset{\mathsf{diff}(D)}{\longrightarrow} \tau \mid \mathsf{list}^\alpha[I]\ \tau \mid$
			$\tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid C \& \tau \mid C \supset \tau \mid U(A_1, A_2) \mid \Box\tau \mid \mathsf{int}[I] \mid$
			$\{P\}\ \overset{\mathsf{diff}(D)}{\exists \vec{\gamma}.\tau}\ \{Q\} \mid \mathsf{Array}_\gamma[I]\ \tau$
Constraints	C	::=	$I_1 = I_2 \mid I_1 < I_2 \mid \neg C \mid I_1 \in I_2$
Heaps	H	::=	$[] \mid [l \rightarrow z] \mid H_1 \uplus H_2$
Arrays	z	::=	$[v_1, \ldots, v_m]$
Assertions	P, Q	::=	$\mathsf{empty} \mid \gamma \rightarrow \beta \mid P \star Q$
Sorts	S	::=	$\mathbb{R} \mid \mathbb{N} \mid \mathbb{B} \mid \mathbb{P} \mid \mathbb{L}$

Figure 4.1: Syntax of ARel.

Loc Env	$\Sigma ::= \emptyset \mid \Sigma, \gamma :: \mathbb{L}$	**Sort Env.**	$\Delta ::= \emptyset \mid \Delta, i :: S$
Unary Type Env.	$\Omega ::= \emptyset \mid \Omega, x : A$	**Relational Type Env.**	$\Gamma ::= \emptyset \mid \Gamma, x : \tau$
Constraint Env.	$\Phi_a ::= \top \mid C \wedge \Phi_a$		

Unary Typing Judgment	$\Sigma; \Delta; \Phi_a; \Omega \vdash^{U}_{L} t : A$
Relational Typing Judgment	$\Sigma; \Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$
Sorting Judgment	$\Sigma; \Delta \vdash I :: S$
Unary Subtyping	$\Sigma; \Delta; \Phi_a \models A_1 \sqsubseteq A_2$
Relational Subtyping	$\Sigma; \Delta; \Phi_a \models \tau_1 \sqsubseteq \tau_2$
Unary Type Well-formedness	$\Sigma; \Delta; \Phi_a \vdash A$ wf
Relational Type Well-formedness	$\Sigma; \Delta; \Phi_a \vdash \tau$ wf
Constraint Well-formedness	$\Delta \vdash C$ wf
Heap Well-formedness	$\Omega \vdash H$ wf
Assertion Well-formedness	$\Sigma; \Delta \vdash P$ wf

Figure 4.2: Notations of ARel.

4.1.2 Impure Constructs

The impure part at the term level includes the constructs to deal with arrays, we highlight them in blue in Figure 4.1. We have constructs for *allocating* arrays ($\mathsf{alloc}\, t_1\, t_2$, where t_1 specifies the number of array cells to be allocated, and t_2 the initial value to be stored in each array cell), for *reading* from arrays ($\mathsf{read}\ t_1\ t_2$, where t_1 specifies the array to read from, and t_2 the position in the array to read from), and for *updating* arrays ($\mathsf{updt}\ t_1\ t_2\ t_3$, where t_1 specifies the array to be updated, t_2 the position in the array to be updated, and t_3 the value to be used for the update). All imperative (array-manipulating) constructs are confined to a monad. The constructs $\mathsf{return}\, t$ and $\mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2$ are the usual return and bind of the monad. We do not distinguish between impure expressions and pure expressions at the syntactic level; this distinction is enforced by types. Impure expressions (expressions of monadic types) are values but can be *forced* using a special forcing semantics that we describe below. Finally, arrays are referenced through

$$\frac{}{v \Downarrow^{0,0} v}\ \text{e-val} \qquad \frac{t \Downarrow^{c,k} v}{\text{inl}\ t \Downarrow^{c,k} \text{inl}\ v}\ \text{e-inl} \qquad \frac{t \Downarrow^{c_1,k_1} \text{inl}\ v \qquad t_1[v/x] \Downarrow^{c_2,k_2} v_r}{\text{case}\ (t, x.t_1, y.t_2) \Downarrow^{c_1+c_2+c_{\text{case}},k_1+k_2+1} v_r}\ \text{e-casel}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} \lambda x.t' \quad t_2 \Downarrow^{c_2,k_2} v \quad t'[v/x] \Downarrow^{c_3,k_3} v_1}{t_1\ t_2 \Downarrow^{c_1+c_2+c_3+c_{\text{app}},k_1+k_2+k_3+1} v_1}\ \text{e-app}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} \text{fix}\, f\, x.t' \quad t_2 \Downarrow^{c_2,k_2} v \quad t'[\text{fix}\, f\, x.t'/f][v/x] \Downarrow^{c_3,k_3} v_1}{t_1\ t_2 \Downarrow^{c_1+c_2+c_3+c_{\text{fapp}},k_1+k_2+k_3+1} v_1}\ \text{e-fix}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} v \quad v; H \Downarrow_f^{c_2,k_2} v_1; H_1 \quad t_2[v_1/x] \Downarrow^{c_3,k_3} v_2 \quad v_2; H_1 \Downarrow_f^{c_4,k_4} v_3; H_2}{\text{let}\{x\} = t_1 \text{ in } t_2; H \Downarrow_f^{c_1+c_2+c_3+c_4+c_{\text{let}},k_1+k_2+k_3+k_4+1} v_3; H_2}\ \text{f-bind}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} l \quad t_2 \Downarrow^{c_2,k_2} n \quad t_3 \Downarrow^{c_3,k_3} v}{\text{updt}\ t_1\ t_2\ t_3; H \Downarrow_f^{c_1+c_2+c_3+c_{\text{updt}},k_1+k_2+k_3+1} (); H(l)[n] \leftarrow v}\ \text{f-updt}$$

$$\frac{t \Downarrow^{c,k} v}{\text{return}\ t; H \Downarrow_f^{c+c_{\text{ret}},k+1} v; H}\ \text{f-ret} \qquad \frac{t_1 \Downarrow^{c_1,k_1} l \quad t_2 \Downarrow^{c_2,k_2} n_2 \quad H(l)[n] = v}{\text{read}\ t_1\ t_2; H \Downarrow_f^{c_1+c_2+c_{\text{read}},k_1+k_2+1} v; H}\ \text{f-read}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} n \quad t_2 \Downarrow^{c_2,k_2} v \quad z = [\overbrace{v,\ldots,v}^{n}] \quad l \text{ fresh}}{\text{alloc}\, t_1\ t_2; H \Downarrow_f^{c_1+c_2+c_{\text{alloc}},k_1+k_2+1} l; H \uplus [l \rightarrow z]}\ \text{f-alloc}$$

Figure 4.3: Selection of rules for pure evaluation and forcing evaluation of ARel.

locations, $l \in Loc$. Although locations do not appear in programs, they are needed for the evaluation, so they are included in the syntax. Types can contain index terms. We use $iVar$ to denote the set of index term variables and $iLoc$ to denote the set of index term variables that refer to locations statically. These static identifiers for locations are denoted γ and belong to a specific sort that is written $\mathbb{L}$.

4.2 OPERATIONAL SEMANTICS

4.2.1 The Definition of Heap

We define a cost-annotated, big-step operational semantics for our language. Part of this semantics is based on manipulation of heaps, also described in Figure 4.1. We represent heaps as mappings $H = [l_1 \to z_1, \ldots, l_n \to z_n]$ from memory locations to concrete arrays $z = [v_1, \ldots, v_n]$. The notation $H(l)[n] = v$ expresses that the value v is stored in the heap H in the array pointed by the location pointer l at the index n, the notation $H(l)[n] \leftarrow v$ represents the heap H where the array pointed by l is updated with the value v at index n, and the notation $H_1 \uplus H_2$, in the spirit of separation logic, denotes a disjoint union of the heaps H_1 and H_2.

4.2.2 Pure and Impure Evaluation

We have two kinds of evaluation judgments: *pure evaluation* $t \Downarrow^{c,k} v$ states that the (pure) expression t evaluates to the value v with cost c, using k steps, while *forcing evaluation* $t; H \Downarrow_f^{c,k} v; H'$ states that the impure expression t can be forced in the heap H to the value v and to the updated heap H' with cost c, consuming k steps. We give a selection of the evaluation rules in Figure 4.3. We include all the rules for the impure part and a selection for the pure part, all the other rules can be found in Appendix A.

Steps k are a proof artifact, needed only in our soundness proof that relies on a step-indexed logical relation in Chapter 5. We count a unit step for every elimination and monadic construct. Readers may ignore steps for now. The costs c are what we seek to upper bound (relatively) using our type system and are, therefore, important. At every elimination form or monadic construct, the semantics adds a construct-dependent cost. For example, the cost c_{app} appearing in the rules stands for the cost of an application. By changing these costs and setting some of them to 0, we can get different cost models.

In a word, our type system is parametric in the costs of individual constructs.

The pure evaluation rules are mostly standard. They track how the cost and the steps change when a pure expression evaluates. The rule e-val says that a value v evaluates itself with no cost and in zero steps. The rules e-inl and e-casel describe the evaluation of the introduction and elimination terms for the sum type (we only show the rule for inl as the rule for inr is similar), where in addition we record the cost c_{case} for the case elimination, and we increment the steps. Rules e-app and e-fix are similar but for the two application cases, the latter one includes recursion.

The forcing evaluation rules are used to evaluate impure (monadic) expressions manipulating heaps (arrays). The rule f-ret forces the evaluation of an expression $\mathsf{return}\, t$, representing the unit of the monad, by evaluating the underlying pure expression t using the pure evaluation semantics. The cost consists of the cost of the pure evaluation of t and the constant cost c_{ret} for the monadic return. As one would expect, the unit of the monad wraps a pure expression into a monadic computation, and it accounts for the cost of this operation by means of the cost c_{ret}. The rule f-bind combines pure and forcing evaluations in order to fully evaluate a monadic let. This rule shows how an impure computation (involving arrays) is evaluated in our semantics. We first evaluate an expression t_1 to a value v using the pure evaluation semantics. Then, we force evaluate this value v to a value v_1 of the underlying type. We can then perform the substitution and evaluate the resulting expression $t_2[v_1/x]$ to a value v_2 using the pure evaluation semantics. The resulting value v_2 is then force evaluated to another value v_3 which is also the result of the overall let expression. The heap also changes accordingly. c_{let} accounts for additional costs associated with the bind operation itself.

The rule f-read forces the evaluation of a read expression in the heap H by first evaluating the heap location l from which to read, then the index of the element n to read,

and then returning the value stored in l at index n. The rule f-updt forces the evaluation of an update expression in a similar way; it returns a unit value. Finally, the rule f-alloc forces the evaluation of an alloc expression by creating a new array with the length specified by the first argument and initial values specified by the second argument, and by allocating it in the heap at a new location l, which is returned.

It is worth noticing that, in the forcing evaluation rules, all the subexpressions evaluate using the pure semantics to values of base types, since in our language we only allow arrays of base types.

4.3 INDEX TERMS AND CONSTRAINTS

To have a precise cost analysis, we need some extra information about the data structures from the corresponding types. The normal types do not support this and adding index terms to the types is a nice way to do that. In the spirit of DML (Xi & Pfenning, 1999), types are indexed using *static* index terms that are defined in Figure 4.1. Index terms include natural numbers and real numbers, which we use to express size and cost information, respectively. We equip index terms with several operations including ceiling, floor, log, min, and max. Moreover, we have special index terms β denoting (potentially infinite) sets of natural numbers, representing sets of array indices. We also have operations over β (we identify these with blue underlines in Figure 4.1). In relational types, the set β tracks where the two related arrays may differ (as explained earlier), while in unary types, they represent the write permissions for the array. We will return to this point later after we explain the types. We can explicitly form a set through an indexed set comprehension of the form $\{I_i\}_{i \in K}$, where $K \subseteq \mathbb{N}$, and we can take the union $\beta_1 \cup \beta_2$ or the difference $\beta_1 \setminus \beta_2$ of two sets β_1, β_2. We use index terms also in the relational type for lists to specify the number of values that differ pointwise in the lists across

two runs. We denote index terms with this specific meaning with the metavariable α.

We consider only *well-sorted* index terms. To this end, we have a sorting judgment of the form $\Delta \vdash I :: S$ in Figure 4.2 where Δ is a *sort environment*, assigning sorts to index variables, and S is a sort. The detailed sorting rules can be found in the appendix A, in particular, Figure A.11. Our language has five sorts: $\mathbb{N}$ for natural numbers, used for sizes of arrays, lists, and other data structures; $\mathbb{R}$ for real numbers used to express costs; $\mathbb{B}$ for booleans; $\mathbb{P}$ for sets of natural numbers as just described; and, $\mathbb{L}$ for static names of arrays (sorting rules can be found in Appendix A). As a convention, we use L, U to represent the unary minimum and maximum costs, and D to denote a maximum relational cost (L, U, and D are always of sort $\mathbb{R}$). Index terms can also appear in constraints C, which express equalities and inequalities over index terms and can be used to represent conditional typing.

4.4 UNARY AND RELATIONAL TYPES

In ARel, we have two typing modes: *unary* and *relational*. This separation is also reflected at the type level where we have two different type languages: *unary types* A and *relational types* τ.

4.4.1 Unary Types

Unary types describe values (expressions) in a single run. They use index terms to represent size information, as in the case of the type $\mathsf{list}[I]\ A$ where I represents the size of the list, and costs, as in the case of the type $A \xrightarrow{\mathrm{exec}(L,U)} A'$ where L and U represent lower and upper bounds on the cost of the body of the function being typed. The cost can also be seen as an effect. We also have other basic types, as well as types for products and sums. Index terms are also used for size in basic types like integers, booleans, etc., for costs in

universal quantifications, and in constraints.

Besides the pure types we just discussed, we have a type for arrays and a type for impure computations (with blue underlines in Figure 4.1). The type $\mathsf{Array}_\gamma[I]\ A$ is the type of arrays of length I containing objects of type A. We limit A to base types like int, bool, etc. to simplify our technical development. In particular, we do not support arrays of arrays here. The annotation γ associates a *static name* to the array that is typed. This static name can be used to refer to the array in other types.

Impure expressions are typed with monadic types. In our case, a monadic unary type is a cost-annotated Hoare triple type of the shape $\{P\}\ \exists\vec{\gamma}.A\ \{Q\}^{\mathrm{exec}(L,U)}$, which is inspired by Hoare Type Theory (Nanevski et al., 2008). Assertions P, Q are sets $\{\gamma_1 \to \beta_1, \ldots, \gamma_n \to \beta_n\}$ assigning to each static location γ_i a set of natural numbers β_i describing (writing) permissions. The idea is that the array named γ_i can be written only at indices in β_i (although it may read anywhere). The domain of Q may be larger than the domain of P, to account for newly allocated arrays. The index terms L and U are lower and upper bounds on the execution cost of the (forcing) evaluation of the typed expression.

Additionally, index terms are used in constraints which can appear in types of the shape $C \supset A$, which reads as "the constraint C implies A", and of the shape $C \& A$, which reads as "A and the constraint C is true". These constraints support conditional typing and they are quite useful to restrict the properties of arrays. For example, the type $I > 0\ \&\ \mathsf{Array}_\gamma[I]\ A$ ascribes non-empty arrays, and the constraint C in the type $C \supset A$ can be used to restrict array index bounds, as we will see in the examples in Chapter 6.

4.4.2 Relational Types

Relational types ascribe pairs of expressions, and, as we will see in Chapter 5, they are interpreted as sets of pairs of values in our model. In relational types, index terms carry

not just size information but also information about the *relation* between the two expressions, between their inputs, and between their outputs. The type $\mathsf{list}^{\alpha}[I]\ \tau$ ascribes a pair of lists, each of length I, whose elements are pointwise related at the type τ. Importantly, the relational refinement α specifies an upper bound on the number of positions at which the corresponding elements may differ. In other words, the two lists must have equal elements in at least $I-\alpha$ positions, even if τ allows them to be unrelated. The type $\mathsf{int}[I]$ represents pairs of integers both of which are equal to I. In arrow types $\tau \overset{\mathrm{diff}(D)}{\longrightarrow} \tau'$, the index term D represents an upper bound on the relative cost of the execution of the underlying pair of functions on two inputs related at τ.

Given a pair of unary types A_1, A_2, the relational type $U(A_1, A_2)$ represents arbitrary pairs of expressions of types A_1, A_2, respectively. This offers a way to relate in a relational type two "unrelated" values. As explained in the imperative mapi example in Section 3.3.1, we also have a comonadic relational type $\Box\tau$ which represents pairs of expressions of type τ which are syntactically equal, and we have corresponding subtyping rule s-r-T and s-r-D in Figure 4.10. In particular, $\Box U(A_1, A_2)$ is the diagonal sub-relation of $U(A_1, A_2)$, i.e. $\Box U(A_1, A_2)$ is the subset of $U(A_1, A_2)$ where the left and right components are equal.

The relational type $\mathsf{Array}_{\gamma}[I]\ \tau$ is similar to the unary array type but it represents two arrays, each of length I, containing values related at τ pointwise. The static name γ is the name for both arrays. As we will see in Chapter 5, our logical relation relates γ to two arrays in two different heaps. Relational impure computations, illustrated in the imperative mapi example, are typed using a relational cost-annotated monadic type of the form $\{P\}\ \exists\vec{\gamma}.\tau\ \{Q\}^{\mathrm{diff}(D)}$. This looks similar to the unary type $\{P\}\ \exists\vec{\gamma}.A\ \{Q\}^{\mathrm{exec}(L,U)}$ but it means something different. In the relational type, the pre and postconditions P, Q of form $\{\gamma_1 \to \beta_1, \ldots, \gamma_n \to \beta_n\}$ have a relational interpretation, namely, that (for all i) the two

arrays named γ_i must carry equal values at all positions not in β (and the values must be related at τ). At positions in β, the values must still be related at τ, but they need not be equal (unless τ forces this, e.g. with a prefix $\Box$). D is an upper bound on the relative cost of forcing the evaluation of the two impure expressions.

As usual, we consider only types that are well-formed. We have well-formedness judgments $\Sigma;\Delta;\Phi_a \vdash A\ \mathsf{wf}$ for unary types, and $\Sigma;\Delta;\Phi_a \vdash \tau\ \mathsf{wf}$ for relational types in appendix A. The unary one can be found in Figure A.12 and the relational well-formedness judgments are presented in Figure A.13 and Figure A.14. Here, Σ is a *location environment* listing the locations that can appear in the rest of the judgment, Δ is a sort environment listing all free index variables and Φ_a is a *constraint environment* to support conditional typing. Additionally, we also define the well-formedness of the heaps in Figure A.15.

4.5 UNARY AND RELATIONAL TYPING

4.5.1 Unary Typing Judgments

ARel's unary typing uses the judgment form

$$\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A$$

where t is an expression, Σ is a location environment, Δ is a sort environment listing all the free index variables as mentioned before, Φ_a is a constraint environment, Ω is a *unary type environment* assigning unary types to variables, A is a unary type, and L and U are index terms representing a lower bound and an upper bound on the cost of evaluating t, respectively. We give a selection of the typing rules for deriving unary typing judgments in Figure 4.4 (for the pure constructs) and Figure 4.5 (for the impure

$$\frac{}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} n : \mathrm{int}[n]}\ \text{u-int} \qquad \frac{\Sigma;\Delta;\Phi_a;x:A, f:A \overset{\mathrm{exec}(L,U)}{\longrightarrow} B, \Omega \vdash^{U}_{L} t : B}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{fix}\, f(x).t : A \overset{\mathrm{exec}(L,U)}{\longrightarrow} B}\ \text{u-fix}$$

$$\frac{\Omega(x) = A}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} x : A}\ \text{u-var} \qquad \frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : A \overset{\mathrm{exec}(L,U)}{\longrightarrow} B \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : A}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+U+U_{\mathsf{app}}}_{L_1+L_2+L+L_{\mathsf{app}}} t_1\ t_2 : B}\ \text{u-app}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A_1 \qquad \Sigma;\Delta;\Phi_a \vdash A_2\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} \mathrm{inl}\ t : A_1 + A_2}\ \text{u-inl}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A_2 \qquad \Sigma;\Delta;\Phi_a \vdash A_1\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} \mathrm{inr}\ t : A_1 + A_2}\ \text{u-inr}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t : A_1 + A_2 \\ \Sigma;\Delta;\Phi_a;x:\ A_1,\Omega \vdash^{U_2}_{L_2} t_1 : A \qquad \Sigma;\Delta;\Phi_a;y:\ A_2,\Omega \vdash^{U_2}_{L_2} t_2 : A\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+U_c}_{L_1+L_2+L_c} \mathrm{case}\ (t, x.t_1, y.t_2) : A}\ \text{u-case}$$

$$\frac{}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{nil} : \mathsf{list}[0]\ A}\ \text{u-nil}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : \mathsf{list}[I]\ A}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \mathsf{cons}(t_1, t_2) : \mathsf{list}[I+1]\ A}\ \text{u-cons}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A \qquad \Sigma;\Delta;\Phi_a \vDash A \sqsubseteq A' \qquad \Delta;\Phi_a \vDash U \le U' \qquad \Delta;\Phi_a \vDash L' \le L}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t : A'}\ \text{u-sub}$$

Figure 4.4: Selection of pure unary typing rules of ARel.

part).

4.5.1.1 Unary Typing Rules for Pure Constructs

Selected rules for pure expressions in Figure 4.4 are similar to the typing rules proposed by Çiçek et al. (2017a), the main difference is that our rules have one more environment Σ, used to store the locations in the heap. Rules u-int and u-var are relatively self-explanatory. Rules u-fix and u-app are similar to the ones available in classical effect systems, where the lower bound and upper bound of the cost of the (recursive) function body is recorded in the function type. If we look at the rule u-fix, the lower bound L and U should be provided in the arrow type for the function $\mathsf{fix}\, f\,(x).t$ such that the hypothesis on the function body t can be proved. The cost of function application considers also the cost of executing the function body. We present these rules here to show how the costs change in typing. Rules u-inl and u-inr are dual. Notice that the introduction of the sum type $A_1 + A_2$ requires well-formedness for the type that is introduced in the sum. The cost remains the same as its subexpression. Rule u-case for eliminating the sum type requires the same upper and lower bound in both branches. Rules u-nil and u-cons are similar and specify the size of lists. Finally, rule u-sub, internalizes weakening for the upper and lower bounds and subtyping.

4.5.1.2 Unary Typing Rules for Impure Constructs

The rules for the unary typing of impure expressions are presented in Figure 4.5. Since costs of operations like reading and writing memory are *variable* on most architectures, these rules rely on given upper- and lower-bounds on the cost of each operation. For example, L_{read} and U_{read} denote the minimum and maximum cost of reading a heap location, respectively. The costs $U_{\mathsf{let}}, U_{\mathsf{alloc}}, L_{\mathsf{read}}, L_{\mathsf{updt}}$ are similar.

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_L^U t : A \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L,U)} \mathsf{return}\ t : \{P\}\ \exists\gamma.A\ \{P\}}\ \text{u-ret}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \{P\}\ \exists\vec{\gamma}_1.A\ \{P'\}^{\mathrm{exec}(L,U)} \qquad \Sigma;\Delta,\vec{\gamma}_1;\Phi_a;\Omega,x:A \vdash_{L_2}^{U_2} t_2 : \{P'\}\ \exists\vec{\gamma}_2.B\ \{Q\}^{\mathrm{exec}(L',U')}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L+L'+L_1+L_2+L_{\mathsf{let}},U+U'+U_1+U_2+U_{\mathsf{let}})} \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 : \{P\}\ \exists\vec{\gamma}_1,\vec{\gamma}_2.B\ \{Q\}}\ \text{u-bind}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \mathrm{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : A \qquad \gamma\ \text{fresh} \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L_1+L_2+L_{\mathsf{alloc}},U_1+U_2+U_{\mathsf{alloc}})} \mathsf{alloc}\ t_1\ t_2 : \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \to \mathbb{N}\}}\ \text{u-alloc}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \mathsf{Array}_\gamma[I]\ A \\ \gamma \in dom(P) \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a \vDash I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L_1+L_2+L_{\mathsf{read}},U_1+U_2+U_{\mathsf{read}})} \mathsf{read}\ t_1\ t_2 : \{P\}\ \exists\_.A\ \{P\}}\ \text{u-read}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \mathsf{Array}_\gamma[I]\ A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : \mathrm{int}[I'] \\ \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_3}^{U_3} t_3 : A \qquad \Delta;\Phi_a \vDash I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Delta;\Phi_a \vDash I' \in \beta\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L_1+L_2+L_3+L_{\mathsf{updt}},U_1+U_2+U_3+U_{\mathsf{updt}})} \mathsf{updt}\ t_1\ t_2\ t_3 : \{P \star \gamma \to \beta\}\ \exists\_.\mathrm{unit}\ \{P \star \gamma \to \beta\}}\ \text{u-updt}$$

Figure 4.5: Selection of impure unary typing rules of ARel.

Rules u-ret and u-bind type the unit and the bind of the monad, respectively. They combine the different costs and assertions in the monadic type, using a style similar to separation logic. For example, the assertion $P_1 \star P_2$ corresponds to disjoint parts of the heap. The rule for allocations, u-alloc, introduces a new static location γ and creates a new monadic type whose postcondition assigns to γ all the natural numbers ($\mathbb{N}$), indicating that the continuation has the permission to write all positions of the array. Additionally, like all other rules, this rule also adds a cost accounting for the forcing of the allocation. Finally, note that the upper and lower bounds on the judgment are 0. This is because $\mathsf{alloc}\, t_1\, t_2$ is a value. Cost arises only when the impure expression $\mathsf{alloc}\, t_1\, t_2$ is forced; this is accounted for in the cost annotations of the monadic type. The rule for reading, u-read, merely checks that the index being read is within the array bounds. The rule for updating, u-updt, also performs a similar check but, in addition, it also requires that the updated index is contained in the permissions available for the array in the precondition.

4.5.2 Relational Typing Judgments

ARel's relational typing uses the judgment form

$$\Sigma; \Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$$

Here, t_1 and t_2 are two expressions, Σ, Δ, and Φ_a are environments similar to the ones used by unary typing judgments, Γ is a *relational type environment* assigning relational types to variables, τ is a relational type for t_1, t_2, and D is an index term representing an *upper bound* on the relative cost of evaluating t_1 and t_2, i.e., $\mathrm{cost}(t_1) - \mathrm{cost}(t_2)$.

As we notice in the relational judgments, two programs, the left one and the right one are related, which naturally results in two kinds of relational typing rules: *synchronous*

rules which relate two structurally similar programs, and *asynchronous rules* for arbitrary programs. We first, present a selection of the *pure* typing rules which include both synchronous rules (Figure 4.6) and asynchronous rules (Figure 4.7), these are again inspired by the work of Çiçek et al. (2017a). Then, we present a selection of the *impure* typing rules which support relational cost analysis for arrays. Again, we distinguish the synchronous rules (Figure 4.8) and the asynchronous rules (Figure 4.9).

4.5.2.1 Relational Synchronous Typing Rules for Pure Constructs

We describe the selected synchronous rules for pure expressions that we present in Figure 4.6. The rule r-int assigns to two copies of the same integer n, a singleton type int$[n]$. The rules r-var, r-fix and r-app are the relational counterpart of the rules for variables, function abstraction, and application we saw in the unary part. The main difference is that we give an upper bound to the relative cost, rather than a lower and upper bound on the execution cost of a single expression. The rules r-inl, r-inr, and r-case talk about the introduction and elimination of the relational sum type. In the elimination rule r-case, notice that we require the relative cost D_2 of the two branches to be the same, similar to what we did for the unary rule. Rule r-split allows reasoning by cases on any constraint in the constraint environment. Rule r-sub is the relational version of the rule u-sub, it allows weakening the upper bound on the relative cost D as well as subtyping. Rule r-nc is the introduction rule for $\Box$-ed types. Briefly, t can be related to itself at the type $\Box\tau$ when t relates to itself at type τ and, additionally, all variables in the context morally have $\Box$-ed types. The latter ensures that variables can only be substituted by equal terms. In this case, the relative cost is 0. This rule allows us to assume that given a pair of functions whose type is refined with $\Box$, if we apply them to the same argument, we have two executions following the same path, and at the same cost. We discussed this intuition

$$\frac{}{\Sigma;\Delta;\Phi_a;\Gamma \vdash n \ominus n \lesssim 0 : \mathrm{int}[n]}\ \text{r-int} \qquad \frac{\Gamma(x) = \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash x \ominus x \lesssim 0 : \tau}\ \text{r-var}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_1}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \lesssim D + D_1 + D_2 : \tau_2}\ \text{r-app}$$

$$\frac{\Sigma;\Delta;\Phi_a;x:\tau, f :\tau \xrightarrow{\mathrm{diff}(D)} \sigma, \Gamma \vdash t_1 \ominus t_2 \lesssim D : \sigma}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{fix}\, f(x).t_1 \ominus \mathsf{fix}\, f(x).t_2 \lesssim 0 : \tau \xrightarrow{\mathrm{diff}(D)} \sigma}\ \text{r-fix}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_1 \qquad \Sigma;\Delta;\Phi_a \vdash \tau_2\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathrm{inl}\ t \ominus \mathrm{inl}\ t' \lesssim D : \tau_1 + \tau_2}\ \text{r-inl}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_2 \qquad \Sigma;\Delta;\Phi_a \vdash \tau_1\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathrm{inr}\ t \ominus \mathrm{inr}\ t' \lesssim D : \tau_1 + \tau_2}\ \text{r-inr}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D_1 : \tau_1 + \tau_2 \\ \Sigma;\Delta;\Phi_a;\Gamma, x:\tau_1 \vdash t_1 \ominus t_1' \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma, y:\tau_2 \vdash t_2 \ominus t_2' \lesssim D_2 : \tau\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash\ \mathrm{case}\ (t, x.t_1, y.t_2) \ominus\ \mathrm{case}\ (t', x.t_1', y.t_2') \lesssim D_1 + D_2 : \tau}\ \text{r-case}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t \lesssim D : \tau \qquad \forall x \in dom(\Gamma).\Sigma;\Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t \lesssim 0 : \Box\tau}\ \text{r-nc}$$

$$\frac{\Sigma;\Delta;\Phi_a, C;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \Sigma;\Delta;\Phi_a, \neg C;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau}\ \text{r-split}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash t \ominus t \lesssim D : \tau \qquad \Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma;\Delta;\Phi_a \models D \leq D'}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t \lesssim D' : \tau'}\ \text{r-sub}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;x:\tau_1, f :\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, \Gamma, f : U(A_1, A_2) \vdash t_1 \ominus t_2 \lesssim D : \tau_2 \\ \Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash_0^0 \mathsf{fix}\, f(x).t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash_0^0 \mathsf{fix}\, f(x).t_2 : A_2\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{fix}\, f(x).t_1 \ominus \mathsf{fix}\, f(x).t_2 \lesssim 0 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2}\ \text{r-fix-ext}$$

Figure 4.6: Selection of pure relational synchronous typing rules of ARel.

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_2}_{L_2} t_2 : A_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim U_1 - L_2 : U(A_1, A_2)}\ \text{r-switch}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t_2 \ominus t_2' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \ominus t_2' \lesssim U_1 + D_2 + c_{\mathsf{lt}} : \tau}\ \text{r-lt-t}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t_1' : A_1' \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1', A_1') \vdash t_2 \ominus t_2' \lesssim D_2 : \tau'}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus \mathsf{let}\ x = t_1'\ \mathsf{in}\ t_2' \lesssim D_2 - L_1 - c_{\mathsf{lt}} : \tau'}\ \text{r-t-lt}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \overset{\text{exec}(L,U)}{\longrightarrow} A_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : U(A_1, A_2')}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_2' \lesssim U_1 + U + D_2 + c_{app} : U(A_2, A_2')}\ \text{r-app-t}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t : A_1 + A_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t_1 \ominus t' \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma, y : U(A_2, A_2) \vdash t_2 \ominus t' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash\ \text{case}\ (t, x.t_1, y.t_2) \ominus t' \lesssim U_1 + D_2 + c_{case} : \tau}\ \text{r-case-t}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t' : A_1 + A_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t \ominus t_1' \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma, y : U(A_2, A_2) \vdash t \ominus t_2' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus\ \text{case}\ (t', x.t_1', y.t_2') \lesssim D_2 - L_1 - c_{case} : \tau}\ \text{r-t-case}$$

Figure 4.7: Selection of pure relational asynchronous typing rules of ARel.

behind the rule r-nc when we looked at the first relational type of mapi in Section 3.3.1. Rule r-fix-ext types fixpoint expressions relationally. This rule also requires *unary* typing for the two functions, which are established in separate premises. We require these additional premises so that we can use the information provided by the unary typing to establish the relative cost. In other words, this rule introduces a weak form of *intersection types* in the environment which can be used in combination with the rule r-switch (Figure 4.7) to give precise bounds on relative cost.

4.5.2.2 *Relational Asynchronous Typing Rules for Pure Constructs*

We present a selection of the pure asynchronous rules in Figure 4.7. Rule r-switch allows switching from relational reasoning about t_1 and t_2, as witnessed by a relational typing judgment in the conclusion, to unary reasoning about the two terms, independently, as witnessed by the two unary typing judgments in the premises. Notice that the relational type in the conclusion is the embedding of the two unary types without any meaningful relation ($U(A_1, A_2)$). The rule uses an erasure map $|\Gamma|_i$ from relational environments to unary environments(1 for left and 2 for the right), whose definition can be found in the appendix. Importantly, the relative cost in the conclusion is the difference of the unary costs in the premises. Rule r-lt-t relates a pure let binding expression to an arbitrary expression. In this rule, we use the metavariable c_{lt} to denote the cost of a let elimination. (This is different from the cost c_{let} of the monadic bind, which we discussed earlier.) Notice that one of the assumptions in this rule, the one for the expression t_1, is a unary typing judgment. This is needed in order to provide guarantees on the typability of t_1 and to provide the cost of evaluating it, which is used in the bound on the relative cost in the conclusion of the rule.

The rule r-t-lt is dual to r-lt-t – it relates an arbitrary expression with a standard let. Notice that while the rule r-lt-t uses the upper bound on the unary cost of t_1, the rule r-t-lt uses the lower bound. Rule r-app-t relates a function application with an arbitrary expression, while rule r-case-t relates a case expression with an arbitrary expression and r-t-case has the opposite direction. Also in these rules, we use some unary typing assumptions to guarantee typability and to provide unary costs that are used in giving upper bounds on the relative costs. We also have dual rules which we present in Appendix A.

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \{P\}\ \exists \vec{\gamma}_1.\tau\ \{P'\}^{\mathrm{diff}(D)} \qquad \Sigma;\Delta,\vec{\gamma}_1:\vec{\mathbb{L}};\Phi_a;\Gamma,x:\tau \vdash t_2 \ominus t_2' \lesssim D_2 : \{P'\}\ \exists \vec{\gamma}_2.\sigma\ \{Q\}^{\mathrm{diff}(D')}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \lesssim 0 : \{P\}\ \exists \vec{\gamma}_1\vec{\gamma}_2 : \sigma\ \{Q\}^{\mathrm{diff}(D+D'+D_1+D_2)}}\ \text{r-bind}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ t_1 \ominus \mathsf{return}\ t_2 \lesssim 0 : \{P\}\ \exists \_.\tau\ \{P\}^{\mathrm{diff}(D)}}\ \text{r-ret}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathrm{int}[I] \quad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau \quad \gamma\ \mathsf{fresh} \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \lesssim 0 : \{P\}\ \exists \gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}^{\mathrm{diff}(D_1+D_2)}}\ \text{r-alloc}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathrm{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \square\tau \qquad \gamma\ \mathsf{fresh} \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \lesssim 0 : \{P\}\ \exists \gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \emptyset\}^{\mathrm{diff}(D_1+D_2)}}\ \text{r-allocb}$$

$$\frac{\gamma \in dom(P) \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \quad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \lesssim 0 : \{P\}\ \exists \_.\tau\ \{P\}^{\mathrm{diff}(D_1+D_2)}}\ \text{r-read}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad \Delta;\Phi_a \models I' \notin \beta}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \lesssim 0 : \{P \star \gamma \to \beta\}\ \exists \_.\square\tau\ \{P \star \gamma \to \beta\}^{\mathrm{diff}(D_1+D_2)}}\ \text{r-readb}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 : \tau \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \lesssim 0 : \{P \star \gamma \to \beta\}\ \exists \_.\mathrm{unit}\ \{P \star \gamma \to \beta \cup \{I'\}\}^{\mathrm{diff}(D_1+D_2+D_3)}}\ \text{r-updt}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 : \square\tau \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \lesssim 0 : \{P \star \gamma \to \beta\}\ \exists \_.\mathrm{unit}\ \{P \star \gamma \to \beta \setminus \{I'\}\}^{\mathrm{diff}(D_1+D_2+D_3)}}\ \text{r-updtb}$$

Figure 4.8: Selection of monadic synchronous relational typing rules of ARel.

4.5.2.3 *Relational Synchronous Typing Rules for Impure Constructs*

Figure 4.8 shows a selection of relational synchronous typing rules pertaining to the monadic constructs and arrays. Rules r-ret and r-bind relationally type the return and bind of our monad. The rules introduce the trivial relational Hoare-triple and combine two relational Hoare triples by sequencing, respectively. In particular, the rule r-bind uses the style of separation logic, similarly to what we have for the corresponding unary rule.

For each operation on arrays, we have two rules, one that is general and the other that works under some assumption about equality of arguments in the two runs. Consider, for example, the rules r-alloc and r-allocb for relationally typing the alloc construct. The rules are similar, e.g., both create a new static name γ for the two allocated arrays, and both account for relative costs very similarly. However, r-allocb applies only when the expressions initializing the two arrays are related at a $\Box$-ed type (the second premise). As a result, it is guaranteed that the arrays allocated in the two runs will have equal values in all positions. This is reflected in the assertion $\gamma \rightarrow \emptyset$ in the postcondition of the monadic type in the rule, which says that there are no locations where the newly allocated arrays (named γ) can differ. In contrast, the rule r-alloc does not require the initializing expressions to be related at a $\Box$-ed type, but it has $\gamma \rightarrow \mathbb{N}$ in the postcondition, meaning that the two arrays may differ anywhere after the allocation. A similar difference arises in the rules r-read and r-readb for relationally typing the construct read. In r-readb, the read index I' must not be in the β of the array being read in the precondition; as a result, the values read must be equal in the two runs. Hence, the resulting type has a $\Box$ on it. r-read is similar, but, here, there is no requirement that I' is not in the β, so two different values may be read, and there is no $\Box$ on the result type. The rules r-updt and r-updtb for updt follow the principle of alloc: In r-updtb, the values being

written in the two runs are known to be equal (via a $\Box$-ed type), so the index I' that is updated is removed from β in the postcondition. This is not the case in r-updt, where it must be added to β, since the two values at index I' might differ after the update. In all these rules, the premise $\Delta;\Phi_a \vDash I' \leq I$ denotes a constraint entailment which reads as follows: under the substitution of all the variables in the index environment Δ, under the assumption of the constraints in Φ_a, the constraint $I' \leq I$ holds. This premise guarantees that the array bound is not exceeded. We omit here the rules for deriving this judgment since they are standard.

Finally, note that all monadic rules "propagate" relative costs from the premises to the monadic types. This is similar to the unary rules; the difference is that the costs propagated here are relative, whereas the unary type system propagates unary lower and upper bounds.

It is worth stressing that the set of γs in any pre or postcondition must be written down explicitly, i.e., we have not introduced sophisticated constructors (like set comprehension) for pre and postconditions. This means that we cannot meaningfully specify monadic computations that allocate a data-dependent number of arrays. This hasn't been a problem for our examples, and we believe an extension to lift this restriction is possible by adding language-level constructors and elimination rules for assigning γ. While this change would make our approach more flexible and more expressive, it would also put an additional burden on the programmer.

4.5.2.4 Relational Asynchronous Typing Rules for Impure Constructs

Figure 4.9 shows the two asynchronous rules r-bind-t and r-t-bind, relating a monadic binding construct and an arbitrary expression. We explain only the rule r-bind-t which relates the monadic binding construct $\mathsf{let}\{x\} = t_1 \;\mathsf{in}\; t_2$ to an arbitrary expression t_2' (the

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : \{P_1\}\ \overset{\mathrm{exec}(L,U)}{\exists \vec{\gamma}_1 : A_1}\ \{Q_1\} \\ \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_2}_{L_2} t_2' : \{P_2\}\ \overset{\mathrm{exec}(L',U')}{\exists \vec{\gamma}_1 : A_1'}\ \{Q_2\} \qquad dom(P) = dom(P_1) \\ \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t_2 \ominus t_2' \lesssim D_2 : \{P \sqcup P_1\}\ \overset{\mathrm{diff}(D')}{\exists \vec{\gamma}_1.\tau}\ \{Q\}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus t_2' \lesssim -L_2 : \quad \overset{\mathrm{diff}(U_1+U+(D_2+U_2)+D'+c_{let})}{\{P\}\ \exists \vec{\gamma}_1.\tau}\ \{Q\}}\ \text{r-bind-t}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t_1' : \{P_1\}\ \overset{\mathrm{exec}(L,U)}{\exists \vec{\gamma}_1 : A_1'}\ \{Q_1\} \\ \Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_2}_{L_2} t_2 : \{P_2\}\ \overset{\mathrm{exec}(L',U')}{\exists \vec{\gamma}_1 : A_1}\ \{Q_2\} \qquad dom(P) = dom(P_1) \\ \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1', A_1') \vdash t_2 \ominus t_2' \lesssim D_2 : \{P \sqcup P_1\}\ \overset{\mathrm{diff}(D')}{\exists \vec{\gamma}_1.\tau'}\ \{Q\}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \lesssim U_2 : \quad \overset{\mathrm{diff}(D'+(D_2-L_2)-L_1-L-c_{\mathrm{let}})}{\{P\}\ \exists \vec{\gamma}_1.\tau'}\ \{Q\}}\ \text{r-t-bind}$$

Figure 4.9: Selection of monadic asynchronous relational typing rules of ARel.

rule r-t-bind is its dual and it can be understood similarly).

The first premise of the rule r-bind-t requires a unary typing for the monadic expression t_1. This typing has two kinds of costs: the lower bound L_1 and upper bound U_1 for the unary execution cost of t_1, and the lower bound L and upper bound U for the execution cost of the resulting computation evaluated from t_1, this is embedded in the monadic type of t_1. The second premise requires a unary typing for the monadic expression t_2'. This gives us an upper bound U_2 on the cost of evaluating this expression. The premise $dom(P) = dom(P_1)$ requires that the execution of the computation resulting from the expression t_1 can only affect arrays that appear in both P_1 and P. Finally, the last premise requires relating the subexpression t_2 to t_2' with the relative cost upper bounded by D_2 under the assumption that the values substituted for the variable x are related at the type $U(A_1, A_1)$. Notice that this is the weakest requirement in terms of types that we can have.

Additionally, this typing judgment also gives us the upper bound D' on the relative cost for executing the two computations resulting from evaluating the two expressions t_2 and t_2'. To put the information of the unary and relational typing together we use the precondition $P \sqcup P_1$ in this premise, where the operation $\sqcup$ gives a precondition where a name γ which is used in P, e.g. $\gamma \to \beta \in P$, and in P_1, e.g. $\gamma \to \beta_1 \in P_1$, now points to the union of the two corresponding sets, e.g. $\gamma \to \beta \cup \beta_1 \in P \sqcup P_1$. The precondition in the unary monadic type of t_1 in the first premise provides the indices where the computation associated with t_1 has writing permission. So, intuitively, P_1 provides the indices that may be overwritten when executing t_1. Hence, we want to use this information when relating the expression t_2 and t_2' because t_1 is supposed to be executed by then. The conclusion of the rule uses all the cost information we discussed to compute an upper bound on the relative cost of the two expressions, where, as usual, we use the metavariable c_{let} to denote the cost of evaluating the monadic binding construct. The bounds on the relative cost here deserve some discussion. Following the definition, and observing that the monadic let is a value, we have that the relative cost of the two expressions is bound by $-L_2$. However, we want also to have a bound of the relative cost of forcing the evaluation of the two expressions, since this must be recorded in the monadic type. The upper bound is $U_1 + U + (D_2 + U_2) + D' + c_{let}$, where $U_1 + U$ upper bounds the cost of forcing the evaluation of t_1, D_2 upper bounds the difference in cost of evaluating t_2 and t_2' to value, U_2 upper bounds the cost of evaluating t_2 to its value, D' upper bounds the difference in cost of forcing the evaluation of the values obtained by evaluating t_2 and t_2' respectively. The costs in the rule r-t-bind can be obtained in a similar way.

One can also design similar asynchronous rules for the other monadic constructs. However, the syntactic forms of the other constructs considerably constrain their asyn-

chronous typing rules, making the scope of application of such rules rather narrow. For this reason, we do not commit to the design of such rules here.

4.5.3 Subtyping

Subtyping is important in ARel. It serves several purposes. First, as in all refinement type systems, subtyping equates terms up to refinement, e.g., it allows replacing int[$2+i$] with int[5] under the constraint $i = 3$. Second, specific to cost analysis, subtyping allows weakening costs, e.g., the relational type $\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2$ can be subtyped to $\tau_1 \xrightarrow{\mathrm{diff}(D')} \tau_2$ when $D \leq D'$ since the D on the arrow is an upper bound on relative cost. Third, subtyping allows "massaging" of modalities $\Box$ and U, e.g., $\Box\tau$ can be subtyped to τ. Finally, specific to the monadic types, subtyping allows weakening of pre and postconditions in monadic types. The first three purposes of subtyping in ARel are relatively standard (e.g., see Çiçek et al. (2017a)) and we will only briefly introduce them, we only describe the last use here at length. The unary and relational subtyping judgments have the forms $\Sigma;\Delta;\Phi_a \models A_1 \sqsubseteq A_2$ and $\Sigma;\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2$, respectively. Figure 4.10 and Figure 4.11 show these selected subtyping rules. The notation $P \subseteq P'$ means that $P = \{\gamma_1 \to \beta_1, \gamma_2 \to \beta_2, \ldots, \gamma_n \to \beta_n\}$, $P' = \{\gamma_1 \to \beta'_1, \gamma_2 \to \beta'_2, \ldots, \gamma_n \to \beta'_n\}$, and $\forall i \in \{1,\ldots,n\}.\beta_i \subseteq \beta'_i$.

The rules s-u-arrow and s-r-arrow subtype unary and relational function types, respectively. The rule s-r-w allows weakening from the relational type τ to its weak version $U(|\tau|_1, |\tau|_2)$, where $|\cdot|_i$ is used to convert from a relational type to a unary type. When $i = 1$, this construct projects the left side of the relational type; when $i = 2$ it projects its right side (see the Appendix for the definitions of these projections). The rules s-r-list and s-r-array subtype relational list and array types, respectively. They impose requirements on lengths. Rule s-r-ua allows weakening the relational type $U(A_1, A_2)$ to $U(A'_1, A'_2)$ if we know that A_1 and A_2 are subtypes of A'_1 and A'_2, respectively. As we have seen before,

$$\frac{\Sigma;\Delta;\Phi_a \models A_1' \sqsubseteq A_1 \qquad \Sigma;\Delta;\Phi_a \models A_2 \sqsubseteq A_2' \qquad \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'}{\Sigma;\Delta;\Phi_a \models A_1 \overset{\text{exec}(L,U)}{\longrightarrow} A_2 \sqsubseteq A_1' \overset{\text{exec}(L',U')}{\longrightarrow} A_2'}\ \text{s-u-arrow}$$

$$\frac{\Sigma;\Delta;\Phi_a \models A_1 \sqsubseteq A_1' \qquad \Sigma;\Delta;\Phi_a \models A_2 \sqsubseteq A_2'}{\Sigma;\Delta;\Phi_a \models U(A_1, A_2) \sqsubseteq U(A_1', A_2')}\ \text{s-r-ua} \qquad \frac{}{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq U(|\tau|_1, |\tau|_2)}\ \text{s-r-w}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box\tau \sqsubseteq \tau}\ \text{s-r-T} \qquad \frac{}{\Sigma;\Delta;\Phi_a \models \Box\tau \sqsubseteq \Box\Box\tau}\ \text{s-r-D}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau_1' \sqsubseteq \tau_1 \qquad \Sigma;\Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta;\Phi_a \models D \leq D'}{\Sigma;\Delta;\Phi_a \models \tau_1 \overset{\text{diff}(D)}{\longrightarrow} \tau_2 \sqsubseteq \tau_1' \overset{\text{diff}(D')}{\longrightarrow} \tau_2'}\ \text{s-r-arrow}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box(\tau_1 \overset{\text{diff}(D)}{\longrightarrow} \tau_2) \sqsubseteq \Box\tau_1 \overset{\text{diff}(0)}{\longrightarrow} \Box\tau_2}\ \text{s-r-bd}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma;\Delta;\Phi_a \models I = I' \ \Sigma;\Delta;\Phi_a \models \alpha \leq \alpha'}{\Sigma;\Delta;\Phi_a \models \mathsf{list}^{\alpha}[I]\ A \sqsubseteq \mathsf{list}^{\alpha'}[I']\ A'}\ \text{s-r-list}$$

Figure 4.10: Selection of subtyping rules of ARel, part 1.

$$\frac{}{\Sigma;\Delta;\Phi_a \models U(A_1 \overset{\text{exec}(L,U)}{\longrightarrow} A_2, A_1' \overset{\text{exec}(L',U')}{\longrightarrow} A_2') \sqsubseteq U(A_1, A_1') \overset{\text{diff}(U-L')}{\longrightarrow} U(A_2, A_2')}\ \text{s-r-execdiff}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma;\Delta;\Phi_a \models I{=}I' \qquad \Sigma;\Delta;\Phi_a \models \gamma{=}\gamma'}{\Sigma;\Delta;\Phi_a \models \mathsf{Array}_{\gamma}[I]\ \tau \sqsubseteq \mathsf{Array}_{\gamma'}[I']\ \tau'}\ \text{s-r-array}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a \models A \sqsubseteq A' \qquad \vec{\gamma}_1 \subseteq \vec{\gamma}_2 \qquad \Delta;\Phi_a \models L'{\leq}L \qquad \Delta;\Phi_a \models U{\leq}U' \\ \Sigma,\Delta;\Phi_a \models P{\subseteq}P' \qquad \Sigma,\vec{\gamma}_1;\Delta;\Phi_a \models Q'{\subseteq}Q\end{array}}{\Sigma;\Delta;\Phi_a \models \{P\}\ \exists\vec{\gamma}_1.A\ \{Q\} \sqsubseteq \{P'\}\ \overset{\text{exec}(L,U)}{\exists\vec{\gamma}_2}.A'\ \{Q'\}}\ \text{s-um}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad \Delta;\Phi_a \models D{\leq}D' \\ \Sigma,\vec{\gamma}_1;\Delta;\Phi_a \models Q{\subseteq}Q' \qquad \Sigma;\Delta;\Phi_a \models P'{\subseteq}P \qquad \vec{\gamma}_1 \subseteq \vec{\gamma}_2\end{array}}{\Sigma;\Delta;\Phi_a \models \{P\}\ \overset{\text{diff}(D)}{\exists\vec{\gamma}_1}.\tau\ \{Q\} \sqsubseteq \{P'\}\ \overset{\text{diff}(D')}{\exists\vec{\gamma}_2}.\tau'\ \{Q'\}}\ \text{s-rm}$$

$$\frac{\beta_i' = \beta_i \cup T_i \cup T_i'}{\begin{array}{c}\Sigma;\Delta;\Phi_a \models U(\{\gamma_i \to T_i\}\ \overset{\text{exec}(L,U)}{\exists\vec{\gamma}_1'}.A_1\ \{Q_1\}, \{\gamma_i \to T_i'\}\ \overset{\text{exec}(L',U')}{\exists\vec{\gamma}_2'}.A_2\ \{Q_2\}) \\ \overset{\text{diff}(U-L')}{\sqsubseteq} \{\gamma_i \to \beta_i\}\ \exists\vec{\gamma}_1,\vec{\gamma}_2.U(A_1,A_2)\ \{\gamma_i \to \beta_i'\}\end{array}}\ \text{s-rum}$$

Figure 4.11: Selection of subtyping rules of ARel, part 2.

the modality $\Box$ applied to a relational type τ requires the two components related by the type τ to be identical. In fact, $\Box$ has a comonadic flavor and it follows the standard comonadic rules s-r-T and s-r-D. Rule s-r-bd can be read as follows: When two equal functions (of the $\Box$-ed relational function type $\Box(\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2)$) are given equal arguments (type $\Box\tau_1$), the results are equal and the relative cost is 0. We used this rule implicitly in the mapi example in Section 3.3.1.

Now, we present the subtyping rules involving monadic types, the first rule s-um allows subtyping on the unary monadic type. It says that we can subtype by weakening the costs, adding more (write) permissions to the precondition and removing permissions from the postcondition, as manifest in the premises $P \subseteq P'$ and $Q' \subseteq Q$. Rule s-rm similarly allows subtyping on the relational monadic type. This rule says that we can subtype by weakening the relative cost, making the precondition more precise and the postcondition less precise, where P' is more precise than P when P' tells us more about which values are equal. In particular, $\gamma \to \beta$ is more precise than $\gamma \to \beta'$ when $\beta' \subseteq \beta$. This is why the premises of s-rm check $P' \subseteq P$ and $Q \subseteq Q'$. Note that the checks on P, P' and Q, Q' are dual in the two rules. This is because the meanings of the pre(post)condition in the unary and relational monadic types are completely different. As a reminder, in the unary setting, the indices in β refer to the writing permission while it refers to the may-different relation of two arrays.

Finally, rule s-rum allows subtyping from modal U applied to two unary monadic types to a single relational monadic type. This rule is best read as follows: If we have two computations that modify an array (associated with the static name γ_i) at positions in T_i and T_i', respectively (left side of $\sqsubseteq$), then running them on two arrays that agree at all positions outside the set β will result in two arrays that agree at all positions outside the set $\beta \cup T_i \cup T_i$' (right side of $\sqsubseteq$). This is because we cannot ignore the fact that the indices

in the set $T_i \cup T_i$' may be overwritten during the execution.

CHAPTER 5

Metatheory and Soundness of ARel

The soundness of ARel is proved through a step-indexed logical relation for its types. The logical relation is a widely accepted proof technique for program properties such as strong normalization, type equivalence, and type safety. It is defined by induction on types. The interpretation of types is depicted by the relations of values, which are well designed to catch the property of interest. Naturally, it can be easily extended to prove properties of type and effect systems when the effects are properly expressed in these relations. The prior work RelCost (Çiçek et al., 2017a) adopts such concept and shows its soundness using a logical relation model which has two interpretations of both the unary and relational types. These two kinds of types interact at the type $U(A_1, A_2)$ as we have seen before. The logical relation model of ARel is more complicated than the one of RelCost due to the existence of mutable arrays, which requires us to consider the heap in the model, through the interpretation of monadic types introduced in Section 4.4.

In the rest of this chapter, we first present the logical relation model of ARel, by showing the unary and relational interpretation of types, and the impure part will be our focus. Then, we present the metatheory of our system, including the proof of the soundness of ARel.

5.1 UNARY INTERPRETATION OF AREL TYPES

The *value interpretation* $[\![A]\!]_{g,k}$ of a unary type A is, a set of values. Also, this interpretation is indexed by a "step-index" $k \in \mathbb{N}$, which is merely a proof device for induction (Ahmed, 2006; Appel & McAllester, 2001). The step-index counts the "steps" in our operational semantics in Section 4.2. Importantly, the interpretation is also indexed by

a *world* g mapping from static names γ to triples (l, n, A) specifying the location, the length of the array, and the *syntactic* type of the elements of the array named γ. This world is the corresponding part we add to reason about the mutable arrays. Technically, we are defining a *Kripke logical relation*, and the world g is a so-called Kripke world (Neis et al., 2011; Turon et al., 2013). We give the clauses defining the value interpretation of unary types in Figure 5.1.

The value interpretations of standard type constructors like pairs and functions are also standard. One thing about the interpretation of functions is that we use step-index to make the recursion of the function $\mathsf{fix}\, f(x).t$ to be well-found. The value interpretation of an array type, $[\![\mathsf{Array}_\gamma[I]\ A]\!]_{g,k}$, is a set of locations. A location l is in this set if $g(\gamma)$ is (l, A, I), i.e., the element type and length for γ in the world g match those in the array type and the location l corresponds to γ.

The value interpretation of monadic types relies on a heap relation $H \models_{g,k} P$, which is defined in Figure 5.3. This heap relation means that the assertion P – which could be a pre-or postcondition from a unary monadic type – holds for the heap H at world g at step k. The relation checks that for every (l, A, n) in the range of g, every element of the array named l in H is in the interpretation of type A. All the static names γ in P will be checked so that there is a mapping in the world g so that $g(\gamma) = (l, A, n)$. To break the cyclicity between the definition of the heap relation and value interpretation, A is interpreted at the smaller step-index $k-1$ in the heap relation.

Back to the unary monadic type, the value interpretation $[\![\{P\}\ \exists\vec{\gamma}.A\ \{Q\}]\!]^{\mathrm{exec}(L,U)}_{g,k}$ is a set of monadic values v that when forced in a heap H validating the precondition P, yield a heap H_1 validating the postcondition Q. We define $v; H \Downarrow^{c,k}_{f}$ to show that the evaluation of v in the heap H will not diverge. Additionally, the interpretation only allows those computations v that update arrays at locations for which the precondition P asserts

$$
\begin{array}{ll}
[\![\mathrm{int}]\!]_{g,k} & = \{\, n \mid n \in \mathbb{N} \,\} \\
[\![\mathrm{unit}]\!]_{g,k} & = \{\, () \,\} \\
[\![\mathrm{int}[I]]\!]_{g,k} & = \{\, n \mid I = n \,\} \\
[\![\mathsf{Array}_\gamma[I]\ A]\!]_{g,k} & = \{\, l \mid I = n \ \wedge\ g(\gamma) = (l, A, n) \,\} \\
\overset{\mathrm{exec}(L,U)}{[\![A \longrightarrow A']\!]_{g,k}} & = \left\{ \mathsf{fix}\, f(x).t \;\middle|\; \begin{array}{l} \forall k' < k,\ g' \supseteq g.\ \forall v.\ v \in [\![A]\!]_{g',k'} \Longrightarrow \\ \qquad t[v/x][\mathsf{fix}\, f(x).t/f] \in [\![A']\!]^{e}_{g',k',(L,U)} \end{array} \right\} \\
\overset{\mathrm{exec}(L,U)}{[\![\{P\}\ \exists \vec{\gamma}.A\ \{Q\}]\!]_{g,k}} & = \left\{ v \;\middle|\; \begin{array}{l} \forall g_1 \supseteq g.\ \forall k_1 \le k,\ k_2 < k_1.\ \forall c.\ \forall H. \\ H \vDash_{g_1,k_1} P \wedge\ v; H \Downarrow^{c,k_2}_{f} \Longrightarrow \\ \qquad \exists g_2 \supseteq g_1.\ \exists H_1,\ v_1,\ \vec{\gamma}.\ v; H \Downarrow^{c,k_2}_{f} v_1; H_1 \ \wedge \\ L \le c \le U \ \wedge\ H_1 \vDash_{g_2,k_1-k_2} Q \ \wedge\ v_1 \in [\![A]\!]_{g_2,k_1-k_2} \ \wedge \\ \Big(\exists n.\ P = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \ \wedge \\ \quad \forall i \in [1,n].\ g(\gamma_i) = (l_i, A, m) \Longrightarrow \\ \quad \forall j. H[l_i][j] \neq H_1[l_i][j] \Longrightarrow j \in T_i \Big) \end{array} \right\} \\
\overset{\mathrm{exec}(L,U)}{[\![\forall i{::}S.\ A]\!]_{g,k}} & = \{\, \Lambda.t \mid \forall I.\ \vdash I :: S\ .\ \wedge\ t \in [\![A[I/i]]\!]^{E,(L[I/i],U[I/i])}_{g,k-1} \,\} \\
[\![\exists i :: S.\ A\,]\!]_{g,k} & = \{\, \mathsf{pack}\, v \mid \exists I.\ \vdash I :: S \ \wedge\ v \in [\![A[I/i]]\!]_{g,k-1} \,\} \\
[\![A_1 + A_2]\!]_{g,k} & = \{\, \mathrm{inl}\ v \mid v \in [\![A_1]\!]_{g,k} \,\} \cup \{\, \mathrm{inr}\ v \mid v \in [\![A_2]\!]_{g,k} \,\} \\
[\![C \supset A]\!]_{g,k} & = \{\, v \mid \not\vDash C \ \vee\ v \in [\![A]\!]_{g,k-1} \,\} \\
[\![C \& A]\!]_{g,k} & = \{\, v \mid \vDash C \ \wedge\ v \in [\![A]\!]_{g,k-1} \,\}
\end{array}
$$

$$
[\![A]\!]^{e}_{g,k,(L,U)} = \{\, t \mid \forall v,\ k', c.\ k' \le k \wedge\ t \Downarrow^{c,k'} v \Rightarrow\ v \in [\![A]\!]_{g,k-k'} \ \wedge\ L \le c \le U \,\}
$$

$$
\begin{array}{ll}
[\![\cdot]\!]_{g,k} & = \{\, \emptyset \,\} \\
[\![\Omega, x : A]\!]_{g,k} & = \{\, (\sigma[v/x]) \mid \sigma \in [\![\Omega]\!]_{g,k} \ \wedge\ v \in [\![A]\!]_{g,k} \,\}
\end{array}
$$

Figure 5.1: The interpretation of the types and contexts of ARel.

permissions. We note that the interpretation quantifies universally over worlds g_1 that extend g ($g_1 \supseteq g$) and step-indices k_1 less than or equal to k. This is standard in step-indexed Kripke logical relations and makes the interpretation "monotonic", i.e., closed under larger worlds and smaller step indices (Lemma 1). Also, let us pay attention to the annotated cost upper and lower bounds (L, U), which bound the cost of forcing evaluation of the impure computation of the monadic type in the interpretation. The writing permission is also expressed in the existential formula which says that if the value at certain index j in an array whose static name is γ_i, then this index must have the writing permission and appear in the precondition P.

We extend the value interpretation to an expression interpretation:

$$[\![A]\!]^{e}_{g,k,(L,U)} = \{\ t \mid \forall v, k', c.\ k' \leq k \wedge\ t \Downarrow^{c,k'} v \Rightarrow\ v \in [\![A]\!]_{g,k-k'} \wedge\ L \leq c \leq U\ \}$$

Compared to the value interpretation, the expression interpretation, which we identify by the superscript e, accounts for lower and upper bound costs L and U. The expression interpretation requires that if the expression t evaluates to value v with the step-index k' and cost c, then the cost satisfies the constraint $L \leq c \leq U$ and the resulting value v is in the value interpretation of the type A with the remaining step-index $k - k'$.

Next, we extend our interpretation to *open* terms. For this, we first extend our value interpretation to unary contexts: Given a substitution σ, we say that $\sigma \in [\![\Omega]\!]_{g,k}$ whose definition in Figure 5.1, if σ maps every variable in Ω to a value in the interpretation of its unary type. We write $\sigma\, t$ to denote the application of the substitution σ to the term t. With this, we can extend our interpretation to typed open terms, i.e., typing judgments, as in the statement of the fundamental theorem (Theorem 2).

$$
\begin{aligned}
(\!|\mathsf{int}|\!)_{G,k} &= \{\, (n_1, n_2) \mid n_1 = n_2 \,\} \\
(\!|\mathsf{unit}|\!)_{G,k} &= \{\, ((), ()) \,\} \\
(\!|\mathsf{int}[I]|\!)_{G,k} &= \{\, (n, n) \mid I = n \,\} \\
(\!|\mathsf{Array}_\gamma[I]\ \tau|\!)_{G,k} &= \{\, (l_1, l_2) \mid I = n \ \wedge\ G(\gamma) = (l_1, l_2, \tau, n) \,\} \\
(\!|\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2|\!)_{G,k} &= \left\{ \begin{pmatrix} (\,\mathsf{fix}\, f(x).t_1, \\ \mathsf{fix}\, f(x).t_2\,) \end{pmatrix} \;\middle|\; \begin{array}{l} \forall G' \supseteq G.\ \forall k' \le k-1.\ \forall v_1,\ v_2. \\ (v_1, v_2) \in (\!|\tau_1|\!)_{G',k'} \implies \\ \quad (t_1[v_1/x][\mathsf{fix}\, f(x).t_1/f], \\ t_2[v_2/x][\mathsf{fix}\, f(x).t_2/f]) \in (\!|\tau_2|\!)^e_{G',k',D} \ \wedge \\ \forall j.\ \Big(\mathsf{fix}\, f(x).t_1 \in [\![|\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\tau_2|_1]\!]_{G|_1,j} \ \wedge \\ \mathsf{fix}\, f(x).t_2 \in [\![|\tau_1|_2 \xrightarrow{\mathrm{exec}(0,\infty)} |\tau_2|_2]\!]_{G|_2,j}\Big) \end{array} \right\} \\
(\!|\{P\}\ \exists\vec{\gamma}.\tau\ \{Q\}|\!)^{\mathrm{diff}(D)}_{G,k} &= \left\{ (v_1, v_2) \;\middle|\; \begin{array}{l} \forall G_1 \supseteq G.\ \forall k_1 \le k,\ k_2 < k_1,\ k_3,\ c_1,\ c_2,\ H_1,\ H_2. \\ (H_1, H_2) \models_{G_1,k_1} P \ \wedge\ v_1; H_1 \Downarrow_f^{c_1,k_2} \ \wedge\ v_2; H_2 \Downarrow_f^{c_2,k_3} \\ \implies \quad \exists G_2 \supseteq G_1,\ H_1',\ H_2',\ v_1',\ v_2'. \\ \Big(\ v_1; H_1 \Downarrow_f^{c_1,k_2} v_1'; H_1' \ \wedge\ v_2; H_2 \Downarrow_f^{c_2,k_3} v_2'; H_2' \ \wedge \\ \quad (H_1', H_2') \models_{G_2,k_1-k_2} Q \ \wedge\ (v_1', v_2') \in (\!|\tau|\!)_{G_2,k_1-k_2} \ \wedge \\ \quad c_1 - c_2 \le D\Big) \end{array} \right\} \\
(\!|U(A_1, A_2)|\!)_{G,k} &= \{\, (v_1, v_2) \mid \forall k'.\ v_1 \in [\![A_1]\!]_{G|_1,k'} \ \wedge\ v_2 \in [\![A_2]\!]_{G|_2,k'} \,\} \\
(\!|\forall i{::}S.\ \tau|\!)^{\mathrm{diff}(D)}_{G,k} &= \left\{ (\Lambda.t_1, \Lambda.t_2) \;\middle|\; \begin{array}{l} \forall I.\ \vdash I :: S \ \wedge\ (t_1, t_2) \in (\!|\tau[I/i]|\!)^{E,D[I/i]}_{G,k-1} \ \wedge \\ \forall j.\ t_1 \in [\![|\tau|_1]\!]^{E,(0,\infty)}_{G|_1,j} \ \wedge\ t_2 \in [\![|\tau|_2]\!]^{E,(0,\infty)}_{G|_2,j} \end{array} \right\} \\
(\!|\exists i :: S.\ \tau\ |\!)_{G,k} &= \{\, (\mathsf{pack}\, v_1, \mathsf{pack}\, v_2) \mid \exists I.\ \vdash I :: S \ \wedge\ (v_1, v_2) \in (\!|\tau[I/i]|\!)_{G,k-1} \,\} \\
(\!|\tau_1 + \tau_2|\!)_{G,k} &= \{\, (\mathrm{inl}\ v_1, \mathrm{inl}\ v_2) \mid (v_1, v_2) \in (\!|\tau_1|\!)_{G,k-1} \ \vee\ (v_1, v_2) \in (\!|\tau_2|\!)_{G,k-1} \} \\
(\!|\Box\tau|\!)_{G,k} &= \{\, (v, v) \mid (v, v) \in (\!|\tau|\!)_{G,k} \,\} \\
(\!|C \supset \tau|\!)_{G,k} &= \{\, (v_1, v_2) \mid \not\models C \ \vee\ (v_1, v_2) \in (\!|\tau|\!)_{G,k-1} \,\} \\
(\!|C\&\tau|\!)_{G,k} &= \{\, (v_1, v_2) \mid \models C \ \wedge\ (v_1, v_2) \in (\!|\tau|\!)_{G,k-1} \,\} \\
(\!|\tau|\!)^e_{G,k,D} &= \left\{ (t_1, t_2) \;\middle|\; \begin{array}{l} \forall k_1 \le k.\ \forall k_2,\ v_1,\ v_2,\ c_1,\ c_2. \\ (t_1 \Downarrow^{c_1,k_1} v_1 \wedge\ t_2 \Downarrow^{c_2,k_2} v_2) \\ \implies (v_1, v_2) \in (\!|\tau|\!)_{G,k-k_1} \ \wedge\ c_1 - c_2 \le D \end{array} \right\} \\
(\!|\cdot|\!)_{G,k} &= \{\, \emptyset \,\} \\
(\!|\Gamma|\!)_{G,k} &= \{\, (\sigma_1, \sigma_2) \mid \forall x \in dom(\Gamma).\ \forall \tau.\ x : \tau \in \Gamma.\ (\sigma_1(x), \sigma_2(x)) \in (\!|\tau|\!)_{G,k} \,\}
\end{aligned}
$$

Figure 5.2: The interpretation of relational types and contexts of ARel.

$$
\begin{array}{rll}
H \vDash_{g,k} empty & \text{iff} & \text{true} \\
H \vDash_{g,k} (\gamma \to \beta) & \text{iff} & \exists l, A, n : g(\gamma) = (l, A, n) \wedge \left(\forall i \le n.\, \left(H[l][i]\right) \in (\!|A|\!)_{g,k-1}\right) \\
H \vDash_{g,k} (P * Q) & \text{iff} & \exists H', H'', g', g'' : \left(H = H' \uplus H''\right) \wedge \left(g = g' \uplus g''\right) \\
 & & \wedge \left(H' \vDash_{g',k} P\right) \wedge \left(H'' \vDash_{g'',k} Q\right) \\
(H_1, H_2) \vDash_{G,k} empty & \text{iff} & \text{true} \\
(H_1, H_2) \vDash_{G,k} (\gamma \to \beta) & \text{iff} & \exists l_1, l_2, \tau, n : G(\gamma) = (l_1, l_2, \tau, n) \\
 & & \wedge \left(\forall i \le n.\, \left(H_1[l_1][i], H_2[l_2][i]\right) \in (\!|\tau|\!)_{G,k-1}\right) \\
 & & \wedge \left(\forall i \le n.\, \left(H_1[l_1][i] \neq H_2[l_2][i] \;\Rightarrow\; i \in \beta\right)\right) \\
(H_1, H_2) \vDash_{G,k} (P * Q) & \text{iff} & \exists H_1', H_1'', H_2', H_2'', G', G'' : \\
 & & \left(H_1 = H_1' \uplus H_1''\right) \wedge \left(H_2 = H_2' \uplus H_2''\right) \wedge \left(G = G' \uplus G''\right) \\
 & & \wedge \left((H_1', H_2') \vDash_{G',k} P\right) \wedge \left((H_1'', H_2'') \vDash_{G'',k} Q\right)
\end{array}
$$

Figure 5.3: Unary and relational heap relation of ARel.

5.2 RELATIONAL INTERPRETATION OF AREL TYPES

We show the value and expression interpretations of relational types, and the interpretation of relational contexts in Figure 5.2. As in the case of the unary interpretation, we use Kripke worlds. Relational Kripke worlds are denoted G and they map static array names γ to 4-tuples (l_1, l_2, n, τ). If $G(\gamma) = (l_1, l_2, n, \tau)$, then l_1, l_2 are the locations where the arrays statically named γ are stored in the two runs, n is the length of each of these two arrays, and τ is the type at whose relational interpretation the two arrays' corresponding elements should be related. We now only support cases when the two arrays have the same length here so that our monadic type can track the meaningful set of indices where the two arrays may differ. The world G now will not track two arrays of different lengths together.

The value interpretation of a relational type τ is written $(\!|\tau|\!)_{G,k}$. It is a set of pairs of related values. Most of the clauses of this definition are straightforward. Somewhat unusually, the interpretation of a function type contains a pair of (recursive) functions that

satisfy not one but two conditions: (i) Given related values as arguments, the functions return related results, and (ii) Each of the two functions, when given an argument in the (unary) interpretation of the argument type's unary projection returns a result in the (unary) interpretation of the result type's unary projection. The first condition is standard. The second condition is needed to make our relational-to-unary projections of types sound.

The interpretation of function type $\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2$ ascribes a pair of functions whose body execution time is bounded by the annotated D when the input are related by τ_1. This interpretation reasons about recursive functions so we can notice the use of step indices, which decreases from k to k' along with the beta reduction. Also, the interpretation of relational function types provides the relational-to-unary projection, by requiring that both the first or second element of the pair of values in this interpretation belong to its corresponding unary interpretation at an arbitrary step-index.

To define the interpretation of the relational monadic types we need a relational heap relation $(H_1, H_2) \vDash_{G,k} P$, defined in Figure 5.3. The relation says when the heaps H_1, H_2 from two runs satisfy a relational assertion P, which could be a pre- or postcondition from a relational monadic type. Intuitively, this relation holds when for every $\gamma \to \beta$ in P, $G(\gamma)$ is also defined, and if $G(\gamma) = (l_1, l_2, \tau, n)$, then for every index up to n, the two arrays l_1, l_2 in heaps H_1, H_2 respectively have elements within the relational interpretation of τ, and every index where the two elements differ is in β. This formalizes the intuitive meaning of β from earlier chapters.

Note that the condition on elements differing is a one-way implication: We do not insist that at every index in β, the two elements necessarily differ. In fact, depending on τ, in some cases, even elements at indices in β might be forced to be equal. For example, when τ is $\mathrm{int}[m]$ (for some m) or even $\exists i.\mathrm{int}[i]$, this forces corresponding elements to be

equal at all indices since the relational interpretation of int[m] is the singleton $\{(m,m)\}$. However, when $\tau = U(A_1, A_2)$, elements at indices in β can be arbitrary values of types A_1, A_2 since the relational interpretation of $U(A_1, A_2)$ is morally $A_1 \times A_2$.

Like the unary heap relation, the relational heap relation is well-found by induction on the step-index.

The relational interpretation for a monadic type $\{P\}\ \exists\vec{\gamma}.\tau\ \{Q\}^{\mathrm{diff}(D)}$ (Figure 5.2) is the set of pairs of values (v_1, v_2) that when forced starting from heaps H_1, H_2 satisfying the precondition P, result in heaps H_1', H_2' satisfying the postcondition Q. The relative cost of forcing must be upper bounded by D.

Next, we extend the value interpretation of relational types to pairs of expressions:

$$(\!|\tau|\!)^{e}_{G,k,D} = \left\{ (t_1, t_2) \;\middle|\; \begin{array}{l} \forall k_1 \le k,\ \forall k_2,\ v_1,\ v_2,\ c_1,\ c_2.\ (t_1 \Downarrow^{c_1,k_1} v_1 \ \wedge\ t_2 \Downarrow^{c_2,k_2} v_2) \Rightarrow \\ (v_1, v_2) \in (\!|\tau|\!)_{G,k-k_1} \ \wedge\ c_1 - c_2 \le D \end{array} \right\}$$

The interpretation simply says that two expressions e_1, e_2 are in the interpretation of type τ if they both evaluate to some values v_1, v_2 and those values are in the value interpretation of τ. Additionally, the costs c_1, c_2 of these two evaluations satisfy $c_1 - c_2 \le D$. Note that the step-index counts steps of only the left evaluation, not both. We could, alternatively, have set up the interpretation to count steps of only the right evaluation or both.

Next, we extend our value interpretation to relational contexts (as for the unary case), and the statement of our fundamental theorem (Theorem 5.3.3) contains the interpretation of typed open terms, i.e., the relational typing judgment.

Note For readers familiar with Kripke logical relations, we note that our worlds g and G are *not* step-indexed (only our logical relations are step-indexed). This is unlike some

prior work (Neis et al., 2011; Turon et al., 2013). We do not need step-indexed worlds since we include syntactic types, A or τ, for mutable locations (arrays) in the worlds. This suffices for our purposes because our language only considers arrays whose elements are of base type like int, bool, etc.

5.3 FUNDAMENTAL THEOREM

In this section, we prove a standard theorem, called a fundamental theorem, for each of our two interpretations, unary and relational. We use $\vdash \delta : \Delta$ to mean that δ is a well-sorted substitution for the index variables in the domain of Δ, and $\vDash \delta\Phi_a$ to denote that δ satisfies the constraint environment Φ_a. Let us have a look at an example. Suppose the index variable context $\Delta = \{(i : \mathbb{N}), (c : \mathbb{R})\}$ has two index variables i (natural number) and c (real number). The constraint environment has the following constraint $\Phi_1 = i < 2 \wedge c > 1.5$. Then $\delta = \{(i : 1), (c : 2.5)\}$ is a well-sorted substitution. As a preliminary step, we show a monotonicity lemma, which is useful in the proofs of our fundamental theorems.

5.3.1 Monotonicity

Lemma 1 (Monotonicity)**.** *1. If $k' \leq k$ and $G \subseteq G'$, then $(\!|\tau|\!)_{G,k} \subseteq (\!|\tau|\!)_{G',k'}$.*

2. If $k' \leq k$ and $g \subseteq g'$, then $[\![A]\!]_{g,k} \subseteq [\![A]\!]_{g',k'}$.

3. If $k' \leq k$ and $G \subseteq G'$, then $(\!|\tau|\!)^{E,D}_{G,k} \subseteq (\!|\tau|\!)^{E,D}_{G',k'}$.

4. If $k' \leq k$ and $g \subseteq g'$, then $[\![A]\!]^{E,(L,U)}_{g,k} \subseteq [\![A]\!]^{E,(L,U)}_{g',k'}$.

5. If $k' \leq k$ and $G \subseteq G'$, then $(\!|\Gamma|\!)_{G,k} \subseteq (\!|\Gamma|\!)_{G',k'}$

6. If $k' \leq k$ and $g \subseteq g'$, then $[\![\Omega]\!]_{g,k} \subseteq [\![\Omega]\!]_{g',k'}$

Proof. Points (1) and (3), related to the interpretation of relational types, are proved simultaneously by induction on τ. Similarly, points (2) and (4), about the interpretation of the unary types, are proved simultaneously by induction on the unary type A. Points (5) and (6), about the contexts, follows directly from points (1) and (2). □

The monotonicity lemma says that all our interpretations are monotone concerning larger worlds and smaller step indexes. Take the value interpretation of a relational type as an instance. If a pair of values (v_1, v_2) is in this interpretation for some step-index k in some world G, then (v_1, v_2) is also in the value interpretation of the same type for any smaller step-index k' and any bigger world G'. It can also be found in the appendix A, Lemma 1.

5.3.2 Fundamental Theorem for Unary Typing

Next, we present the fundamental theorem of ARel's unary typing. The theorem states the following. Suppose we have an expression t that is well-typed at a unary type A in contexts Σ, Δ, Φ_a and Ω with cost lower and upper bounds L and U. If we close everything using a substitution δ for the index variables in Δ (satisfying Φ_a) and a substitution σ for the term variables Ω satisfying the (context) interpretation of $\delta\Omega$ (at world g and step-index k), then the closed term $\sigma\, t$ is in the interpretation of the closed type $\delta\, A$ with bounds L and U (at world g and step-index k).

Theorem 2 (Fundamental Theorem for Unary Typing)**.** *If* $\Sigma;\Delta;\Phi_a;\Omega \vdash_L^U t : A$, $\vdash \delta : \Delta$ *and* $\vDash \delta\Phi_a$, *and* $\sigma \in [\![\delta\Omega]\!]_{g,k}$, *then* $(\sigma\ t) \in [\![\delta\, A]\!]^e_{g,k,(\delta L,\delta U)}$.

Proof. The proof is by induction on the derivation of the judgment $\Sigma;\Delta;\Phi_a;\Omega \vdash_L^U t : A$. We present some of the most relevant cases. The full proof of this theorem can be found in the Appendix A, Section A.5.0.1.

$$\textbf{Case}\ \frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \mathsf{Array}_\gamma[I]\ A \qquad \gamma \in dom(P) \\ \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : \mathbf{int}[I'] \qquad \vDash I' \leq I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^0 \mathsf{read}\ t_1\ t_2 : \overset{\mathrm{exec}(L_1+L_2+L_{\mathsf{read}},U_1+U_2+U_{\mathsf{read}})}{\{P\}\ \exists\_ : A\ \{P\}}}\ \textbf{u-read}$$

By assumption we have $\vdash \delta : \Delta$, $\vDash \delta\Phi_a$ and $\sigma \in [\![\delta\Omega]\!]_{g,k}$. We need to show:

$$\mathsf{read}\ (\sigma\ t_1)\ (\sigma\ t_2) \in [\![\delta \overset{\mathrm{exec}(L_1+L_2+L_{\mathsf{read}},U_1+U_2+U_{\mathsf{read}})}{\{P\}\ \exists\_ : A\ \{P\}}]\!]^e_{g,k,(0,0)}.$$

Since $\mathsf{read}\ (\sigma\ t_1)\ (\sigma\ t_2)$ is a value, and its evaluation incurs no cost, it is sufficient to show:

$$\mathsf{read}\ (\sigma\ t_1)\ (\sigma\ t_2) \in [\![\delta \overset{\mathrm{exec}(L_1+L_2+L_{\mathsf{read}},U_1+U_2+U_{\mathsf{read}})}{\{P\}\ \exists\_ : A\ \{P\}}]\!]_{g,k}$$

So, unfolding the definition of interpretation, for $k' \leq k$, $g' \supseteq g$, and an arbitrary heap H such that $H \vDash_{g',k'} P$ and $\mathsf{read}\ (\sigma\ t_1)\ (\sigma\ t_2); H \Downarrow^{c,k''}$ terminates with some cost c and step $k'' < k'$. By the definition of forcing evaluation we have:

$$\frac{\sigma\ t_1 \Downarrow^{c_1,k_1} l \quad \sigma\ t_2 \Downarrow^{c_2,k_2} n_2 \quad H(l)[n] = v}{\mathsf{read}\ (\sigma\ t_1)\ (\sigma\ t_2); H \Downarrow_f^{c_1+c_2+c_{\mathsf{read}},k_1+k_2+1} v; H}\ \text{f-read}$$

Now we have $k_1 + k_2 + 1 < k' \leq k$. we must show now:

1. $\delta\, L_1 + \delta\, L_2 + \delta\, L_{\mathsf{read}} \leq c_1 + c_2 + c_{\mathsf{read}} \leq \delta\, U_1 + \delta\, U_2 + \delta\, U_{\mathsf{read}}$.
2. $H \vDash_{g',k'-(k_1+k_2+1)} P$.
3. $v \in [\![\delta A]\!]_{g',k'-(k_1+k_2+1)}$.
4. $\exists n.P = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \ \wedge\ \forall i \in [1,n].g(\gamma_i) = (l_i, A, m) \Rightarrow \forall j.(H[l_i][j]) \neq H[l_i][j] \Rightarrow j \in T_i$.

By induction hypothesis, from the first premise and Lemma 1, we have:

$$\sigma\ t_1 \in [\![\mathsf{Array}_\gamma[\delta\ I]\ (\delta\ A)]\!]^e_{g',k',(\delta\ L_1,\delta\ U_1)}.$$

From this we have:

$$l \in [\![\mathsf{Array}_\gamma[\delta\ I]\ (\delta\ A)]\!]_{g',k'-k_1}\ \wedge\ \delta\ L_1 \leq c_1 \leq \delta\ U_1$$

which in turn tells us: $g'(\gamma) = (l, \delta\ A, \delta\ I)$. By induction hypothesis, from the premise $\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : \mathrm{int}[I']$ and Lemma 1 we have:

$$\sigma\ t_2 \in [\![\mathrm{int}[\delta\ I']]\!]^e_{g',k',(\delta\ L_2,\delta\ U_2)}.$$

From this we have that: $n_2 \in [\![\mathrm{int}[\delta\ I']]\!]_{g',k'-k_2}$, which in turn tells us that

$$n_2 = \delta\ I'\ \wedge\ \delta\ L_2 \leq c_2 \leq \delta\ U_2$$

Now to conclude we can proceed as follows:

1. the inequality $\delta\ L_1 + \delta\ L_2 + \delta\ L_{\mathsf{read}} \leq c_1 + c_2 + c_{\mathsf{read}} \leq \delta\ U_1 + \delta\ U_2 + \delta\ U_{\mathsf{read}}$ can be shown using the fact that $L_{\mathsf{read}} \leq c_{\mathsf{read}} < U_{\mathsf{read}}$ and the fact that in our language L_{read} and U_{read} are constants , which means $\delta\ L_{\mathsf{read}} = L_{\mathsf{read}}$ and $\delta\ U_{\mathsf{read}} = U_{\mathsf{read}}$.

2. $H \vDash_{g',k'-(k_1+k_2+1)} P$ is proved by unfolding the definition of our assumption $H \vDash_{g',k'} P$ and using Lemma 1.

3. For $v \in [\![\delta\ A]\!]_{g',k'-(k_1+k_2+1)}$, we know that from the heap relation $H \vDash_{g',k'} P$ and the premise $\gamma \in dom(P)$, we have $\forall i \leq n, (H_1(l)(i)) \in [\![A]\!]_{g',k'-1}$, which in turn tells us $v \in [\![A]\!]_{g',k'-1}$ because $H(l)[n] = v$. We can then show our goal by using Lemma 1.

4. $\exists n.P = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \ \wedge \ \forall i \in [1,n].g(\gamma_i) = (l_i, A, m) \Rightarrow \forall j.(H[l_i][j]) \neq H[l_i][j] \Rightarrow j \in T_i$ follows because the heap H is not changed, so the claim is trivial.

$$\textbf{Case}\ \frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : \mathsf{Array}_\gamma[I]\ A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : \mathbf{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_3}_{L_3} t_3 : A \qquad \Delta;\Phi_a \vDash I' \leq I \qquad \Sigma;\Delta;\vdash P\ \mathsf{wf} \qquad \Delta;\Phi_a \vDash I' \in \beta}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{updt}\ t_1\ t_2\ t_3 :\ \{P \star \gamma \to \beta\}\ \exists\_ : \mathbf{unit}\ \{P \star \gamma \to \beta\}}^{\mathrm{exec}(L_1+L_2+L_3+L_{\mathsf{updt}}, U_1+U_2+U_3+U_{\mathsf{updt}})}\ \textbf{u-updt}$$

By assumption we have $\vdash \delta : \Delta$, $\vDash \delta\Phi_a$ and $\sigma \in [\![\delta\Omega]\!]_{g,k}$, we need to show:

$$\mathsf{updt}\ (\sigma\ t_1)\ (\sigma\ t_2)\ (\sigma\ t_3) \in [\![\delta\ \{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta\}\ ]\!]^{e}_{g,k,(0,0)}$$

Since $\mathsf{updt}\ (\sigma\ t_1)\ (\sigma\ t_2)\ (\sigma\ t_3)$ is a value, it is sufficient to show:

$$\mathsf{updt}\ (\sigma\ t_1)\ (\sigma\ t_2)\ (\sigma\ t_3) \in [\![\delta\ \{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta\}\ ]\!]_{g,k}$$

By the definition of forcing evaluation we have:

$$\frac{\sigma\ t_1 \Downarrow^{c_1,k_1} l \quad \sigma\ t_2 \Downarrow^{c_2,k_2} n \quad \sigma\ t_3 \Downarrow^{c_3,k_3} v}{\mathsf{updt}\ (\sigma\ t_1)\ (\sigma\ t_2)\ (\sigma\ t_3); H \Downarrow_f^{c_1+c_2+c_3+c_{\mathsf{update}}, k_1+k_2+k_3+1} (); H(l)[n] \leftarrow v}\ \text{f-updt}$$

So, unfolding the definition of the interpretation, for $k' \leq k$, $g' \supseteq g$, for an arbitrary heap H such that $H \vDash_{g',k'} P \star \gamma \to \beta$, and $k_1 + k_2 + k_3 + 1 < k'$, we must show:

1. $\delta\, L_1 + \delta\, L_2 + \delta\, L_3 + \delta\, L_{\mathsf{updt}} \leq c_1 + c_2 + c_3 + c_{\mathsf{updt}} \leq \delta\, U_1 + \delta\, U_2 + \delta\, U_3 + \delta\, U_{\mathsf{updt}}$.

2. $(H(l)[n] \leftarrow v) \vDash_{g',k'-(k_1+k_2+k_3+1)} P \star \gamma \to \beta$.

3. $v \in [\![\mathrm{unit}]\!]_{g',k'-(k_1+k_2+k_3+1)}$.

4. $\exists n.P \star \gamma \to \beta = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \ \wedge \ \forall i \in [1,n].g(\gamma_i) = (l_i, A, m)$
   $\Rightarrow \forall j.(H[l_i][j]) \neq (H(l)[n] \leftarrow v)[l_i][j] \Rightarrow j \in T_i$.

By induction hypothesis, from the first premise $\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : \mathsf{Array}_\gamma[I]\ A$ and lemma 1, we have:

$$\sigma\ t_1 \in [\![\mathsf{Array}_\gamma[\delta\ I]\ (\delta A)]\!]^e_{g',k',\delta\,(\delta\,L_1,\delta\,U_1)}$$

From this we have:

$$l \in [\![\mathsf{Array}_\gamma[\delta\ I]\ (\delta A)]\!]_{g',k'-k_1} \Rightarrow g'(\gamma) = (l, \delta A, \delta\ I)\ \wedge\ \delta\ L_1 \leq c_1 \leq \delta\ U_1$$

By induction hypothesis, from the second premise $\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : \mathrm{int}[I']$ and lemma 1, we conclude:

$$\sigma\ t_2 \in [\![\mathrm{int}[\delta\ I]]\!]^e_{g',k',(\delta\,L_2,\delta\,U_2)}$$

From this we have that $n \in [\![\mathrm{int}[\delta\ I]]\!]_{g',k'-k_2}$, which in turn tells us that

$$n = I\ \wedge\ \delta\ L_2 \leq c_2 \leq \delta\ U_2$$

By induction hypothesis, from the third premise $\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_3}_{L_3} t_3 : A$ and Lemma 1, we have:

$$\sigma\ t_3 \in [\![\delta A]\!]^e_{g',k',(\delta\,L_3,\delta\,U_3)}$$

From this we have:

$$v \in [\![\delta A]\!]_{g',k'-k_3}\ \wedge\ \delta\ L_3 \leq c_1'' \leq \delta\ U_3$$

To conclude, we proceed as follows:

1. the inequality $\delta\ L_1 + \delta\ L_2 + \delta\ L_3 + \delta\ L_{\mathsf{updt}} \leq c_1 + c_2 + c_3 + c_{\mathsf{updt}} \leq \delta\ U_1 + \delta\ U_2 + \delta\ U_3 + \delta\ U_{\mathsf{updt}}$ can be shown using the fact that $L_{\mathsf{updt}} \leq c_{\mathsf{updt}} < U_{\mathsf{updt}}$ and L_{updt} and U_{updt} are constants, which means $\delta\ L_{\mathsf{updt}} = L_{\mathsf{updt}}$ and $\delta\ U_{\mathsf{updt}} = U_{\mathsf{updt}}$.

2. $(H(l)[n] \leftarrow v) \vDash_{g',k'-(k_1+k_2+k_3+1)} P \star \gamma \rightarrow \beta$ is proved by unfolding the definition of our assumption $H \vDash_{g',k'} P \star \gamma \rightarrow \beta$.

We can conlude that $\forall i \le I.H(l)[i] \in (\!|\delta A|\!)_{g',k'-1}$, and then we can show that $\forall i \le I.(H(l)[n] \leftarrow v)(l)[i] \in (\!|\delta A|\!)_{g',k'-(k_3+1)}$ because of the our previous conclusion $v \in [\![\delta A]\!]_{g',k'-k_3}$. Then our goal can be proved by using Lemma 1.

3. $v \in [\![\text{unit}]\!]_{g',k'-(k_1+k_2+k_3+1)}$ is proved by the definition of the interpretation of unit type.

4. $\exists n.P \star \gamma \to \beta = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \wedge \forall i \in [1,n].g(\gamma_i) = (l_i, A, m)$
$\Rightarrow \forall j.(H[l_i][j]) \neq (H(l)[n] \leftarrow v)[l_i][j] \Rightarrow j \in T_i$. It is proved depending on γ_i. When $\gamma_i \neq \gamma$, the array is not changed such that $H[l_i] = (H(l)[n] \leftarrow v)[l_i]$. When $\gamma_i = \gamma$, then $T_i = \beta$, we can show that the only updated index n at this array $H[l]$ is in β from our premise $\Delta;\Phi_a \vDash I' \in \beta$ in the rule, which proves this case.

$$\textbf{CASE}\ \frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : \textbf{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : A \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{alloc}\ t_1\ t_2 : \{P\}\ \exists\gamma : \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \to \mathbb{N}\}}^{\text{exec}(L_1+L_2+L_a, U_1+U_2+U_a)}\ \textbf{U-ALLOC}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and there exists Ω'. s.t. $FV(t) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(\sigma) \in (\!|\delta\Omega'|\!)_{g,k}$

TS: $(\delta\sigma\ \mathsf{alloc}\ t_1\ t_2) \in [\![\{P\}\ \exists\gamma : \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \to \mathbb{N}\}]\!]^{E,(0,0)}_{g,k}$

Because $\mathsf{alloc}\ t_1\ t_2$ is a value. STS: $(\delta\sigma\ \mathsf{alloc}\ t_1\ t_2) \in [\![\{P\}\ \exists\gamma : \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \to \mathbb{N}\}]\!]_{g,k}$

Unfold the definition of $[\![\{P\}\ \exists\gamma : \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \to \mathbb{N}\}]\!]_{g,k}$. We choose $g' \supseteq g$, $k' \le k$, $k'' < k'$ and H. Assume $H \vDash_{g',k'} P$ and $H;\mathsf{alloc}\ t_1\ t_2 \Downarrow^{k''}$.

From $H;\mathsf{alloc}\ t_1\ t_2 \downarrow^{k''}$, because t_1, t_2 are sub terms of $\delta\sigma\ \mathsf{alloc}\ t_1\ t_2$,we get

$\delta\sigma\, t_1 \Downarrow^{k_1}_f \wedge \delta\sigma\, t_2 \Downarrow^{k_2}_f$ (1). From (1) , we get:

$\exists v_1, v_2, k_1, k_2.\delta\sigma\, t_1 \Downarrow^{k_1,c_1} v_1 \wedge \delta\sigma\, t_2 \Downarrow^{k_2,c_1'} v_2 \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_1' + c_{alloc}(a)$.

From (a) and evaluation rules we get: $\exists l.H, (\mathsf{alloc}\ t_1\ t_2)\delta\sigma \Downarrow^{k'',c}_f l; H[l \to [v_2, \ldots v_2](b)$.

By IH on the first premise instantiated with $\sigma \in [\![\delta\Omega']\!]_{g,k''}$using lemma 1 and $FV(t_1) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$, we get $(\delta\sigma\, t_1) \in [\![\text{int}[I]]\!]^{E,(\delta L_1, \delta U_1)}_{g',k''}(*)$.

Unfold the definition of $[\![\text{int}[I]]\!]^{E,(\delta L_1,\delta U_1)}_{g',k''}$ and using (a) and $k_1 \le k''$, we know:

$$(v_1) \in [\![\text{int}[I]]\!]_{G',k''-k_1} \Rightarrow v_1 = I \wedge \delta L_1 \le c_1 \le \delta U_1 \qquad \text{IH1}$$

By IH on the second premise instantiated with $(\sigma) \in [\![\delta\Gamma]\!]_{g',k'-k_1}$ using lemma 1 and $FV(t_2) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$, we get: $(\delta\sigma\, t_2) \in [\![\delta A]\!]^{E,(\delta L_2,\delta U_2)}_{g',k'-k_1}(**)$.
Unfold the definition of (**) and using (a), (b) and $k_2 \le k' - k_1$, we know

$$(v_2) \in [\![\delta A]\!]_{g',k'-k_1-k_2} = [\![\delta A]\!]_{g',k'-k''+1} \wedge \delta L_2 \le c_1' \le \delta U_2 \qquad \text{IH2}$$

Let us assume: $g'' = g'[r \to (l, A, I)](c)$, $H' = H, \left[l \to [v_2, v_2, \ldots, v_2]\right](d)$. We want to show 4 cases.

TS1 $(l) \in [\![\text{Array}_\gamma[I]\ A]\!]_{g'',k'-k''}$

it is proved by unfolding the definition and using the assumption (c).

TS2 $(H') \vDash_{g'',k'-k''} p \star \gamma \to \mathbb{N}$

we know that $g''(\gamma) = (l, A, I)$, $H' = H \uplus H_l \wedge H_l = l \to [v_2, v_2, \ldots, v_2]$

STS1 $\forall i \le I, (H_l(l)[i] \in [\![A]\!]_{g'',k'-k''-1}$. It is proved by using IH2 and Lemma 1.

TS3 $\delta(L_1 + L_2 + L_a) \le c \le \delta(U_1 + U_2 + U_a)$

It is proved by IH1 and IH2 and $\delta L_a \le c_{alloc} \le \delta U_a$.

TS4 $\exists n.P = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \wedge \forall i \in [1, n].g(\gamma_i) = (l_i, A, m)$
$\Rightarrow \forall j.(H[l_i][j]) \neq H'[l_i][j] \Rightarrow j \in T_i \Big\}$

Because new allocated γ is not in P, so for all the γ_i, it is not changed.

□

5.3.3 Fundamental Theorem for Relational Typing

Similarly, we have a fundamental theorem for the relational typing. The theorem says the following. Suppose we have a pair of expressions (t_1, t_2) that is well-typed with the relational type τ in the contexts Σ, Δ, Φ_a and Γ with upper bound D on the relative cost. Suppose we close Δ with the substitution δ for index variables, which also satisfies the constraint Φ_a, and close Γ with a pair of closing substitutions (σ_1, σ_2) satisfying the interpretation of the relational context $\delta\Gamma$. Then, the pair of closed terms $(\sigma_1\, t_1, \sigma_2\, t_2)$ is in the expression interpretation of the closed relational type $\delta\tau$ with the cost upper bound D.

Theorem 2 (Fundamental Theorem for Relational Typing)**.** *If* $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ *and* $\vdash \delta : \Delta$ *and* $\vDash \delta\Phi_a$ *and* $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$*, then* $(\sigma_1\, t_1, \sigma_2\, t_2) \in (\!|\delta\tau|\!)^e_{G,k,(\delta D)}$.

Proof. The proof is by induction on $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$.

Case $\dfrac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_1}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1\, t_2 \ominus t_1'\, t_2' \lesssim D + D_1 + D_2 : \tau_2}$ **r-app**

By assumption we have $\vdash \delta : \Delta$, $\vDash \delta\Phi_a$ and $(\sigma_1, \sigma_2) \in [\![\delta\Gamma]\!]_{G,k}$, we need to show:

$$\big((\sigma_1\, t_1)\,(\sigma_1\, t_2), (\sigma_2\, t_1')\,(\sigma_2\, t_2')\big) \in (\!|\delta\tau_2|\!)^e_{G,k,(\delta D + \delta D_1 + \delta D_2)}$$

By the definition of the evaluation we have:

$$\frac{\sigma_1\, t_1 \Downarrow^{c_1,k_1} \mathsf{fix}\, f(x).t \quad \sigma_1\, t_2 \Downarrow^{c_2,k_2} v \quad t[\mathsf{fix}\, f(x).t/f][v/x] \Downarrow^{c_3,k_3} v_1}{(\sigma_1\, t_1)\,(\sigma_1\, t_2) \Downarrow^{c_1+c_2+c_3+c_{\mathsf{fapp}},k_1+k_2+k_3+1} v_1}\ \text{e-fix}$$

$$\frac{\sigma_2\, t_1' \Downarrow^{c_1',k_1'} \mathsf{fix}\, f(x).t' \quad \sigma_2\, t_2' \Downarrow^{c_2',k_2'} v' \quad t'[\mathsf{fix}\, f(x).t'/f][v'/x] \Downarrow^{c_3',k_3'} v_1'}{(\sigma_2\, t_1')\,(\sigma_2\, t_2') \Downarrow^{c_1'+c_2'+c_3'+c_{\mathsf{fapp}},k_1'+k_2'+k_3'+1} v_1'}\ \text{e-fix}$$

So, unfolding the interpretation, for $k_1 + k_2 + k_3 + 1 \leq k$, we must show:

1. $(v_1, v_1') \in (\!|\delta\tau_2|\!)_{G,k-(k_1+k_2+k_3+1)}$.

2. $(c_1 + c_2 + c_3 + c_{\mathsf{fapp}}) - (c_1' + c_2' + c_3' + c_{\mathsf{fapp}}) \leq \delta\, D + \delta\, D_1 + \delta\, D_2$

By induction hypothesis, from the first premise $\Sigma; \Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D_1 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2$, we have:

$$(\sigma_1\, t_1, \sigma_2\, t_1') \in (\!|\delta\tau_1 \xrightarrow{\mathrm{diff}(\delta\, D)} \delta\tau_2|\!)^e_{G,k,\delta\, D_1}$$

From this we know:

$$(\mathsf{fix}\, f(x).t, \mathsf{fix}\, f(x).t') \in (\!|\delta\tau_1 \xrightarrow{\mathrm{diff}(\delta\, D)} \delta\tau_2|\!)_{G,k-k_1} \ \wedge\ c_1 - c_1' \leq \delta\, D_1$$

By induction hypothesis, from the second premise $\Sigma; \Delta; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_1$, we conclude:

$$(\sigma_1\, t_2, \sigma_2\, t_2') \in (\!|\delta\tau_1|\!)^e_{G,k,\delta\, D_2}$$

From this we have:

$$(v, v') \in (\!|\delta\tau_1|\!)_{G,k-k_2} \ \wedge\ c_2 - c_2' \leq \delta\, D_2$$

So, we unfold the definition of $(\mathsf{fix}\, f(x).t, \mathsf{fix}\, f(x).t') \in (\!|\delta\tau_1 \xrightarrow{\mathrm{diff}(\delta\, D)} \delta\tau_2|\!)_{G,k-k_1}$, using the previous conclusion $(v, v') \in (\!|\delta\tau_1|\!)_{G,k-(k_2+k_1+1)}$, we have:

$$(t[\mathsf{fix}\, f(x).t/f][v/x], t'[\mathsf{fix}\, f(x).t'/f][v'/x]) \in (\!|\delta\tau_2|\!)^e_{G,k-(k_1+k_2+1),\delta\, D}$$

From this we know:

$$(v_1, v_1') \in (\!|\delta\tau_2|\!)_{G,k-(k_1+k_2+k_3+1),\delta\, D} \wedge c_3 - c_3' \leq \delta\, D$$

Now to conclude we can proceed as follows:

1. $(v_1, v_1') \in (\!|\delta\tau_2|\!)_{G,k-(k_1+k_2+k_3+1)}$ is proved by our aforementioned conclusions

2. $(c_1 + c_2 + c_3 + c_{\mathsf{fapp}}) - (c_1' + c_2' + c_3' + c_{\mathsf{fapp}}) \le \delta D + \delta D_1 + \delta D_2$ is proved by our aforementioned conclusions.

$$\textbf{Case}\ \frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathbf{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\, t_1\, t_2 \ominus \mathsf{alloc}\, t_1'\, t_2' \lesssim 0 : \{P\}\ \exists\gamma.\ \overset{\mathrm{diff}(D_1+D_2)}{\mathsf{Array}_\gamma[I]\ \tau}\ \{P \star \gamma \to \mathbb{N}\}}\ \textbf{r-alloc}$$

By assumption we have $\vdash \delta : \Delta$, $\vDash \delta\Phi_a$ and $(\sigma_1, \sigma_2) \in [\![\delta\Gamma]\!]_{G,k}$, we need to show:

$$(\mathsf{alloc}\,(\sigma_1\, t_1)\,(\sigma_1\, t_2), \mathsf{alloc}\,(\sigma_2\, t_1')\,(\sigma_2\, t_2')) \in (\!|\{P\}\ \exists\vec{\gamma}.\ \overset{\mathrm{diff}(\delta D_1+\delta D_2)}{\mathsf{Array}_\gamma[I]\ \delta\tau}\ \{P \star \gamma \to \mathbb{N}\}|\!)^e_{G,k,0}$$

Since $\mathsf{alloc}\,(\sigma_1\, t_1)\,(\sigma_1\, t_2)$ and $\mathsf{alloc}\,(\sigma_2\, t_1')\,(\sigma_2\, t_2')$ are values, it is sufficient to show:

$$(\mathsf{alloc}\,(\sigma_1\, t_1)\,(\sigma_1\, t_2), \mathsf{alloc}\,(\sigma_2\, t_1')\,(\sigma_2\, t_2')) \in (\!|\{P\}\ \exists\gamma.\ \overset{\mathrm{diff}(\delta D_1+\delta D_2)}{\mathsf{Array}_\gamma[I]\ (\delta\tau)}\ \{P \star \gamma \to \mathbb{N}\}|\!)_{G,k}$$

By the definition of the forcing evaluation we have:

$$\frac{\sigma_1\, t_1 \Downarrow^{c_1,k_1} n \quad \sigma_1\, t_2 \Downarrow^{c_2,k_2} v \quad z = [\overbrace{v,\ldots,v}^{n}] \quad l \text{ fresh}}{\mathsf{alloc}\,(\sigma_1\, t_1)\,(\sigma_1\, t_2); H_1 \Downarrow_f^{c_1+c_2+c_{\mathsf{alloc}},k_1+k_2+1} l; H_1 \uplus [l \to z]}\ \text{f-alloc}$$

$$\frac{\sigma_2\, t_1' \Downarrow^{c_1',k_1'} n' \quad \sigma_2\, t_2' \Downarrow^{c_2',k_2'} v' \quad z' = [\overbrace{v',\ldots,v'}^{n'}] \quad l' \text{ fresh}}{\mathsf{alloc}\,(\sigma_2\, t_1')\,(\sigma_2\, t_2'); H_2 \Downarrow_f^{c_1'+c_2'+c_{\mathsf{alloc}},k_1'+k_2'+1} l'; H_2 \uplus [l \to z']}\ \text{f-alloc}$$

So, unfolding the definition of interpretation, for $G' \supseteq G$, $k' \le k$, and arbitrary heaps H_1, H_2 such that $(H_1, H_2) \vDash_{G',k'} P$ and $k_1 + k_2 + 1 < k' \le k$. Now we define $G_2 = G'[r \to (l, l', \delta\tau, \delta I)$, $H_1' = H_1 \uplus [l \to z]$ and $H_2' = H_2 \uplus [l' \to z']$, we must show:

1. $(H_1', H_2') \vDash_{G_2,k'-(k_1+k_2+1)} P \star \gamma \to \mathbb{N}$

2. $(l, l') \in (\!|\mathsf{Array}_\gamma[\delta\, I]\ (\delta\tau)|\!)_{G_2, k'-(k_1+k_2+1)}$.

3. $(c_1 + c_2 + c_{\mathsf{alloc}}) - (c_1' + c_2' + c_{\mathsf{alloc}}) \leq (\delta\, D_1 + \delta\, D_2)$.

By induction hypothesis, from the first premise $\Sigma; \Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathrm{int}[I]$ and Lemma 1 we conclude:

$$(\sigma_1\, t_1, \sigma_2\, t_1') \in (\!|\mathrm{int}[\delta\, I]|\!)^e_{G', k', \delta\, D_1}$$

From this we have: $(n, n') \in (\!|\mathrm{int}[\delta\, I]|\!)_{G', k'-k_1}$, which in turn tells us:

$$n = n' = \delta\, I \ \wedge\ c_1 - c_1' \leq \delta\, D_1$$

By induction hypothesis, from the second premise $\Sigma; \Delta; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau$ and Lemma 1 we have:

$$(\sigma_1\, t_2, \sigma_2\, t_2') \in (\!|\delta\tau|\!)^e_{G', k', \delta\, D_2}$$

From this we have:

$$(v, v') \in (\!|\delta\tau|\!)_{G', k'-k_2} \ \wedge\ c_2 - c_2' \leq \delta\, D_2$$

Now to conclude we can proceed as follows:

1. $(H_1', H_2') \vDash_{G_2, k'-(k_1+k_2+1)} P \star \gamma \to \mathbb{N}$. It is proved depending on γ_i. When $\gamma_i \neq \gamma$, the corresponding arrays is not changed, so the claim is trivial. When $\gamma_i = \gamma$, we can show that $\forall i < \delta\, I. H_1'(l)[i] \neq H_2'(l')[i] \Rightarrow i \in \mathbb{N}$.

2. $(l, l') \in (\!|\mathsf{Array}_\gamma[\delta\, I]\ (\delta\tau)|\!)_{G_2, k'-(k_1+k_2+1)}$ is proved by unfolding its definition using $G_2 = G'[r \to (l, l', \delta\tau, \delta\, I)$.

3. $(c_1+c_2+c_{\mathsf{alloc}})-(c_1'+c_2'+c_{\mathsf{alloc}}) \leq (\delta\, D_1+\delta\, D_2)$ is proved by the previous conclusions.

$$\textbf{Case}\ \frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash_{L_1}^{U_1} t_1' : \{P_1\}\ \exists\vec{\gamma}_1 : A_1'\ \{Q_1\}^{\mathrm{exec}(L,U)} \\ \Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash_{L_2}^{U_2} t_2 : \{P_2\}\ \exists\vec{\gamma}_1 : A_1\ \{Q_2\}^{\mathrm{exec}(L',U')} \qquad dom(P) = dom(P_1) \\ \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1', A_1') \vdash t_2 \ominus t_2' \lesssim D_2 : \{P \sqcup P_1\}\ \exists\vec{\gamma}_1.\tau'\ \{Q\}^{\mathrm{diff}(D')}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus \mathsf{let}\{x\} = t_1' \ \mathsf{in}\ t_2' \lesssim U_2 : \qquad \{P\}\ \exists\vec{\gamma}_1.\tau'\ \{Q\}^{\mathrm{diff}(D'+(D_2-L_2)-L_1-L-c_{\mathsf{let}})}}\ \textbf{r-e-bind}$$

By assumption we have $\vdash \delta : \Delta$, $\vDash \delta\Phi_a$ and $(\sigma_1, \sigma_2) \in [\![\delta\Gamma]\!]_{G,k}$, we need to show:

$$(\sigma_1\ t_2, \mathsf{let}\{x\} = (\sigma_2\ t_1')\ \mathsf{in}\ (\sigma_2\ t_2')) \in (\!|\ \{P\}\ \exists\vec{\gamma}_1.\delta\tau'\ \{Q\}^{\mathrm{diff}(\delta(D'+(D_2-L_2)-L_1-L-c_{\mathsf{let}}))}\ |\!)^e_{G,k,\delta\,U_2}$$

By the definition of the evaluation of $\sigma_1\ t_2$ of the form $\sigma_1\ t_2 \Downarrow^{c_1,k_1} v_1$, and the evaluation of the monadic bind of the form, we have:

$$(\mathsf{let}\{x\} = \sigma_2\ t_1'\ \mathsf{in}\ \sigma_2\ t_2') \Downarrow^{0,0} \mathsf{let}\{x\} = \sigma_2\ t_1'\ \mathsf{in}\ \sigma_2\ t_2'$$

So, unfolding the definition of the expression interpretation, we must show:

1. $c_1 - 0 \le \delta\,U_2$.
2. $(v_1, (\mathsf{let}\{x\} = (\sigma_2\ t_1')\ \mathsf{in}\ (\sigma_2\ t_2'))) \in (\!|\delta(\ \{P\}\ \exists\vec{\gamma}_1.\delta\tau\ \{Q\}^{\mathrm{diff}(\delta(D'+(D_2-L_2)-L_1-L-c_{\mathsf{let}}))}\ )|\!)_{G,k-k_1}$.

By Theorem 2, from the second premise $\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash_{L_2}^{U_2} t_2 : \{P_2\}\ \exists\vec{\gamma}_1 : A_1\ \{Q_2\}^{\mathrm{exec}(L',U')}$, instantiated with $\sigma_1 \in [\![\delta|\Gamma|_1]\!]_{|G|_1,k}$ inferred from $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$, we have:

$$\sigma_1\ t_2 \in [\![\{P_2\}\ \exists\vec{\gamma}_1 : \delta A_1\ \{Q_2\}^{\mathrm{exec}(\delta L',\delta U')}]\!]^e_{|G|_1,k,(\delta L_2,\delta U_2)}$$

From this we have:

$$v_1 \in [\![\{P_2\}\ \exists\vec{\gamma}_1 : \delta A_1\ \{Q_2\}^{\mathrm{exec}(\delta L',\delta U')}]\!]_{|G|_1,k} \wedge \delta\,L_2 \le c_1 \le \delta\,U_2$$

which proves our first goal: $c_1 - 0 \leq \delta\, U_2$. We then need to show:

$$(v_1, \mathsf{let}\{x\} = \sigma_2\, t_1' \;\mathsf{in}\; \sigma_2\, t_2') \in (\!|\overset{\mathrm{diff}(\delta(D'+(D_2-L_2)-L_1-L-c_{\mathsf{let}}))}{\{P\}\ \exists\vec{\gamma}_1.\delta\tau\ \{Q\}}|\!)_{G,k-k_1}$$

By the definition of the forcing evaluation we have:

$$v_1; H \Downarrow_f^{c_2,k_2} v_2; H_2$$

$$\frac{\sigma_2\, t_1' \Downarrow^{c_1',k_1'} v' \quad v'; H' \Downarrow_f^{c_2',k_2'} v_1'; H_1' \quad \sigma_2\, t_2'[v_1'/x] \Downarrow^{c_3',k_3'} v_2' \quad v_2'; H_1' \Downarrow_f^{c_4',k_4'} v_3'; H_2'}{\mathsf{let}\{x\} = \sigma_2\, t_1' \;\mathsf{in}\; \sigma_2\, t_2'; H' \Downarrow_f^{c_1'+c_2'+c_3'+c_4'+c_{\mathsf{let}},k_1'+k_2'+k_3'+k_4'+1} v_3'; H_2'}\ \text{f-bind}$$

So, unfolding the definition of the interpretation, for $G' \supseteq G$, $k' \leq k - k_1$, arbitrary heaps H, H' such that $H, H' \vDash_{G',k'} P$ and $k_2 < k' \leq k - k_1$, we must show:

1. $(H_2, H_2') \vDash_{G',k'-k_2} Q$.
2. $(v_2, v_3') \in (\!|\delta\tau|\!)_{G',k'-k_2}$.
3. $c_2 - (c_1' + c_2' + c_3' + c_4' + c_{\mathsf{let}}) \leq \delta(D' + (D_2 - L_2) - L_1 - L - c_{\mathsf{let}})$.

By induction hypothesis using Theorem 2, from the first premise, we have:

$$\sigma_2\, t_1' \in [\![(\{P_1\}\ \exists\vec{\gamma}_1 : \delta A_1'\ \{Q_1\})]\!]^{\mathrm{exec}(\delta L,\delta U)}_{|G|_2,k',(\delta L_1,\delta U_1)}$$

From this we have:

$$v' \in [\![\{P_1\}\ \exists\vec{\gamma}_1 : \delta A_1'\ \{Q_1\}]\!]^{\mathrm{exec}(\delta L,\delta U)}_{|G|_2,k'-k_1'} \ \wedge\ \delta L_1 \leq c_1' \leq \delta U_1$$

which in turn tells us the following when we define $g_2 = |G|_2$:

$$H_1' \vDash_{g_2,k'-k_2'} Q_1 \ \wedge\ v_1' \in [\![\delta A_1']\!]_{g_2,k'-k_2'} \ \wedge\ \delta L \leq c_2' \leq \delta U$$

From this, we can conclude:

$$(\sigma_1[v_1'/x], \sigma_2[v_1'/x]) \in (\!|\delta(\Gamma, x : U(A_1', A_1'))|\!)_{G_2,k}$$

By induction hypothesis, from the third premise instantiated with $(\sigma_1[v_1'/x], \sigma_2[v_1'/x]) \in (\!|\delta(\Gamma, x : U(A_1', A_1'))|\!)_{G_2,k}$, we have:

$$(\sigma_1\ t_2[v_1'/x], \sigma_2\ t_2'[v_1'/x]) \in (\!|\{P \cup P_1\}\ \overset{\text{diff}(\delta\, D')}{\exists \vec{\gamma}_1}.\delta\tau'\ \{Q\}|\!)^{e}_{G_2,k,\delta\, D_2}$$

From this we have:

$$(v_1, v_2') \in (\!|\{P \cup P_1\}\ \overset{\text{diff}(\delta\, D')}{\exists \vec{\gamma}_1}.\delta\tau'\ \{Q\})|\!)_{G_2,k-k_1}\ \wedge\ c_1 - c_3' \leq \delta\, D_2$$

We unfold $(v_1, v_2') \in (\!|\{P \cup P_1\}\ \overset{\text{diff}(\delta\, D')}{\exists \vec{\gamma}_1}.\delta\tau'\ \{Q\})|\!)_{G_2,k-k_1}$ and have: $(H, H_1') \vDash_{G',k'} P \sqcup P_1$ by using the definition of the heap relation and the premise $dom(P) = dom(P_1)$. We choose $G_2 = G'$ and then have:

$$(v_2, v_3') \in (\!|\delta\tau'|\!)_{G',k'-k_2}\ \wedge\ c_2 - c_4' \leq \delta\, D'\ \wedge\ (H_2, H_2') \vDash_{G',k'-k_2}$$

Now to conclude we can proceed as follows:

1. $(H_2, H_2') \vDash_{G',k'-k_2} Q$ is already proved by the previous conclusions.
2. $(v_2, v_3') \in (\!|\delta\tau|\!)_{G',k'-k_2}$ is already proved by the previous conclusions.
3. $c_2 - (c_1' + c_2' + c_3' + c_4' + c_{\mathsf{let}}) \leq \delta(D' + (D_2 - L_2) - L_1 - L - c_{\mathsf{let}})$ is proved by previous conclusions.

□

5.3.4 Soundness of ARel.

When we have the fundamental theorems for both the unary and relational typing, we can easily have the soundness of our system. The soundness statement covers four possible cases which are the combination of the two kinds of computations we have in ARel, pure and impure, and the two kinds of typing, unary and relational. It can also be found in appendix A, Lemma 3.

Lemma 3 (Soundness for Costs)**.**

1. *If* $\vdash_L^U t : A$ *and* $t \Downarrow^c v$, *then* $L \le c \le U$.
2. *If* $\vdash_0^0 v : \{P\}\ \overset{\mathrm{exec}(L',U')}{\exists \gamma : A}\ \{Q\}$, *and there exists* H *such that* $H \vDash_{g,k'} P$ *and* $v; H \Downarrow_f^{c',k''} v', H'$ *and* $k'' < k'$, *then* $L' \le c' \le U'$.
3. *If* $\vdash t_1 \ominus t_2 \lesssim D : \tau$ *and* $t_1 \Downarrow^{c_1} v_1$ *and* $t_2 \Downarrow^{c_2} v_2$, *then* $c_1 - c_2 \le D$.
4. *If* $\vdash v_1 \ominus v_2 \lesssim 0 : \{P\}\ \overset{\mathrm{diff}(D')}{\exists \gamma : \tau}\ \{Q\}$, *and there exist heaps* H_1, H_2 *such that* $H_1, H_2 \vDash_{G,k'} P$, *and* $v_1; H_1 \Downarrow_f^{c_1',k''} v_1'; H_1'$, $v_2; H_2 \Downarrow_f^{c_2',-} v_2'; H_2'$ *and* $k'' < k'$, *then* $c_1' - c_2' \le D'$.

Proof.

1. Proof of statement (1) follows by the fundamental theorem for unary typing directly . If $\vdash_L^U t : A$, then $t \in [\![A]\!]_{G,k}^{E,(L,U)}$ for some k and g. We can then conclude the following by unfolding the definition of the expression interpretation : $L \le c \le U$ and $t \Downarrow^c v$.

2. Proof of statement (2) is proved by the interpretation of the unary monadic types. From the fundamental theorem, we can have $v \in [\![\{P\}\ \overset{\mathrm{exec}(L',U')}{\exists \gamma : A}\ \{Q\}]\!]_{G,k}^{E,(L,U)}$ for some g and $k \ge k'$. We can conclude the following by unfolding the definition of the interpretation:$L' \le c' \le U'$.

3. Proof of statement (3). From the fundamental theory of relational types, if $\vdash t_1 \ominus t_2 \lesssim D : \tau$, we know closed terms (t_1, t_2) will be in the interpretation of re-

lational type τ, $(\!|\tau|\!)^{E,D}_{G,k}$ for some G and $k \geq k'$. We can conclude the following by unfolding the definition of expression interpretation of relational type: $c_1 - c_2 \leq D$.

4. The proof of statement (4). From the fundamental theory,
   If $\vdash v_1 \ominus v_2 \lesssim 0 : \{P\}\ \exists\gamma : \tau\ \{Q\}$, we know closed terms (v_1, v_2) in the value interpretation $(\!|\{P\}\ \exists\gamma : \tau\ \{Q\}|\!)_{G,k}$ for some G and $k \geq k'$, we conclude the following by unfolding the definition of the value interpretation of the relational monadic type: $c_1' - c_2' \leq D'$.

□

In ARel, we require the elements of the arrays to be of the base type so that the return type of the monadic type will be base type, as mentioned in Section 4.2.2. So we do not consider cases that the return type of a monadic type is also another monadic type. In the unary setting, the soundness considers two cases. Statement (1) talks about the pure evaluation cost c of a program t of unary type A, showing that c is bounded by the lower bound L and U in the typing judgment. This statement is general to A so that A can be either monadic or not. The statement (2) shows that the forcing evaluation cost c' of a value v with a monadic type $\{P\}\ \exists\gamma : A\ \{Q\}$ will be bounded by the lower bound L' and upper bound U' in the monadic type.

The statement (3) and (4) talk about the relational cases. Statement (3) shows if two programs t_1 and t_2 are related by a judgment in ARel, then the relative cost of the two programs is upper bounded by the bound D of the judgment. In this case, the relational type τ can be either a monadic type or not. Statement (4) only talks about impure computations. The difference of the forcing evaluation costs of two values related at a relational monadic type $\{P\}\ \exists\gamma : \tau\ \{Q\}$ is upper bounded by the bound D' of the relational monadic type.

CHAPTER 6

Examples Analyzed by ARel

Examples are easier to understand than words, so in this chapter, we show five more examples demonstrating how we perform relational cost analysis on programs with arrays. To improve readability, we omit some annotations and use syntactic sugar. For example, we abbreviate $\mathsf{let}\{x\} = t_1 \ \mathsf{in} \ t_2$ to $x \leftarrow t_1 \ ; \ t_2$ and even to $t_1; t_2$ when x does not appear in t_2. We also shorten the type $U(A, A)$ to $U(A)$ and use $\mathsf{if} \ t \ \mathsf{then} \ t_1 \ \mathsf{else} \ t_2$ as syntactic sugar for $\mathsf{case}(t, x.t_1, y.t_2)$ when x and y do not appear in t_1 and t_2, respectively.

In some of the examples, such as the Cooley-Tukey FFT Algorithm below, we add a dummy first argument of type unit to a recursive function. This allows us to make the recursive function polymorphic in index terms.

6.1 COOLEY-TUKEY FAST FOURIER TRANSFORMATION ALGORITHM

Fast Fourier Transform (short for 'FFT') is an algorithm that computes discrete Fourier Transform of a sequence, as in our example, in the form of an array. As the most widely used FFT algorithm, the Cooley-Tukey algorithm uses the idea of divide-and-conquer and recursively breaks the sequence into smaller parts. Throughout this example, we want to show how to use relational cost analysis to reason about programs with imperative updates. We consider the following implementation of the Cooley-Tukey algorithm for fast Fourier transforms (Cooley & Tukey, 1965) in Figure 6.1. The full detail can be found in Appendix A, Example A.6.1.

The recursive function FFT uses divide-and-conquer. The variable x is the input array, y is another array used for temporary storage, m is the length of the range of the array to be transformed, and p is an index that specifies the starting index of the range of the array to be split in the recursive call. This function uses a helper function separate

```
fix FFT ().λx.λy.λm.λp.
  if 2 ≤ m then
    separate () x m y p ;
    FFT () x y (m/2) p ;
    FFT () x y (m/2) (p + m/2) ;
    loop () 0 m x p
  else
    return ()

fix separate ().λx.λm.λy.λp.
  sp() x m 0 y p ;
  cp() y x p (m + p)

fix loop ().λk.λm.λx.λp.
  if k < (m/2) then
    e ←read x (k + p) ;
    o ←read x (k + p + m/2) ;
    let w = exp(−2πk/m) in
    updt x (k + p) (e + w ∗ o) ;
    updt x (k + p + m/2) (e − w ∗ o) ;
    loop () (k + 1) m x p
  else
    return ()
```

Figure 6.1: Code of FFT.

to relocate elements in even positions to the lower half of the array x and elements in odd positions to the upper half of the array respectively, using y as a scratchpad. The function separate also uses two helper functions – the function sp does the separation work using temporary storage and the function cp copies the separated part from the temporary storage back to the original array. We omit the code of sp and cp here, which is standard; this code can be found in Appendix A, Example A.6.1.

Another helper function loop simulates a for loop in which the input array x is updated using the mathematical manipulations needed for the Fourier transform. Intuitively, this example is constant time (for arrays with fixed length) because array manipulations depend only on the positions of the elements, and not on their values (assuming constant time addition and multiplication). One way to internalize this observation in the typing process is using only relational rules and relational types, always with the relative cost 0. We do this for the auxiliary functions separate and loop first, the following

is the relational typing of separate and loop.

$$\begin{aligned}
&\qquad\qquad \mathrm{unit} \to \forall \gamma_1, \gamma_2, \beta_1, M, N, P.(P + M < N) \supset \\
&\vdash \mathsf{separate} \ominus \mathsf{separate} \lesssim 0 : \ \big(\mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}) \to \mathrm{int}[M] \to \mathsf{Array}_{\gamma_2}[N]\ U(\mathrm{int}) \to \\
&\qquad\qquad \mathrm{int}[P] \to \{\gamma_1 \to \beta_1, \gamma_2 \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists}_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\} \big)
\end{aligned}$$

$$\begin{aligned}
&\qquad\qquad \mathrm{unit} \to \forall \gamma_1, \beta_1. \forall K, M, N, P.(P + M < N) \supset \\
&\vdash \mathsf{loop} \ominus \mathsf{loop} \lesssim 0 : \ \big(\mathrm{int}[K] \to \mathrm{int}[M] \to \mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}) \to \mathrm{int}[P] \to \\
&\qquad\qquad \{\gamma_1 \to \beta_1\}\ \overset{\mathrm{diff}(0)}{\exists}_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}\} \big)
\end{aligned}$$

The constraint $P + M < N$ guarantees the array bound limit. With the relational type of these auxiliary functions, we can easily give the following relational type to FFT, witnessing the constant-time nature of this function.

$$\begin{aligned}
&\qquad\qquad \mathrm{unit} \to \forall \gamma_1, \gamma_2, \beta_1, M, N, P.(P + M < N) \supset \\
&\vdash \mathsf{FFT} \ominus \mathsf{FFT} \lesssim 0 : \ \big(\mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}) \to \mathsf{Array}_{\gamma_2}[N]\ U(\mathrm{int}) \to \mathrm{int}[M] \to \\
&\qquad\qquad \mathrm{int}[P] \to \{\gamma_1 \to \beta_1, \gamma_2 \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists}_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\} \big)
\end{aligned}$$

Another way to achieve the same result is to first compute the precise lower and upper bounds on the unary cost of FFT and then show that they are, in fact, equal. However, computing the precise unary cost of FFT is more difficult. We first establish the precise cost of the auxiliary functions. For example, we need to give the following unary type to separate and loop.

$$\begin{aligned}
&\qquad\qquad \mathrm{unit} \to \forall \gamma_1, \gamma_2, M, N, P.(M + P < N) \supset \\
&\vdash \mathsf{separate} : \ \big(\mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \mathrm{int}[M] \to \mathsf{Array}_{\gamma_2}[N]\ \mathrm{int} \to \mathrm{int}[P] \to \\
&\qquad\qquad \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\}\ \overset{\mathrm{exec}(4*M, 4*M)}{\exists}_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\} \big)
\end{aligned}$$

$$\vdash \mathsf{loop} : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1, K, M, N, P. (P + M < N) \supset \\ \left(\mathrm{int}[K] \to \mathrm{int}[M] \to \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \mathrm{int}[P] \to \{\gamma_1 \to \mathbb{N}\}\ \exists \_. \mathrm{unit}^{\mathrm{exec}(4*(M-K),4*(M-K))}\ \{\gamma_1 \to \mathbb{N}\} \right) \end{array}$$

Once these unary costs are available, we can conclude that the function FFT has the same min and max costs: $8 * M * \log(M)$ and is, thus, constant time (using the rule r-switch in Figure 4.7).[1]

While both the unary and relational reasoning can show that this example is constant time, the relational reasoning is *much* easier in this case since the relative cost is 0 everywhere.

6.2 NAIVE STRING SEARCH

We show how a combination of unary and relational reasoning can give a precise relative cost to a classic array algorithm: naive substring search. The full detail can be found in Appendix A, Example A.6.2.

As in our example, strings are represented as arrays of integers (storing the ASCII code of each character). In Figure 6.2, the function NSS takes as input, a "long" string s and a "short" string w in the form of arrays, the lengths l_s and l_w of these arrays, and an array p of length l_s (we call this the result array). This function iteratively searches the substring w at each position in s and records whether the substring is found at that position (1) or not (0).

To do this, NSS uses helper function search, which is also in Figure 6.2.

The function search has the same inputs as NSS except for the additional index i, that

[1] It is not hard to see that FFT has unary cost in $O(M * \log(M))$: The unary costs of separate and the call to loop() are both linear in M, so the cost $f(M)$ of FFT satisfies the recurrence $f(M) = 2 * f(M/2) + O(M)$, which has the standard solution $O(M * \log(M))$. However, proving this in the type system is much harder than the direct relational proof of 0 relative cost.

```
fix NSS (s).λw.λm.λl_s.λl_w.λp.
  if (m + l_w) ≤ l_s then
    search s w m 0 l_s l_w p ;
    NSS s w (m+1) l_s l_w p
  else
    return ()
```

```
fix search ().λs.λw.λm.λi.λl_s.λl_w.λp.
  x ←read s (m + i) ;
  y ←read w i ;
  if i + 1 == l_w then
    if x == y then
      updt p m 1
    else
      updt p m 0
  else
    if x == y then
      search () s w m (i + 1) l_s l_w p
    else
      updt p m 0
```

Figure 6.2: Code of NSS.

iterates over the positions of l_w. The two conditionals check whether search is in its final step ($i + 1 == l_w$), and whether the two corresponding characters in s and w coincide. When the two characters differ, p is updated with 0. When the two conditionals are satisfied at the same time, p is updated with 1. Thanking its recursive nature, search allows the substring search index by index until a result is obtained.

Intuitively, search runs fastest when the first character of w does not appear in s. It runs slowest when the suffix of w starting at index i occurs in s at offset $m + i$. The difference between these two costs is a bound on the relative cost of search. However, in the specific case where we consider two runs of search on the same string s, the same index i and where the two ws agree on some prefix, we can see that the two runs behave identically until we reach an index where the ws start to differ. In this case, we can give a better bound. To write this bound, we need to express the first index in the range $[i, l_w]$ where the two ws differ. In ARel, the index term $MIN(\beta_2 \cap [I, \infty))$ represents this index (assuming β_2 is the relational precondition of w and I is the static index refinement for i's size). Then, search incurs a nontrivial relative cost only *after* this index is reached.

Using this idea, we can show:

$$\vdash \mathsf{search} \ominus \mathsf{search} \lesssim 0 : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1, \gamma_2, \gamma_3, I, M, R, N, \beta_2, \beta_3. \\ (I < R < N \wedge M + I < N) \supset \\ \overset{\mathrm{diff}((R-1-min(MIN(\beta_2 \cap [I,\infty)),R-1))*r)}{\big(T \to \{P, \gamma_3 \to \beta_3\}\ \exists \_.\mathrm{unit}\ \{P, \gamma_3 \to \beta_3 \cup \{M\}\}\big)} \end{array}$$

where $T = \mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}) \to \mathsf{Array}_{\gamma_2}[R]\ U(\mathrm{int}) \to \mathrm{int}[M] \to \mathrm{int}[I] \to \mathrm{int}[N] \to \mathrm{int}[R] \to \mathsf{Array}_{\gamma_3}[N]\ U(\mathrm{bool})$

where $P = \gamma_1 \to \emptyset, \gamma_2 \to \beta_2$, R is the static size of l_w, and r is the (constant) cost of two read operations. The constraint $I < R$ guarantees that the search will not exceed the length of the array w and the constraint $R < N$ guarantees that w is shorter than s. The other constraint $M + I < N$ guarantees that the search will not exceed the length of the array s. The post-condition modifies only the set associated with γ_3, from β_3 to $\beta_3 \cup \{M\}$, unveiling that only the result array p will be overwritten. To account for the case where w is the same in the two executions we also add a lower bound $R-1$ in the cost. The relative cost we establish here is more precise than the one we would achieve with a non-relational analysis $((R-1-I)*r)$.

We stress here that to obtain this relative cost, the rule r-fix-ext in Figure 4.6 is essential. At a high level, typing proceeds by case analysis on $I \in \beta_2$. When $I \notin \beta_2$ we can proceed relationally with relative cost 0 in the recursive call. When $I \in \beta_2$ the control flows may differ in the two runs and we need to switch to unary reasoning via the rule r-switch. To obtain our bound using unary worst- best-case analysis we need the precise unary type of search, which is available in the context thanks to the rule r-fix-ext. The details of this proof are in Appendix A. By using the typing above for search, we can also obtain an improved relative cost for NSS relative to itself: $(R-1-min(MIN(\beta_2 \cap [I,\infty)),R-1))*$

$r * (N - M - R)$. This is simply the number of times search is called $(N - M - R)$ multiplied by the relative cost of search.

6.3 MERGESORT

As our next example, we consider an imperative version of mergesort. Similar to what we did for the *mapi* example, we present two relational types for this mergesort, corresponding to two different assumptions on the inputs. The full detail can be found in Appendix A, Example A.6.3.

Consider the function msort in Figure 6.3 which sorts the elements of an array a from index l to u, using another array b as buffer. We use here an auxiliary function merge, merging the two sorted partitions of the array, defined in the same figure. The function merge takes in input the array a, the buffer array b, the starting index l and ending index u, and additionally asks for a "midpoint" index m at which the array range is divided. It uses two helper functions: the function $\mathsf{merge}_{\mathsf{lp}}$ which implements the standard merging process in a recursive way, and the function copy which copies the merged buffer array back to the working array. We omit the code of the function copy here (it can be found in Example A.6.3.) and we show just the code of the function $\mathsf{merge}_{\mathsf{lp}}$.

The argument k tells the position where to store the merged element in the buffer array b, the four arguments l_s, l_e, r_s and r_e separate the array a into two portions, the left one is specified by the variables l_s and l_e representing its starting and ending indices, whereas the right portion is specified by r_s and r_e. The idea underlying this function is to check if there is an element available in the left portion and right portion using conditionals, and then compare two elements, each one from one side, before storing the smaller one in the buffer array at the position pointed by k. Then this pointer is updated, and the process is repeated recursively.

```
fix msort (). λa.λb.λl.λu.
  if l ≥ u then
    return ()
  else
    msort a b l (l + ⌊(u − l)/2⌋) ;
    msort a b (l + ⌊(u − l)/2⌋ + 1) u ;
    merge a b l (l + ⌊(u − l)/2⌋) u
```

```
merge = λa.λb.λl.λm.λu.
  merge_lp l a l m (m + 1) u b ;
  copy l a b u
```

```
fix merge_lp (k) λa.λl_s.λl_e.λr_s.λr_e.λb.
  if l_s < l_e then
    if r_s < r_e then
      x ←read a l_s ;
      y ←read a r_s ;
      if x < y then
        updt b k x ;
        merge_lp (k + 1) a (l_s + 1) l_e r_s r_e b
      else
        updt b k y ;
        merge_lp (k + 1) a l_s l_e (r_s + 1) r_e b
    else
      x ←read a l_s ;
      updt b k x ;
      merge_lp (k + 1) a (l_s + 1) l_e r_s r_e b
  else
    if r_s < r_e then
      x ←read a r_s ;
      updt b k x ;
      merge_lp (k + 1) a l_s l_e (r_s + 1) r_e b
    else
      return ()
```

Figure 6.3: Code of msort.

The first relational type we want to show for msort assumes that the two arrays, which are input to the two runs, are equal in the range $[l,u]$. Intuitively, the fact that we have the same array elements in the range $[l,u]$ implies that we have the same execution path, and so we can prove that the relative cost is 0.

$$\begin{aligned}\vdash \mathsf{msort} \ominus \mathsf{msort} \lesssim 0 : \; & \mathrm{unit} \to \forall\, l,u,\beta,\gamma,\gamma',N. \\ & l \le u \le N \wedge \beta \cap [l,u] = \emptyset \supset \\ & \big(\mathsf{Array}_{\gamma}[N]\ U(\mathrm{int}) \to \mathsf{Array}_{\gamma'}[N]\ U(\mathrm{int}) \to \mathrm{int}[l] \to \\ & \mathrm{int}[u] \to \{\gamma \to \beta, \gamma' \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists}\_.\mathrm{unit}\ \{\gamma \to \beta, \gamma' \to \mathbb{N}\}\big)\end{aligned}$$

The assumption that the two arrays from the two runs are equal in the range $[l,u]$ is represented by the constraint $(\beta \cap [l,u] = \emptyset)$, where β is a variable representing the indices at which the two arrays can differ. Using this assumption, we can derive relational types asserting relative cost 0 for both $\mathsf{merge}_{\mathsf{lp}}$ and copy, and thus for merge. We first show the type of $\mathsf{merge}_{\mathsf{lp}}$ and then merge below.

$$\begin{aligned}\vdash \mathsf{merge}_{\mathsf{lp}} \ominus \mathsf{merge}_{\mathsf{lp}} \lesssim 0 : \; & \forall N, l_s, l_e, r_s, r_e, K, \gamma, \gamma', \beta. \\ & l_s \le l_e \le r_s \le r_e \le N \wedge \beta \cap [K, r_e] = \emptyset \supset \\ & \big(\mathrm{int}[K] \to \mathsf{Array}_{\gamma}[N]\ U(\mathrm{int}) \to \mathrm{int}[l_s] \to \mathrm{int}[l_e] \to \\ & \mathrm{int}[r_s] \to \mathrm{int}[r_e] \to \mathsf{Array}_{\gamma'}[N]\ U(\mathrm{int}) \to \\ & \{\gamma \to \beta, \gamma' \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists}\_.\mathrm{unit}\ \{\gamma \to \beta, \gamma' \to \mathbb{N} \setminus [l,u]\}\big)\end{aligned}$$

$$\begin{aligned}\vdash \mathsf{merge} \ominus \mathsf{merge} \lesssim 0 : \; & \forall\, l,m,u,\beta,\gamma,\gamma',N. \\ & l \le u \le N \wedge \beta \cap [l,u] = \emptyset \supset \\ & \big(\mathsf{Array}_{\gamma}[N]\ U(\mathrm{int}) \to \mathsf{Array}_{\gamma'}[N]\ U(\mathrm{int}) \to \mathrm{int}[l] \to \\ & \mathrm{int}[m] \to \mathrm{int}[u] \to \{\gamma \to \beta, \gamma' \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists}\_.\mathrm{unit}\ \{\gamma \to \beta, \gamma' \to \mathbb{N}\}\big)\end{aligned}$$

Notice that we also restrict the value of u to be between the starting index l and the array length of a. Moreover, notice that precondition and postcondition are equal. This follows again from the assumption we have for this typing: since the two arrays coincide in the range $[l,u]$ before the sort, they will also coincide after the sort, so no new element will be added to β.

We now want to show a more general relational type for msort that doesn't rely on the assumption about the two input arrays to be equal in the range $[l,u]$. We start from reasoning about a general type for the function $\mathsf{merge}_{\mathsf{lp}}$. Without the previous assumption, we may now have different possible execution paths for two runs of this function. We use the rule r-switch to switch to the unary analysis to get the relative cost and the rule R-X to use the subtyping rule r-rum for the proper monadic type. Using this approach we can get the following relational type:

$$\begin{aligned} &\forall K, l_s, l_e, r_s, r_e, \beta, \gamma, \gamma', N. \\ \vdash \mathsf{merge}_{\mathsf{lp}} \ominus \mathsf{merge}_{\mathsf{lp}} \lesssim 0 : \;& l_s \le l_e \le r_s \le r_e \le N \wedge K \le r_e \supset \\ &\left(T \to \{\gamma \to \beta, \gamma' \to \mathbb{N}\}\ \exists\_.\mathrm{unit}\ \{\gamma \to \beta, \gamma' \to \mathbb{N}\}\right)^{\mathsf{diff}(\mathsf{max}(l_e - l_s, r_e - r_s))} \end{aligned}$$

where $T = \mathrm{int}[K] \to \mathsf{Array}_{\gamma}[N]\ U(\mathrm{int}) \to \mathrm{int}[l_s] \to \mathrm{int}[l_e] \to \mathrm{int}[r_s] \to \mathrm{int}[r_e] \to \mathsf{Array}_{\gamma'}[N]\ U(\mathrm{int})$

So, we have that the relative cost of two executions of $\mathsf{merge}_{\mathsf{lp}}$ is $\mathsf{max}(l_e - l_s, r_e - r_s)$. The function copy does not contain branches and so it does not introduce a relative cost.

Using the relational type described above we can derive the following relational type

for merge.

$$\begin{aligned}
&\forall l, m, u. \forall \beta. \forall \gamma, \gamma'. \forall N : \mathbb{N}.\ l \le m \le u \le N \supset \\
\vdash \mathsf{merge} \ominus \mathsf{merge} \lesssim 0 :\ & \mathsf{Array}_{\gamma}[N]\ U(\mathrm{int}) \to \mathsf{Array}_{\gamma'}[N]\ U(\mathrm{int}) \to \mathrm{int}[l] \to \mathrm{int}[m] \\
& \to \mathrm{int}[u] \to \{\gamma \to \beta, \gamma' \to \overset{\mathrm{diff}(max(m-l, u-(m+1)))}{\mathbb{N}}\}\ \exists _.\mathrm{unit}\ \{\gamma \to \beta \cup [l,u], \gamma' \to \mathbb{N}\}
\end{aligned}$$

With it, we can now give a relational typing to msort.

$$\begin{aligned}
&\forall l, u, \beta, \gamma, \gamma', N.\ l \le u \le N \supset \\
\vdash \mathsf{msort} \ominus \mathsf{msort} \lesssim 0 :\ & \big(\mathsf{Array}_{\gamma}[N]\ U(\mathrm{int}) \to \mathrm{int}[l] \to \mathrm{int}[u] \to \mathsf{Array}_{\gamma'}[N]\ U(\mathrm{int}) \\
& \to \{\gamma \to \beta, \gamma' \to \overset{\mathrm{diff}(Q(u-l+1, |\beta \cap [l,u]|))}{\mathbb{N}}\}\ \exists _.\mathrm{unit}\ \{\gamma \to \beta \cup \{l,u\}, \gamma' \to \mathbb{N}\}\big)
\end{aligned}$$

where $Q(n,\alpha) = \Sigma_{i=0}^{H} h(\lceil 2^{i-1} \rceil) \cdot min(\alpha, 2^{H-i})$ and $H = \lceil \log_2(n) \rceil$. To prove that msort has this relative cost, we need some algebraic properties of the recurrence relation Q, which we postpone to the appendix. While the cost looks complicated, we prove that it is in $O(n \cdot (1 + \log_2(|\beta \cap [l,u]|)))$.

6.4 INPLACE INSERTION SORT

Our next example, *inplace insertion sort*, implements the insertion sort algorithm without any temporary arrays. This is an involved example that combines most of the ideas we discussed in other examples, in order to derive a precise relative cost. The relative cost is complex but we can show that under reasonable assumptions, ARel provides a more precise relative cost than a unary analysis. The algorithm is written in our language in Figure 6.4. The full detail can be found in Appendix A, Example A.6.4. The function ISort sorts the input array s in the range $[i, l_s]$. Intuitively, we observe that the cost of ISort relative to itself should be the sum of the possible cost variation of every recursive call, which is mainly decided by the auxiliary function insert shown in Figure 6.4.

```
fix ISort (). λs.λi.λl_s.
  if i < l_s then
    a ←read s i ;
    b ← insert () s a 0 i ;
    ISort () s (i + 1) l_s
  else
    return ()
```

```
fix insert (). λs.λa.λx.λi.
  b ←read s x ;
  if a ≥ b then
    insert () s a (x + 1) i ;
  else
    shift () s x (i − 1) ;
    updt s x a
fix shift () λs.λidx.λi.
  if idx ≤ i then
    c ←read s i ;
    updt s i + 1 c ;
    shift () s idx (i − 1)
  else
    return ()
```

Figure 6.4: Code of ISort.

The recursive function insert implements the standard operation of inserting an element into an array by finding the right position x to insert the element at and shifting elements behind x in the array backward before updating the value at index x to a. The input arguments x and i specify the range in the array to insert. This function uses a helper function shift, which performs the shift operation. It is not hard to observe that the auxiliary function shift, shifting the elements in the range $[idx, i]$ backward by one index, uses one read operation and one update operation at every index, and finding the right position only needs one read operation. The unary cost of ISort is maximum when the input array is initially sorted in descending order. In contrast, the unary cost is minimum when the input array is initially sorted ascending. Assuming that read and update operations incur a unit cost, the unary type of insert is as follows.

$$\begin{aligned}\vdash \mathsf{insert}: \quad & \mathsf{unit} \to \forall \gamma_1, N, X, I.\ X \le N \wedge I \le N \supset \\ & \big(\mathsf{Array}_{\gamma_1}[N]\ \mathsf{int} \to \mathsf{int} \to \mathsf{int}[X] \to \mathsf{int}[I] \to \{\gamma_1 \to \mathbb{N}\}\ \exists\_.\mathsf{unit}\ \{\gamma_1 \to \mathbb{N}\}^{\mathsf{exec}(I-X+1,2*(I-X)+2)}\big)\end{aligned}$$

With this unary type of insert in hand, we can obtain the relative cost of insert by switch-

ing to unary reasoning and then taking the difference. An interesting observation is that if the input arrays of the two runs coincide in the insertion range $[X, I]$ and the elements 'a' being inserted also agree, then insert's cost relative to itself is 0. The corresponding relational type is shown below, where the constraint $\beta_1 \cap [X, I] = \emptyset$ describes our aforementioned assumption.

$$\vdash \mathsf{insert} \ominus \mathsf{insert} \lesssim 0 : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1, \beta_1, N, A, X, I. \\ X \le N \wedge I \le N \wedge \beta_1 \cap [X, I] = \emptyset \supset \\ \big(\mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}) \to \mathrm{int} \to \mathrm{int}[X] \to \mathrm{int}[I] \to \\ \{\gamma_1 \to \beta_1\}\ \exists^{\mathrm{diff}(0)}\_.\mathrm{unit}\ \{\gamma_1 \to \beta_1\} \big) \end{array}$$

This observation can be used in typing ISort: For every I, we split cases on whether $\beta_1 \cap [0, I] = \emptyset$ or not (using rule r-split). While $\beta_1 \cap [0, I] = \emptyset$, we proceed relationally (with 0 relative cost). Once $\beta_1 \cap [0, I] \neq \emptyset$, we switch to unary reasoning using rule r-switch since control flow may differ in the two runs. We need the rule r-fix-ext to allow us to switch back to unary typing when the control flow actually differs at some point. Using this idea, we obtain a very precise relational cost for ISort.

$$\vdash \mathsf{ISort} \ominus \mathsf{ISort} \lesssim 0 : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1, \beta_1, N, I.\ I \le N \supset \\ \big(\mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}) \to \mathrm{int}[I] \to \mathrm{int}[N] \to \{\gamma_1 \to \beta_1\}\ \exists^{\mathrm{diff}(\frac{N*(N+1)-k*(k+1)}{2})}\_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}\} \big) \end{array}$$

where the index term $k = max(I, min(MIN(\beta_1), N))$ represents the first index where the two arrays differ. The relative cost $\frac{N*(N+1)-k*(k+1)}{2}$ is the sum of all the relative costs generated in the recursive calls corresponding to indices in the range $[k, N]$. Recursive calls up to index k incur 0 relative cost, as noted above. More details are provided in the appendix. We note that the cost obtained here is more precise than the relative cost that can be obtained using unary reasoning alone.

```
fix loop (A). λm.λn.                loopOp = if b then
  if m < n then                                fix loop′ (A). λm.λn.
    if b then                                    if m < n then
      h ←read A m ;                                h ←read A m ;
      let _ = f h in                               let _ = f h in
      loop A (m+1) n                               loop′ A (m+1) n
    else                                         else
      return ()                                    return ()
  else                                         else
    return ()                                    λA.λm.λn.return ()
```

Figure 6.5: Code of Loop Unswitching.

6.5 LOOP UNSWITCHING

In the next example, we will have a look at one technique applied in compiler optimization, known as "loop unswitching". In this example, we show ARel can provide a more precise relative cost than the standard worst-case/best-case analysis when dealing with two programs that are not structurally similar. This is the first example we compare two different programs.

We propose a function loop in Figure 6.5, which operates a simple loop over an input array A, from index m, to the end of the array specified by the length of the array n of type int[N]. Inside the loop, there is an if conditional if b..., whose then branch reads the value in the array at index m and then does some pure computation $f\ h$ on the value h just read before going into the next iteration. For simplicity, we let the else branch do nothing, return unit.

Actually, this program can be transformed a little bit, by pulling out the if conditional from the function body, as we can see in the right part of Figure 6.5, we call it loopOp, which is the optimized version of loop. We suppose the original loop and the optimized loopOp operates on the same array A, has the identical pure computation f inside the loop, and shares the boolean input b in two runs. Intuitively, we think the relative cost

is upper bounded by N when we count one unit cost for elimination forms because the first one checks b at each iteration while the optimized one checks only once. When people use the worst-case/best-case analysis, they obtain the following unary types at first.

$$\vdash \lambda f.\lambda b.\mathsf{loop}: \begin{array}{l}(A \to \mathrm{unit}) \to \mathrm{bool} \to \forall \gamma_1, N, M. \\ \mathsf{Array}_{\gamma_1}[N]\ A \to \mathrm{int}[M] \to \mathrm{int}[N] \to \{\gamma_1 \to \emptyset\}\ \exists \_.\mathrm{unit}\ \{\gamma_1 \to \emptyset\}^{\mathrm{exec}(1,4*(N-M)+1)}\end{array}$$

$$\vdash \lambda f.\lambda b.\mathsf{loopOp}: \begin{array}{l}(A \to \mathrm{unit}) \to \mathrm{bool} \to \forall \gamma_1, N, M. \\ \mathsf{Array}_{\gamma_1}[N]\ A \to \mathrm{int}[M] \to \mathrm{int}[N] \to \{\gamma_1 \to \emptyset\}\ \exists \_.\mathrm{unit}\ \{\gamma_1 \to \emptyset\}^{\mathrm{exec}(1,3*(N-M)+1)}\end{array}$$

In the above unary type, both precondition and postcondition have location γ_1 referring to the empty set. It is consistent with our assumption that f does some pure computation, hence indicating the arrays remain intact within the execution of either loop or loopOp, which requires no writing permission. The unary upper bound loop is $4*(N-M)+1$ when we count the cost of reading the array as one unit cost, the function application fh as one unit cost, and the test of the conditional of the if statement as one unit cost. Similarly for loopOP. With the help of the subtyping rule s-rum, we obtain the relative cost, which is $4*(N-M)$. When we start the loop from the beginning, which means $M=0$, then the relative cost is bounded by $4*N$, instead of our intuition N since the only difference is the test of b for every iteration.

On the other hand, we can do better if we use the relational analysis, in particular, the relational asynchronous rule r-t-case in Figure 4.7. Remember, we use if t then $t_1$else t_2 as the syntactic sugar for the construct case $(t, x.t_1, y.t_2)$. To be precise, we can have a

simplified asynchronous rule r-t-if from r-t-case.

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t' : \text{bool} \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t'_1 \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t'_2 \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus \text{if } t' \text{ then } t'_1 \text{else } t'_2 \lesssim D_2 - L_1 - c_{case} : \tau}\ \text{r-t-if}$$

The above asynchronous rule allows us to compare loop with respect to the inner recursive function loop′ inside loopOp. When we compare the body of loop and loop′, and type the "then" branch of the first if conditional if $m < n$ …, we come across the structurally dissimilar pieces of code: the red(left) part in loop and the green(right) part in loop′ in Figure 6.5. In this turn, we use a similar asynchronous rule r-case-t in Figure 4.7. In particular, we want to avoid comparing the "else" branch return () in the red part of loop with the green part of loop′, when we know the conditional b the same in two runs. To this end, we can refine our boolean type bool as we do for the integer type, bool$[B]$ for unary boolean type and bool$_r[B]$ for the relational one, where $B \in \{\text{true}, \text{false}\}$ so that the erasure operation over the relational type gives $|\text{bool}_r[B]|_i = \text{bool}[B]$. Then, accordingly, our asynchronous rule r-t-if also reflects this modification, we call it r-t-if-br to distinguish the previous one.

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t' : \text{bool}[B] \qquad \Sigma;\Delta;\Phi_a \wedge B;\Gamma \vdash t \ominus t'_1 \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a \wedge \neg B;\Gamma \vdash t \ominus t'_2 \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus \text{if } t' \text{ then } t'_1 \text{else } t'_2 \lesssim D_2 - L_1 - c_{case} : \tau}\ \text{r-t-if-br}$$

In this sense, we can compare the red code if b… with the green one using this modified rule, since now b of type bool$[B]$ will hold due to the constraint environment $\Phi_a \wedge B$, we only need to compare the then branch of the red code with the green one. The comparison between the else branch and the green part is trivial because the constraint envi-

ronment will not be satisfied. In that case, we can provide any cost we want.

Now we are able to provide the relational type of the two programs with a precise cost upper bound as follows:

$$\begin{aligned} & & & (U(A) \to \text{unit}) \to \forall B :: \{\text{true}, \text{false}\}.\text{bool}_r[B] \\ \vdash \lambda f.\lambda b.\mathsf{loop} \ominus \lambda f.\lambda b.\mathsf{loopOp} \lesssim 0 : & & \xrightarrow{\text{diff}(-1)} & \forall \gamma_1, N, M.\ \mathsf{Array}_{\gamma_1}[N]\ U(A) \to \text{int}[M] \\ & & & \to \text{int}[N] \to \{\gamma_1 \to \emptyset\} \overset{\text{diff}(N-M)}{\exists\_}.\text{unit}\ \{\gamma_1 \to \emptyset\} \end{aligned}$$

Notice, the negative cost -1 in the type comes from checking b at the beginning in the optimized version. And the relative cost embedded in the monadic type reflects the difference between the red(the boxed code in loop) and green(the boxed code in loop$'$) parts for every iteration.

CHAPTER 7

Related Work of ARel

This chapter discusses the related work that inspires the design of ARel. It consists of two main parts: Static cost analysis that ARel is working on, and Relational Verification which reasons about relational properties such as relative cost in ARel.

7.1 STATIC COST ANALYSIS

There is a lot of prior work that has studied static cost analysis, however, all these systems consider the unary cost. On the other side, ARel provides static cost analysis in the relational setting.

Type and Effect Systems Reistad & Gifford (1994) present a type and effect system for cost analysis, like ARel, and the cost can depend on the size of the input. Danielsson (2008) uses a cost-annotated monad similar in spirit to the one we use in ARel. Dal Lago & Gaboardi (2011) present a linear dependent type system using index terms to analyze time complexity. Wang et al. (2017) develop a type system for cost analysis with time complexity annotations in types.

Recurrence Extraction Recurrence extraction is a classic approach to reason about the cost. The recurrences relations, which express the run time cost in terms of sizes of inputs, can be extracted under either call-by-value, call-by-name or call-by-push-value evaluation strategies (Danner et al., 2015; Kavvos et al., 2019; Cutler et al., 2020).

Amortized Resource Analysis Charguéraud & Pottier (2015) present an amortized resource analysis based on an extension of separation logic with time credits. Our use of

triples and separation-based management of arrays references is similar to theirs. However, their technique is based on separation logic, while ours is based on a type-and-effect system. Moreover, they consider only unary reasoning while we are interested primarily in relational reasoning.

Lichtman & Hoffmann (2017) present an amortized resource analysis for arrays and references using. Hoffmann et al. (2012) present an automated amortized cost analysis for programs with complex data structures such as matrices. Their technique represents the available "potential" before and after a computation, similar to our triples. Again, they focus only on unary cost analysis and consider mostly first-order programs and linear potentials.

7.2 RELATIONAL VERIFICATION

Outside of cost analysis, a lot of work has considered relational verification techniques for other applications.

Probabilistic Relational Verification Lahiri et al. (2010) present a differential static analysis to find code defects looking at two pieces of code relationally. Probabilistic relational verification has seen many applications in cryptography (Barthe et al., 2014) and differential privacy (Gaboardi et al., 2013; Barthe et al., 2015a). Barthe et al. (2015a) propose HOARe^2 which uses relational refinements to reason about differential privacy and other probabilistic relational properties. ARel also relies on relational refinements to reason about pairs of arrays via assertions P, Q in our monadic types. The difference is that we choose lightweight assertions and use them to reason only about the difference of arrays. Conversely, HOARe^2 uses arbitrary relational refinement types, which are more expressive, but which also require many more annotations.

Amortized Relational Resource Reasoning Ngo et al. (2017) combine information flow and amortized resource analysis to guarantee constant-resource implementations. Their type system allows relational reasoning about resources through precise unary analysis. Their focus is on first-order functional programs and on the constant time guarantee, while we want to support functional-imperative programs and more general relative costs.

Relational Refinement Types The indexed types used by Gaboardi et al. (2013) are similar in spirit to ours. Their indices cover the size of the data types as we do, but they also track the sensitivity, which is useful for differential privacy. Our indices instead focus more on the effects and differences of arrays. Zhang et al. (2015) introduce dependent labels into the type of SecVerilog, an extension of Verilog with information flow control. The use of a lightweight invariant on variables and security levels in SecVerilog is similar to our use of β, which is also an invariant on static location variables. Unno et al. (2017) present an automated approach to verification based on induction for Horn clauses, which can also be used for relational verification. Benton et al. (2014, 2016) introduce abstract effects to reason about abstract locations. This is conceptually similar to the way our preconditions and postconditions allow us to reason about different independent locations.

Relational Cost Analysis Our work is directly inspired by RelCost (Çiçek et al., 2017a) and DuCostIt (Çiçek et al., 2016). These are refinement type and effect systems for pure functional languages *without mutable state*. RelCost supports relational cost analysis of pure programs. In contrast, ARel supports imperative arrays. The difference is substantial: Besides significant changes to the model, the type system has to be enriched with Hoare-like triples, whose design is a key contribution of our work. RelCost has an

implementation via an SMT back-end (Çiçek et al., 2019); we extend this approach with imperative features and support for sets of indices (our βs). Radicek et al. (2018a) add a cost monad to a relational refinement type system, where refinements reason about relational cost, for programs without the state. This system is expressive: it supports a combination of cost analysis with value-sensitivity and full functional specifications (RelCost can also be embedded in it). However, it requires a framework for full functional verification. Our approach is complementary in that we use lighter refinements that are easier to implement but do not support full functional verification.

II BIAREL

In the next few chapters, we will have a look at the implementation of ARel, a bidirectional type checker named BiARel. Since ARel has some interesting points compared to the standard type and effect system, the implementation starts from a simple system, and then step by step to reach BiARel.

CHAPTER 8

Towards Bidirectional Type Checking

We have seen the type system of ARel and how it is capable of type checking many examples and verifying the cost bounds along with the type both in Chapter 6 and in the appendix A. Naturally, the implementation of ARel becomes the next target.

As we have seen, relational type systems such as ARel, are appealing for relational properties because they deliver simpler and more precise verification than what could be derived from typing the two programs separately. However, relational type systems do not yet achieve the practical appeal of their non-relational counterpart, in part because of the lack of a general foundation for implementing them. Additionally, how to handle the relational effects(relative cost upper bound) and refinements(the $\Box$ modality) bring us more challenges in ARel's implementation.

In this chapter, we will formally introduce some concepts such as relational type system, relational effect, and refinement, in particular, bidirectional type checking. The declarative type system of ARel is not suitable to be implemented as a type checker due to its non-deterministic nature. We then algorithmize ARel's bidirectional type system, which eliminates the non-determinism.

8.1 BACKGROUND OF IMPLEMENTATION

ARel is a relational type and effect system, with relational refinements. We first look at these concepts, which are important for the implementation, one by one.

Relational type systems (Abadi et al., 1993; Pottier & Simonet, 2003; Gaboardi et al., 2013; Barthe et al., 2014, 2015b; Çiçek et al., 2016; Çiçek et al., 2017b; Aguirre et al., 2017, 2018) aim to repeat the success of type systems, but for so-called relational properties,

which consider pairs of execution traces. Typical examples of relational properties include noninterference in information flow systems, continuity, and robustness analysis of programs, differential privacy, and relational cost analysis. The key difference of relational type systems is that they consider two expressions simultaneously, and maximally exploit structural similarities between them to achieve simpler and more precise verification than would be possible with unary analysis of the individual expressions. Similarities are exploited through two main ingredients: relational refinement types and relational effects.

Relational refinements types (Pottier & Simonet, 2003; Gaboardi et al., 2013; Barthe et al., 2015b; Çiçek et al., 2016; Çiçek et al., 2017b) relate two executions of two expressions and are akin to standard refinement types (Xi & Pfenning, 1999). However, their interpretation is a relation between the values in the two executions. For example, in information flow control, a relational refinement is used to describe equivalence between the values that are observable at a specific security level. In ARel, remember we know the array size N and its static name γ from the relational array type $\mathsf{Array}_{\gamma}[N]\ \tau$.

Relational effects (Pottier & Simonet, 2003; Gaboardi et al., 2013; Barthe et al., 2015b; Çiçek et al., 2016; Çiçek et al., 2017b) are often of a quantitative nature and measure some quantitative difference between two executions of the two expressions. These relational effects are similar in spirit to their standard unary counterpart (Lucassen & Gifford, 1988; Nielson & Nielson, 1999; Petricek et al., 2013; Brunel et al., 2014) but their interpretation is a relation between the effects of the two executions. For example, in differential privacy, a relational effect is used to measure the level of indistinguishability between the observable outputs on two inputs differing in one data element. Also in ARel, the relational arrow type $\tau \xrightarrow{\mathrm{diff}(D)} \tau$ carries the relational effect D.

Bidirectional type checking (Pierce & Turner, 2000) is a very successful method of implementing type systems through a combination of type inference and type checking (Bracha et al., 1998; Odersky et al., 2001; Peyton Jones et al., 2007; Bierman et al., 2007; Abel et al., 2017). The appeal of bidirectional type checking lies in its ability to minimize typing annotations—in most cases, type annotations are needed only on recursive functions, or on reducible expressions—while supporting disciplines that are too expressive to fall under the purview of type inference. Furthermore, bidirectional type systems offer a formal framework based on rules that resemble standard typing rules. This simplifies proofs of soundness and completeness of the algorithmic implementation relative to the declarative type system.

While so many works cited above come with implemented type checkers, there is, so far, no common understanding of the challenges and solutions for implementing relational type systems. Hence, the broad goal of our work is to investigate issues in *implementing* a type checker for relational type systems with relational refinements and relational effects. Bidirectional type checking is a natural starting point for the reasons mentioned above, and because it has been used for implementing refinement type systems (Xi & Pfenning, 1999; Davies & Pfenning, 2000; Dunfield & Krishnaswami, 2019) and subtyping (Pierce & Turner, 2000), which are important common features in most of the type systems we are inspired from. However, bidirectional type checking has not been extensively applied to effect systems, although some examples exist (Toro & Tanter, 2015), and it has not been applied to relational type systems at all.

8.2 STUDY ON BIDIRECTIONAL TYPE CHECKING

We present the overview of our study of *bidirectional* type checking for *relational type and effect systems* step by step. The details of the study can be found in Chapter 9.

The outline of our step-by-step bidirectional type checking study on relational refinement type and effect systems is shown in Figure 8.1. We start with the study of a basic relational type system, named relSTLC, that includes judgment of the form $\Gamma \vdash t_1 \backsim t_2 : \tau$, to only relate two expressions with the same top-level structure, with types to represent related and non-related boolean values, no relational refinements, and no relational effects. This can be seen as the relational analog of the simply-typed lambda calculus over a base type with subtyping, so the relational type τ in relSTLC is quite standard. For this system, bidirectional type checking works as expected and it delivers a sound and complete algorithm implementing the declarative system.

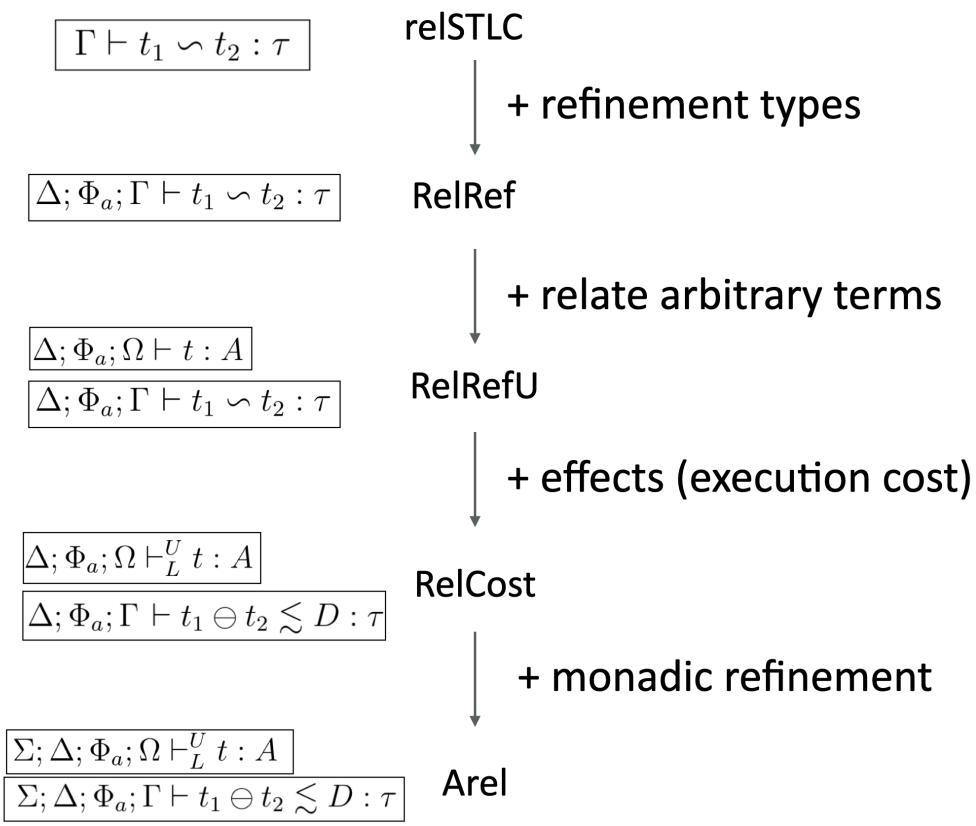


Figure 8.1: Overview of step-by-step study on bidirectional typechecking.

Next, we extend relSTLC in two steps inspired by the features of previously pro-

posed relational type systems. Our first step, named RelRef, adds *relational refinement types* over lists (as an example of an inductive data type), and a comonadic type (the $\Box$ type we have seen in mapi example) that represents *syntactic equality* of values. The judgment of RelRef in Figure 8.1 of the form of $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau$ has two more environments compared to relSTLC: the index variable environment Δ which stores the index variables used in the refinement types, and the constraint environment Φ_a which supports conditional reasoning in RelRef.

Our second step, named RelRefU, adds to RelRef the possibility to *relate arbitrary programs* of possibly dissimilar syntactic structure, by switching to a complementary unary type system. Hence, as we seen in Figure 8.1, RelRefU has both the unary and relational judgments. The connection between relational types and unary types is the $U(A_1, A_2)$ type, as we have seen before. In comparison to RelRef, the addition of the unary type system, as well as the appearance of $U(A_1, A_2)$ type, are the most important part for RelRefU. At the same time, these two extensions (RelRef and RelRefU) add intrinsic *non-determinism* to the type system to allow a programmer flexibility in writing programs. The source of non-determinism in both these systems is non-syntax-directed typing and subtyping rules. RelRef has such rules for relational refinement types and for subtyping, while RelRefU has such a rule for switching to unary typing and more such rules for subtyping.

To overcome the challenges introduced by non-determinism, we introduce a two-step methodology. We first show that every well-typed program can be translated to a well-typed program in a core language with term constructors that resolve the non-determinism. This translation is type derivation-directed; it introduces annotations to resolve the non-determinism in applying the (non-syntax-directed) typing rules and eliminates relational subtyping by replacing it with explicit coercions defined within the

core language. Next, we develop a bidirectional type system and prove it sound and complete with respect to the core system. It follows that every typeable program can be annotated to remove non-determinism, and then bidirectionally type-checked. Although this indirection via a core language does not directly lead to an implementation strategy, it makes a strong theoretical point, namely that the bidirectional type checking is complete modulo non-determinism. We show that this methodology is applicable to both RelRef and RelRefU.

Naturally, our next step is to add *relational effects* to RelRefU, specifically, it comes to RelCost (Çiçek et al., 2017a), a prior work ARel is inspired from. This type system extends RelRefU with a relational effect to enable relational cost analysis. Since RelCost extends RelRefU, it inherits the latter's many sources of non-determinism. It can be shown that the two-step method also extends to relational effects. The detail is described in the thesis of Ezgi Cicek (Çiçek, 2018), in which the implementation of RelCost uses this method. We can see that the judgments of RelCost shown in Figure 8.1 now also contains the cost, L, U in the unary one and D for the relation one.

Finally, we will reach our target, developing a bidirectional type checking system for ARel. We then need to handle the monadic refinements that are used to reason about mutable arrays and also consider the non-determinism introduced along with the imperative feature. We show this two-step method works well for ARel.

8.3 CONTRIBUTIONS

To show the effectiveness of bidirectional type checking for relational type systems, we have implemented a prototype for ARel, called BiARel. Our implementation handles the two steps of our method simultaneously. To implement the first step, the translation of the source language to the core language, we rely on example-guided heuristics rather

than programmer-specified annotations to eliminate non-determinism. This reduces the programmer's annotation burden by compromising completeness to some degree. We explain these heuristics used in implementing ARel, and also include some heuristics used in implementing RelCost, called BiRelCost from Ezgi Cicek's thesis (Çiçek, 2018), which can be applied in the implementation of ARel as well. For the second step, we implement the bidirectional typing rules for ARel. Both type checking mode and type inference mode generate constraints that capture arithmetic relationships between refinements (e.g., list sizes, array lengths) of various subterms, relational refinements, and relationships between unary and relational costs. Our constraints contain existentially quantified variables over integers and reals. Therefore, we use the algorithm to eliminate existential variables in BiRelCost by finding substitutions for them and using SMT solvers to discharge the substituted constraints, which works well for constraintS generated in BiARel.

Summing up, our contributions are:

- We present several bidirectional relational type systems that combine relational and non-relational typing, refinements, and unary and relational effects. It provides a clear study on how to build the implementation of ARel step by step.
- We present the implementations of ARel, using heuristics to get rid of the inherent non-determinism. We use the implementation to type-check several examples, including most of the interesting examples in this dissertation.

CHAPTER 9

Bidirectional Type Checking for Relational Refinement Type and Effect Systems

In this section, we take a step in this direction by developing *bidirectional relational* type checking for systems with relational refinements and effects. Our approach achieves the benefits of bidirectional type checking, in a relational setting. In particular, it significantly reduces the need for typing annotations through the combination of type checking and type inference. In order to highlight the foundational nature of our approach, we develop bidirectional versions of several relational type systems which incrementally combine many different components needed for expressive relational analysis. We only show selected rules in this section which we think is of interest. For more details, please check the appendix B.

9.1 RELATIONAL STLC (RELSTLC)

As an introduction to how relational reasoning works, we consider relSTLC, a rehash of the simply-typed lambda calculus (STLC) with relational reasoning. The full detail of relSTLC can be found in Appendix B, Section B.1. relSTLC has the following type and expression grammar:

$$\begin{array}{ll} \text{Types} & \tau ::= \text{bool}_r \mid \text{bool}_u \mid \tau_1 \to \tau_2 \\ \text{Term} & t ::= x \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \lambda x.t \mid t_1\ t_2 \end{array}$$

A type τ is interpreted as a set of pairs of values. The values are true, false and the lambda construct. For instance, the primitive type bool_r ascribes pairs of *identical* booleans (the diagonal relation on booleans) whereas the type bool_u ascribes pairs of arbitrary booleans (the complete relation on booleans). In particular, $\text{bool}_r \sqsubseteq \text{bool}_u$. The function

type $\tau_1 \to \tau_2$ relates pairs of functions that, given a pair of related arguments of type τ_1, return a pair of related values of type τ_2. Even though relSTLC is quite primitive, it forms the basis of our development and we find it instructive to discuss challenges in its algorithmization.

Declarative typing The typing judgment $\Gamma \vdash t_1 \backsim t_2 : \tau$ ascribes the expressions t_1 and t_2 the relational type τ under the environment Γ. The typing rules and subtyping rules are standard. A selection is shown in Figure 9.1. These relational typing rules can be found in Figure B.4, the subtyping rules in Figure B.5 in Appendix B. Note how the rule **r-bool** relates two identical booleans at type bool_r, while **r-u-bool** relates two arbitrary booleans at type bool_u. This difference manifests in the rule **r-if**: If the branch condition of an if-then-else has type bool_r, then we only need to type the two "then" branches and the two "else" branches separately, but do not need to type a "then" and an "else" branch together. Finally, note that the calculus has (standard) subtyping induced by the base relation $\text{bool}_r \sqsubseteq \text{bool}_u$.

Algorithmic (bidirectional) typing The type system presented above is declarative, i.e. it doesn't prescribe an algorithm for building a typing derivation. In fact, two aspects of relSTLC make it difficult to straightforwardly algorithmize. First, the obvious approach of *inferring* the type of a term bottom-up by starting at the leaves (variables) does not work because relSTLC does not have type annotations on variable-bindings in the **r-lam**. The other obvious approach of *checking* a term against a given type top-down also runs into a problem, this time in the **r-app** rule, where the argument type τ_1 must be guessed. Second, the **trans** subtyping rule is also not syntax-directed (the type τ_2 must be guessed). Hence, the typing and subtyping rules of relSTLC cannot be directly interpreted as a type checking algorithm.

$\boxed{\Gamma \vdash t_1 \backsim t_2 : \tau}$

$$\frac{\mathtt{b} \in \{\text{true}, \text{false}\}}{\Gamma \vdash \mathtt{b} \backsim \mathtt{b} : \text{bool}_r}\ \textbf{r-bool} \qquad \frac{\mathtt{b}_1, \mathtt{b}_2 \in \{\text{true}, \text{false}\}}{\Gamma \vdash \mathtt{b}_1 \backsim \mathtt{b}_2 : \text{bool}_u}\ \textbf{r-u-bool}$$

$$\frac{\Gamma \vdash t \backsim t' : \text{bool}_r \qquad \Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Gamma \vdash t_2 \backsim t_2' : \tau}{\Gamma \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 \backsim \text{if } t' \text{ then } t_1' \text{ else } t_2' : \tau}\ \textbf{r-if} \qquad \frac{\Gamma, x : \tau_1 \vdash t_1 \backsim t_2 : \tau_2}{\Gamma \vdash \lambda x.t_1 \backsim \lambda x.t_2 : \tau_1 \to \tau_2}\ \textbf{r-lam}$$

$$\frac{\Gamma \vdash t_1 \backsim t_1' : \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 \backsim t_2' : \tau_1}{\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' : \tau_2}\ \textbf{r-app} \qquad \frac{\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \models \tau \sqsubseteq \tau'}{\Gamma \vdash t_1 \backsim t_2 : \tau'}\ \textbf{r-}\sqsubseteq$$

$\boxed{\models \tau \sqsubseteq \tau'}$

$$\frac{}{\models \text{bool}_r \sqsubseteq \text{bool}_u}\ \textbf{bool} \qquad \frac{\models \tau_1' \sqsubseteq \tau_1 \qquad \models \tau_2 \sqsubseteq \tau_2'}{\models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'} \to \qquad \frac{}{\models \tau \sqsubseteq \tau}\ \textbf{refl}$$

$$\frac{\models \tau_1 \sqsubseteq \tau_2 \qquad \models \tau_2 \sqsubseteq \tau_3}{\models \tau_1 \sqsubseteq \tau_3}\ \textbf{trans}$$

Figure 9.1: Typing and subtyping rules of relSTLC.

A well-established way of making typing rules syntax-directed (hence obtaining a type checking algorithm) is to make the rules *bidirectional* (Pierce & Turner, 2000; Xi & Pfenning, 1999; Xi, 1998). In comparison to fully type-annotating all bound variables, which could be tedious for a programmer, the main idea behind bidirectional type checking is to only annotate programs at the top-level and at explicit β-redexes (which are usually rare) and infer all other types by combining top-down and bottom-up analysis.

In the case of relSTLC, bidirectional type checking splits the usual typing judgment $\Gamma \vdash t_1 \backsim t_2 : \tau$ into two judgments: (1) the checking judgment $\Gamma \vdash t_1 \backsim t_2 \downarrow \tau$, where the type τ is an input (the type is checked), and (2) the inference judgment $\Gamma \vdash t_1 \backsim t_2 \uparrow \tau$, where the type τ is an output (the type is inferred). As a convention, we write all outputs in red and all inputs in black. Figure 9.2 shows selected algorithmic typing rules. The full typing rules can be found in Figure B.6 in Appendix B. We explain below the basic principles behind the bidirectional typing rules. These principles are completely standard for *unary* type systems (Xi & Pfenning, 1999; Xi, 1998); our observation thus far is simply that they apply as-is to relational type systems as well and, for relSTLC, they suffice to ensure completeness of bidirectional type checking (this will cease to be the case for later type systems).

1. Types of introduction forms are checked (e.g., rule **alg-r-lam**). Types of elimination forms are, in general, inferred (e.g., rule **alg-r-app**). However, types of case-like elimination forms (e.g., rule **alg-r-if**) are checked. In all cases, the type of an expression in an elimination position is inferred (e.g., the branch condition $t \backsim t'$ in rule **alg-r-if**). Types of variables and constants can always be inferred. However, here, we treat constants as constructors (introduction forms), so their types are checked (this is an arbitrary design choice).

2. In checking mode, the rule **alg-r-↑↓** allows switching to inference mode. The requirement is that the inferred type must be a subtype of the checked type.

3. In inference mode, it is permissible to switch to the checking mode when an expression's type has been explicitly annotated by the programmer (rule **alg-r-anno-↑**). It can be shown that, for completeness, it suffices to annotate only at explicit β-redexes (although there is no prohibition on annotating at other places).

Subtyping also has an algorithmic counterpart, $\models \tau_1 \leq \tau_2$, shown in Figure 9.2. The full typing rules can be found in Figure B.7 in Appendix B. We introduce two additional rules for reflexivity of base types (rules **alg-bl-u** and **alg-bl-r**). Importantly, it can be proved that reflexivity and transitivity of subtyping are *admissible*, so, in particular, there is no need for an explicit rule of transitivity, which, as mentioned, is difficult to use in an algorithm.

The bidirectional type system's rules, when read bottom-up, can be interpreted as a syntax-directed algorithm for type checking. This algorithm is *sound* relative to the declarative type system in the following sense:

If either $\Gamma \vdash t_1 \backsim t_2 \uparrow \tau$ or $\Gamma \vdash t_1 \backsim t_2 \downarrow \tau$, then $\Gamma \vdash |t_1| \backsim |t_2| : \tau$, where $|t|$ is obtained by erasing type annotations from t.

The bidirectional type system is also *complete* relative to the declarative type system:

If $\Gamma \vdash t_1 \backsim t_2 : \tau$, then there are type-annotated variants, t_1', t_2' of t_1, t_2 such that $\Gamma \vdash t_1' \backsim t_2' \uparrow \tau$. These annotations can be limited to the top-level and any explicit β-redexes.

The proofs of these statements are in Appendix B, Section B.1.2 .

9.2 RELREF

Next, we extend bidirectional type checking to *relational refinements*. Relational refinements (Pottier & Simonet, 2003; Gaboardi et al., 2013; Barthe et al., 2015b; Çiçek et al.,

$\boxed{\Gamma \vdash t_1 \backsim t_2 \uparrow \tau, \Gamma \vdash t_1 \backsim t_2 \downarrow \tau}$

$$\frac{\Gamma \vdash t \backsim t' \uparrow \mathrm{bool}_r \qquad \Gamma \vdash t_1 \backsim t_1' \downarrow \tau \qquad \Gamma \vdash t_2 \backsim t_2' \downarrow \tau}{\Gamma \vdash \mathrm{if}\ t\ \mathrm{then}\ t_1\ \mathrm{else}\ t_2 \backsim \mathrm{if}\ t'\ \mathrm{then}\ t_1'\ \mathrm{else}\ t_2' \downarrow \tau}\ \textbf{alg-r-if}$$

$$\frac{\Gamma, x : \tau_1 \vdash t_1 \backsim t_2 \downarrow \tau_2}{\Gamma \vdash \lambda x.t_1 \backsim \lambda x.t_2 \downarrow \tau_1 \to \tau_2}\ \textbf{alg-r-lam}$$

$$\frac{\Gamma \vdash t_1 \backsim t_1' \uparrow \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 \backsim t_2' \downarrow \tau_1}{\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' \uparrow \tau_2}\ \textbf{alg-r-app} \qquad \frac{\Gamma \vdash t \backsim t' \uparrow \tau' \qquad \models \tau' \leq \tau}{\Gamma \vdash t \backsim t' \downarrow \tau}\ \textbf{alg-}\uparrow\downarrow$$

$$\frac{\Gamma \vdash t \backsim t' \downarrow \tau}{\Gamma \vdash (t : \tau) \backsim (t' : \tau) \uparrow \tau}\ \textbf{alg-r-anno-}\uparrow$$

$\boxed{\models \tau \leq \tau'}$

$$\frac{}{\models \mathrm{bool}_r \leq \mathrm{bool}_r}\ \textbf{alg-bl-r} \qquad \frac{}{\models \mathrm{bool}_u \leq \mathrm{bool}_u}\ \textbf{alg-bl-u} \qquad \frac{}{\models \mathrm{bool}_r \leq \mathrm{bool}_u}\ \textbf{alg-bl}$$

$$\frac{\models \tau_1' \leq \tau_1 \qquad \models \tau_2 \leq \tau_2'}{\models \tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'}\ \textbf{alg-}\to$$

Figure 9.2: Algorithmic typing and subtyping rules of relSTLC.

2016; Çiçek et al., 2017b) express fine-grained relations between pairs of expressions. They have been used for many different purposes ranging from information flow control to differential privacy. We consider here a simple setting, which still suffices to bring out key challenges in applying bidirectional type checking to relational refinements.

We extend relSTLC with primitive lists and a relational refinement type $\mathrm{list}[n]^{\alpha}\,\tau$, which ascribes a pair of lists, both of length n, that differ pointwise in at most α positions (n and α are natural numbers). $\mathrm{list}[n]^{\alpha}\,\tau$ *refines* the standard list type with n and α and the refinement is *relational* since α expresses a constraint on the two lists together. To construct lists of this type when $\alpha \neq n$, we also need a way to express that at least $(n-\alpha)$ elements are pointwise equal. To this end, we introduce the comonadic type $\Box\,\tau$, which ascribes pairs of expressions of type τ that are equal (i.e. the diagonal relation on τ). $\Box\,\tau$ generalizes the refinement r in bool_r to arbitrary types. Type-level terms like n and α are called index terms or indices, generically denoted I. The type system also supports quantification over such terms. To write recursive programs on lists, we also add a fixpoint operator, which poses no additional difficulty for bidirectional type checking. The resulting system, called RelRef, has the following syntax.

$$
\begin{array}{lll}
\text{Types } \tau ::= & \ldots \mid \mathrm{list}[n]^{\alpha}\,\tau \mid \forall i{::}S.\,\tau \mid \exists i{::}S.\,\tau \mid \Box\,\tau \mid C \,\&\, \tau \mid C \supset \tau \\
\text{Term } t ::= & \ldots \mid \mathrm{fix}\ f(x).t \mid \mathrm{nil} \mid \mathrm{cons}(t_1, t_2) \mid (\mathrm{case}\ t\ \mathrm{of\ nil} \rightarrow t_1 \mid h :: tl \rightarrow t_2) \\
& \mid \Lambda.t \mid t[] \mid \mathrm{let}\ x = t_1\ \mathrm{in}\ t_2 \mid \mathrm{clet}\ t_1\ \mathrm{as}\ x\ \mathrm{in}\ t_2 \mid \mathrm{celim}\ t \\
& \mid \mathrm{pack}\ t \mid \mathrm{unpack}\ t_1\ \mathrm{as}\ x\ \mathrm{in}\ t_2 \\
\text{Indices } I, n, \alpha ::= & i \mid 0 \mid I+1 \mid I_1 + I_2 \mid I_1 - I_2 \mid \frac{I_1}{I_2} \mid I_1 \cdot I_2 \mid \lceil I \rceil \mid \lfloor I \rfloor \mid \mathtt{min}(I_1, I_2) \mid \mathtt{max}(I_1, I_2)
\end{array}
$$

Types can quantify over index variables, i, as in $\forall i{::}S.\,\tau$ and $\exists i{::}S.\,\tau$. The sort S can only be natural number now. The constructs pack t and unpack t_1 as x in t_2 are the introduction and elimination forms for existentially quantified types. The constructs $\Lambda.t$

and $t[]$ are the introduction and elimination forms for universally quantified types. To represent arithmetic relations over index variables, constraints denoted C, sets of predicates over index terms, appear in types as in $C \,\&\, \tau$ and $C \supset \tau$. The type $C \,\&\, \tau$ means the type τ and that C holds, while $C \supset \tau$ means that, if the constraint C holds, then the type is τ. The construct clet t_1 as x in t_2 is the elimination form for the constrained type $C \,\&\, \tau$. By design, index terms do not appear in RelRef expressions. The full syntax of RelRef can be found in Appendix B, Section B.2.1.

Example (map) As an example, we can write the standard list map function, (the standard version of mapi) and give it the following very informative relational type in RelRef (for any τ_1, τ_2).

$$
\begin{array}{l}
\text{fix map}(f).\Lambda.\Lambda.\lambda l. \\
\quad \text{case } l \text{ of} \\
\quad\quad \text{nil} \rightarrow \text{nil} \\
\quad\quad \mid h :: tl \rightarrow \text{cons}(f\ h, \texttt{map}\ f\,[]\,[]\ tl)
\end{array}
$$

$$\texttt{map} : (\Box(\tau_1 \rightarrow \tau_2)) \rightarrow \forall n, \alpha.\text{list}[n]^{\alpha}\,\tau_1 \rightarrow \text{list}[n]^{\alpha}\,\tau_2$$

The type means that two runs of map with *equal* mapping functions and two lists that differ in at most α positions result in two lists with the same property. Notice how α is universally quantified in the type, and how $\Box$ represents that the mapping function be equal in the two runs. Notice the use of expression such $\Lambda.t$ and $t[]$, which is used for the introduction and elimination of index variables n, α in the type.

Declarative typing RelRef's typing judgment has the form $\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_2 : \tau$ and means that t_1 and t_2 have the relational type τ if the constraints Φ_a hold. Δ is a (universally quantified) context of index variables and Γ, as usual, is the typing context for

program variables. Figure 9.3 shows selected typing rules that use refinements and constraints in interesting ways, and make bidirectional type checking difficult. The full details can be found in Appendix B, Figure B.12, B.13. To start, as a result of the relational refinement α in the list type, there are *two* rules for typing the list cons constructor. Rule **rr-cons1** applies when the head elements of the constructed lists may differ. Note how the relational refinement α changes to $\alpha + 1$ from the premise to the conclusion. Rule **rr-cons2** applies when the head elements are equal, witnessed by the comonadic type $\Box\tau$. α does not change in this rule. Dually, the cons branch of list case analysis (rule **rr-caseL**) is typed *twice* with different index constraints—once for each of these two possible ways of constructing the cons-ed list. A consequence of this double typing of the same branch with different constraints is that expressions cannot contain index terms (else such typing may be impossible). The rule **rr-split** case-splits on an arbitrary constraint C in the context. This is useful for typing recursive functions (Çiçek et al., 2016; Çiçek et al., 2017b). Finally, the rule for introducing the type $\Box\tau$, **rr-nochange**, is interesting. It says that if t relates to itself at type τ and all variables in Γ morally have $\Box$-ed types (checked via subtyping), then t also relates to itself at type $\Box\tau$.

Declarative subtyping RelRef subtyping $\tau \sqsubseteq \tau'$ is complex. Some of the rules are shown in Figure 9.3 (The full subtyping rules are in Appendix B, Figure B.14). First, subtyping is constraint-dependent, because it must, for instance, be able to show that $\mathrm{list}[n]^\alpha\,\tau \sqsubseteq \mathrm{list}[m]^\alpha\,\tau$ when $n = m$. Second, in RelRef, $\Box$'s comonadic properties manifest themselves via subtyping. This results in interactions between $\Box$ and other connectives as, for instance, in the rules $\to\Box_{\mathbf{diff}}$, **l2** and **l**$\Box$.

We explain some of the subtyping rules. The rule **l1** allows α, the upper-bound on the number of elements that differ in the two lists, to be weakened covariantly. The rule **l2** allows two related lists with zero differences to be retyped as two related lists whose

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \mathrm{list}[n]^{\alpha}\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}(t_1,t_2) \backsim \mathrm{cons}(t_1',t_2') : \mathrm{list}[n+1]^{\alpha+1}\,\tau}\ \textbf{rr-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \mathrm{list}[n]^{\alpha}\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}(t_1,t_2) \backsim \mathrm{cons}(t_1',t_2') : \mathrm{list}[n+1]^{\alpha}\,\tau}\ \textbf{rr-cons2}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \mathrm{list}[n]^{\alpha}\,\tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau, tl:\mathrm{list}[i]^{\alpha}\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h:\tau, tl:\mathrm{list}[i]^{\beta}\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \ \mathrm{case}\ t\ \mathrm{of\ nil}\ \to t_1 \mid h :: tl \to t_2 \backsim\ \mathrm{case}\ t'\ \mathrm{of\ nil}\ \to t_1' \mid h :: tl \to t_2' : \tau'}\ \textbf{rr-caseL}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \Delta \vdash C\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau}\ \textbf{rr-split}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t : \tau \qquad \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \backsim t : \Box\tau}\ \textbf{rr-nochange}$$

$$\frac{}{\Delta;\Phi_a \models \Box(\tau_1 \to \tau_2) \sqsubseteq \Box\tau_1 \to \Box\tau_2} \to \Box\textbf{diff}$$

$$\frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \le \alpha' \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \sqsubseteq \mathrm{list}[n']^{\alpha'}\,\tau'}\ \textbf{l1}$$

$$\frac{\Delta;\Phi_a \models \alpha \doteq 0}{\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \sqsubseteq \mathrm{list}[n]^{\alpha}\,\Box\tau}\ \textbf{l2} \qquad \frac{}{\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\Box\tau \sqsubseteq \Box\,(\mathrm{list}[n]^{\alpha}\,\tau)}\ \textbf{l}\Box$$

$$\frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \tau}\ \textbf{T}$$

Figure 9.3: Typing and subtyping rules of RelRef.

elements are in the diagonal relation. The rule **l**$\Box$ allows two related lists whose elements are equal to be retyped as two equal lists, represented by the outer $\Box$. The rule **T** coerces $\Box\tau$ to τ by forgetting that the two related elements are, in fact, equal.

Towards algorithmization An algorithm for type checking RelRef faces two difficulties beyond those seen in relSTLC. Both difficulties arise due to RelRef's relational refinements. First, there is additional non-syntax-directedness in the rules: Rules **rr-cons1** and **rr-cons2** apply to expressions of the *same* shape (the rules differ only in their treatment of index terms), and rules **rr-split** and **rr-nochange** are not syntax-directed (their use overlaps with other rules). Second, owing to the interaction between $\Box$ and other type constructs, it is infeasible to re-define subtyping in a way that makes transitivity admissible. As a result of these two problems, bidirectional typing alone does not yield an *algorithm* for type checking.

An obvious way to address the first of these problems is to force additional annotations in expressions to remove the non-syntax-directness. However, this will not address the problem with subtyping. Importantly, it also will not allow us to theoretically connect the algorithmic type system to the declarative type system (with its non-syntax-directedness).

Consequently, we follow a slightly different approach. First, we introduce a simpler core calculus RelRef Core, which annotates expressions to resolve the lack of syntax-directedness in typing rules. Additionally, RelRef Core features only *type equivalence*, not subtyping. We show that every RelRef expression can be *elaborated* to a semantically equivalent expression in RelRef Core by adding enough annotations and expressing subtyping as definable type coercions. Next, we build a bidirectional, algorithmic type system for RelRef Core and prove it relatively sound and complete. Although the elaboration to RelRef Core cannot be directly *implemented* without losing complete-

ness or relying on programmer annotations, this approach does establish a strong *theoretical* point, end-to-end: There is a calculus (RelRef Core) that is as expressive as RelRef, and that is fully amenable to bidirectional type checking.

RelRef Core syntax The core calculus RelRef Core is similar to RelRef but has explicit syntactic markers to indicate which typing rules to apply where, thus resolving the nondeterminism caused by the aforementioned typing rules such as **rr-split** and **rr-cons1/rr-cons2**. The details are in Section B.2.1, Appendix B. The expression syntax of RelRef Core is as follows.

$$\text{Terms } t ::= \quad \ldots \mid \text{split}\,(t_1, t_2) \text{ with } C \mid \text{NC}\ t \mid \Lambda i.t \mid t[I] \mid \text{cons}_{NC}(t_1, t_2) \mid \text{cons}_C(t_1, t_2) \mid \begin{pmatrix} \text{case } t \text{ of nil } \to t_1 \\ \mid h ::_{NC} tl \to t_2 \mid h ::_C tl \to t_3 \end{pmatrix}$$

The list constructor cons is separated into two—cons_C and cons_{NC}—to disambiguate the rules **rr-cons1** and **rr-cons2**. Dually, the list-case construct now has three branches, one each for nil , cons_C and cons_{NC}. "split (t_1, t_2) with C" indicates that rule **rr-split** must be applied and records the constraint C to split on. To write meaningful Cs, expressions now carry index terms. For instance, the elimination form for universally quantified types in RelRef Core is $t[I]$ as opposed to RelRef's $t[]$. The construct NC t indicates an application of the rule **rr-nochange**.

RelRef Core typing rules Selected rules of RelRef Core's typing judgment

$$\Delta; \Phi_a; \Gamma \vdash t \backsim t' :^c \tau$$

are shown in Figure 9.4. The details are in Appendix B, Figure B.16, B.17. Φ_a and Δ are interpreted as in RelRef. RelRef Core's rules are similar to RelRef's, but there are im-

$$\frac{\Delta;\Phi_a;\square\Gamma \vdash t \backsim t :^c \tau}{\Delta;\Phi_a;\square\Gamma,\Gamma' \vdash \mathrm{NC}\ t \backsim \mathrm{NC}\ t :^c \square\tau}\ \textbf{c-nochange}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \backsim t_2 :^c \tau \quad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1' \backsim t_2' :^c \tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{split}\ (t_1,t_1')\ \mathrm{with}\ C \backsim \mathrm{split}\ (t_2,t_2')\ \mathrm{with}\ C :^c \tau}\ \textbf{c-s}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau \quad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \mathrm{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}_C(t_1,t_2) \backsim \mathrm{cons}_C(t_1',t_2') :^c \mathrm{list}[n+1]^{\alpha+1}\,\tau}\ \textbf{c-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \square\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \mathrm{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}_{NC}(t_1,t_2) \backsim \mathrm{cons}_{NC}(t_1',t_2') :^c \mathrm{list}[n+1]^\alpha\,\tau}\ \textbf{c-r-cons2}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau'}\ \textbf{c-r-}\equiv$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1,f:\tau_1\to\tau_2,\Gamma \vdash t_1 \backsim t_2 :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).t_1 \backsim \mathrm{fix}\ f(x).t_2 :^c \tau_1 \to \tau_2}\ \textbf{c-r-fix}$$

Figure 9.4: Selection of typing rules of RelRef Core.

portant differences. First, rules are now *syntax-directed*. Second, there is no relational subtyping. Instead, there is type-equivalence, $\equiv$, which is much simpler than subtyping. It only lifts equality modulo constraints to types (e.g., $\text{list}[1+2]^{\alpha}\,\tau \equiv \text{list}[3]^{\alpha}\,\tau$) and it can be easily implemented algorithmically (modulo constraint solving). We omit its straightforward details.

Simulating RelRef's subtyping A key property of RelRef Core is that it can simulate RelRef's subtyping via explicit coercion functions, as formalized in the following lemma. Such elimination of subtyping is a common technique for simplifying type checking in the unary setting (Breazu-Tannen et al., 1991; Crary, 2000); here, we lift the idea to the relational setting and our comonad.

Lemma 4. *[Existence of coercions for relational subtyping] If* $\Delta;\Phi_a \models \tau \sqsubseteq \tau'$ *in RelRef then there exists* $t \in$ *RelRef Core*
s.t. $\Delta;\Phi_a;\cdot \vdash t \rightharpoonup t :^{c} \tau \to \tau'$.

Proof. By induction on the subtyping derivation. □

The detail proof of this lemma can be found in the appendix B, Section B.2.3.1.

Elaboration Given Lemma 4, we define a straightforward type derivation-directed embedding from RelRef to RelRef Core. Briefly, we use the RelRef typing derivation to insert additional syntactic annotations that RelRef Core needs and use Lemma 4 wherever subtyping appears in the RelRef derivation. The embedding preserves well-typedness (see the appendix B, Theorem 31 for details). This shows that RelRef Core is as expressive as RelRef. (In fact, this is expressiveness in a strong sense, dubbed macro-expressiveness by Felleisen (Felleisen, 1991).)

$$\frac{i,\beta \in \mathrm{fresh}(\mathbb{N}) \qquad \Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau \Rightarrow \Phi_1 \qquad \Delta;i,\beta,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \mathrm{list}[i]^{\beta}\,\tau \Rightarrow \Phi_2 \qquad \Phi_2' = n \doteq (i+1) \wedge \exists \beta :: \mathbb{N}.\Phi_2 \wedge \alpha \doteq \beta + 1}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{cons}_C(t_1,t_2) \ominus \mathrm{cons}_C(t_1',t_2') \downarrow \mathrm{list}[n]^{\alpha}\,\tau \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}$$

alg-r-consC-↓

$$\frac{i \in \mathrm{fresh}(\mathbb{N}) \qquad \Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \square\tau \Rightarrow \Phi_1 \qquad \Delta;i,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \mathrm{list}[i]^{\alpha}\,\tau \Rightarrow \Phi_2 \qquad \Phi_2' = \Phi_2 \wedge n \doteq (i+1)}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{cons}_{NC}(t_1,t_2) \ominus \mathrm{cons}_{NC}(t_1',t_2') \downarrow \mathrm{list}[n]^{\alpha}\,\tau \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}$$

alg-r-consNC-↓

$$\frac{\Delta;\psi_a;C \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\neg C \wedge \Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau \Rightarrow \Phi_2 \qquad \Delta \vdash C\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{split}\,(t_1,t_2)\ \mathrm{with}\ C \ominus \mathrm{split}\,(t_1',t_2')\ \mathrm{with}\ C \downarrow \tau \Rightarrow C \to \Phi_1 \wedge \neg C \to \Phi_2}$$

alg-r-split↓

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \equiv \mathrm{list}[n']^{\alpha'}\,\tau' \Rightarrow \Phi \wedge n \doteq n' \wedge \alpha \doteq \alpha'}\ \textbf{alg-r-list}$$

Figure 9.5: Selection of algorithmic typing and equivalence rules of BiRelRef.

Algorithmic (bidirectional) type checking We now build an algorithmic, bidirectional type system for RelRef Core. We call this system BiRelRef. Selected rules of BiRelRef are shown in Figure 9.5. The details can be found in Appendix B, Figure B.22, B.23, B.24. As before, BiRelRef has two typing judgments: one to check types and the other to infer them. The key addition over relSTLC is that BiRelRef's typing judgments output *constraints* between index terms, which must be verified for typing. The checking judgment has the form $\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \downarrow \tau \Rightarrow \Phi$. Here, the type τ is an input, but the constraints Φ are an output. The intuitive meaning is that *if constraints* Φ *hold* (assuming Φ_a), then $\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_2 :^c \tau$ holds in RelRef Core. (The reader may ignore the index variable context ψ_a; it contains variables universally quantified in Φ.) As an example, consider the rule **alg-r-consC-↓** for relating cons-ed lists at type $\mathrm{list}[n]^\alpha \tau$ when the heads may differ. To do this, the rule relates the tails at type $\mathrm{list}[i]^\beta \tau$ for some i and β. For this to be sound, $n \doteq i+1$ and $\alpha \doteq \beta + 1$ must hold, so these appear as constraints in Φ. Note that Φ quantifies over i and β existentially. Constraint solvers cannot handle such existential quantification easily, a point to which we return in our implementation, which uses heuristics to eliminate these existential variables.

In the inference judgment $\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \uparrow \tau \Rightarrow \Phi$, both Φ and τ are outputs. The meaning of the judgment is similar: if Φ holds (assuming Φ_a), then $\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_2 :^c \tau$.

Like typing, the algorithmic type equivalence judgment $\Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi$ also generates constraints.

Soundness and completeness We prove that BiRelRef is sound and complete w.r.t. RelRef Core's declarative type system. Soundness says that any inference or checking judgment provable in the algorithmic type system can be simulated in RelRef Core if the output constraints Φ are satisfied. Dually, completeness says that any pair of typeable RelRef Core programs can be sufficiently annotated with types to make their

type checkable in BiRelRef, with satisfiable output constraints. As before, we write $|t|$ for the erasure of typing annotations from a BiRelRef expression t to yield a RelRef Core expression.

Theorem 5 (Soundness)**.**

1. *Assume* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi$, $FIV(\Phi_a,\Gamma,\tau) \subseteq dom(\Delta,\psi_a)$, *and* θ_a *is a valid substitution for* ψ_a *s.t.* $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds.*
   Then, $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \backsim |t'| :^c \tau[\theta_a]$.

2. *Assume* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau \Rightarrow \Phi$, $FIV(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$, *and* θ_a *is a valid substitution for* ψ_a *s.t.* $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds.*
   Then, $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \backsim |t'| :^c \tau[\theta_a]$.

Proof. By simultaneous induction on the given BiRelRef derivations. □

The details of soundness proof can be found in Appendix B, Section B.2.3.2.

Theorem 6 (Completeness)**.**

1. *Assume that* $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 :^c \tau$. *Then, there exist* t_1', t_2' *such that* $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau \Rightarrow \Phi$ *and* $\Delta;\Phi_a \models \Phi$ *and* $|t_1'| = t_1$ *and* $|t_2'| = t_2$.

Proof. By induction on the given RelRef Core typing derivation. □

The details of completeness proof can be found in Appendix B, Section B.2.3.3.

9.3 RELREFU

In many cases, it is possible to prove a relation between two expressions by analyzing them *individually*. For example, if we can prove that t_1 and t_2 individually produce lists of length n with elements of type τ, then it is immediate that $\vdash t_1 \backsim t_2 : \mathrm{list}[n]^n\,\tau$. Falling

back to such *unary* analysis during the relational analysis is not allowed in RelRef. In this section, we extend RelRef to allow such fallback to unary analysis. We call the new system RelRefU.

RelRefU adds a new class of unary types, A, which ascribe individual expressions. These are the "standard" types from existing refinement systems like DML (Xi & Pfenning, 1999). For example, the unary type for lists, $\text{list}[n]\,A$, carries a refinement n, the length of the list. Unary types also include quantification over index variables, which we elide here for brevity. Importantly, we also add a new *relational* type $U(A_1, A_2)$ which ascribes any pair t_1, t_2, whose unary types are A_1 and A_2, respectively. The details of the syntax can be found in Section B.4.1 of Appendix B.

$$\begin{array}{lrcl} \text{Unary types} & A & ::= & \text{bool} \mid A_1 \to A_2 \mid \text{list}[n]\,A \mid \ldots \\ \text{Relational types} & \tau & ::= & \ldots \mid U(A_1, A_2) \end{array}$$

Declarative typing RelRefU has two typing judgments, unary and relational. The unary judgment's rules are exactly those of standard (unary) refinement type systems like DML (Xi & Pfenning, 1999), so we elide them here. The relational rules are those of RelRef and the following new rule, **r-switch**, which allows the use of unary typing in relational typing. Here, $|.|_i$ is a projection function that converts a relational type to its left ($i = 1$) or right ($i = 2$) unary type by forgetting relational refinements. For example, $|U(A_1, A_2)|_1 = A_1$ and $|\text{list}[n]^\alpha\,\tau|_1 = \text{list}[n]\,(|\tau|_1)$.

$$\frac{|\Gamma|_1 \vdash t_1 : A_1 \qquad |\Gamma|_2 \vdash t_2 : A_2}{\Gamma \vdash t_1 \backsim t_2 : U(A_1, A_2)}\ \textbf{r-switch}$$

Besides this rule, the second interesting aspect of RelRefU is subtyping for $U(A_1, A_2)$, which makes unary typing useful in relational reasoning. For instance, the example

given at the beginning of this section is typed using **r-switch** and the subtyping rule $U\,(\text{list}[n]\,A_1, \text{list}[n]\,A_2) \sqsubseteq \text{list}[n]^n\,(U\,(A_1, A_2))$. The details of the typing rules can be found in Figure B.44, B.45 and the relational subtyping rules in Figure B.46, and unary subtyping rules in Figure B.47.

Algorithmization Algorithmizing RelRefU faces new hurdles: The new rule **r-switch** is also not syntax-directed and subtyping for $U\,(A_1, A_2)$ is complex and cannot be redefined to make transitivity admissible. Consequently, we follow the approach we used for RelInf. We define a core language, RelRefU Core, that has the syntactic markers of RelRef Core and a new construct switch t (which marks the rule **r-switch**) to resolve the ambiguity in typing rules. The language has simple type equivalence, not subtyping. We then define an elaboration of RelRefU into RelRefU Core. Finally, we define a bidirectional, algorithmic type system called BiRelRefU and prove it sound and complete relative to RelRefU Core. The entire development is not particularly more challenging than that of RelInf (see in Section B.3.1 and Section B.3.2 of Appendix B) but is more tedious because we have to handle both unary and relational typing. Conceptually, the interesting aspect is that, in typing examples, we found it convenient to apply the **r-switch** rule in both checking and inference mode in the bidirectional type system, so this type system features two versions of the rule, one in each mode. This does not cause ambiguity in the rules since the mode is always uniquely known. The algorithmic subtyping rules can be found in Figure B.57, relational type equivalence rules in Figure B.58. The algorithmic unary typing rules are in Figure B.59, B.60. The algorithmic relational typing rules are in Figure B.61, B.62, B.63. The metatheory of RelRefU can be found in Section B.4.2.

9.4 RELCOST

Next, we add a relational effect, namely, relative cost to RelRefU. This results in the type system RelCost of Çiçek et al. (2017b). A quick preview, RelCost allows establishing an upper bound on the *relative cost* of two expressions t_1 and t_2, i.e. on $\text{cost}(t_1) - \text{cost}(t_2)$. For this, RelCost extends RelRefU's types and judgments with cost effects. Expressions are syntactically those of RelRefU, but the evaluation of elimination forms like list-case and function application produces non-trivial cost in the operational semantics. This section builds on the prior work on the bidirectional typing system for RelCost from the Ph.D. thesis of Ezgi Cicek Çiçek (2018). This section helps the readers to understand the bidirectional typing system of ARel.

It starts with the types. The relational function type $\tau_1 \to \tau_2$ is refined to $\tau_1 \xrightarrow{\text{diff}(D)} \tau_2$, where t (an index term of sort real) is an upper-bound on the relative cost of the bodies of the two functions that the type ascribes. Similarly, the *unary* function type $A_1 \to A_2$ is refined to $A_1 \xrightarrow{\text{exec}(L,U)} A_2$, where L and U are lower and upper bounds on the cost of the body of the function. Other than this, the types match those of RelRefU.

As an example, the list `map` function can be given the following more informative type in RelCost:

$$\forall D.(\Box(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2)) \to \forall n, \alpha.\text{list}[n]^{\alpha}\,\tau_1 \xrightarrow{\text{diff}(D\cdot\alpha)} \text{list}[n]^{\alpha}\,\tau_2$$

This type says that if two runs of `map` are given the *same* mapping function whose body's cost may vary by at most D across inputs, and two lists that differ in no more than α elements, then the relative cost of the two runs is no more than $D \cdot \alpha$. Intuitively, this makes sense. The two runs differ only in applications of the mapping function to list elements that differ. Each such application results in a relative cost at most D and there

$$\frac{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a; x:\tau_1, f:\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, \Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).t_1 \ominus \mathrm{fix}\ f(x).t_2 \lesssim 0 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2}\ \textbf{r-fix}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2 \\ \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_1\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \lesssim D_1 + D_2 + D : \tau_2}\ \textbf{r-app}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t \lesssim D : \tau \\ \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta;\Phi_a;\Gamma,\Gamma';\Omega \vdash t \ominus t \lesssim 0 : \Box\tau}\ \textbf{nochange}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma| \vdash_{L_1}^{U_1} t_1 : A \\ \Delta;\Phi_a;|\Gamma| \vdash_{L_2}^{U_2} t_2 : A\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim U_1 - L_2 : U\,A}\ \textbf{switch}$$

Figure 9.6: Selection of relational typing rules of RelCost.

are at most α such applications.

Declarative typing Like RelRefU, RelCost has two typing judgments, one unary and one relational. The difference from RelRefU is that these judgments now carry *cost effects*—upper and lower bounds on the cost of the expression being typed in the unary judgment and an upper bound on the relative cost of the two expressions in the relational judgment. The unary judgment, $\Delta;\Phi_a;\Omega \vdash_L^U t : A$, means that t has the unary type A and its cost is upper- and lower-bounded by U and L, respectively. The relational judgment, $\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$, means that t_1, t_2 have the relational type τ and their relative cost is upper-bounded by D. Here, we describe only the relational judgment, since the unary judgment has simpler rules.

The rules of the relational judgment are obtained by augmenting the rules of RelRefU to track costs. Selected, interesting rules are shown in Figure 9.6. The full de-

tails are in Figure B.75 to Figure B.81 in Appendix B. The rule **r-fix** types two recursive functions at type $\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2$ if their bodies have the relative cost D. Rule **r-app** relates $t_1\ t_2$ and $t_1'\ t_2'$ with a relative cost obtained by *adding* the relative costs of (t_1, t_1'), (t_2, t_2') and of the bodies of the applied functions (obtained from the function type in the first premise). Rule **nochange** says that if t relates to itself in a context that has only variables of $\Box$-ed types (i.e. they will get substituted by equal values in the two runs), then t's cost relative to itself is 0. Finally, rule **switch** allows a fallback to unary reasoning: If t_1's unary cost is upper-bounded by U_1 and t_2's unary cost is lower-bounded by L_2, then t_1, t_2's relative cost is (trivially) upper-bounded by $U_1 - L_2$.

Declarative subtyping RelCost's subtyping is directly based on RelRef and RelRefU, but the rules are additionally aware of costs. For example, the RelRef rule $\rightarrow\Box$**diff** (Figure 9.3) gets refined to: $\Box(\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2) \sqsubseteq \Box\tau_1 \xrightarrow{\mathrm{diff}(0)} \Box\tau_2$, which intuitively means that two equal functions when given two equal arguments reduce with exactly the same cost.

Towards algorithmization RelCost inherits all the non-syntax-directedness and subtyping complexity of RelRefU, and additionally adds costs. Implementing RelCost naively results in an immediate challenge: the relational aspects of RelCost create nondeterminism in the type system. This non-determinism comes from two aspects. First, some of the rules are not syntax-directed. For example, in the split rule r-split in Figure 9.6, we can choose any constraint to split on (and this is an infinite choice); we can apply the switch rule r-switch in Figure 9.6 anywhere; Second, the existence of the modality $\Box$ and its interaction with other types makes the algorithmization of relational subtyping quite tricky. To be concrete, it is challenging to define relational subtyping algorithmically while preserving the transitivity of the subtyping relation.

To build an algorithmic type system for RelCost, BiRelCost follows the approach

of (RelRef and) RelRefU. It first defines a simpler core language, RelCostCore, which resolves all rule ambiguity and has type equivalence in place of subtyping, and elaborate RelCost into this core language. This step is not significantly harder than for RelRefU since RelCost does not add more rule-ambiguity.

The interesting step is the second one—the bidirectional type system for RelCostCore. This bidirectional system uses constraints to relate not just type refinements but also costs of subexpressions, as explained next.

Algorithmic (bidirectional) type checking For RelCost, the bidirectional type checking must not only check/infer the type but also the cost. Consequently, one might expect that, for each of unary and relational typing, one would need not two but four bidirectional judgments—one judgment for each combination of checking and inferring the type and checking and inferring the cost. However, after some experimentation with the design, it is realized that the judgments where the type is checked and the cost is inferred or vice-versa is actually not required. In hindsight, this is because the cost can be viewed as an extension of the type even if the two are written separately for convenience, so the type and the cost always follow the same mode (checking or inference).

As a result, RelCost's bidirectional type system, called BiRelCost, uses two judgments – one for type checking and one for type inference – for each of unary and relational typing. We describe here only the two relational judgments as these are more complicated than the unary judgments. The relational checking judgment $\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \downarrow \tau, D \Rightarrow \Phi$ means that if Φ holds (assuming Φ_a), then $\Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ is provable in RelCostCore. Like the type τ, the relative cost D is also an input in this judgment, i.e. it is also checked. The relational inference judgment $\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \uparrow \tau \Rightarrow [\psi], D, \Phi$ has a similar meaning, but both the cost D and the type τ are outputs, i.e. both are inferred. This judgment also has an additional output ψ. This is a set of existentially

quantified cost variables generated by the rules. Substitutions for these variables must be found to satisfy Φ.

To understand the need for ψ, let us examine a few typing rules (Figure 9.7). The details of algorithmic typing rules can be found in Figure B.99 to Figure B.109. Consider the rule **alg-r-app-**$\uparrow$ for function application. In this rule, the relative cost D_1 of the functions is inferred (first premise), but the cost of the arguments must be checked. Since this cost is not available upfront, the rule creates a fresh existential variable D_2 and "checks" the arguments against that. This results in appropriate constraints on D_2 getting added to Φ_2 in the second premise. D_2 is included in the total cost and, importantly, it is added to ψ in the conclusion to indicate that it is existentially quantified (any substitution for it that satisfies the final constraints is okay). This pattern of creating existential variables for the costs that need to be checked but aren't already known is pervasive in the rules and is the key technical increment in BiRelCost relative to BiRelRefU.

There are two other interesting rules. The rule **alg-r-**$\uparrow\downarrow$, which switches from inference to checking mode when read top-down, closes the existential variables ψ in the premise by explicitly introducing an existential quantifier over them in the conclusion. In the rule **alg-r-nochange-**$\downarrow$ (which implements the rule **nochange** from Figure 9.6), the final cost must be 0. So, a constraint equating the given cost t to 0 is generated.

BiRelCost is sound and complete relative to RelCostCore in the sense of Theorems 66 and Theorem 67 (appropriately adapted to the judgments of RelCost). We defer the details to the appendix B.

9.5 AREL

Finally, we add mutable arrays to RelCost to reach ARel. Implementing ARel has one extra challenge besides those we have seen in implementing RelCost in Section 9.4:

$$\frac{D' \in \text{fresh}(\mathbb{R}) \qquad \Delta; D', \psi_a; \Phi_a; \Box\Gamma \vdash t \ominus t \downarrow \tau, D' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma', \Box\Gamma \vdash \text{NC}\ t \ominus \text{NC}\ t \downarrow \Box\tau, D \Rightarrow 0 \doteq D \wedge (\exists D' :: \mathbb{R}.\Phi)}\ \textbf{alg-r-nochange-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \uparrow \tau' \Rightarrow [\psi], D', \Phi_1 \qquad \Delta; \psi, \psi_a; \Phi_a \models \tau' \equiv \tau \Rightarrow \Phi_2}{\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi_2 \wedge D' \leq D}\ \textbf{alg-r-}\uparrow\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \uparrow \tau_1 \xrightarrow{\text{diff}(D_e)} \tau_2 \Rightarrow [\psi], D_1, \Phi_1 \\ D_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; D_2, \psi, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1, D_2 \Rightarrow \Phi_2\end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \uparrow \tau_2 \Rightarrow [D_2, \psi], D_1 + D_2 + D_e, \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-app-}\uparrow$$

Figure 9.7: Selection of algorithmic typing rules of BiRelCost.

the relational aspects of ARel create non-determinism in the type system. This non-determinism comes from the r-fix-ext shown in Figure 4.8, we have to guess the unary types of the functions (again an infinite choice); simultaneously, there are two rules for every array operator, which makes a choice between them non-deterministic because $\Box\tau$ is a subtype of τ.

We use the two-step method to algorithmize ARel. We first introduce an intermediate core language denoted ARelCore corresponding to an annotated, syntax-directed version of ARel. An elaboration of ARel to ARelCore eliminates the non-determinism of ARel in two steps as we have seen in previous systems. First, it adds annotations on term constructors to resolve non-syntax-directed typing rules. Second, it eliminates relational subtyping in the core language to avoid its corresponding non-determinism. The elaboration replaces relational subtyping with explicit coercions in the core language. The language ARelCore is sound and complete with respect to ARel, modulo finding the right elaboration. We then develop bidirectional type checking for ARelCore as the main component of our implementation.

This section is organized as follows. We first present the ARelCore language through

its syntax and typing rules. Then, we demonstrate the elaboration from ARel to ARelCore by showing some of the elaboration rules. We also justify its soundness and completeness. Then, after the introduction of ARelCore and the elaboration, we focus on the algorithmization and discuss how to construct a bidirectional type system with respect to ARelCore.

9.5.1 ARelCore

The difficulties encountered when trying to algorithmize ARel disappear when we instead algorithmize a core language suitable for bidirectional type checking. This core language can be seen as a theoretical medium for algorithmization, which is sound and complete with respect to our declarative type system ARel.

The syntax of ARelCore is an extension of the syntax of ARel with the annotations and corresponding term constructs shown in Figure 9.8. These annotations and new constructs make ARelCore's type system syntax-directed. For instance, the term construct split t with C can be used to mark a use of the rule r-split, which splits on the constraint C in the type-checking of t. The use of the rule r-fix-ext of Figure 4.6 is indicated by the construct FIXEXT $f(x).t$ with A that provides the unary type A of fix$f(x).t$. Similarly, the constructs NC t and switch t specify the use of the rule r-nc which introduces the $\Box$ modality, and the switch rule r-split, respectively. The construct contra t allows us to assign an arbitrary type to the expression t if there is a contradiction in our constraint environment. In addition, we introduce two variants of each array operation, e.g., alloc and alloc_b correspond to the two rules r-alloc and r-allocb, respectively.

The main purpose of ARelCore is to provide a syntax-directed type system that we can use as the target for a translation of ARel programs. Like ARel, ARelCore has two typing judgments. We have the unary judgment $\Delta;\Phi_a;\Omega \vdash^U_L t :^c A$ that assigns to a single

Terms $t ::= \cdots \mid \mathsf{return}\ t \mid \mathsf{alloc}\ t_1\ t_2 \mid \mathsf{alloc_b}\ t_1\ t_2 \mid \mathsf{updt}\ t_1\ t_2\ t_3 \mid \mathsf{updt_b}\ t_1\ t_2\ t_3$
$\mid \mathsf{read}\ t_1\ t_2 \mid \mathsf{read_b}\ t_1\ t_2 \mid \mathsf{let}\{x\} = t\ \mathsf{in}\ t_1 \mid \mathsf{switch}\ t \mid \mathsf{NC}\ t \mid \mathsf{split}\ t\ \mathsf{with}\ C$
$\mid \mathsf{contra}\ t \mid \mathsf{FIXEXT}\ t\ \mathsf{with}\ A$

Values $v ::= \cdots \mid \mathsf{return}\ t \mid \mathsf{alloc}\ t_1\ t_2 \mid \mathsf{alloc_b}\ t_1\ t_2 \mid \mathsf{updt}\ t_1\ t_2\ t_3 \mid \mathsf{updt_b}\ t_1\ t_2\ t_3$
$\mid \mathsf{read}\ t_1\ t_2 \mid \mathsf{read_b}\ t_1\ t_2 \mid \mathsf{let}\{x\} = t\ \mathsf{in}\ t_1$

Figure 9.8: Syntax of values and terms of ARelCore.

term t a type A, and bounds L and U to t's evaluation cost, under the environment $\Delta;\Phi_a;\Omega$. We also have relational typing $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \tau$, which says that two terms t_1 and t_2 are related at type τ, and that their relative cost is bounded by D, under the context Γ.

We present in Figure 9.9 and Figure 9.10 the selection of the typing rules for ARelCore. The full details can be found in Figure B.121 to Figure B.124 in Appendix B. We select rules for terms that may incur non-determinism in ARel, and we show how the non-determinism can be resolved at the syntactic level in ARelCore. As an example, to resolve the non-determinism caused by array-based operations we now have two different constructors for each operation and two different rules for them. For instance, for the read operation, we have a pair of terms $\mathsf{read}\ t\ t$ and $\mathsf{read_b}\ t\ t$ which correspond to the two typing rules **c-r-read** and **c-r-readb**, respectively. These typing rules of ARelCore are similar to their counterparts in ARel. Consider the rule **c-r-read**. It is easy to see that it has a structure similar to that of the rule r-read in Figure 4.8. For example, the same premise $\Delta;\Phi \vDash I' \leq I$ guarantees the array bound limit. Such similarity can also be found between the rule c-r-readb and the rule r-readb, and other pairs of array-related rules.

Subtyping is managed like what we do in BiRelCost. For unary subtyping, we have the rule **c-$\sqsubseteq$** which has a form similar to its counterpart in ARel. This is mainly because

the unary subtyping relation allows a direct algorithmization. On the other hand, relational subtyping in ARelCore is limited to type *equivalence* as in rule **c-r- ≡**. ARel's subtyping is then simulated using explicit coercion functions. The main reason for this approach comes from the fact that the modalities $\Box$ and U prevent easy algorithmization. We then show that every relational subtyping can be simulated by applying a coercion expression in ARelCore.

Lemma 7 (Simulation of Binary Subtyping in ARelCore)**.** *If* $\Delta;\Phi \models \tau \sqsubseteq \tau'$ *then there exists a term* t *in ARelCore, which we also denote* $coerce_{\tau,\tau'}$ *, such that* $\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c \tau \xrightarrow{diff(0)} \tau'$.

Proof. By induction on the given subtyping derivation. □

The detailed proof can be found in Appendix B, Section B.6.1.1.

9.5.2 Elaboration

Now that we have ARelCore, we want to show that we can elaborate any well-typed program from ARel into a well-typed program in ARelCore. The unary elaboration judgment $\Delta;\Phi_a;\Omega \vdash^U_L t \rightsquigarrow t^* : A$ represents the translation of a term t in ARel to its counterpart t^* in ARelCore. Both terms have the unary type A and bounds L, U in the contexts $\Sigma;\Delta;\Phi_a;\Omega$. In the same vein, the relational elaboration judgment has the shape $\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim D : \tau$. It represents the translation of a pair of terms (t_1, t_2) to (t_1^*, t_2^*). Both pairs are typed at τ in the given contexts, and both pairs have the same relative cost bound D. We show a selection of the unary and relational elaboration rules in Figure 9.11. The details can be found in Figure B.114 to Figure B.118 in Appendix B. In the unary elaboration subsumption rule **e-u-sub** and the relational rule **e-r-sub**, we see the two different approaches to subtyping we discussed previously. The rule **e-u-sub** preserves the unary subtyping relation during the translation. As we mentioned

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 :^c \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}}\ \textbf{c-r-alloc}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \Box\tau \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc_b}\ t_1\ t_2 \ominus \mathsf{alloc_b}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 :^c \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \emptyset\}}\ \textbf{c-r-allocB}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathsf{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 :^c \{P\}\ \exists\_.\tau\ \{P\}}\ \textbf{c-r-read}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathsf{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad I' \notin \beta \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read_b}\ t_1\ t_2 \ominus \mathsf{read_b}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 :^c \{P \star \gamma \to \beta\}\ \exists\_.\Box\tau\ \{P \star \gamma \to \beta\}}\ \textbf{c-r-readb}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathsf{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 :^c \tau \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \overset{\mathrm{diff}(D_1+D_2+D_3)}{\lesssim} 0 :^c \{P \star \gamma \to \beta\}\ \exists\_.\mathrm{unit}\ \{P \star \gamma \to \beta \cup \{I'\}\}}$$

c-r-update

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathsf{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 :^c \Box\tau \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt_b}\ t_1\ t_2\ t_3 \ominus \mathsf{updt_b}\ t_1'\ t_2'\ t_3' \overset{\mathrm{diff}(D_1+D_2+D_3)}{\lesssim} 0 :^c \{P \star \gamma \to \beta\}\ \exists\_.\mathrm{unit}\ \{P \star \gamma \to \beta \setminus \{I'\}\}}$$

c-r-updateb

Figure 9.9: Selection of typing rules of ARelCore, part 1.

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t :^c A \qquad \Delta;\Phi_a \models A \sqsubseteq A' \qquad \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'}{\Delta;\Phi_a;\Omega \vdash_{L'}^{U'} t :^c A'}\ \textbf{c-}\sqsubseteq$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \tau \qquad \Delta;\Phi_a \models \tau \equiv \tau' \qquad \Delta;\Phi_a \models D \leq D'}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D' :^c \tau'}\ \textbf{c-r-}\equiv$$

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash_{L_1}^{U_1} t_1 :^c A \qquad \Delta;\Phi_a;|\Gamma| \vdash_{L_2}^{U_2} t_2 :^c A}{\Delta;\Phi_a;\Gamma \vdash \text{switch } t_1 \ominus \text{switch } t_2 \lesssim U_1 - L_2 :^c U\ A}\ \textbf{c-switch}$$

Figure 9.10: Selection of typing rules of ARelCore, part 2.

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 \rightsquigarrow t_1^* : \mathsf{Array}_\gamma[I]\ A \\ \Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 \rightsquigarrow t_2^* : \text{int}[I'] \qquad \Delta;\Phi_a \models I' \leq I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Delta;\Phi_a;\Omega \vdash_0^0 \mathsf{read}\ t_1\ t_2 \rightsquigarrow \mathsf{read}\ t_1^*\ t_2^* : \overset{\text{exec}(L_1+L_2+L_r,U_1+U_2+U_r)}{\{P\}\ \exists\gamma.A\ \{P\}}}\ \textbf{e-u-read}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \\ \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \text{int}[I'] \\ \Delta;\Phi_a \models I' \leq I \qquad \Delta;\Phi_a \models I' \notin \beta \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \rightsquigarrow \mathsf{read}_\mathsf{b}\ t_1^*\ t_2^* \ominus \mathsf{read}_\mathsf{b}\ t_1'^*\ t_2'^* \lesssim 0 : \overset{\text{diff}(D_1+D_2)}{\{P \star \gamma \to \beta\}\ \exists\gamma.\square\tau\ \{P \star \gamma \to \beta\}}}\ \textbf{e-r-readb}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash_L^U t \rightsquigarrow t^* : A \qquad \Delta;\Phi_a \models A \sqsubseteq A' \\ \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'\end{array}}{\Delta;\Phi_a;\Omega \vdash_{L'}^{U'} t \rightsquigarrow t^* : A'}\ \textbf{e-u-sub}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau \\ \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \text{coerce}_{\tau,\tau'} \qquad \Delta;\Phi_a \models D \leq D'\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t'\ t_1^* \ominus t'\ t_2^* \lesssim D' : \tau'}\ \textbf{e-r-sub}$$

Figure 9.11: Selection of unary and relational elaboration rules.

before, ARelCore eliminates relational subtyping using coercions, which are provided by Lemma 7. In the rule **e-r-sub**, the ARelCore term t' is such a coercion. Elaboration rules for array operations have structures that are quite similar to the structure of the rules in ARel, except that they map terms to their annotated versions.

Elaboration is sound and complete in the sense of the following theorems.

Theorem 8 (Soundness of ARelCore & Type Preservation of Embedding)**.** *The following holds.*

1. *If* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A$*, then* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t^* :^{c} A$ *and* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t : A$.
2. *If* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim D : \tau$*, then* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1^* \ominus t_2^* \lesssim D :^{c} \tau$ *and* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$.

Proof. By simultaneous induction on the given elaboration derivations. □

The details of the proof can be found in Appendix B, Section B.6.1.2.

Theorem 9 (Completeness of ARelCore)**.** *The following holds.*

1. *If* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t : A$ *then* $\exists\, t^*$ *such that* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A$.
2. *If* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ *then* $\exists\, t_1^*$ *and* $\exists\, t_2^*$ *such that* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim D : \tau$.

Proof. By simultaneous induction on the given typing derivations. □

The details of the proof can be found in Appendix B, Section B.6.1.3.

9.5.3 Algorithmization

Next, we algorithmize ARelCore's type system. Here, we face the usual challenge of algorithmizing any type system: The need to either annotate or infer the types of bound

Terms $t ::= \cdots \mid \mathsf{return}\ t \mid \mathsf{alloc}\ t_1\ t_2 \mid \mathsf{alloc}_\mathsf{b}\ t_1\ t_2 \mid \mathsf{updt}\ t_1\ t_2\ t_3 \mid \mathsf{updt}_\mathsf{b}\ t_1\ t_2\ t_3$
$\mid \mathsf{read}\ t_1\ t_2 \mid \mathsf{read}_\mathsf{b}\ t_1\ t_2 \mid \mathsf{let}\{x\} = t\ \mathsf{in}\ t_1 \mid \mathsf{switch}\ t \mid \mathsf{NC}\ t \mid \mathsf{split}\ t\ \mathsf{with}\ C$
$\mid \mathsf{contra}\ t \mid \mathsf{FIXEXT}\ f(x).t\ \mathsf{with}\ A \mid (t : \tau, D) \mid (t : A, L, U)$

Figure 9.12: Syntax of BiARel.

variables. The problem is more nuanced than would be in a simply typed or even a refinement type calculus, since we must also deal with cost bounds in function and monadic types. To address this challenge, we rely on bidirectional type-checking or local type inference (Pierce & Turner, 2000), where type annotations must be provided only at explicit beta-redexes and at the top level, but everything else can be inferred. Our algorithmic system, which we call BiARel, inherits ARelCore's syntax and adds two annotated constructs $(t : \tau, D)$ and $(t : A, L, U)$ as shown in Figure 9.12. These are required for bidirectional type checking. Our bidirectional type system for ARelCore is similar to the one for RelCost (Çiçek et al., 2019), but extended in a nontrivial way to support array-related operations and for our extended fixpoint operator.

The core language has two typing judgments, the unary and relational one, while we have four of them in BiARel: two relational and two unary judgments. The relational typing judgment of ARelCore splits into two relational judgments in its bidirectional version, one for the "checking mode" and one for "inference mode". The relational checking judgment has the form:

$$\Sigma; \Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \downarrow \tau, D \Rightarrow \Phi.$$

Given the location environment Σ, the index variable environment Δ, the existential variable context ψ_a, the current constraint environment Φ_a, the relational typing context Γ, and terms t_1 and t_2, we *check* against the relational type τ and the relative cost D,

and we generate the constraint Φ, which must be discharged separately. In contrast, the relational inference judgment has the form:

$$\Sigma; \Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \uparrow \tau \Rightarrow [\psi], D, \Phi.$$

Here, we *synthesize* the relational type τ and the relative cost D, and we generate the constraint Φ with all the newly generated (existential) variables in ψ.

Similarly, we have two judgments for the unary case. The unary checking judgment has the form $\Sigma; \Delta; \psi_a; \Phi_a; \Omega \vdash t \downarrow A, L, U \Rightarrow \Phi$, while the unary inference judgment has the form $\Sigma; \Delta; \psi_a; \Phi_a; \Omega \vdash t \uparrow A \Rightarrow [\psi], L, U, \Phi$. Both these judgments can be understood in a way similar to their relational counterparts. In all the judgments, we write all the outputs (inferred components) in red and inputs in black. We can think of our unary checking judgment as a mutually recursive function $\mathsf{check}(t, A, L, U)$, whose inputs include a term t, a type A, the bounds L and U. The generated output is a constraint Φ, which holds exactly when t indeed has type A with execution cost bounded by L and U in our semantics. Likewise, the unary inference judgment performs like a function whose input is a term t. The outputs cover the inferred type A, the constraint Φ, the bounds U, L and an index variable environment ψ that tracks all the new generated index variables during the procedure of the inference. Here, the type and bounds hold iff $\exists \psi.\ \Phi$ holds. Notice that algorithmic typing judgments have one more input context ψ_a, which records previously eliminated existential variables.

We show selected algorithmic typing judgments in Figure 9.13 to explain how we handle ARel's non-determinism. The details can be found in Appendix B, from Figure B.126 to Figure B.132. The switch rule (r-switch) exists in both checking and inference modes. Both algorithmic rules relate the annotated terms switch t_1 and switch t_2 at the type $U\,(A_1, A_2)$ and generate the final constraint based on the constraints from

subterms t_1 and t_2 obtained in unary mode. The relative cost D is the difference of the maximal unary cost of t_1 (U_1) and the minimal unary cost of t_2 (L_2). In the checking rule, **alg-r-switch**$\downarrow$, this is forced in the output constraint. The split rule (r-split) exists only in checking mode (**alg-r-split**$\downarrow$). The terms split t_1 with C and split t_2 with C determine that this rule must be applied, splitting on constraint C. The final output constraint $C \to \Phi_1 \wedge \neg C \to \Phi_2$ also analyzes C.

The algorithmic counterpart of the rule r-fix-ext in checking mode, **alg-fixext**$\downarrow$, relates the annotated terms $\mathsf{FIXEXT}\ f(x).t$ with A_1 and $\mathsf{FIXEXT}\ f(x).t$ with A_2 and checks the subterms $\mathsf{fix} f(x).t$ and $\mathsf{fix} f(x).t'$ at the unary types A_1 and A_2, respectively. The final constraint is the combination of the constraints generated from the unary checking of the two subterms and the relational checking of the two function bodies.

The rule **alg-r-**$\uparrow\downarrow$ provides the possibility of transferring from inference mode to checking mode, while the rule **alg-r-anno-**$\uparrow$ allows the opposite transfer. Notice that we check the equivalence of two types in the rule **alg-r-**$\uparrow\downarrow$. In most bidirectional type systems, one would check subtyping here but, as explained earlier, ARelCore only has type equivalence.We emphasize the presence of the annotated term $(t : \tau, D)$ in the rule **alg-r-**$\uparrow\downarrow$. This annotated term allows the user to provide the type τ and the effect (relative cost D) to be checked in checking mode, as shown in the rule **alg-r-**$\uparrow\downarrow$. It helps when the bidirectional type checker has difficulty inferring the type of a term t by transferring the inferring challenge to a task that is easier, namely, checking a user-provided type. The unary annotated term (t, A, L, U) is useful in a similar way.

Next, we discuss selected rules for array operations. These operations constitute the main challenge in our bidirectional type system relative to prior work of RelCost. We show a selection of bidirectional rules for array operations in Figure 9.14 and Figure 9.15 . As mentioned, to resolve the non-determinism between the $\Box$-ed and non-$\Box$-

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;|\Gamma| \vdash t_1 \uparrow A_1 \Rightarrow [\psi_1], \_, U_1, \Phi_1 \\ \Delta;\psi_a;\Phi_a;|\Gamma| \vdash t_2 \uparrow A_2 \Rightarrow [\psi_2], L_2, \_, \Phi_2\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{switch } t_1 \ominus \text{switch } t_2 \uparrow U(A_1, A_2) \Rightarrow [\psi_1, \psi_2], U_1 - L_2, \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-switch}\uparrow$$

$$\frac{\begin{array}{c}L_1, U_1, L_2, U_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; U_1, L_1, \psi_a; \Phi_a; |\Gamma| \vdash t_1 \downarrow A_1, L_1, U_1 \Rightarrow \Phi_1 \\ \Delta; U_2, L_2, \psi_a; \Phi_a; |\Gamma| \vdash t_2 \downarrow A_2, L_2, U_2 \Rightarrow \Phi_2 \\ \Phi = \Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi_2 \wedge U_1 - L_2 \doteq D\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{switch } t_1 \ominus \text{switch } t_2 \downarrow U(A_1, A_2), D \Rightarrow \exists L_1, U_1 :: \mathbb{R}.(\Phi)}\ \textbf{alg-r-switch}\downarrow$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;C \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau, D \Rightarrow \Phi_1 \\ \Sigma;\Delta;\psi_a;\neg C \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau, D \Rightarrow \Phi_2 \qquad \Delta \vdash C\ \text{wf}\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{split } (t_1) \text{ with } C \ominus \text{split } (t_1') \text{ with } C \downarrow \tau, D \Rightarrow C \to \Phi_1 \wedge \neg C \to \Phi_2}$$

alg-r-split↓

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau' \Rightarrow [\psi], D', \Phi_1 \qquad \Sigma;\Delta;\psi,\psi_a;\Phi_a \models \tau' \equiv \tau \Rightarrow \Phi_2}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi_2 \wedge D' \leq D}\ \textbf{alg-r-}\uparrow\downarrow$$

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash \tau\ \text{wf} \qquad \text{FIV}(\tau, D) \in \Delta}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash (t : \tau, D) \ominus (t' : \tau, D) \uparrow \tau \Rightarrow [\cdot], D, \Phi}\ \textbf{alg-r-anno-}\uparrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;|\Gamma| \vdash \text{fix} f(x).t \downarrow A_1, 0, 0 \Rightarrow \Phi_1 \\ \Delta;\psi_a;\Phi_a;|\Gamma| \vdash \text{fix} f(x).t' \downarrow A_2, 0, 0 \Rightarrow \Phi_2 \\ \Sigma;\Delta;\psi_a;\Phi_a;f : \tau_1 \xrightarrow{\text{diff}(D')} \tau_2, f : U(A_1, A_2), x : \tau_1, \Gamma \vdash t \ominus t' \downarrow \tau_2, D' \Rightarrow \Phi \\ \Phi_r = \Phi \wedge \Phi_1 \wedge \Phi_2\end{array}}{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{FIXEXT} f(x).t \text{ with } A_1 \ominus \text{FIXEXT} f(x).t' \text{ with } A_2 \downarrow \\ \tau_1 \xrightarrow{\text{diff}(D')} \tau_2, D \Rightarrow \Phi_r \wedge 0 \doteq D\end{array}}\ \textbf{alg-fixext}\downarrow$$

Figure 9.13: Selection of algorithmic typing rules of BiARel.

$$\frac{\begin{array}{c} D_1, D_2 \in \mathsf{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_1,\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \mathrm{int}[I], D_1 \Rightarrow \Phi_1 \\ \Sigma;\Delta;D_2,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau, D_2 \Rightarrow \Phi_2 \qquad \Phi = \Phi_2 \wedge D_1 + D_2 \doteq D \\ \Sigma \vdash \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf} \qquad \Phi_r = \exists D_1 :: \mathbb{R}.\Phi_1 \wedge (\exists D_2 :: \mathbb{R}.\Phi) \end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \downarrow \{P\}\ \exists\gamma.\ \overset{\mathrm{diff}(D)}{\mathsf{Array}_\gamma}[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}, 0 \Rightarrow \Phi_r}\ \textbf{alg-r-alc-}\downarrow$$

$$\frac{\begin{array}{c} D_1, D_2 \in \mathsf{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_1,\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \mathrm{int}[I], D_1 \Rightarrow \Phi_1 \\ \Sigma;\Delta;D_2,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \square\tau, D_2 \Rightarrow \Phi_2 \qquad \Phi = \Phi_2 \wedge D_1 + D_2 \doteq D \\ \Sigma \vdash \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf} \qquad \Phi_r = \exists D_1 :: \mathbb{R}.\Phi_1 \wedge (\exists D_2 :: \mathbb{R}.\Phi) \end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{alloc_b}\ t_1\ t_2 \ominus \mathsf{alloc_b}\ t_1'\ t_2' \downarrow \{P\}\ \exists\gamma.\ \overset{\mathrm{diff}(D)}{\mathsf{Array}_\gamma}[I]\ \tau\ \{P \star \gamma \to \emptyset\}, 0 \Rightarrow \Phi_r}$$

alg-r-alcB-↓

$$\frac{\begin{array}{c} \Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1 \qquad P = P' \star \gamma \to \_ \\ \Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathrm{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2 \qquad \Delta;\psi_a;\Phi_a \vDash I' \le I \\ \Phi = \Phi_2 \wedge D_1 + D_2 \doteq D \qquad \Sigma;\Delta \vdash P \text{ wf} \qquad \Phi_r = \exists(\psi_1).\Phi_1 \wedge (\exists(\psi_2).\Phi) \end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \downarrow \{P\}\ \exists\_.\overset{\mathrm{diff}(D)}{\tau}\ \{P\}, 0 \Rightarrow \Phi_r}\ \textbf{alg-r-read-}\downarrow$$

$$\frac{\begin{array}{c} \Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1 \qquad P = P' \star \gamma \to \beta \\ \Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathrm{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2 \qquad \Delta;\psi_a;\Phi_a \vDash I' \le I \wedge \neg(I' \in \beta) \\ \Phi = \Phi_2 \wedge D_1 + D_2 \doteq D \qquad \Sigma;\Delta \vdash P \text{ wf} \qquad \Phi_r = \exists(\psi_1).\Phi_1 \wedge (\exists(\psi_2).\Phi) \end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{read_b}\ t_1\ t_2 \ominus \mathsf{read_b}\ t_1'\ t_2' \downarrow \{P\}\ \exists\_.\overset{\mathrm{diff}(D)}{\square\tau}\ \{P\}, 0 \Rightarrow \Phi_r}$$

alg-r-readB-↓

Figure 9.14: Selection of algorithmic typing rules for array operations of BiARel, part 1.

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1 \\ D_3 \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathrm{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2 \\ \Delta;\psi_a;\Phi_a \vDash I' \leq I \wedge \beta' = \beta \cup \{I'\} \qquad \Delta; D_3,\psi_2,\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \downarrow \tau, D_3 \Rightarrow \Phi_3 \\ P = P' \star \gamma \to \beta \qquad Q = P' \star \gamma \to \beta' \qquad \Phi = \Phi_2 \wedge D_1 + D_2 + D_3 \doteq D \\ \Sigma;\Delta \vdash P'\ \mathsf{wf} \qquad \Phi_r = \exists(\psi_1).(\Phi_1 \wedge (\exists(\psi_2).(\Phi_2 \wedge \exists D_3 :: \mathbb{R}.\Phi)))\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \downarrow\ \{P\}\ \exists\_.\mathrm{unit}\ \overset{\mathrm{diff}(D)}{\{Q\}}, 0 \Rightarrow \Phi_r}$$

alg-r-updt-↓

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1 \\ D_3 \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathrm{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2 \\ \Delta;\psi_a;\Phi_a \vDash I' \leq I \wedge \beta' = \beta \setminus \{I'\} \qquad \Delta; D_3,\psi_2,\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \downarrow \square\tau, D_3 \Rightarrow \Phi_3 \\ P = P' \star \gamma \to \beta \qquad Q = P' \star \gamma \to \beta' \qquad \Phi = \Phi_2 \wedge D_1 + D_2 + D_3 \doteq D \\ \Sigma;\Delta \vdash P'\ \mathsf{wf} \qquad \Phi_r = \exists(\psi_1).(\Phi_1 \wedge (\exists(\psi_2).(\Phi_2 \wedge \exists D_3 :: \mathbb{R}.\Phi)))\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{updt}_\mathsf{b}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}_\mathsf{b}\ t_1'\ t_2'\ t_3' \downarrow\ \{P\}\ \exists\_.\mathrm{unit}\ \overset{\mathrm{diff}(D)}{\{Q\}}, 0 \Rightarrow \Phi_r}$$

alg-r-updtB-↓

Figure 9.15: Selection of algorithmic typing rules for array operations of BiARel, part 2.

ed rules for each array operation, we use distinct expressions, e.g., $\mathsf{alloc_b}\ t_1\ t_2$ vs $\mathsf{alloc}\ t_1\ t_2$. Notice that the conclusion of every array operation is typed in checking mode. The two allocation rules **alg-r-alc-↓** and **alg-r-alcB-↓** check the first arguments t_1 and t_1' against the relational type int$[I]$ and relative cost D_1, then check the second arguments t_2 and t_2' against the relational type τ (or $\Box\tau$) and relative cost D_2. The final constraint $\Phi_r = \exists D_1 :: \mathbb{R}.(\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\Phi)$ requires that there exist D_1 and D_2 such that Φ_1 and Φ_2 hold and that $D_1 + D_2$ equals the given cost D.

The algorithmic typing rules for read and updt have other interesting aspects. These rules are in checking mode but the types of the first two arguments are inferred, not checked. This is because, although we know that the first argument of $\mathsf{read}\ t_1\ t_2$ or $\mathsf{updt_b}\ t_1\ t_2\ t_3$ must be an array and the second argument must be a number, we do not know the size of the array or the size (refinement index) of the number. Hence, we must infer this information. Additionally, these rules check the pre and postconditions. As an example, the condition $\neg(I' \in \beta)$ is checked in the rule **alg-r-readB-↓** to guarantee that we indeed read the same value on the two sides. Similarly, in the rules **alg-r-updt-↓** and **alg-r-updtB-↓**, the β' in the postcondition, representing the differences between the two arrays, must be the same as the β in the precondition except for the index I' which has been updated. For this, in the rule **alg-r-updt-↓** we check that $\beta' = \beta \cup \{I'\}$, while in the rule **alg-r-updtB-↓** we check that $\beta' = \beta \setminus \{I'\}$, consistent with the corresponding declarative typing rules of ARel.

Finally, we show the soundness and completeness of BiARel with respect to ARelCore. Soundness says if the constraints in the output of a provable BiARel typing judgment for term t are satisfiable, then a corresponding typing judgment for the type-erased term $|t|$ is provable in ARelCore. Completeness is the converse: If a term t has a typing derivation in ARelCore, then an annotation of t has a typing derivation

in BiARel and the constraints in the output of this derivation are satisfiable. In the following theorem, the function FIV(r)eturns the free index variables in its argument. $\Delta \rhd \theta_a : \psi_a$ means θ_a is a valid substitution for ψ_a under the index variable environment Δ. This substitution is used in the theorem, e.g., free index variables in constraints Φ are substituted using ψ_a, written $\Phi[\theta_a]$. We also define the type erasure operation $|t|$, which erases $(t : A, L, U)$ and $(t : \tau, D)$ to t.

Theorem 10 (Soundness of the Algorithmic Type checking in ARelCore)**.** *The following hold.*

1. *Assume that* $\Sigma; \Delta; \psi_a; \Phi_a; \Omega \vdash t \downarrow A, L, U \Rightarrow \Phi$ *and*

   1.1. $FIV(\Phi_a, \Omega, A, L, U) \subseteq dom(\Delta, \psi_a)$

   1.2. $\Sigma; \Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable for some* θ_a *such that* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

   Then $\Sigma; \Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a]$.

2. *Assume that* $\Sigma; \Delta; \psi_a; \Phi_a; \Omega \vdash t \uparrow A \Rightarrow [\psi], L, U, \Phi$ *and*

   2.1. $FIV(\Phi_a, \Omega) \subseteq dom(\Delta, \psi_a)$

   2.2. $\forall \theta\ \forall \theta_a.\ \Delta; \Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *is provable s.t* $\Delta \rhd \theta : \psi$ *and* $\Delta \rhd \theta_a : \psi_a$ *are derivable*

   Then $\Sigma; \Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta\,\theta_a]}_{L[\theta\,\theta_a]} |t| :^c A[\theta\,\theta_a]$.

3. *Assume that* $\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi$ *and*

   3.1. $FIV(\Phi_a, \Gamma, \tau, D) \subseteq dom(\Delta, \psi_a)$

   3.2. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable for some* θ_a *such that* $\Delta \rhd \theta_a : \psi_a$ *is derivable*

   Then $\Sigma; \Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta_a] :^c \tau[\theta_a]$.

4. *Assume that* $\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \uparrow \tau \Rightarrow [\psi], D, \Phi$ *and*

   4.1. $FIV(\Phi_a, \Gamma) \subseteq dom(\Delta, \psi_a)$

   4.2. $\forall \theta\ \forall \theta_a.\ \Delta; \Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *is provable s.t* $\Delta \rhd \theta : \psi$ *and* $\Delta \rhd \theta_a : \psi_a$ *are derivable*

   Then $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta\,\theta_a] :^c \tau[\theta\,\theta_a]$.

Proof. Statements (1–4) follow from simultaneous structural induction on the given algorithmic typing derivations. □

The details of the proof can be found in Appendix B, Section B.6.1.4.

Theorem 11 (Completeness of the Algorithmic Type checking in ARelCore)**.** *The following hold.*

1. *Assume that* $\Sigma; \Delta; \Phi_a; \Omega \vdash^U_L t :^c A$. *Then,* $\exists t'$ *such that*

   1.1. $\Sigma; \Delta; \cdot; \Phi_a; \Omega \vdash t' \downarrow A, L, U \Rightarrow \Phi$

   1.2. $\Delta; \Phi_a \models \Phi$

   1.3. $|t'| = t$

2. *Assume that* $\Sigma; \Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \tau$. *Then,* $\exists t'_1, t'_2$ *such that*

   2.1. $\Sigma; \Delta; \cdot; \Phi_a; \Gamma \vdash t'_1 \ominus t'_2 \downarrow \tau, D \Rightarrow \Phi$

   2.2. $\Delta; \Phi_a \models \Phi$

   2.3. $|t'_1| = t_1$ *and* $|t'_2| = t_2$

Proof. By simultaneous induction on the given ARelCore typing derivations. □

The details of the proof can be found in Appendix B, Section B.6.1.5.

Summary Since bidirectional type checking for effects has received relatively little attention even in the context of unary analysis, we briefly recapitulate the insights we gained from designing BiARel. First, bidirectional type checking extends very well to type systems with effects, even when combined with refinements and relational reasoning. Second, the mode of the effect (cost in our case) seems to mirror the mode of the type: The checking judgment checks both the type and the cost, while the inference judgment infers both. We did not find the need for a judgment that checks one but infers the other. Finally, bidirectional type checking generates more existential variables than it would without effects, but effects do not complicate the meta-theory (soundness and completeness) substantially. The creation of additional existential variables has consequences for constraint solving, since SMT solvers do not handle such variables well, we return to the issue in the next chapter.

CHAPTER 10

Typechecker BiARel

In the previous chapter, we have seen our two-step method to algorithmize relational refinement type and effect system such as ARel. In this chapter, we look at the implementation in the real world, that is, a bidirectional type checker. Before we introduce the type checker BiARel for ARel, we first discuss a prior work, BiRelCost Çiçek (2018). Its heuristics to alleviate the burden of annotations, and the constraint solving techniques can be also helpful in BiARel. And then, we show BiARel, especially how to cope with the array-based operations and the more complicated constraint.

10.1 HEURISTICS AND CONSTRAINT SOLVING TECHNIQUES IN PRIOR WORK

We list the heuristics in BiRelCost that can also be applied in BiARel, and its techniques to handle the constraint. These heuristics are listed below.

1. Subtyping is only invoked in three places: (a) for switching from checking to inference mode (rule **alg-r-↑↓** in Figure 9.7), (b) for the algorithmic version of the **nochange** rule (Figure 9.6), which checks subtyping on all variables in the context, and (c) as mentioned in the next point.

2. Relational subtyping rules that mention $\Box$ are applied lazily at specific elimination points. For instance, in typing a function application, if the applied expression's inferred type is $\Box(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2)$, it tries to complete the typing by subtyping to $\Box\tau_1 \xrightarrow{\text{diff}(0)} \Box\tau_2$ and $\tau_1 \xrightarrow{\text{diff}(D)} \tau_2$, in that order.

3. It switches to the unary reasoning (algorithmic analogue of rule **switch** from Figure 9.6) only when necessary i.e. when (a) eliminating expressions of the type

$U(A_1, A_2)$, (b) checking related expressions at type $U(A_1, A_2)$, and (c) no other relational rules apply.

BiARel also applies some techniques in constraint solving from BiRelCost. In the prior work, in principle, the constraints Φ output can be passed by the checking and inference judgments to an SMT solver that understands the domain of integers (for sizes) and real numbers (for costs). However, the constraint typically contains many existentially quantified variables and current SMT solvers do not eliminate such variables well. To solve this problem, BiRelCost has a simple pre-processing pass that finds candidate substitutions for existentially quantified variables. For any such variable n, it looks for constraints of the form $n = I$ and $n \leq I$. In either case, it considers I a candidate substitution for n. In this way, it generates a set of candidate substitutions for all existentially quantified variables. For each such substitution, BiRelCost tries to check the constraint's satisfiability using an SMT solver. This pre-processing of the constraint to eliminate existentially quantified variables of BiRelCost Çiçek et al. (2019) is also used in BiARel.

10.2 IMPLEMENTATION OF AREL

We develop a bidirectional type checker in Ocaml for ARel. We checked all the examples described in and some other examples described in the Appendix. Our type checker is implemented in OCaml and the code as well as the examples are available in the public Git repository at https://github.com/haddyclipk/ICFP2019_BiArel. We summarize salient points of our implementation and the results of our experiments in this section.

10.2.1 Heuristics for ARel

Our implementation uses heuristics to automatically backtrack over some typing rules of ARel. We do this to reduce the annotations programmers have to write manually.

One heuristic of our type checker is to automatically determine whether to apply the □-ed rule or the non-□-ed rule, rather than forcing the programmer to make this choice by providing an annotation on every array operation. Our heuristic applies the □-ed rule first and tries to solve the generated constraints (with an SMT-solver, as explained later). If the constraint cannot be solved, the heuristic tries the non-□-ed rule. For example, when processing a read operation we always try the **alg-r-readB-↓** rule first. The generated constraint is $I \notin \beta$. We pass this constraint to the SMT solver and if it says yes (satisfiable), we just continue. If the SMT solver says no, we backtrack and try the **alg-r-read-↓** rule.

Another heuristic is that we switch from relational to unary reasoning only when absolutely necessary. There are three cases when this happens: a) the unary type is explicitly mentioned with the construct FIXEXT t with A_1, b) the switch term switch t is used, and c) no other relational rules apply.

These heuristics suffice for our examples and reduce our annotation burden at the cost of some extra type checking time.

10.2.2 Constraint Solving in ARel

The primary difficulty in our implementation (and the most time-consuming step in type checking) is solving the constraints that the bidirectional type system generates. For this, we rely on an SMT solver. Specifically, we use Alt-Ergo (Bobot et al., 2013) through the Why3 frontend (Filliâtre & Paskevich, 2013). A fundamental difficulty here is that the SMT solver struggles with constraints that have too many existential quanti-

fiers. To alleviate this concern, we rely on a solution proposed in the implementation of RelCost (Çiçek et al., 2019): We implement a simple algorithm that generates candidate substitutions for existentially quantified variables by examining equality and inequality constraints that mention the variables. From a simple inspection of the algorithmic rules, we can see that generated constraints contain inequalities on index terms such as $I' \leq I$ to check the array bound limit (for instance, in the rule **alg-r-read-**$\downarrow$), and equalities such as $D_1 + D_2 = D$ to show that the inferred relative cost matches the relative cost we want to check. Çiçek et al. (2019)'s simple algorithm works remarkably well on these kinds of constraints.

A new challenge for ARel is how to represent and solve constraints involving the sets of integers β. There are three kinds of constraints involving these sets. 1) equalities of two sets $\beta = \beta'$ which are generated when the rule compares whether the precondition and postcondition are the same (e.g., in the rule **alg-r-read-**$\downarrow$); 2) containments between sets such as $\beta \subseteq \beta'$ which are generated when the postcondition is updated (e.g., **alg-r-updt-**$\downarrow$); 3) index inclusions $I \in \beta$ which are generated when certain indexes appear or disappear after the execution of the computation (e.g., **alg-r-updtB-**$\downarrow$). Index inclusions are also used by our heuristic to decide whether to use $\Box$-ed rules or not. To express these constraints, we rely on the library for set theory from Why3. Its operations for membership, equality, inclusion, empty set, union, intersection, and difference are enough to solve our constraints, and work well in our experience.

10.2.3 Type Checking Example in ARel

We illustrate our implementation of type checking by walking the reader through the type checking of the following *annotated* version of the mapi function from Chapter 3.

$$
\begin{array}{l}
\mathsf{let}\ \leq = (\mathsf{contra} : \mathrm{int} \to \mathrm{int} \to \mathrm{bool}, 0)\ \mathsf{in} \\
\mathsf{let}\ \mathsf{plusOne} = (\mathsf{contra} : \forall x.\mathrm{int}[x] \to \mathrm{int}[x+1], 0)\ \mathsf{in} \\
\mathsf{fix}\ \mathsf{mapi}\ (f).\lambda a.\lambda k.\lambda n. \\
\quad \mathsf{if} (\leq\ k\ n)\ \mathsf{then} \\
\quad\quad \boxed{\mathsf{SPLIT}}\{ \\
\quad\quad \mathsf{let}\ \{x\} = \boxed{(\mathsf{read}\ a\ k, \tau_1, 0\ ;\ \tau_1', 0)}\ \mathsf{in} \\
\quad\quad \mathsf{let}\ \{\_\} = \boxed{(\mathsf{updt}\ a\ k\ (f\ x), \tau_2, 0\ ;\ \tau_2', 0)}\ \mathsf{in} \\
\quad\quad \mathsf{mapi}\ f\ a\ (\mathsf{plusOne}\ k)\ n \\
\quad\quad \}\ \mathrm{with}\ (i \in b) \\
\quad \mathsf{else} \\
\quad\quad \mathsf{return}\ () \\
\leq 0\ :\ \forall r : .\Box(U(\mathrm{int},\mathrm{int}) \xrightarrow{\mathrm{diff}(r)} U(\mathrm{int},\mathrm{int})) \to \forall i, m, g, b.(i < m) \supset \\
\mathsf{Array}_g[m]\ U(\mathrm{int},\mathrm{int}) \to \mathrm{int}[i] \to \mathrm{int}[m] \to \{g \xrightarrow{\mathrm{diff}(|b \cap [i,m]| * r)} b\}\ \exists g.\mathrm{unit}\ \{g \to b\}
\end{array}
$$

First, we introduce two primitive functions for ≤ and plusOne. We use a trivial expression contra to simplify the actual implementations of these primitives. For ≤, contra appears in the term $(\mathsf{contra}, \mathrm{int} \to \mathrm{int} \to \mathrm{bool}, 0)$, which also includes the relational type and relative cost of ≤. We omit the relational costs on the arrows here because they are 0, which is also the default cost in our concrete syntax. (In experimental results below, we do not count these functions as annotations, since they are just primitive functions that actually should be inserted by the compiler automatically.)

Different from the mapi function, the map example verified by our type checker uses

the annotated term $\mathsf{SPLIT}\{t\}$ with C to specify the use of the split rule r-split in Figure 4.6. This eliminates non-determinism and guides the type checking. We also have the annotated terms $(\mathsf{read}\ a\ k, \tau_1, 0\ ;\ \tau_1', 0)$ and $(\mathsf{updt}\ a\ k\ (f\ x), \tau_2, 0\ ;\ \tau_2', 0)$, which vary from our standard terms because they now contain the two relational types and two relative cost upper bounds. As a reminder, the annotated terms aim to provide the necessary type and effect to help the type checker.

When we want to provide the relational type and relative cost for terms related to array operations, we need to provide two types and two relative costs, one for the $\Box$-ed rule and the other for the non-$\Box$-ed rule, as depicted in our heuristics in Section 10.2.1. As an example, consider the expression $(\mathsf{updt}\ a\ k\ (f\ x), \tau_2, 0\ ;\ \tau_2', 0)$. In this expression, $\tau_2 = \{g \to b\}\ \exists g.\overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{g \to b \setminus \{i\}\}$ is the relational type we want our type checker to use for the expression $\mathsf{updt}\ a\ k\ (f\ x)$ when it tries the rule **alg-r-updtB-↓**. The second type, $\tau_2' = \{g \to b\}\ \exists g.\overset{\mathrm{diff}(r)}{\mathrm{unit}}\ \{g \to b \cup \{i\}\}$ is the type for the non-$\Box$-ed rule **alg-r-updt-↓**, which is tried only if the rule **alg-r-updtB-↓** generates an unsatisfiable constraint.

Overall, this example uses three annotations, which are shown in boxes in the code above.

10.2.4 Experimental Evaluation of ARel

Table 10.1 summarizes some statistics about the performance of our type checker on different examples. For each example, we show the number of lines of code (LOC), the number of type annotations that are needed (#TYP), the number of annotations needed to disambiguate rules (#ESF), the time needed for type checking (TC), the time needed for solving the constraints that arise as premises during type checking (TC-SMT), and the time needed for solving the final constraint which is the output of the type checking (TF-SMT). Our experiments were performed on a 3.1 GHz Intel Core i5 processor with

8GB of RAM.

The programs mapi(1), mapi(2), boolOr, FFT, NSS and ISort are implementations of the corresponding examples discussed in Chapter 3 and Section 6. The example mapi(1) is mapi where we do not assume that the input functions are equal in the two runs. In mapi(2), we assume that the inputs functions are equal in the two runs (we presented the full annotated code for this example in Section 10.2.3). For FFT, which uses the auxiliary functions separate and loop, we report statistics for the whole program and individually for each auxiliary function. The program ISort uses helper functions insert and shift. These are also shown separately. The programs merge(1) and merge(2) are the two typings of imperative merge discussed in Chapter 6.

The function SAM (square-and-multiply) computes a positive power of a number represented as an array of bits, while comp checks the equality of two passwords represented as arrays of bits. These last two examples are array-based implementations of similar list-based implementations presented in Çiçek et al. (2017a). More details of these examples are in the Appendix A.

The results in Table 10.1 show that ARel can be used to reason about the relative cost of functional-imperative programs. Unsurprisingly, examples combining relational and unary reasoning (using rules r-fix-ext and r-switch) such as boolOr, NSS and ISort need more annotations and need more time for both type checking and SMT solving. In some examples like ISort, the time taken for solving constraints in the premises of the rules (TC-SMT), is very high. This is because of the heuristic we described at the beginning of this section where we try □-ed rules before non-□-ed rules. The SMT solver first tries to prove that the □-ed rule can be applied, but in some cases, it *times out*. This timeout period is counted in TC-SMT. It is set to 1s in all examples, except ISort and Insert, where we need 2s. TF-SMT, the time taken to check the final output constraint, is also high for

Table 10.1: Statistics from BiARel examples.

Benchmark	LOC	#TYP	#ESF	TC	TC-SMT	TF-SMT
mapi(1)	12	2	0	0.802	1.051	0.01
mapi(2)	19	3	1	1.247	0.994	0.02
boolOr	48	8	3	1.574	1.131	2.38
separate	36	8	0	1.351	2.148	0.01
loop	23	5	0	1.167	2.114	0.01
FFT	66	17	0	2.591	4.268	0.01
Search	62	10	3	3.753	4.430	6.56
NSS	94	12	3	4.158	4.413	10.03
shift	14	3	0	0.660	1.394	0.01
insert	22	6	0	1.001	3.019	0.01
iSort	134	12	3	2.897	6.181	10.70
merge(1)	29	8	0	2.203	2.232	0.01
merge(2)	64	11	2	3.231	0.349	0.02
sam	19	4	1	0.946	0.083	0.02
comp	20	3	0	1.138	0.112	0.01

some examples like ISort, but this is due to the complexity of the constraint.

10.2.5 Limitations and Future Directions of BiARel

One obvious limitation of our current prototype is efficiency, as we mentioned for examples NSS and iSort. Type checking slows down for two reasons: 1) The heuristic to determine whether to apply a $\Box$-ed rule to array-based operations has to wait for SMT to timeout in some cases. The time for alt-ergo (our SMT solver) to solve constraints varies considerably depending on the examples. When dealing with examples with many array-based operations, the problem is exacerbated. Unfortunately, we use Why3 to connect to alt-ergo and have to set a large timeout to guarantee enough time for alt-ergo to deal with the constraint on all connections, which accounts for the unnecessary time consumption. 2) The complexity of the final constraint grows with the number of array-based operations. This complexity translates to longer SMT-solving times.

Another limitation of our implementation is that some annotations are still needed

(despite our heuristics). We saw this in the example of Section 10.2.3.

We plan to improve our prototype by improving our heuristics and the constraint solving process. We would like to find a way to decrease connection times, the constraint solving time, and to make our backtracking more efficient. We also plan to investigate the use of other SMT solvers in order to improve efficiency further.

10.3 DISCUSSION ABOUT THE IMPLEMENTATION

Our observation so far is that, to a large extent, bidirectional relational type systems follow the same broad principles as (the well-studied) bidirectional unary type systems. However, relational type systems usually have more nondeterminism in both typing and subtyping rules. This non-determinism can be resolved by additional annotations, but an alternate approach is to use the annotations (and an elaboration to programs with these annotations) only as a theoretical tool in proving the completeness of the bidirectional type system relative to a standard declarative one, and to use heuristics to resolve the nondeterminism in an implementation. This is the approach we follow here. Bidirectionality also extends to type-and-effect systems, with the general principle that the mode of the effect follows the mode of the type. In the following, we explain the extent to which some other features of relational type systems that we have not considered so far can be brought under the purview of bidirectional typing.

First, in the type systems we have considered so far, the *same* variable is used for related inputs in the two typed expressions. A more expressive alternative is to allow the two expressions to have different free variables and type them relative to assumptions about relations between the variables. Bidirectionality extends to this setup fairly easily, without any new principles. The rule for typing variables still remains in inference mode, but the variables on the two sides do not have to be the same.

Second, many relational type systems feature *asynchronous* rules that allow analysis of only one expression, while maintaining relational reasoning, e.g., (Abel & Scherer, 2012)(Aguirre et al., 2017). These rules require care in a bidirectional type system. As a simple example, suppose we add to RelRefU the following rule, which allows relating $t_1\ t_2$ to t_2' by relating t_2 to t_2'.

$$\frac{|\Gamma|_1 \vdash t_1 : A_1 \to A_2 \qquad \Gamma \vdash t_2 \backsim t_2' : U\,(A_1, A_2')}{\Gamma \vdash t_1\ t_2 \backsim t_2' : U\,(A_2, A_2')}$$

The question is what modes – inference or checking – the two premises and the conclusion of the rule's bidirectional version should follow? The usual principle for function application says that the first premise and the conclusion should have inferred types, and the second premise should have a checked type. However, with this choice, the type A_2' must be guessed in an implementation. Consequently, the standard principle does not work for this (and other) asynchronous rules. Here, one possible way is to infer the type in the first premise and check the types in the second premise and the conclusion. This deviation from the usual bidirectional principle for function application arises because the types of the two sides (A_2, A_2') are coupled in a single type in the conclusion, while the (asynchronous) rule analyzes the two sides differently. This suggests possible future work on bidirectional typing where the different components of a *single* type can have different modes. Here, for instance, we may say that in $U\,(A_2, A_2')$, A_2 will be inferred, while A_2' will be checked. Then, one could stay with the usual bidirectional principle for function application on the left side.

Finally, we note that for type systems with refinements or dependencies, any algorithmization (bidirectional or not) is limited by the tractability of the underlying logic of assertions. There are many relational refinement type systems that are designed for

manual proofs and rely on very powerful assertion logic, in some cases as expressive as HOL or CiC (Aguirre et al., 2017; Nanevski et al., 2011). While bidirectional principles can be used to direct the generation of constraints in these cases, solving the constraints automatically is fundamentally intractable.

A particular case of the previous point is that in type systems with even simple quantitative refinements like list lengths and costs, the verification constraints often have existential quantifiers, which SMT solvers have significant difficulty handling. In our BiARel and its prior work BiRelCost Çiçek et al. (2019), a custom algorithm to eliminate these quantifiers is used, which works well. An alternative approach, used for example in liquid types (Vazou et al., 2014) and by Dunfield and Krishnaswami for GADTs (Dunfield & Krishnaswami, 2019), is to engineer or restrict the type system so that generated constraints do not have existential quantifiers. In the context of bidirectional type checking, this seems fundamentally difficult for quantitative *effects* like costs. The problem, as explained earlier, is that in trying to check a composite expression, the expression's given cost must be nondeterministically split between the subexpressions. Other (co)effect type systems (Dal Lago & Gaboardi, 2011; Dal Lago & Petit, 2013; Gaboardi et al., 2013; de Amorim et al., 2014) use bounded exponentials $!_n\tau$ (n copies of τ) from bounded linear logic to express quantitative (co)effects. The difficulty for bidirectionality is similar: Now, in a rule with two or more premises, one must nondeterministically split the context. Further work is needed to understand whether these type systems can be redesigned to not generate existential quantifiers.

CHAPTER 11

Related Work of BiARel

There is a lot of literature on implementing various combinations of refinement types, effect systems, modal types, and subtyping. A distinctive feature of our work is that it combines all these aspects in a relational setting.

11.1 BIDIRECTIONAL TYPE CHECKING

The idea of bidirectional type systems appeared in literature early on. However, the idea was popularized only more recently by Pierce and Turner Pierce & Turner (2000). The technique has shown great applicability—it has been used for dependent types Coquand (1996), indexed refinement types Xi & Pfenning (1999); Xi (1998), intersection and union types Davies & Pfenning (2000); Dunfield & Pfenning (2003), higher-rank polymorphism Peyton Jones et al. (2007); Dunfield & Krishnaswami (2013, 2019), contextual modal types Pientka (2008), algebraic effect handlers Lindley et al. (2017) and gradual typing Toro & Tanter (2015). Our approach is inspired by many of these papers, in particular DML Xi & Pfenning (1999); Xi (1998), but departs in the technical design of the algorithmic type system due to new challenges offered by relational and modal types, and costs. In particular, in these works is *unary*, i.e. a single program is checked (inferred) in isolation. Moreover, none of these works consider effects explicitly, i.e. as a type and effect system. One exception is the bidirectional effect system by Toro & Tanter (2015), which uses bidirectional type checking for gradual unary effects. However, their end goal is different since they infer minimal effects at compile time and then check dynamic effects at runtime.

11.2 LIGHTWEIGHT DEPENDENT TYPES

Numerous other systems use lightweight dependent types for program verification including, for instance, F* Swamy et al. (2011, 2016) and LiquidHaskell Vazou et al. (2014). However, these developments also do not consider comonadic types and costs. The DML approach has also been used in combination with linear types for asymptotic complexity analysis Dal Lago & Gaboardi (2011); Dal Lago & Petit (2013) and for reasoning about differential privacy Gaboardi et al. (2013); de Amorim et al. (2014). Besides lightweight dependent types, these papers also consider the comonadic modality of linear logic. This modality's structural properties are quite different from those of the comonadic $\Box$ we consider here. Another way to extend dependent types with cost information is to index a monad with the execution cost. Gundry considers this approach in a unary setting for a subset of Haskell with support for bidirectional typechecking Gundry (2013). None of these papers consider relational typing.

11.3 RELATIONAL VERIFICATION SYSTEMS

Some other type systems establish relational properties of programs. Barthe et al. (2014) consider a relational variant of a fragment of F* for the verification of cryptographic implementations, and similarly Barthe et al. (2015b) consider a relational refinement type system for differential privacy. However, some of the key technical challenges of our system, including those that arise from the interaction between unary and relational typing, as well as costs, do not show up in these settings. In the realm of incremental computing, some work Çiçek et al. (2015, 2016) has proposed declarative type systems for reasoning about update costs of incremental programs. These systems share similarities with the systems we considered here and we believe that the ideas developed in this paper can be applied to obtain algorithmic versions of these type systems as well.

11.4 ELIMINATION OF SUBTYPING

Prior work has also studied methods of eliminating subtyping as a way of simplifying type checking, e.g. Breazu-Tannen et al. (1991); Crary (2000). While our approach is similar in motivation, our technical challenges are quite different. Main difficulties in simplifying subtyping in our work arise from the interaction of the modalities $\Box$ and U with other connectives.

III ADAPTIVE DATA ANALYSIS

In the next few chapters, we present a program analysis that can help data analysts design adaptive data analyses controlling their generalization errors. Given an input program implementing an adaptive data analysis, our program analysis generates an upper bound on the total number of queries that the data analysis will run, and more interestingly also an upper bound on the depth of the chain of queries. These two measures can be used to select the right technique to guarantee a bound on the generalization error of the data analysis. Our program analysis is based on an analysis of the dependency graph between different queries, representing the potential chain an adaptive data analysis may generate. We show how the proposed program analysis can help to analyze the generalization error of several concrete data analyses with different adaptivity.

CHAPTER 12

Program Analysis for Adaptive Data Analysis

Data analysts run their data analysis on sample data drawn from the population they want to study because it is usually impossible to run it on the population. The generalization error is used to measure whether the results from the sample data generalize to the population. Hence, guaranteeing a low generalization error is an important topic in the data analysis area.

An adaptive data analysis can be seen as a process composed by multiple queries interrogating some data, where the choice of which query to run next may rely on the results of previous queries. The generalization error of each individual query/analysis can be controlled by using an array of well-established statistical techniques. However, when queries are arbitrarily composed, the different errors can propagate through the chain of different queries and result in a generalization error that is too high. To address this issue, data analysts are designing several techniques that not only guarantee the bounds on the generalization errors of single queries, but that also guarantee bounds on the generalization error of the composed analyses. The total number of queries and the depth of the chain of queries are of great significance when attempting to limit the generalization error when the composed data analyses are adaptive. The choice of which of these techniques to use often depends on the depth of the chain of queries that an adaptive data analysis can generate.

12.1 ADAPTIVE DATA ANALYSIS

Consider a dataset X consisting of n independent samples from some unknown population P. How can we ensure that the conclusions drawn from X *generalizes* to the population P? Despite decades of research in statistics and machine learning on methods for

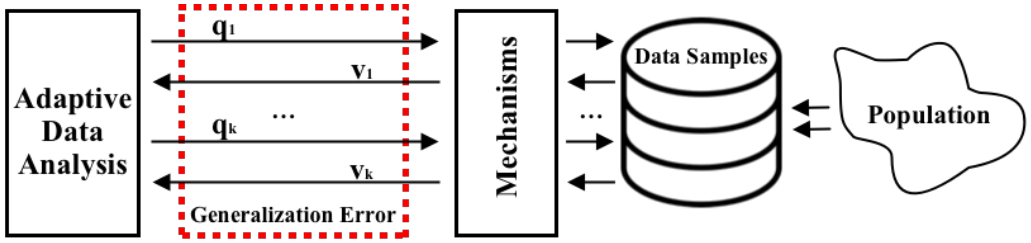


Figure 12.1: Overview of our adaptive data analysis model.

ensuring generalization, there is an increased recognition that many scientific findings generalize poorly (e.g. Ioannidis (2005); Gelman & Loken (2014)). While there are many reasons a conclusion might fail to generalize, one that is receiving increasing attention is *adaptivity*, which occurs when the choice of method for analyzing the dataset depends on previous interactions with the same dataset Gelman & Loken (2014).

Adaptivity can arise from many common practices, such as exploratory data analysis, using the same data set for feature selection and regression, and the re-use of datasets across research projects. Unfortunately, adaptivity invalidates these traditional methods for ensuring generalization and statistical validity, which assume the method is selected independently of the data. The misinterpretation of adaptively selected results has even been blamed for a "statistical crisis" in empirical science Gelman & Loken (2014).

A line of work initiated by Dwork et al. (2015c), Hardt & Ullman (2014) posed the question: Can we design *general-purpose* methods that ensure generalization in the presence of adaptivity, together with guarantees on their accuracy? The idea that has emerged in these works is to use randomization to help ensure generalization. Specifically, these works have proposed to mediate the access of an adaptive data analysis to the data by means of queries from some pre-determined family (we will consider here statistical or linear queries) that are sent to a *mechanism* which uses some randomized process to guarantee that the result of the query does not depend too much on the spe-

cific sampled dataset. This guarantees that the results of the queries generalize well. This approach is described in Figure 12.1. We have a population that we are interested in studying, and a dataset containing individual samples from this population. The adaptive data analysis we are interested in running has access to the dataset through queries of some pre-determined family (e.g., statistical or linear queries) mediated by a mechanism. This mechanism uses randomization to reduce the generalization error of the queries issued to the data. This line of work has identified many new algorithmic techniques for ensuring generalization in adaptive data analysis, leading to algorithms with greater statistical power than all previous approaches. Common methods proposed by these works include, the addition of noise to the result of a query, data splitting, etc. Moreover, these works have also identified problematic strategies for adaptive analysis, showing limitations on the statistical power one can hope to achieve. Subsequent works have then further extended the methods and techniques in this approach and further extended the theoretical underpinning of this approach, e.g. Dwork et al. (2015b,a); Bassily et al. (2016); Ullman et al. (2018); Feldman & Steinke (2017); Jung et al. (2020); Steinke & Zakynthinou (2020); Rogers et al. (2020).

A key development in this line of work is that the best method for ensuring generalization in an adaptive data analysis depends to a large extent on the number of *rounds of adaptivity*, the depth of the chain of queries. As an informal example, the program $x \leftarrow q_1(D); y \leftarrow q_2(D,x); z \leftarrow q_3(D,y)$ has three rounds of adaptivity, since q_2 depends on D not only directly because it is one of its input but also via the result of q_1, which is also run on D, and similarly, q_3 depends on D directly but also via the result of q_2, which in turn depends on the result of q_1. The works we discussed above showed that, not only does the analysis of the generalization error depend on the number of rounds, but knowing the number of rounds actually allows one to choose methods that lead to the small-

est possible generalization error. As an example, when a study includes queries with a large number of rounds of adaptivity, then a low generalization error can be achieved by adding Gaussian noise scaled to the number of rounds to the result of each query. When instead a study includes queries with a low number of rounds of adaptivity, then a low generalization error can be achieved by using more specialized methods, such as the reusable holdout technique from Dwork et al. (2015c).

12.1.1 Some Results in Adaptive Data Analysis

In Adaptive Data Analysis an *analyst* is interested in studying some distribution P over some domain $\mathscr{X}$. Following previous works Dwork et al. (2015c); Hardt & Ullman (2014); Bassily et al. (2016), we focus on the setting where the analyst is interested in answers to *statistical queries* (also known as *linear queries*) over the distribution. A statistical query is usually defined by some function $f : \mathscr{X} \to [-1,1]$ (often other codomains such as $[0,1]$ or $[-R,+R]$, for some R, are considered). $\mathbb{E}$ represents the expectation. The analyst wants to learn the *population mean*, which (abusing notation) is defined as

$$f(P) = \mathop{\mathbb{E}}_{X \sim P}\big[f(X)\big].$$

However, the distribution P can only be accessed via a set of *samples* $X_1, \ldots, X_n$ drawn from P. We assume that the samples are drawn independently and identically distributed (i.i.d.). These samples are held by a mechanism $M(X_1, \ldots, X_n)$ who receives the query f and computes an answer

$$a \approx f(P).$$

The naïve way to approximate the population mean is to use the *empirical mean*,

which (abusing notation) is defined as

$$f(X_1,\ldots,X_n) = \frac{1}{n}\sum_{i=1}^{n} f(X_i).$$

However, the mechanism M can then adopt some methods for improving the generalization error.

In this work, we consider analysts that ask a sequence of k queries $f_1,\ldots,f_k$. If the queries are all chosen in advance, independently of the answers of each one of them, then we say they are *non-adaptive*. If the choice of each query f_j depend on the prefix $f_1,a_1,\ldots,f_{j-1},a_{j-1}$ then they are *fully adaptive*. An important intermediate notion is r-*round adaptive*, where the sequence can be partitioned into r batches of non-adaptive queries. Note that non-interactive queries are 1-round and fully adaptive queries are k rounds.

We now review what is known about the problem of answering r-round adaptive queries.

Theorem 12. *For any distribution P, and any k* non-adaptive *statistical queries, the empirical mean satisfies*

$$\max_{j=1,\ldots,k} |a_j - f_j(P)| = O\left(\sqrt{\frac{\log k}{n}}\right)$$

*For any $r \geq 2$ and any r-*round adaptive *statistical queries, it satisfies*

$$\max_{j=1,\ldots,k} |a_j - f_j(P)| = O\left(\sqrt{\frac{k}{n}}\right)$$

These bounds are tight (up to constant factors) which means that even allowing one extra round of adaptivity leads to an exponential increase in the generalization error of the empirical mean, from $\log k$ to k.

Dwork et al. (2015c) and Bassily et al. (2016) showed that by using an alternative mechanism M which uses randomization in order to limit the dependency of a single query on the specific data instance, one can actually achieve a much stronger generalization error as a function of the number of queries, specifically.

Theorem 13 (Dwork et al. (2015c); Bassily et al. (2016))**.** *For any k, there exists a mechanism such that for any distribution P, and any $r \geq 2$ any r-*round adaptive *statistical queries, it satisfies*

$$\max_{j=1,\ldots,k} |a_j - f_j(P)| = O\left(\frac{\sqrt[4]{k}}{\sqrt{n}}\right)$$

Notice that Theorem 13 has different quantification in that the optimal choice of the mechanism depends on the number of queries. Thus, we need to know the number of queries *a priori* to choose the best mechanism.

Dwork et al. (2015c) also gave more refined bounds in terms of the number of rounds of adaptivity.

Theorem 14 (Dwork et al. (2015c))**.** *For any r and k, there exists a mechanism such that for any distribution P, and any $r \geq 2$ any r-*round adaptive *statistical queries, it satisfies*

$$\max_{j=1,\ldots,k} |a_j - f_j(P)| = O\left(\frac{r\sqrt{\log k}}{\sqrt{n}}\right)$$

This suggests that if one knows a good *a priori upper bound on the number of rounds of adaptivity*, one can get a much better guarantee of the generalization error, but only by using an appropriate choice of the mechanism.

This scenario motivates us to explore the design of program analysis techniques that can be used to estimate the number of *rounds of adaptivity* that a program implementing a data analysis can perform. These techniques could be ultimately be integrated into a tool for adaptive data analysis such as the *Guess and Check* framework by Rogers et al.

(2020).

12.2 CHALLENGES AND OUR SOLUTIONS

12.2.1 Formalizing an Adaptive Data Analysis Model

The first problem we face is *how to define formally* a model for adaptive data analysis which is general enough to support the methods we discussed above and would permit to formulate the notion of adaptivity these methods use. We take the approach of designing a programming framework for submitting queries to some *mechanism* giving access to the data mediated by one of the techniques we mentioned before, e.g., adding Gaussian noise, randomly selecting a subset of the data, using the reusable holdout technique, etc. In this approach, a program models an *analyst* asking a sequence of queries to the mechanism. The mechanism runs the queries on the data applying one of the methods discussed above and returns the result to the program. The program can then use this result to decide which query to run next. Overall, we are interested in controlling the generalization of the results of the queries which are returned by the mechanism, by means of the adaptivity.

To define adaptivity we consider a dependency graph between the different queries that we synthesize from the possible execution traces of the program representing the data analysis. The dependency graph is built by inspecting all the possible traces of execution and by identifying situations where the execution of a query *causes* the execution of another query. Intuitively, a query Q may depend on another query P, if there are two values that P can return which affect in different ways the execution of Q. For example, as depicted in Dwork et al. (2015b), a machine learning algorithm for constructing a classifier can be modeled by first computing each feature's correlations with the label via a sequence of queries and then constructing the classifier based on the correlation

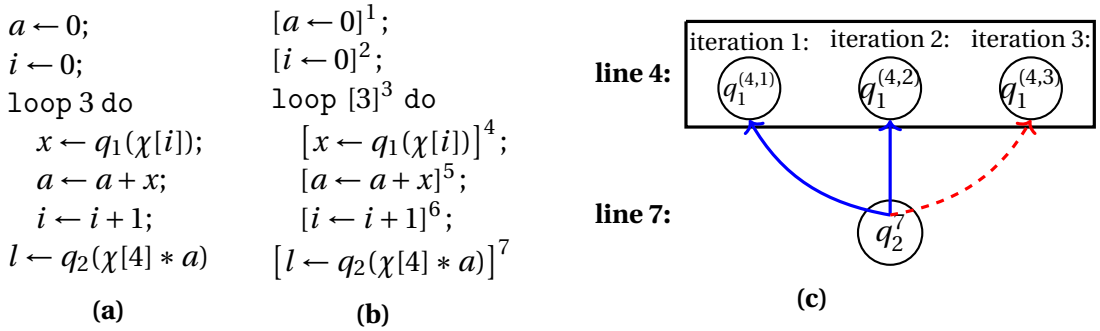


Figure 12.2: (a) Example of a program with two rounds of adaptivity, (b) Labeled program for the same example, (c) The corresponding query-based dependency graph.

values. If one feature's correlation changes, the classifier depending on features is also affected. This notion of dependency builds on the execution trace as a *causal history*. In particular, we are interested in the history or provenance of a query up until this is executed, we are not then concerned about how the result is used — except for tracking whether the result of the query may further cause some other query. This is because we focus on the generalization error of queries and not their post-processing.

Through an Example We motivate the definition of adaptivity we will use through a simple example illustrated in Figure 12.2(a), which implements a simple "two rounds strategy".

In this example, the analyst asks queries to the mechanism in two phases. In the first phase, the analyst asks a fixed number k of queries (in the example $k = 3$) and stores the answers that are provided by the mechanism. In the second phase, the analyst constructs a new query based on the results of the previous k queries and sends this query to the mechanism. More specifically, we assume that, in this example, the domain X contains at least four numeric attributes, which we index just by natural numbers. The queries inside the loop correspond to the first phase and compute an approximation of the empirical mean of the first three attributes. The query outside the loop corresponds

to the second phase and computes an approximation of the empirical mean where each record is weighted by the sum of the empirical mean of the first three attributes. Queries are of the form $q(e_q)$ where e_q is a query expression with a special variable χ representing a possible row. Mainly e_q represents a function from X to some domain U, for example U could be $[-1,1]$ or $[0,1]$. This function characterizes the linear query we are interested in running. As an example, $x \leftarrow q(\chi[2])$ computes an approximation, according to the used mechanism, of the empirical mean of the second attribute, identified by $\chi[2]$. Notice that we don't materialize the mechanism but we assume that it is implicitly run when we execute the query.

In order to analyze programs like the one we just discussed, it is convenient to work with a version of the program where similar commands can be easily distinguished. For this reason, we use labeled versions of programs, where labels correspond to lines of code. As an example, we give the labeled version of the two rounds example program in Figure 12.2(b).

This example is intuitively 2-rounds adaptive. The reason is that we have two distinguished phases and the queries that we ask in the first phase do not depend on each other, while the last query depends on all the previous queries. However, capturing this concept formally is surprisingly difficult. The difficulty comes from the fact that a query can depend on the result of another query in multiple ways, by means of data dependency or control flow dependency. In order to find the right definition for our goal, we take inspiration from the known results of the data analysis model we discussed above. This theory tells us that what we want to measure is the generalization error on the result of a query and not an arbitrary manipulation of the query. Indeed, arbitrary manipulations can change the generalization error. As an example, suppose that v is the result we get from running a query, if we multiply this result by some constant, we are also

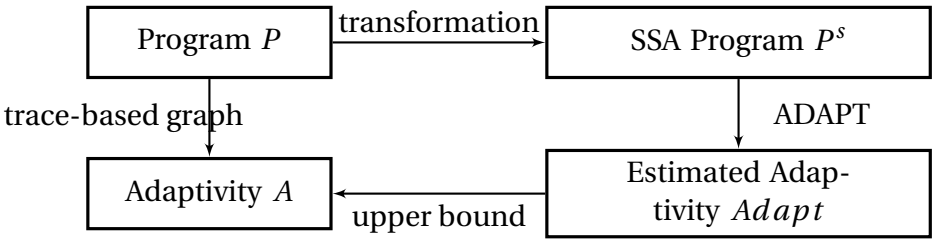


Figure 12.3: High level architecture of ADAPT.

changing the incurred error. Moreover, this theory tells us that we can always consider a non-adaptive set of queries as to being adaptive, and more importantly, that we can transform an adaptive query into a non-adaptive one, incurring an exponential blow-up of the number of queries. For example, we could ask many queries upfront, and depending on the results of some of them, we could return the results of others. For these reasons, we define adaptivity in terms of the possible execution traces of the program on all possible inputs. A trace of execution is a list of query requests of the form $[q(v_1)^{(l_1,w_1)},\ldots,q(v_n)^{(l_n,w_n)}]$, where every occurrence of a query is labeled with the line of code l it appears at, and the counter w identifying the possible loop iteration happening when a query is called. For example, in $q(v_1)^{(3,1)}$, the superscript $(3,1)$ indicates that the query is asked at line 3 and that the query is requested in the first iteration of the loop. When the query is not in a loop, we omit the counter.

Using traces we can identify situations when one query can affect the execution of another one. Using this information we can build a directed graph, called the query-based dependency graph, where the nodes represent the queries that are executed and the edges between two nodes represent the fact that one query may depend on the other. We show such a graph for our running example in Figure 12.2(c). We can then define adaptivity as the longest possible path in this graph. Looking again at our example, it is easy to see that the longest path in the graph in Figure 12.2(c), which we mark with a red dashed arrow, is 2, as we were expecting.

12.2.2 Static Analysis for Adaptivity

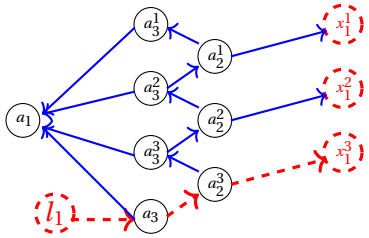
The second problem we face is *how to estimate the adaptivity of a given program.* The adaptive data analysis model we consider and our definition of adaptivity suggest that for this task we can use a program analysis that is reminiscent of information flow control. However, this is not sufficient since, in general, a query Q is not a monolithic block but rather it may depend, through the use of variables and values, on other parts of the program. Hence, we also need to consider some form of data-flow analysis. Our program analysis, named ADAPT, combines information flow and data-flow analysis using an adjacency matrix M representing the dependency between different variables, and a vector V representing different queries. These two components allow us to over-approximate the dependency graph and estimate the adaptivity of the program.

To simplify our analysis, we do not directly apply the program analysis to the source program. Instead, we first transform the program into static single assignment form, SSA form. In this form all the variables are assigned once, including variables in loops, and this helps our analysis in avoiding the complexity of handling variables reassignment. Moreover, we show that by analyzing programs in SSA form, we get a bound on the number of rounds of adaptivity that is also a bound for the source program.

The high-level architecture of our static analysis framework is shown in Figure 12.3. The input of the analysis is a labeled program P for which the adaptivity A is defined by means of a trace-based definition, as discussed above. In order to estimate an upper bound on A, our program analysis first transforms the program P into the static single assignment (SSA) form. The goal of this step is to guarantee that each variable is assigned only once. We show the result of this transformation applied to our two rounds strategy example in Figure 12.4(a). This transformation, when applied to a loop, introduces some extra variables that serve as intermediate storage. For example, in 12.4(a) there is

$[a_1 \leftarrow 0]^1$;
$[i_1 \leftarrow 0]^2$;
`loop` $[3]^3$ `do`
  $[(i_3, i_1, i_2), (a_3, a_2, a_1)]$
  $\left[x_1 \leftarrow q_1(\chi[i_3])\right]^4$;
  $[a_2 \leftarrow a_3 + x_1]^5$;
  $[i_2 \leftarrow i_3 + 1]^6$;
$\left[l_1 \leftarrow q_2(\chi[4] * a_3)\right]^7$

(a)



(b)

Figure 12.4: (a) Example of a SSA program with two rounds of adaptivity (b) The corresponding variable-based dependency graph.

a new instruction $[(i_3, i_1, i_2), (a_3, a_1, a_2)]$ after the loop. This instruction asserts that the value of the new variables i_3 and a_3, depending on the execution step, may come from i_1 or i_2, a_1 or a_2, respectively. To be concrete, if it is in the first iteration of the loop, $i_3 = i_1$, otherwise, $i_3 = i_2$. For those readers who are familiar with SSA, it can be regarded as another form of the phi node. The transformation of a program into SSA form preserves the execution traces, and so, in turns, it preserves the adaptivity.

The main component of our framework is an algorithm, which we call $ADAPT$. This algorithm constructs a variable-based weighted directed dependency graph where nodes are annotated variables and edges represent potential dependencies between the variables. We show the variable-based dependency graph for our running two rounds example in Figure 12.4(b). The algorithm builds this dependency graph by traversing the SSA program P^S and collecting the information about dependencies between the different variables in an adjacency matrix M, and information about the relations between queries and variables in a vector V. The matrix M collects information about both data dependency and control flow dependency. The vector V is used to assign a weight to the different nodes. In particular, to each variable related to a query, the algorithm assigns weight 1 and to any other variables, the algorithm assigns no weight. The estimated upper bound for the adaptivity is the weight of the path in the graph with

maximal weight. In Figure 12.4(b), we have a dependency graph between variables. The nodes are the assigned variables and the edge between two nodes means one variable may have data dependency or control flow dependency(or both) on the other variable. Only the variable x which is assigned with the query result via $x \leftarrow q(e_q)$ will be marked as the weighted node with the unit weight. The nodes in a dashed circle are those special nodes associated with query requests. We want to find the path in the graph with the highest weights (the most special nodes). The path with maximal weight is the dashed one in the graph. The weight of this path is 2, providing us a tight upper bound on the adaptivity of P. More in general, we prove that the upper bound estimated by ADAPT gives a sound overapproximation of the adaptivity of the analyzed program.

12.3 CONTRIBUTIONS OF ADAPT

To Summarize, our work aims at the design of a static analysis for programs implementing adaptive analysis that can estimate their rounds of adaptivity. Specifically, our contributions are as follows:

1. A programming framework for adaptive data analyses where the program represents an analyst that can query a generalization-preserving mechanism mediating the access to some data.

2. A formal definition of the notion of adaptivity under the aforementioned analyst-mechanism model. This definition is built on a query-based dependency graph built out of all the possible program execution traces.

3. A program analysis algorithm ADAPT which provides an estimated upper bound on the adaptivity via a variable-based dependency graph.

4. A soundness proof of the program analysis showing that the adaptivity estimated by ADAPT bounds the true adaptivity of the program.

CHAPTER 13

Formal Definition of Adaptivity

In this chapter, we formally introduce the language we will focus on for writing data analyses. This is a simple loop language with some primitives for calling queries. After defining the syntax of the language and showing an example, we will define its trace-based operational semantics. This is the main technical ingredient we will use to define the program's adaptivity.

13.1 SYNTAX OF LOOP LANGUAGE

We introduce the syntax of the loop language we use to write our data analyses. It is standard that expressions can be either arithmetic expressions or boolean expressions. An arithmetic expression can be a constant n denoting integer, a variable x from some countable set `Var`, a combination of arithmetic expressions by means of the symbol $\oplus_a$, denoting basic operations including addition, product, subtraction, etc. A boolean expression can be `true` or `false`, the negation of a boolean expression, or a combination of boolean expressions by means of $\oplus_b$, denoting basic boolean connectives, or the result of some basic comparison $\sim$ between arithmetic expressions, e.g., $\leq, =, <,$, etc. Additionally, list over expressions is supported and [] stands for the empty list. The access to elements in the list can be achieved through $x[a]$ when variable x is referred to a list. The value v now contains the natural number n, the boolean primitives true and false, the empty list [] and non-empty list $[v, \ldots, v]$.

As we have seen in previous examples, that the query request may look like $q(\chi[2] + 3)$. We have the query expressions e_q for the query request. The query expressions include the special variable χ representing a row of the database, and access to values at a certain index in χ, as $\chi[a]$. The arithmetic expressions can also appear in e_q. We also

Arithmetic Operators	$\oplus_a$	::=	$+ \mid - \mid \times \mid \div$
Boolean Operators	$\oplus_b$	::=	$\vee \mid \wedge \mid \neg$
Relational Operators	$\sim$	::=	$< \mid \leq \mid ==$
Arithmetic Expressions	a	::=	$n \mid x \mid a \oplus_a a$
Boolean Expressions	b	::=	$\text{true} \mid \text{false} \mid \neg b \mid b \oplus_b b \mid a \sim a$
Expressions	e	::=	$a \mid b \mid [] \mid [e, \ldots, e]$
Values	v	::=	$n \mid \text{true} \mid \text{false} \mid [] \mid [v, \ldots, v]$
Query expressions	e_q	::=	$a \mid \chi \mid \chi[a] \mid e_q \oplus_a e_q$
Query Values	v_q	::=	$n \mid \chi \mid \chi[n] \mid v_q \oplus_a v_q$
Commands	c	::=	$\texttt{skip} \mid x \leftarrow e \mid x \leftarrow q(e_q) \mid \texttt{loop}\ a\ \texttt{do}\ c$ $\mid c; c \mid \text{if}\ (b, c, c)$

Figure 13.1: Syntax of loop language.

allow the arithmetic operations on query expressions, such as $\chi[2]+3$. The normal form v_q of the query expression can be the natural number, χ and access to it $\chi[n]$, and also the arithmetic operations over two query values $v_q \oplus_a v_q$. We also define the equality relation between the query values, denoted as $=_q$, such that $\chi[2]+2 =_q 2+\chi[2]$. The details can be seen in Appendix C, Section C.1. This design of having query expressions and their normal form is important in the trace-based operational semantics, in the next section.

A command c can either be `skip`, an assignment command $x \leftarrow e$, the composition of two commands $c; c$, an if statement if (b, c, c), a loop statement `loop` a `do` c. The main novelty of the syntax is the query request command $x \leftarrow q(e_q)$. Notice inside the query, we have the query expression e_q. As a reminder, we focus on linear queries, specified by a function from rows to $[0, 1]$ or $[-1, +1]$. To express these functions, we introduce the special variable χ to represent the rows of the database in the query expressions. In this sense, a simple linear query which returns the first element of the row is written as $q(\chi[1])$, representing a query $q(\chi) = \chi(1)$. The query can also take variables as input, the aforementioned two round example in Figure 12.2.(a) uses the loop counter i to

construct the query $q(\chi[i])$, and a more complicated one $q(\chi[4]+a)$ at the end. However, when the query is sent to the database, the argument inside q should be in the normal form. So the variable a should be evaluated first in $q(\chi[4]+a)$.

13.2 TRACE-BASED OPERATIONAL SEMANTICS

We evaluate programs in our loop language by means of our trace-based operational semantics, to capture the dependency between queries. For distinguishing elements in the trace, we add a label to commands in the loop language as follow:

$$\begin{array}{rrcl} \text{Labeled commands} & c & ::= & [x \leftarrow e]^l \mid [x \leftarrow q\,(e_q)]^l \mid \texttt{loop}\ [a]^l\ \texttt{do}\ c \mid c;c \\ & & & \mid \text{if}\ ([b]^l, c, c) \mid [\texttt{skip}]^l \end{array}$$

Each command is now labeled with a label l, a natural number standing for the line of code where the command appears. Notice that we associate the label l to the conditional predicate b in the if statement, and to the loop counter a in the loop statement. We will also use Loop map w as defined below.

$$\begin{array}{llllllll} \text{Loop Map} & w & \in & \text{Label} \rightarrow \mathbb{N} & \text{Memory} & m & ::= & [] \mid m[x \rightarrow v] \\ \text{Annotated Query} & \mathcal{AQ} & ::= & \{q(v_q)^{(l,w)}\} & \text{Trace} & t & ::= & [] \mid q(v_q)^{(l,w)} :: t \end{array}$$

Loop map are a map from the label l to the iteration number n. A mapping $[k \rightarrow n]$ gives accurate information on which loop a statement is in by its key k (label at loop counter), and which iteration n the statement belongs to. For example, the loop map $w = [3:1, 4:2]$ indicates that the statement is currently in a nested loop, the outer loop starting from label 3 and in its first iteration, the statement is now in the inner loop starting from label 4 and in the second iteration. We use $\emptyset$ to represent an empty map,

indicating the statement is not in any loop. We define operations on w as follows.

$$\begin{array}{lll} w \setminus l & = w & l \notin Keys(w) \\ & = w_l & Otherwise \end{array} \qquad \begin{array}{lll} w + l & = w[l \rightarrow 1] & l \notin Keys(w) \\ & = w[l \rightarrow w(l)+1] & Otherwise \end{array}$$

We use $w \setminus l$ to remove the mapping of the key l from the loop map w. This is used when exiting the loop at line l. We denote w_l to express a map identical to w but without the mapping of label l. We record in w the first iteration of a loop marked by label l by assigning l with the iteration 1. The mapped number increase when going into another iteration of the same loop. We use $Keys(w)$ to return all the keys of the loop map w.

A memory is standard, a map from variables to values. Queries can be uniquely annotated as $\mathscr{AQ}$, and the annotation (l, w) considers the location of the query by line number l and which iteration the query is at when it appears in a loop statement, specified by w. A trace t is a list of annotated queries accumulated along with the execution of the program.

A trace can be regarded as the program history, where this history consists of the queries asked by the analyst during the execution of the program. We collect the trace with a trace-based small-step operational semantics based on transitions of the form $\langle m, c, t, w\rangle \rightarrow \langle m', \texttt{skip}, t', w'\rangle$. It states that a configuration $\langle m, c, t, w\rangle$ evaluates to another configuration with the trace and loop map updated along with the evaluation of the command c to the normal form of the command `skip`. A configuration contains four elements: a memory m, the command c to be evaluated, a starting trace t, a starting loop map w. The loop map remains empty until the evaluation goes into a loop. Correspondingly, we also have the evaluation of expressions in Figure 13.2. The arithmetic expression evaluation of the form $\langle m, a\rangle \rightarrow_a \langle m, a'\rangle$, the boolean expression evaluation has the form of $\langle m, b\rangle \rightarrow_b \langle m, b'\rangle$. In particular, we have the evaluation of the query ex-

pressions, of the form $\langle m, e_q \rangle \rightarrow_q \langle m, e'_q \rangle$. These expressions will evaluate to the normal form, e to v and e_q to v_q.

We give a selection of rules of the trace-based operational semantics in Figure 13.3. The rule **l-query-e** evaluates the argument e_q of a query request $q(e_q)$ using the query evaluation $\rightarrow_q$. When the query expression is in the normal form, this query will be answered. The rule **l-query-v** modifies the starting memory m to $m[v_q/x]$ using the answer v of the query $q(v_q)$ from the mechanism, with the trace expanded by appending the query $q(v_q)$ with the current annotation (l, w). The rule for assignment is standard and the trace remains unchanged. The sequence rule keeps tracking the modification of the trace, and the evaluation rule for if conditional goes into one branch based on the result of the conditional predicate b. The rule **l-loop-a** first evaluates the loop counter a, when the loop counter is a number, then the evaluation will start to execute the loop body. The rules for loop modify the loop map w. In the rule **l-loop**, the loop map w is updated by $w + l$ because the execution goes into another iteration when the condition $v_N > 0$ is satisfied. When v_N reaches 0, the loop exits and the loop map w eliminates the label l of this loop statement by $w \setminus l$ in the rule **l-loop-exit**.

13.3 QUERY-BASED DEPENDENCY GRAPH

We define adaptivity through a query-based dependency graph. In our model, an *analyst* asks a sequence of queries to the mechanism, and the analyst receives the answers to these queries from the mechanism. A query is adaptively chosen by the analyst when the choice of this query is affected by answers from previous queries. In this model, the adaptivity we are interested in is the length of the longest sequence of such adaptively chosen queries, among all the queries the data analyst asks to the mechanism. Also, when the analyst asks a query, the only information the analyst will have will be the an-

$$\boxed{\langle m, a\rangle \to_a \langle m, a'\rangle}$$

$$\frac{n = m(x)}{\langle m, x\rangle \to_a \langle m, n\rangle}\ \text{a-var} \qquad \frac{\langle m, a_1\rangle \to_a \langle m, a_1'\rangle}{\langle m, a_1 \oplus_a a_2\rangle \to_a \langle m, a_1' \oplus_a a_2\rangle}\ \text{a-aop1}$$

$$\frac{\langle m, a_2\rangle \to_a \langle m, a_2'\rangle}{\langle m, n_1 \oplus_a a_2\rangle \to_a \langle m, n_1 \oplus_a a_2'\rangle}\ \text{a-aop2} \qquad \frac{n = n_1 \oplus n_2}{\langle m, n_1 \oplus_a n_2\rangle \to_a \langle m, n\rangle}\ \text{a-aop3}$$

$$\boxed{\langle m, b\rangle \to_b \langle m, b'\rangle}$$

$$\frac{\langle m, a_1\rangle \to_a \langle m, a_1'\rangle}{\langle m, a_1 \sim a_2\rangle \to_b \langle m, a_1' \sim a_2\rangle}\ \text{b-rop1} \qquad \frac{\langle m, a_2\rangle \to_a \langle m, a_2'\rangle}{\langle m, n_1 \sim a_2\rangle \to_b \langle m, n_1 \sim a_2'\rangle}\ \text{b-rop2}$$

$$\frac{\text{true} = n_1 \sim n_2}{\langle m, n_1 \sim n_2\rangle \to_a \langle m, \text{true}\rangle}\ \text{b-rop3} \qquad \frac{\text{false} = n_1 \sim n_2}{\langle m, n_1 \sim n_2\rangle \to_a \langle m, \text{false}\rangle}\ \text{b-rop4}$$

$$\boxed{\langle m, e_q\rangle \to_q \langle m, e_q'\rangle}$$

$$\frac{\langle m, a\rangle \to_a \langle m, a'\rangle}{\langle m, a\rangle \to_q \langle m, a'\rangle}\ \text{q-a} \qquad \frac{\langle m, a\rangle \to_a \langle m, a'\rangle}{\langle m, \chi[a]\rangle \to_q \langle m, \chi[a']\rangle}\ \text{q-chi}$$

$$\frac{\langle m, e_{q_1}\rangle \to_q \langle m, e_{q_1}'\rangle}{\langle m, e_{q_1} \oplus_a e_{q_2}\rangle \to_q \langle m, e_{q_1}' \oplus_a e_{q_2}\rangle}\ \text{q-aop1}$$

$$\frac{\langle m, e_{q_2}\rangle \to_a \langle m, e_{q_2}'\rangle}{\langle m, v_{q_1} \oplus_a e_{q_2}\rangle \to_q \langle m, v_{q_1} \oplus_a e_{q_2}'\rangle}\ \text{q-aop2}$$

Figure 13.2: Operational semantics of loop language, expression evaluation

$$\boxed{\langle m, c, t, w\rangle \to \langle m', c', t', w'\rangle}$$

$$\frac{\langle m, e\rangle \to \langle m, e'\rangle}{\langle m, [x \leftarrow e]^l, t, w\rangle \to \langle m, [x \leftarrow e']^l, t, w\rangle}\ \textbf{l-assn1}$$

$$\frac{}{\langle m, [x \leftarrow v]^l, t, w\rangle \to \langle m[v/x], [\texttt{skip}]^l, t, w\rangle}\ \textbf{l-assn2}$$

$$\frac{\langle m, a\rangle \to_a \langle m, a'\rangle}{\langle m, \texttt{loop}\ [a]^l\ \texttt{do}\ c, t, w\rangle \to \langle m, \texttt{loop}\ [a']^l\ \texttt{do}\ c, t, (w + l)\rangle}\ \textbf{l-loop-a}$$

$$\frac{v_N > 0}{\langle m, \texttt{loop}\ [v_N]^l\ \texttt{do}\ c, t, w\rangle \to \langle m, c; \texttt{loop}\ [(v_N - 1)]^l\ \texttt{do}\ c, t, (w + l)\rangle}\ \textbf{l-loop}$$

$$\frac{v_N = 0}{\langle m, \texttt{loop}\ [v_N]^l\ \texttt{do}\ c, t, w\rangle \to \langle m, [\texttt{skip}]^l, t, (w \setminus l)\rangle}\ \textbf{l-loop-exit}$$

$$\frac{\langle m, e_q\rangle \to_q \langle m, e'_q\rangle}{\langle m, [x \leftarrow q(e_q)]^l, t, w\rangle \to \langle m, [x \leftarrow q(e'_q)]^l, t, w\rangle}\ \textbf{l-query-e}$$

$$\frac{q(v_q) = v}{\langle m, [x \leftarrow q(v_q)]^l, t, w\rangle \to \langle m[v/x], \texttt{skip}, t +\!\!+ [q(v_q)^{(l,w)}], w\rangle}\ \textbf{l-query-v}$$

$$\frac{\langle m, c_1, t, w\rangle \to \langle m', c'_1, t', w'\rangle}{\langle m, c_1; c_2, t, w\rangle \to \langle m', c'_1; c_2, t', w'\rangle}\ \textbf{l-seq1} \qquad \frac{}{\langle m, [\texttt{skip}]^l; c_2, t, w\rangle \to \langle m, c_2, t, w\rangle}\ \textbf{l-seq2}$$

$$\frac{\langle m, b\rangle \to_b b'}{\langle m, \text{if } ([b]^l, c_1, c_2), t, w\rangle \to \langle m, \text{if } ([b']^l, c_1, c_2), t, w\rangle}\ \textbf{l-if}$$

$$\frac{}{\langle m, \text{if } ([\text{true}]^l, c_1, c_2), t, w\rangle \to \langle m, c_1, t, w\rangle}\ \textbf{l-if-t}$$

$$\frac{}{\langle m, \text{if } ([\text{false}]^l, c_1, c_2), t, w\rangle \to \langle m, c_2, t, w\rangle}\ \textbf{l-if-f}$$

Figure 13.3: Trace-based operational semantics of loop language.

swers to previous queries and the state of the program. It means that when we want to know if this query is adaptively chosen, we only need to check whether the choice of this query will be affected by changes of answers to previous queries. There are two possible situations that can affect the choice of a query, either the query argument directly uses the results of previous queries (data dependency), or the control flow of the program with respect to a query (whether to ask this query or not) depends on the results of previous queries (control flow dependency).

As a first step, we need to first think about when one query may depend on a previous query, which is supposed to consider both control dependency and data dependency. We first look at two possible candidates:

1. One query may depend on a previous query if and only if a change of the answer to the previous query may also change the result of the query.

2. One query may depend on a previous query if and only if a change of the answer to the previous query may also change the choice of this query by the analyst.

The first candidate works well by witnessing the result of one query according to the change of the answer of another query. We can easily find that the two queries have nothing to do with each other in a simple example $p = x \leftarrow q(\chi[1]); y \leftarrow q(\chi[2])$. This candidate definition works well with respect to data dependency. However, it fails to handle control dependency. The key point is that this query may also not be asked according to the answers of previous queries. An example of this situation is shown in program p_1 as follows.

$$p_1 = x \leftarrow q(\chi[1]); \text{if } (x > 2, y \leftarrow q(\chi[2]), \texttt{skip})$$

We choose the second candidate, which performs well by witnessing the appearance of

the query $q(\chi[2])$ upon the change of the result of the previous query $q(\chi[1])$ in p_1. It considers the control dependency, and at the same, does not miss the data dependency. In particular, the arguments of a query characterize it. In this sense, if the data used in the arguments changes due to a different answer to a certain previous query, the appearance of the query may change as well. This situation is also captured by our definition. Let us look at another program p_2, in which the queries are equipped with functions using previously assigned variables storing the answer of its previous query.

$$p_2 = x \leftarrow q(\chi[2]); y \leftarrow q(x + \chi[3])$$

As a reminder, in the loop language, the query request is composed of two components: a symbol q representing a linear query type and the argument e_q, a query expression that represents the function specifying what the query asks. When we think two queries are not the same, we use the equality relation between the query values (the normal form of the argument) of the two queries, denoted as $=_{vq}$. This allows us to compare two queries. For example, we think query $q(\chi[1] + \chi[2])$ is the same query as $q(\chi[2] + \chi[1])$, the query $q(\chi[1] + 3)$ is the same as $q(3 + \chi[1])$. See details in Section C.1.

Informally, we think $q(x + \chi[3])$, whose normal form is $q(v + \chi[3])$ assuming v is the result of $q(\chi[2])$ associated with variable x such that $v = q(\chi[2])(D)$, may depend on the query $q(\chi[2])$. Because the normal form of the query $q(x + \chi(3))$ asked by the analyst of depend on the value of x, which is associated with the return value of $q(\chi(2))$. Suppose the value of x changes from 1 to 3, we do not think $q(1 + \chi[3])$ is the same query as $q(3 + \chi[3])$ even though both of them are evaluated from $q(x + \chi[3])$.

We give a formal definition of query may dependency based on the trace-based operational semantics as follows. The notations $\in_q$ and $\notin_q$ are defined in Section C.1.

Definition 1 (Query may dependency)

One query $q(v_{q_1})$ may depend on another query $q(v_{q_2})$ in a program c, with a starting loop maps w, a starting memory m, hidden database D, denoted as $\mathsf{DEP}(q(v_{q_1})^{(l_1,w_1)}, q(v_{q_2})^{(l_2,w_2)}, c, w, m, D)$ is defined below.

$$
\begin{array}{l}
\forall t.\exists m_1, m_3, t_1, t_3, c_2. \\
\left(\begin{array}{l}
\langle m, c, t, w\rangle \rightarrow^* \langle m_1, [x \leftarrow q(v_{q_1})]^{l_1}; c_2, t_1, w_1\rangle \rightarrow \\
\langle m_1[q(v_{q_1})(D)/x], c_2, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle m_3, \mathtt{skip}, t_3, w_3\rangle \\
\wedge \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \wedge q(v_{q_2})^{(l_2,w_2)} \in_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), m_3', t_3', w_3'. \\
\langle m_1[v/x], c_2, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle m_3', \mathtt{skip}, t_3', w_3'\rangle \\
\wedge (q(v_{q_2})^{(l_2,w_2)}) \notin_q (t_3' - t_1)\Big) \\
\wedge \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \wedge q(v_{q_2})^{(l_2,w_2)} \notin_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), m_3', t_3', w_3'. \\
\langle m_1[v/x], c_2, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle m_3', \mathtt{skip}, t_3', w_3'\rangle \\
\wedge (q(v_{q_2})^{(l_2,w_2)}) \in_q (t_3' - t_1)\Big)
\end{array}\right)
\end{array}
$$

We give a formal definition of the query-based dependency graph with the formal definition of may dependency between the two queries above.

Definition 2 (Query-based Dependency Graph)

Given a program c, a database D, a starting memory m, an initial loop map w, the query-based dependency graph $G(c, D, m, w) = (V, E)$ is defined as:

$V = \{q(v_q)^{l,w} \in \mathscr{AQ} \mid \forall t.\exists m', w', t'.\langle m, c, t, w\rangle \rightarrow^* \langle m', \mathtt{skip}, t', w'\rangle \wedge q(v_q)^{l,w} \in (t' - t)\}$.

$E = \left\{(q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')}) \in \mathscr{AQ} \times \mathscr{AQ} \;\middle|\; \mathsf{DEP}(q(v_q{}')^{(l',w')}, q(v_q)^{(l,w)}, c, w, m, D)\right\}$.

The edge is directed, when an annotated query $q(v_q)^{(l,w)}$ may depend on its previous

query $q(v_q{}')^{(l',w')}$, we have the directed edge $(q(v_q)^{(l,w)}, q(v_q{}')^{(l'.w')})$, from $q(v_q)^{(l,w)}$ to $q(v_q{}')^{(l'.w')}$.

The query-based dependency graph only considers the newly generated annotated queries during the execution of the program c, so we see the nodes coming from the trace $t' - t$. The previous trace before the execution of c is excluded when constructing the graph. To summarize, for every execution of a program c staring with different configurations, we can construct a corresponding dependency graph.

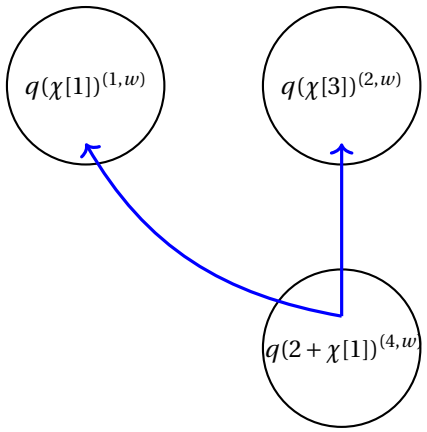
I will show an example showing how our definition works and how we build the graph for the following program p_3. We use if b then c else c as another form of if (b, c, c) to better present the example.

$$
\begin{aligned}
& \left[x \leftarrow q[\chi[2]]\right]^1; \\
& \left[y \leftarrow q(\chi[3])\right]^2; \\
p_3 = \ & \text{if } [(x > 2)]^3 \\
& \text{then } \left[x \leftarrow q(x + \chi[1])\right]^4 \\
& \text{else } \left[\texttt{skip}\right]^5
\end{aligned}
$$

In our definition, we assume that the same query $q(\chi[1])$ sent to the same database D will always get the same result.

Now, let us assume we have an initial memory m, which is a trivial one $m = [(x : 0), (y : 0), (z : 0)]$. We also assume a hidden database D, such that we have: $q(\chi[2])(D) = 2$, $q(\chi[3])(D) = 3$. We have the empty loop map w (w to be empty here), start with the empty trace t. We first evaluate the program p_3 with our operational semantics: $\langle m, p_3, t, w\rangle \rightarrow \langle m', \texttt{skip}, t', w'\rangle$. We know that the newly generated trace $t' - t$ will have three elements: $[q(\chi[2])^{(1,w)}, q(\chi[3])^{(2,w)}, q(2 + \chi[1])^{(4,w)}]$. So these three query requests consist of nodes of our dependency graph. Then we need to check the may-dependency

relation between them. From the definition, we know that $q(\chi[2])^{(1,w)}$, $q(\chi[3])^{(2,w)})$ are independent. We have $\mathsf{DEP}(q(\chi[2])^{(1,w)}, q(2+\chi[1])^{(4,w)}, p_3, w, m, D)$ because when we choose another value for $q(\chi[2])(D)$ in all the possible values of this query, say 3, then the resulting trace will change to $q(\chi[2])^{(1,w)}, q(\chi[3])^{(2,w)}, q(3+\chi[1])^{(4,w)}]$. Also, we know $q(2+\chi[1])^{(4,w)}$ may depend on the query $q(\chi[3])^{(2,w)}$ because we know it is true that $\mathsf{DEP}(q(\chi[3])^{(2,w)}, q(2+\chi[1])^{(4,w)}, p_3, w, m, D)$. This time, we can find another value of $q(\chi[3])(D) = 1$, which results in a different resulting trace $q(\chi[2])^{(1,w)}, q(\chi[3])^{(2,w)}]$ with only two nodes. Of course, we need to find out the codomain for these query requests in our system. Then we have the following dependency graph between query requests.



Finally, we reach the definition of adaptivity, by means of the query-based dependency graph.

Definition 3 (Adaptivity in loop language)

Given a program c, and a memory m, a database D, a starting loop map w, the adaptivity of the dependency graph $G(c, D, m, w) = (V, E)$ is the length of the longest path in this graph. We denote the path from $q(v_q)^{(l,w)}$ to $q(v_q{}')^{(l',w')}$ as $p(q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')})$.

The adaptivity denoted as $A(c, D, m, w)$.

$$A(c, D, m, w) = \max_{q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')} \in V} |p(q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')})|$$

CHAPTER 14

The SSA Loop Language

This chapter contains the SSA version of the loop language we use to express the data analysis algorithm. Also, the necessity of the transformation from the loop language to the SSA language is presented.

14.1 THE LIMITATIONS OF LOOP LANGUAGE FOR STATIC ANALYSIS

The labeled loop language supports the notion of adaptivity semantically, through a query-based dependency graph. However, syntactically, it is not so suitable for program analysis. The reason is that it allows variables to be reassigned, making the decision on where used variables come from tricky, especially there are controlled branches. We use three examples in loop language to show the dilemma, assuming q_1, q_2, q_3 are three linear queries.

$$
s_1 = \begin{array}{l} [x \leftarrow q_1]^1; \\ \text{if } [(x<0)]^2 \\ \text{then } [x \leftarrow q_2]^3 \\ \text{else } [\texttt{skip}]^4; \\ [y \leftarrow q(x+\chi[3])]^5 \end{array}
\qquad
s_2 = \begin{array}{l} [x \leftarrow q_1]^1; \\ \text{if } [(x<0)]^2 \\ \text{then } [x \leftarrow q_2]^3 \\ \text{else } [x \leftarrow q_3]^4; \\ [y \leftarrow q(x+\chi[3])]^5 \end{array}
\qquad
s_3 = \begin{array}{l} [x \leftarrow q_1]^1; \\ \text{if } [(x<0)]^2 \\ \text{then } [z \leftarrow q_2]^3 \\ \text{else } [\texttt{skip}]^4; \\ [y \leftarrow q(x+\chi[3]]^5 \end{array}
$$

In these three examples, the variable x used in the query $q(x+\chi[3])$ at line 5 is implicit, when we statically analyze the statement $[y \leftarrow q(x+\chi[3])]^5$. In program s_1, it refers to the either x at line 1, or x at line 3. When we have a look at the other two programs s_2 and s_3, the query $q(x+\chi[3])$ may depend on either q_2(x at line 3) or q_3(x at line 4) in s_2, while it only depends on q_1 at line 1 in program s_3. These structural similar three examples, however, have quite dissimilar dependencies between variables. It increases

the challenge to track dependency in static analysis. Still look at the analysis on the statement $\left[y \leftarrow q(x+\chi[3])\right]^5$, extra information is needed such as value of x used in the statement may come from the result of q_1 or the answer to q_2 when this statement lies in s_1. Similarly, when in program s_2, the extra information that the value of x used in the same statement relies on answers to queries q_2 or q_3 in both branches of the if statement starting from line 2 is necessary for static analysis on dependency. Additionally, in program s_2, we also need to update the information that x assigned at line 1 is overwritten by both branches when analyzing the statement at line 5.

To simplify the program analysis, we choose to conduct the static analysis on the SSA form of our target programs.

$$
s_1^s = \begin{array}{l} \left[\mathbf{x_1} \leftarrow q_1\right]^1; \\ \text{if } [(\mathbf{x_1} < 0)]^2 \\ ([], [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}], []) \\ \text{ then } \left[\mathbf{x_2} \leftarrow q_2\right]^3 \\ \text{ else } \left[\texttt{skip}\right]^4; \\ \left[\mathbf{y_1} \leftarrow q(\mathbf{x_3} + \chi[3])\right]^5 \end{array}
\quad
s_2^s = \begin{array}{l} \left[\mathbf{x_1} \leftarrow q_1\right]^1; \\ \text{if } [(\mathbf{x_1} < 0)]^2 \\ ([\mathbf{x_4}, \mathbf{x_2}, \mathbf{x_3}], [], []) \\ \text{ then } \left[\mathbf{x_2} \leftarrow q_2\right]^3 \\ \text{ else } \left[\mathbf{x_3} \leftarrow q_3\right]^4; \\ \left[\mathbf{y_1} \leftarrow q(\mathbf{x_4} + \chi[3])\right]^5 \end{array}
\quad
s_3^s = \begin{array}{l} \left[\mathbf{x_1} \leftarrow q_1\right]^1; \\ \text{if } [(\mathbf{x_1} < 0)]^2 \\ ([], [], []) \\ \text{ then } \left[\mathbf{z_1} \leftarrow q_2\right]^3 \\ \text{ else } \left[\texttt{skip}\right]^4; \\ \left[\mathbf{y_1} \leftarrow q(\mathbf{x_1} + \chi[3])\right]^5 \end{array}
$$

To distinguish between the loop language and in SSA form, we denote the SSA variable $\mathbf{x}$ in bold. As we can see, the reachability of assigned variables becomes explicit in the SSA form. In the SSA version s_1^s of s_1, still looking at the statement at line 5, which becomes $\left[\mathbf{y_1} \leftarrow q(\mathbf{x_1} + \chi[3])\right]^5$, we can syntactically figure out that the query may depend on the variable $\mathbf{x_3}$, which may come from $\mathbf{x_1}$ or $\mathbf{x_2}$, without extra information like in s_1. This benefit also applies to the analysis over the same statement at line 5 in s_2^s and s_3^s.

14.2 SYNTAX OF THE SSA LOOP LANGUAGE

We present the syntax of the SSA loop language, a language based on the loop language, representing programs in the static single assignment form.

The expressions inherit from the loop language, except that the SSA arithmetic expressions **a** now contain SSA variable $\mathbf{x} \in \mathscr{SV}$. The boolean expressions in the SSA loop language are denoted as **b**. In the language, variables can also be annotated, denoted as $\mathscr{LV}$, in a similar way as the annotated queries in the loop language. For instance, $\mathbf{x}^{(l,w)} \in \mathscr{LV}$. The SSA memory now is a map from SSA variables to values.

The SSA labeled command **c** inherits from the loop language, except that the expressions and variables in these commands are now in its SSA version as shown below.

$$\begin{aligned}\mathbf{c} \quad ::= \quad & [\mathbf{x} \leftarrow \mathbf{e}]^l \mid [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l \mid ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}') \mid [\texttt{skip}]^l \mid \texttt{loop}\ [\mathbf{a}]^l, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c} \mid \\ & \mathbf{c}; \mathbf{c} \mid \text{if } ([\mathbf{b}]^l, ([\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]), \mathbf{c}, \mathbf{c})\end{aligned}$$

The if command now contains the extra part $([\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}])$, which helps to track the dependency of new assigned variables in both branches$([\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}])$, then branch $[\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]$, and else branch $[\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]$. The $\bar{\mathbf{x}}$ is a list of SSA variables, in which every element **x** may depend on the corresponding element(at same location), $\mathbf{x_1}$ from $\bar{\mathbf{x_1}}$ collected in the then branch or the corresponding element $\mathbf{x_2}$ from $\bar{\mathbf{x_2}}$ collected in the else branch. The size of these three lists are required to be the same.

Every tuple $(\mathbf{x}, \mathbf{x_1}, \mathbf{x_2})$ from $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$ can be understood as $\mathbf{x} = \phi(\mathbf{x_1}, \mathbf{x_2})$ in the normal SSA form. The previous example s_2^s can be used for reference. The second part $[\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]$ focuses on the then branch. The list of SSA variables $\bar{\mathbf{y_1}}$ stores the assigned SSA variables before the if statement, whose non-SSA version (variables in the loop language) will be modified only in the then branch. We can look at program s_1 as a reference, in which x at line 1 may be modified only in the then branch at line 3. The list $\bar{\mathbf{y_2}}$ tracks the SSA

variables assigned only in the then branch. If the variables are assigned in both branches such as in the program s_2, they go into $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$. Then we think every SSA variable in $\bar{\mathbf{y}}$ may come from the corresponding variable $\mathbf{y_1}$ in $\bar{\mathbf{y_1}}$ before the if command or $\mathbf{y_2}$ in $\bar{\mathbf{y_2}}$ in the then branch. In this sense, we can also regard every tuple $(\mathbf{y}, \mathbf{y_1}, \mathbf{y_2})$ from $[\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]$ as $\mathbf{y} = \phi(\mathbf{y_1}, \mathbf{y_2})$. The rest part $[\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]$ focus on the else branch and can be understood similarly.

To better understand these $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$, $[\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]$, $[\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]$ in the SSA if statement. Let us look at our previous example s_1, s_2, s_3. We can see in s_1, variable x is only assigned in the then branch at line 3. So what is the value of variable x at line 5 comes from, it is not clear. It may comes from line 1 or line 3. In its SSA version, we want to eliminate this unclear source of x. So If we look at s_1^s, $[\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]$ is instantiated with $[\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]$. In this sense, we claim that $\mathbf{x_3}$ used at line 5 will either comes from $\mathbf{x_1}$ at line 1 or $\mathbf{x_2}$ at line 3. These $\mathbf{y}$s are specific for those variables that are assigned before the if statement, and only reassigned in the then branch. Similarly, $[\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]$ are for those variables that are assigned before the if statement, and only reassigned in the else branch. $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$ will be used for variables assigned in both branches, see s_2.

Also, the loop command also has similar part $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$, focusing on the loop body. The new command $ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$ does not have explicit label because it is only used for evaluation internally, we will discuss more about it when used in the operational semantics for the SSA loop language.

14.3 TRACE-BASED OPERATIONAL SEMANTICS OF THE SSA LOOP LANGUAGE

When switching to the SSA loop language, we show that we are still able to achieve what we can get in the loop language (Chapter 13). The operational semantics of the SSA loop language mimics its counterpart, of the form $\langle \mathbf{m}, \mathbf{c}, t, w \rangle \rightarrow \langle \mathbf{m}', \texttt{skip}, t', w' \rangle$. The SSA

memory $\mathbf{m}$ is a map from SSA variable $\mathbf{x}$ to values. It still uses a trace to track the query requests during the execution, starting from an SSA configuration with SSA memory $\mathbf{m}$ and program in its SSA form $\mathbf{c}$, which allows a similar construction of the query-based dependency graph in the SSA language as in the loop language.

We show selected evaluation rules in Figure 14.1. The details can be found in Section C.2.2. The key idea underneath the operational semantics is to have the trace and the execution path being constructed in a similar way as in the loop language.

Take the query request as an example, the argument $\mathbf{e_q}$ which may contain SSA variables will be evaluated to a value v_q first before the request is sent to the mechanism in rule **SSA-query-arg**. The trace expands in the rule **SSA-query** likewise in the loop language. The query q, a primitive symbol representing the query , makes no difference in the two languages.

Since we add the extra part $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]$ in the if statement compared to its counterpart in the loop language introduced before, the rules relevant to the if conditional (**SSA-if-t** and **SSA-if-f**) use the extra command $ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$ to update the SSA memory $\mathbf{m}$ with the the mapping from all the new generated variable $\mathbf{x}$ in the list $\bar{\mathbf{x}}$ to the appropriate value $\mathbf{m}(\mathbf{x}')$. The SSA variable $\mathbf{x}'$ is the corresponding variable with respect to $\mathbf{x}$ in $\bar{\mathbf{x}}'$. There is a one-on-one correspondence between the two SSA variable lists $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$, based on the position in the list, which requires the two lists of the same length. The rule **SSA-ifvar** reflects the usage of $ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$. It is easier to understand the usage of $ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$ in the rule **SSA-if-t** when we think about how SSA works: in the SSA form, when a variable to be used may come from two sources (e.g. $\mathbf{x_1}$ and $\mathbf{x_2}$ in the rule), it generates a new SSA variable $\mathbf{x}$, assigning it with $\phi(\mathbf{x_1}, \mathbf{x_2})$, and replaces the variable to be used with this newly assigned $\mathbf{x}$. We know that in the future program after this if statement, only the variables appeared in $\bar{\mathbf{x}}$ will be available, instead of $\bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}$ from

$$\boxed{\langle \mathbf{m}, \mathbf{c}, t, w\rangle \rightarrow \langle \mathbf{m}', \mathbf{c}', t', w'\rangle}$$

$$\frac{q(v_q) = v}{\langle \mathbf{m}, [\mathbf{x} \leftarrow q(v_q)]^l, t, w\rangle \rightarrow \langle \mathbf{m}[v/\mathbf{x}], \texttt{skip}, t +\!\!+ [q(v_q)^{(l,w)}], w\rangle}\ \textbf{SSA-query}$$

$$\frac{}{\langle \mathbf{m}, ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}'), t, w\rangle \rightarrow \langle \mathbf{m}[\mathbf{m}(\bar{\mathbf{x}}')/\bar{\mathbf{x}}], \texttt{skip}, \mathbf{t}, \mathbf{w}\rangle}\ \textbf{SSA-ifvar}$$

$$\frac{\langle \mathbf{m}, \mathbf{e_q}\rangle \rightarrow \langle m, \mathbf{e}'_\mathbf{q}\rangle}{\langle \mathbf{m}, [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l, t, w\rangle \rightarrow \langle \mathbf{m}, [\mathbf{x} \leftarrow q(\mathbf{e}'_\mathbf{q})]^l, t, w\rangle}\ \textbf{SSA-query-arg}$$

$$\frac{}{\begin{array}{c}\langle \mathbf{m}, \text{if } ([\text{true}]^\mathbf{l}, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}], [\bar{\mathbf{y}}, \bar{\mathbf{y}_1}, \bar{\mathbf{y}_2}], [\bar{\mathbf{z}}, \bar{\mathbf{z}_1}, \bar{\mathbf{z}_2}], \mathbf{c_1}, \mathbf{c_2}), t, w\rangle \rightarrow \\ \langle \mathbf{m}, \mathbf{c_1};\ ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}_1}); ifvar(\bar{\mathbf{y}}, \bar{\mathbf{y}_2}); ifvar(\bar{\mathbf{z}}, \bar{\mathbf{z}_1}), t, w\rangle\end{array}}\ \textbf{SSA-if-t}$$

$$\frac{}{\begin{array}{c}\langle \mathbf{m}, \text{if } ([\text{false}]^\mathbf{l}, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}], [\bar{\mathbf{y}}, \bar{\mathbf{y}_1}, \bar{\mathbf{y}_2}], [\bar{\mathbf{z}}, \bar{\mathbf{z}_1}, \bar{\mathbf{z}_2}], \mathbf{c_1}, \mathbf{c_2}), t, w\rangle \rightarrow \\ \langle \mathbf{m}, \mathbf{c_2}; ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}_2}); ifvar(\bar{\mathbf{y}}, \bar{\mathbf{y}_1}); ifvar(\bar{\mathbf{z}}, \bar{\mathbf{z}_2}), t, w\rangle\end{array}}\ \textbf{SSA-if-f}$$

$$\frac{v_N > 0 \qquad n = 0 \implies i = 1 \qquad n > 0 \implies i = 2}{\begin{array}{c}\langle \mathbf{m}, \texttt{loop } [v_N]^l, n, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}] \texttt{ do } \mathbf{c}, t, w\rangle \rightarrow \\ \langle \mathbf{m}, \mathbf{c}[\bar{\mathbf{x}_i}/\bar{\mathbf{x}}]; \texttt{loop } [(v_N - 1)]^l, n+1, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}] \texttt{ do } \mathbf{c}, t, (w + l)\rangle\end{array}}\ \textbf{SSA-loop}$$

$$\frac{v_N = 0 \qquad n = 0 \implies i = 1 \qquad n > 0 \implies i = 2}{\langle \mathbf{m}, \texttt{loop } [v_N]^l, n, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}] \texttt{ do } \mathbf{c}, t, w\rangle \rightarrow \langle \mathbf{m}[\mathbf{m}(\bar{\mathbf{x}_i})/\bar{\mathbf{x}}], [\texttt{skip}]^l, t, (w \setminus l)\rangle}\ \textbf{SSA-loop-exit}$$

Figure 14.1: Operational semantics for the ssa loop language.

two branches. Let us look at the example s_2, when we transform s_2 to its SSA version s_2^s, what will the variable x at line 5 in s_2 be replaced with in s_2^s? It is $\mathbf{x_4}$. However, if we look at the evaluation rule of SSA if. Let us choose **SSA-if-t**, after the then branch is executed, in the resulting memory, there is no mapping from $\mathbf{x_4}$ to some value. Instead, there is only a mapping from $\mathbf{x_2}$ to some value. However, in s_2^s, at line 5, we have $\mathbf{x_4}$ instead of $\mathbf{x_2}$. This is the reason we use $ifvar(\mathbf{x_4},\mathbf{x_2})$ in the evaluation rule to modify the resulting memory. For the evaluation of the program after this if statement, we need to tell the memory the exact value of the newly generated variable $\mathbf{x}$, which is the value stored in $\mathbf{x_1}$ when the conditional predicate $\mathbf{b}$ is true, or the value in $\mathbf{x_2}$ when $\mathbf{b}$ is false. To this end, the internal command $ifvar(\bar{\mathbf{x}},\bar{\mathbf{x}}')$ plays its role. For the if rule, we need to instantiate those variables from $\bar{\mathbf{x}}$ whose values come from two branches, $\bar{\mathbf{y}}$ whose values from then branch or assignment before the if command, and $\bar{\mathbf{z}}$ whose values from else branch or before the if command. Correspondingly, we need to have three extra $ifvar$ commands.

The evaluation of loop depends on the loop counter $\mathbf{a}$ in the rule **SSA-loop**, which will be evaluated to a value v_N. When v_N is greater than 0, the loop is still executing, and all the variables $\mathbf{x}$ in $\bar{\mathbf{x}}$ of the loop body $\mathbf{c}$ are replaced as the corresponding variables in $\bar{\mathbf{x_1}}$ in the first iteration($n = 0$), or $\bar{\mathbf{x_2}}$ in other iterations($n > 0$). The loop turns to an exit described in the rule **SSA-loop-exist** when $v_N > 0$, and the memory $\mathbf{m}$ updates the mapping of variables in $\bar{\mathbf{x}}$ with $\bar{\mathbf{x_1}}$ if the iteration counter goes to zero($n = 0$), which means the loop body is not executed once. When the loop enters the exit after executing the body a few times(n), the variables in $\bar{\mathbf{x}}$ is instantiated with the value from the body $\mathbf{m}(\bar{\mathbf{x_2}})$.

The trace-based operational semantics of the SSA loop language allow us to provide our query-based dependency graph in the SSA version.

Definition 4 (Query may dependency in SSA)

One query $q(v_{q_2})$ *may depend* on another query $q(v_{q_1})$ in a program $\mathbf{c}$, with a starting loop maps w, a starting memory $\mathbf{m}$, hidden database D, denoted as
$\mathsf{DEP}_{\mathsf{ssa}}(q(v_{q_1})^{(l_1,w_1)}, q(v_{q_2})^{(l_2,w_2)}, \mathbf{c}, w, \mathbf{m}, D)$.

$$
\begin{array}{l}
\forall t.\exists \mathbf{m_1}, \mathbf{m_3}, t_1, t_3, c_2. \\
\left(\begin{array}{l}
\langle \mathbf{m}, c, t, w\rangle \rightarrow^* \langle \mathbf{m_1}, [\mathbf{x} \leftarrow q(v_{q_1})]^{l_1}; \mathbf{c_2}, t_1, w_1\rangle \rightarrow \\
\langle \mathbf{m_1}[q(v_{q_1})(D)/\mathbf{x}], \mathbf{c_2}, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle \mathbf{m_3}, \mathtt{skip}, t_3, w_3\rangle \\
\wedge \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \wedge q(v_{q_2})^{(l_2,w_2)} \in_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), \mathbf{m'_3}, t'_3, w'_3. \\
\langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle \mathbf{m'_3}, \mathtt{skip}, t'_3, w'_3\rangle \\
\wedge (q(v_{q_2})^{(l_2,w_2)}) \notin_q (t'_3 - t_1) \Big) \\
\wedge \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \wedge q(v_{q_2})^{(l_2,w_2)} \notin_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), \mathbf{m'_3}, t'_3, w'_3. \\
\langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle \mathbf{m'_3}, \mathtt{skip}, t'_3, w'_3\rangle \\
\wedge (q(v_{q_2})^{(l_2,w_2)}) \in_q (t'_3 - t_1) \Big)
\end{array} \right)
\end{array}
$$

Definition 5 (Dependency Graph in SSA)

.

Given a program $\mathbf{c}$, a database D, a starting memory $\mathbf{m}$, an initial loop maps w, the dependency graph $G_s(\mathbf{c}, D, \mathbf{m}, w) = (V, E)$ is defined as:
$V = \{q(v_q)^{l,w} \in \mathscr{AQ} \mid \forall t.\exists \mathbf{m}', w', t'.\langle \mathbf{m}, \mathbf{c}, t, w\rangle \rightarrow^* \langle \mathbf{m}', \mathtt{skip}, t', w'\rangle \wedge q(v_q)^{l,w} \in (t' - t)\}$.
$E = \left\{ (q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')}) \in \mathscr{AQ} \times \mathscr{AQ} \;\middle|\; \mathsf{DEP}_{\mathsf{ssa}}(q(v_q{}')^{(l',w')}, q(v_q)^{(l,w)}, \mathbf{c}, w, \mathbf{m}, D) \right\}$.

Definition 6 (Adaptivity in SSA)

Given a program $\mathbf{c}$, and a meory $\mathbf{m}$, a database D, a starting loop maps w, the adaptivity of the dependency graph $G_s(\mathbf{c}, D, \mathbf{m}, w) = (V, E)$ is the length of the longest path in this

graph. We denote the path from $q(v_q)^{(l,w)}$ to $q(v_q{}')^{(l',w')}$ as $p_s(q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')})$. The adaptivity denoted as $A_s(\mathbf{c}, D, \mathbf{m}, w)$.

$$A_s(\mathbf{c}, D, \mathbf{m}, w) = \max_{q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')} \in V} |p_s(q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')})|$$

14.4 TRANSFORMATION

We build a bridge between the two languages through a transformation in spirit of the work Vekris et al. (2016a). The command transformation of the form $\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'$ translates the labelled command c in the loop language to its counterpart in SSA loop language. The SSA name environment Σ, a set of ssa variables already used before the transformation process, is used to generate a fresh SSA variable via a function $fresh(\Sigma)$. Additionally, translating variables read in the program in the loop language to its unique SSA variable requires a translation environment δ, a map from variable $x \in \mathcal{VAR}$ to its SSA form $\mathbf{x} \in \mathcal{SV}$. Also, the translation environment δ and the SSA name environment Σ will be updated to δ' and Σ' respectively, along the transformation of the target command. The transformation of the expression is much simpler, of the form $\delta; e \hookrightarrow \mathbf{e}$, which transforms the variables in e to ssa variables stored in the translation environment δ, shown in the rule **s-var**.

We present selected transformation rules in Figure 14.2. The rules **s-assn** as well as **s-query** both use $fresh(\Sigma)$ to generate a new fresh SSA variable $\mathbf{x}$ to guarantee the unique assignment of SSA variables. The translation environment is updated with the mapping from variable x in loop language to the new generated SSA variable $\mathbf{x}$ for reference to x used in the future. The SSA name environment is also modified by recording $\mathbf{x}$. The transformation of the sequence is standard, with both environments δ and Σ updated during the transformation procedure.

We look at the rule **s-if** by first introducing the binary operation $\bowtie\!\triangleleft$ on two translation environments δ_1 and δ_2.

$$\delta_1 \bowtie\!\triangleleft \delta_2 = \{(x, \mathbf{x_1}, \mathbf{x_2}) \in \mathcal{VAR} \times \mathcal{SV} \times \mathcal{SV} \mid x \mapsto \mathbf{x_1} \in \delta_1, x \mapsto \mathbf{x_2} \in \delta_2, \mathbf{x_1} \neq \mathbf{x_2}\}$$

$$\delta_1 \bowtie\!\triangleleft \delta_2 / \bar{x} = \{(x, \mathbf{x_1}, \mathbf{x_2}) \in \mathcal{VAR} \times \mathcal{SV} \times \mathcal{SV} \mid x \notin \bar{x} \wedge x \mapsto \mathbf{x_1} \in \delta_1, x \mapsto \mathbf{x_2} \in \delta_2, \mathbf{x_1} \neq \mathbf{x_2}\}$$

This operation $\delta_1 \bowtie\!\triangleleft \delta_2$ combines two translation environments by only keeping the mappings of the same key in both environments. It returns a set of tuples with three elements $(x, \mathbf{x_1}, \mathbf{x_2})$ to show that a variable x in the loop language may be translated to either $\mathbf{x_1}$ or $\mathbf{x_2}$, depending on the control flow. We use $\bar{x}$ to represent a list of variables x, in this sense, the results of $\delta_1 \bowtie\!\triangleleft \delta_2$ is denoted as $[\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$ as follows.

$$[\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \{(x, x_1, x_2) | \forall 0 \leq i < |\bar{x}|, x = \bar{x}[i] \wedge x_1 = \bar{x_1}[i] \wedge x_2 = \bar{x_2}[i] \wedge |\bar{x}| = |\bar{x_1}| = |\bar{x_2}|\}$$

In the rule **s-if**, a variable x in loop language may be translated to two possible SSA variables in three cases: (1) the variable x is assigned in both two branches, whose mapping of x is stored in δ_1(then branch) and δ_2(else branch). (2) the variable x is assigned before the if statement (in δ) and only assigned in the then branch δ_1 (3) the variable x is assigned before the if statement (in δ) and only assigned in the else branch δ_2. This corresponds to the aforementioned discussion of the if statement in SSA loop language. We leave these mappings explicitly in the if command of the SSA loop language syntactically. We also use the variant of $\delta \bowtie\!\triangleleft \delta_1$, $\delta \bowtie\!\triangleleft \delta_1 / \bar{x}$ to guarantee that the variables stored in $[\bar{y}, \bar{\mathbf{y}_1}, \bar{\mathbf{y}_2}]$ only appear in the then branch, not in the else branch. Similarly for $[\bar{z}, \bar{\mathbf{z}_1}, \bar{\mathbf{z}_2}]$. After the transformation, the variable in $\bar{x}, \bar{y}, \bar{z}$ is replaced with the fresh SSA variables stored in $\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}$ and the translation environment is updated accordingly.

The loop transformation rule **s-loop** deserves a further discussion. Besides the nor-

mal transformation of the loop counter a to $\mathbf{a}$, an additional iteration counter is added to its SSA form to support the evaluation, as we have seen in the rule **SSA-loop** in Figure 14.1, and is set to 0. For the variables assigned in the loop body c, we leave $[\bar{\mathbf{x}'}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]$ in the transformed SSA loop command, which tracks variables in the loop body whose value may come from two sources: assignment before the loop(δ) or assignment in the loop body δ_1. We have two transformations on the body c using the same SSA name environment Σ but different translation environments δ and δ_1. The premise $\Sigma;\delta;c \hookrightarrow \mathbf{c_1};\delta_1;\Sigma_1$ corresponds to the transformation of the loop body in the first iteration with the variables assigned before the loop execution. The second premise $\Sigma;\delta;c \hookrightarrow \mathbf{c_2};\delta_1;\Sigma_1$ corresponds to the transformation in the later iteration with the assigned variables updated by previous execution of the body. Thanks to the extra part $[\bar{\mathbf{x}'}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]$ in the SSA loop command, we know that those variables used in the first iteration are stored in $\bar{\mathbf{x}_1}$ and those updated by the loop body are stored in $\bar{\mathbf{x}_2}$. To finish the SSA transformation, we get the fresh SSA variables $\bar{\mathbf{x}'}$ to replace the appearance of $\bar{\mathbf{x}_1}$ in $\mathbf{c_1}$ or $\bar{\mathbf{x}_2}$ in $\mathbf{c_2}$. We use $\mathbf{c_1}[\bar{\mathbf{x}'}/\bar{\mathbf{x}_1}]$ and $\mathbf{c_2}[\bar{\mathbf{x}'}/\bar{\mathbf{x}_2}]$ to represent the replacement, and only the read variables(except for the assigned variables) are replaced. Finally, we get the loop body $\mathbf{c}$ in its SSA form.

14.5 THE SOUNDNESS OF TRANSFORMATION

In this section, we show our transformation from the loop language to its SSA form is sound with respect to adaptivity. To be specific, a transformed program $\mathbf{c}$ starting with appropriate configuration, generates the same trace as the program before the transformation c, in its corresponding configuration.

We first define a well defined memory in the loop language m or in the SSA loop language $\mathbf{m}$ with respect to a translation environment δ, denoted as $m \vDash \delta$ and $\mathbf{m} \vDash \delta$ respectively.

$$\boxed{\delta; e \hookrightarrow \mathbf{e}} \qquad \frac{}{\delta; x \hookrightarrow \delta(x)}\ \textbf{s-var}$$

$$\boxed{\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'}$$

$$\frac{\begin{array}{c}\delta; b \hookrightarrow \mathbf{b} \quad \Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \quad \Sigma_1; \delta; c_2 \hookrightarrow \mathbf{c_2}; \delta_2; \Sigma_2 \\ [\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \delta_1 \bowtie \delta_2 \quad [\bar{y}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}] = \delta \bowtie \delta_1 / \bar{x} \\ [\bar{z}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}] = \delta \bowtie \delta_2 / \bar{x} \qquad \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'][\bar{y} \mapsto \bar{\mathbf{y}}'][\bar{z} \mapsto \bar{\mathbf{z}}'] \\ \bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}'\ fresh(\Sigma_2) \quad \Sigma' = \Sigma_2 \cup \{\bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}'\}\end{array}}{\Sigma; \delta; [\text{if } (b, c_1, c_2)]^l \hookrightarrow [\text{if } (\mathbf{b}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}', \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}', \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^l; \delta'; \Sigma'}\ \textbf{s-if}$$

$$\frac{\delta; e \hookrightarrow \mathbf{e} \quad \delta' = \delta[x \mapsto \mathbf{x}] \quad \mathbf{x}\ fresh(\Sigma) \quad \Sigma' = \Sigma \cup \{\mathbf{x}\}}{\Sigma; \delta; [x \leftarrow e]^l \hookrightarrow [\mathbf{x} \leftarrow \mathbf{e}]^l; \delta'; \Sigma'}\ \textbf{s-assn}$$

$$\frac{\delta; e_q \hookrightarrow \mathbf{e_q} \quad \delta' = \delta[x \mapsto \mathbf{x}] \quad \mathbf{x}\ fresh(\Sigma) \quad \Sigma' = \Sigma \cup \{\mathbf{x}\}}{\Sigma; \delta; [x \leftarrow q(e_q)]^l \hookrightarrow [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l; \delta'; \Sigma'}\ \textbf{s-query}$$

$$\frac{\begin{array}{c}\delta; a \hookrightarrow \mathbf{a} \qquad \Sigma; \delta; c \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma; \delta_1; c \hookrightarrow \mathbf{c_2}; \delta_1; \Sigma_1 \\ [\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \delta \bowtie \delta_1 \qquad \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'] \\ \bar{\mathbf{x}}'\ fresh(\Sigma_1) \qquad \mathbf{c}' = \mathbf{c_1}[\bar{\mathbf{x}}'/\bar{\mathbf{x_1}}] \qquad \mathbf{c}'[\bar{\mathbf{x_2}}/\bar{\mathbf{x}}'] = \mathbf{c_2}\end{array}}{\Sigma; \delta; [\texttt{loop}\ a\ \texttt{do}\ c]^l \hookrightarrow [\texttt{loop}\ \mathbf{a}, \mathbf{0}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}']^l; \delta_1[\bar{x} \to \bar{\mathbf{x}}']; \Sigma \cup \{\bar{\mathbf{x}}'\}}\ \textbf{s-loop}$$

Figure 14.2: Key transformation rules from loop language to ssa language.

Definition 7 (Well defined memory) 1. $m \vDash \delta \triangleq \forall x \in \texttt{dom}(\delta), \exists v, (x, v) \in m$.

2. $\mathbf{m} \vDash_{ssa} \delta \triangleq \forall \mathbf{x} \in \texttt{codom}(\delta), \exists v, (\mathbf{x}, v) \in \mathbf{m}$.

Part of the SSA memory $\mathbf{m}$ can also be reverted to a corresponding part of the memory m with an inverse of δ.

Definition 8 (Inverse of trans env)
$m = \delta^{-1}(\mathbf{m}) \triangleq \forall x \in \texttt{dom}(\delta), (\delta(x), m(x)) \in \mathbf{m}$.

We also show that the expression e in the loop language and its translated SSA version $\mathbf{e}$ by some translation environment δ evaluates to the same value in Lemma 15.

Lemma 15 (Value remains the same during transformation)**.** *Given* $\delta; e \hookrightarrow \mathbf{e}$, $\forall m. m \vDash \delta. \forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *then* $\langle m, e\rangle \rightarrow v$ *and* $\langle \mathbf{m}, \mathbf{e}\rangle \rightarrow v$.

Finally, we show the soundness of the transformation. When a program c is transformed to its SSA form $\mathbf{c}$ through a transformation environment δ, when executing these two programs with the corresponding configuration(memories are well-defined w.r.t the transformation environment δ), the newly generated traces in the two languages will be the same and the resulting memory m' and $\mathbf{m}'$ will also be related.

Theorem 16 (Soundness of transformation)**.** *Given* $\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'$, *and* $\forall m. m \vDash \delta$. $\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *if there exist an execution of c in the loop language, starting with a trace t and loop maps w,* $\langle m, c, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle$, *then there also exists a corresponding execution of* $\mathbf{c}$ *in the ssa language so that* $\langle \mathbf{m}, \mathbf{c}, t, w\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t', w'\rangle$ *and* $m' = \delta'^{-1}(\mathbf{m}')$.

The proof can be found in Appendix C, Section C.2.1.1.

Our dependency graph is constructed based on the trace, we give a lemma that says that the adaptivity remains the same during the transformation.

Lemma 17. *Given* $\Sigma;\delta;c \hookrightarrow \mathbf{c};\delta';\Sigma'$, $\forall m.m \vDash \delta.\forall \mathbf{m},\mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *starting with a trace* t *and loop maps* w, *then* $A(c,D,m,w) = A_s(\mathbf{c},D,\mathbf{m},w)$.

CHAPTER 15

The Program Analysis Algorithm for Adaptivity

In this chapter, we describe the algorithm ADAPT that analyzes the adaptivity of a target program in the SSA loop language, which consists of three auxiliary algorithms: a variable estimation algorithm VE, a matrix-vector based graph generating algorithm GG to generate the weighted variable-based dependency graph, and a path-searching algorithm PS to find the most weighted path in the graph. We do not show details of PS as we use a standard graph algorithm. We start with the ideas of ADAPT, and illustrate the first two auxiliary algorithms, variable estimation algorithm VE and dependency graph generating algorithm GG. In the end, we show the soundness of our algorithm with respect to adaptivity.

15.1 IDEAS BEHIND THE ALGORITHM

In consideration of the definition of adaptivity, the longest path in its query-based dependency graph, our analysis targeting a tight upper bound on the adaptivity is supposed to take care of paths (possible adaptivity candidates) in all the possible dependency graphs (per configuration). To this end, ADAPT aims to statically construct a weighted directed dependency graph, in which the nodes are annotated SSA variables and directed edges show when one annotated variable may depend on another. The weight of the node shows if the variable is assigned with a query request.

The algorithm ADAPT, summarized in Figure 15.1, estimates the adaptivity from the weighted variable-based dependency graph. Given the input SSA program c to be analyzed, the first step of ADAPT is the variable estimation via the auxiliary algorithm VE, which specifies the nodes of the variable-based dependency graph. The result of VE is stored in a global variable list G, fed to the next step. The second step is the graph

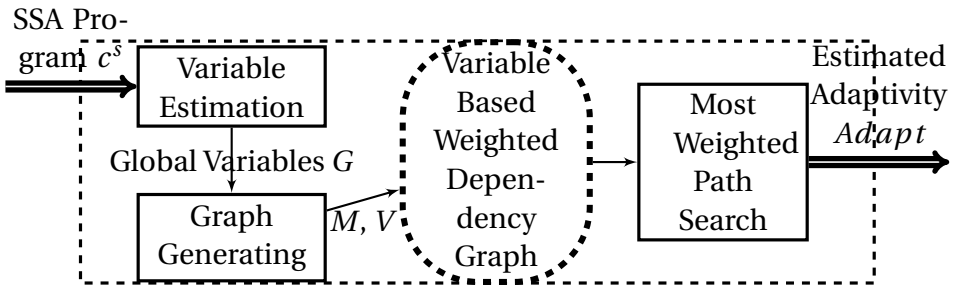


Figure 15.1: The overview of ADAPT.

generation, via a matrix-vector-based graph generating algorithm GG. The matrix M records the may-dependency between annotated variables in the global list G. It has size $|G| \times |G|$. The vector V has the same size as G and gives a weight to each variable in G. This weight is 1 when the variable is assigned with a query request and 0 otherwise.

To be precise, the ith row, jth column of the matrix M, written $M[i][j]$, is 1 when there may be a dependency from variable $G[i]$ to $G[j]$. Dually, $M[i][j] = 0$ means no dependency. In a similar way, $V[i] = 1$ means the variable $G[i]$ is assigned with a query request.

The SSA variable-based weighted dependency graph is constructed by the graph generating algorithm GG. The final step is to find out the estimated adaptivity – the most weighted path in the graph. We use another auxiliary algorithm PS to find the path with the most weights in the graph.

15.2 VARIABLE ESTIMATION ALGORITHM

We first show the algorithm VE, which adds variables to the global variable list G. VE has the form $\mathsf{VE}(G; w; \mathbf{c}) \to (G'; w')$, as shown in Figure 15.2. The input of VE is a list of annotated variables G collected before the program $\mathbf{c}$, a loop map w consistent with previous estimation, and an input SSA program $\mathbf{c}$. The output of the algorithm is the updated global list G', along with the updated loop maps w, for later estimation.

$$\frac{}{\mathsf{VE}(G; w; [\mathbf{x} \leftarrow \mathbf{e}]^{\mathbf{l}}) \rightarrow (G + +[\mathbf{x}^{(l,w)}]; w)}\ \textbf{ag-asgn}$$

$$\frac{}{\mathsf{VE}(G; w; [\mathbf{x} \leftarrow q(\mathbf{e_q})]^{l}) \rightarrow (G + +[\mathbf{x}^{(l,w)}]; w)}\ \textbf{ag-query}$$

$$\frac{\begin{array}{c}\mathsf{VE}(G; w; \mathbf{c_1}) \rightarrow (G_1; w_1) \qquad \mathsf{VE}(G_1; w; \mathbf{c_2}) \rightarrow (G_2; w_2) \\ G_3 = G_2 + +[\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w})}] + +[\bar{\mathbf{y}}^{(\mathbf{l},\mathbf{w})}] + +[\bar{\mathbf{z}}^{(\mathbf{l},\mathbf{w})}]\end{array}}{\mathsf{VE}(G; w; [\text{if } (\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}], [\bar{\mathbf{y}}, \bar{\mathbf{y}_1}, \bar{\mathbf{y}_2}], [\bar{\mathbf{z}}, \bar{\mathbf{z}_1}, \bar{\mathbf{z}_2}], \mathbf{c_1}, \mathbf{c_2})]^{l}) \rightarrow (G_3; w)}\ \textbf{ag-if}$$

$$\frac{\begin{array}{c}G_0 = G \quad w_0 = w \qquad \forall 0 \leq z < N. \mathsf{VE}(G_z + +[\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w_z}+\mathbf{l})}]; (w_z + l); \mathbf{c}) \rightarrow (G_{z+1}; w_{z+1}) \\ G_f = G_N + +[\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w_N}\setminus\mathbf{l})}] \qquad \mathbf{a} = N\end{array}}{\mathsf{VE}(G; w; [\texttt{loop } \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}] \texttt{ do } \mathbf{c}]^{l}) \rightarrow (G_f; w_N \setminus l)}\ \textbf{ag-loop}$$

Figure 15.2: The key rule of variable estimation algorithm.

The algorithm adds variables to G at assignments. In the case of expression assignment **ag-asgn** and query request **ag-query**, the output global list is expanded by $\mathbf{x}^{(l,w)}$. When it comes to if statements (rule **ag-if**), variables assigned in both branches, as well as generated variables $\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}$ in $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]$ are all added to G. Sequencing $\mathbf{c_1}; \mathbf{c_2}$ is handled as expected.

For loops, we assume that a loop counter is a natural number (rule **ag-loop**). The algorithm accounts for variables assigned in every iteration, including newly created variables in $\bar{\mathbf{x}}$, and adds them to G with annotations representing the iteration number.

15.3 MATRIX AND VECTOR BASED ALGORITHM

Next, we describe the algorithm *GG*, which takes the list G generated by the previous algorithm. GG has the form: $\mathsf{GG}(\Gamma; \mathbf{c}; i) \rightarrow (M; V; i')$. The input is a tuple consisting of three elements: (1) a 1-row-N-column matrix Γ storing the (annotated) variables on which the execution of the current instruction is control dependent; this is needed for if statements. (2) the SSA program $\mathbf{c}$ to be analyzed. (3) an index i specifying the location of the

first assigned variable of the program **c** in the global list G. The output of GG also consists of three elements: (1) A matrix M representing the may-dependencies from **c**. (2) A vector V representing the (annotated) variables assigned with queries in **c**. (3) the index i' that refers to the next position of the last assigned variable in **c**, if it exists. The existence of the index i' helps to locate the first assigned variable when we need to analyze the continuation of **c**.

We first define some functions which use the indices in G. The function L(i) generates a 1-column-N-rows matrix, where only the i-th row is 1 and all the other rows are 0. This function is used to locate the right row when we calculate the matrix during assignment and query commands.

The function R(**e**, i) generates a 1-row-N-column matrix. For every variable used in **e**, it finds the corresponding index i in G so that $G[i]$ maps to the variable and marks the ith column as 1. If the variable is not found, we do not mark it. When we say $G[i]$ maps to a target variable, we take off the annotation of $G[i]$ and check if the left variable with no annotation is the same as the target variable. The extra argument i is used to handle loops. For instance, a variable y may appear many times in G, but with different annotations (iteration numbers). In this case, i helps find the most recently assigned variable version of y in G. It is used when analyzing assignment and query request commands. Thanks to our SSA language, where every variable is assigned once, our choice of the most recent assigned variable is reasonable because the variable used in the loop refers to the most recent assignment of itself.

We define $M_1; M_2 := M_2 \cdot M_1 + M_1 + M_2$, where $M_1 + M_2$ is the standard sum of two matrices. We also define the operator $\uplus$ to combine two vectors.

$$V_1 \uplus V_2 := \begin{cases} 1 & (V_1[i] = 1 \vee V_2[i] = 1) \wedge 1 \leq i \leq N \wedge |V_1| = |V_2| \\ 0 & \text{otherwise} \end{cases}$$

For the sake of brevity, we also use some annotations when the algorithm GG handles the extra part $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$ in the if and loop statements. First, we give unique names for variables in lists $\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}$ respectively, as follows:

$$\forall 0 \leq z < |\bar{\mathbf{x}}|.\bar{\mathbf{x}}(z) = \mathbf{x_z}, \bar{\mathbf{x_1}}(z) = \mathbf{x_{1z}}, \bar{\mathbf{x_2}}(z) = \mathbf{x_{2z}}$$

We treat every tuple $(\mathbf{x_z}, \mathbf{x_{1z}}, \mathbf{x_{2z}})$ in $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$ as the simple may dependency case : $\mathbf{x_z}$ may depend on both $\mathbf{x_{1z}}$ and $\mathbf{x_{2z}}$, just like $\mathbf{x_z} \leftarrow \mathbf{x_{1z}} + \mathbf{x_{2z}}$, defined as follows.

$\mathsf{GG}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i) \rightarrow (M; V_\emptyset; i + |\bar{\mathbf{x}}|) \triangleq$

$\forall 0 \leq z < |\bar{\mathbf{x}}|.\mathsf{GG}(\Gamma; \mathbf{x_z} \leftarrow \mathbf{x_{1z}} + \mathbf{x_{2z}}; i + z) \rightarrow (M_{x_z}; V_\emptyset; i + z + 1)$ where $M = \sum_{o \leq z < |\bar{\mathbf{x}}|} M_{x_z}$.

One key idea of algorithm GG is to track the indices i, i' in the input and output to synchronize with the previous algorithm VE: The index in GG increases in the same way as the global list expands after the analysis of a program **c** by VE, which helps GG record the dependency relation from the program **c** in the right place of the matrix. For example, in the cases **ad-asgn** and **ad-query**, the index increases by 1, which corresponds to the addition of one variable in the algorithm **VE**. The if and loop commands have the extra part $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$ and the output index increases by also considering this part as we do in collecting G in VE.

We show one simple example sa to illustrate the construction of the matrix.

$$sa \triangleq [x_1 \leftarrow 2]^1; [x_2 \leftarrow x_1 + 2]^2; [x_3 \leftarrow x_1 + x_2]^3$$

In the program sa, only simple assignment is involved. When we assume an empty Γ,

$$\frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e}, i) + \Gamma)}{\mathsf{GG}(\Gamma; [\mathbf{x} \leftarrow \mathbf{e}]^l; i) \rightarrow (M; V_\emptyset; i+1)}\ \textbf{ad-asgn}$$

$$\frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e_q}, i) + \Gamma) \qquad V = \mathsf{L}(i)}{\mathsf{GG}(\Gamma; [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l; i) \rightarrow (M; V; i+1)}\ \textbf{ad-query}$$

$$\frac{\begin{array}{c}\mathsf{GG}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_1}; i_1) \rightarrow (M_1; V_1; i_2) \\ \mathsf{GG}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_2}; i_2) \rightarrow (M_2; V_2; i_3) \qquad \mathsf{GG}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i_3) \rightarrow (M_x; V_\emptyset; i_3 + |\bar{\mathbf{x}}|) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]; i_3 + |\bar{\mathbf{x}}|) \rightarrow (M_y; V_\emptyset; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \rightarrow (M_y; V_\emptyset; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}| + |\bar{\mathbf{z}}|) \\ M = (M_1 + M_2) + M_x + M_y + M_z\end{array}}{\mathsf{GG}(\Gamma; \text{if } ([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}); i_1) \rightarrow (M; V_1 \uplus V_2; i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)}$$

ad-if

$$\frac{\mathsf{GG}(\Gamma; \mathbf{c_1}; i_1) \rightarrow (M_1; V_1; i_2) \qquad \mathsf{GG}(\Gamma; \mathbf{c_2}; i_2) \rightarrow (M_2; V_2; i_3)}{\mathsf{GG}(\Gamma; (\mathbf{c_1}; \mathbf{c_2}); i_1) \rightarrow ((M_1; M_2); V_1 \uplus V_2; i_3)}\ \textbf{ad-seq}$$

$$\frac{\begin{array}{c}B = |\bar{\mathbf{x}}| \qquad A = |\mathbf{c}| \\ \forall 0 \le j < N. \mathsf{GG}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + j * (B + A)) \rightarrow (M_{1j}; V_{1j}; i + B + j * (B + A)) \\ \mathsf{GG}(\Gamma; \mathbf{c}; i + B + j * (B + A)) \rightarrow (M_{2j}; V_{2j}; i + B + A + j * (B + A)) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + N * (B + A)) \rightarrow (M; V; i + N * (B + A) + B) \\ \mathbf{a} = N \qquad M' = M + \sum_{0 \le j < N} (M_{1j} + M_{2j}) \qquad V' = V \uplus \sum_{0 \le j < N} (V_{1j} \uplus V_{2j})\end{array}}{\mathsf{GG}(\Gamma; \texttt{loop } [\mathbf{a}]^l,\ 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do } \mathbf{c}, i) \rightarrow (M'; V'; i + N * (B + A) + B)}\ \textbf{ad-loop}$$

Figure 15.3: The key rules of the graph generating algorithm.

when analyzing the assignment at line 3, the matrix is built as follows.

$$\textbf{line3: } [x_3 \leftarrow x_1 + x_2]^3 : \quad \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} * \begin{matrix} x_1 & x_2 & x_3 \end{matrix} \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} = \begin{matrix} & x_1 & x_2 & x_3 \\ x_1 & & & \\ x_2 & & & \\ x_3 & & & \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Γ is needed to handle the case when the control flow diverges in an if statement , where the execution of either branch may depend on the conditional predicate **b**. Following this intuition, the analysis of either branch considers the variables used in the conditional **b**, tracked in Γ. In the rule **ad-if**, the analysis of the two branches $\mathbf{c_1}$ and $\mathbf{c_2}$ is in $\Gamma + \mathsf{R}(\mathbf{b}, i)$. Here, $\mathsf{R}(\mathbf{b}, i)$ gives variables used in the conditional predicate **b**.

We compose the matrices and vectors in sequencing in rule **ad-seq**. The non-zeros or, as we call them, the *effect ranges*, of the matrix and the vector are decided by input and output indices. In rule **ad-seq**, the two programs $\mathbf{c_1}$ and $\mathbf{c_2}$ must have disjoint effect ranges $[i_1, i_2)$ and $[i_2, i_3)$, so it is safe to combine them without losing information. In rule **ad-seq**, we use $B = |\bar{\mathbf{x}}|$ and $A = |\mathbf{c}|$ to estimate the number of variables assigned in $\bar{\mathbf{x}}$ and **c**. $|\mathbf{c}|$ is defined with the help of the algorithm VE, defined as $|\mathbf{c}| = |G|$ when $\mathsf{VE}([]; \mathbf{c}; \emptyset) \rightarrow (G; \emptyset)$. The algorithm then determines the number N of loop iterations from the loop counter **a**. For every iteration, it first records the dependency relations between variables in $[\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]$ by constructing a corresponding matrix M_{1j} (j is the iteration number) and an empty vector V_{1j}, and analyzes the loop body **c**, resulting in a matrix M_{2j} and vector V_{2j}. We know that for all possible iteration numbers j, M_{1j} and M_{2j} have disjoint effect ranges so we can safely combine them. Similarly, we can combine the vectors V_{1j} and V_{2j}.

End-to-end, ADAPT yields a variable-based weighted dependency graph defined by

edges M and weights V. The definition of the estimated adaptivity is the weight of the most weighted path in the graph, defined as follows.

Definition 9 (Estimated adaptivity)

Given a program $\mathbf{c}$, the global list G, and $\mathsf{GG}(\Gamma;\mathbf{c};i_1) \rightarrow (M,V,i_2)$, the weighted dependency graph $G_{ssa}(M,V,G,i_1,i_2) = (Nodes, Edges, Weights)$ is defined as:

Nodes $Vt = \{G(j) \in \mathscr{LV} \mid i_1 \leq j < i_2\}$

Edges $E = \{(G(j_1), G(j_2)) \in \mathscr{LV} \times \mathscr{LV} \mid M[j_1][j_2] \geq 1 \wedge i_1 \leq j_1, j_2 < i_2\}$

Weights $Wt = \{(G(j),1) \in \mathscr{LV} \times \mathscr{N} \mid i_1 \leq j < i_2 \wedge V[j] = 1\} \cup \{(G(j),0) \in \mathscr{LV} \times \mathscr{N} \mid i_1 \leq j < i_2 \wedge V[j] = 0\}$.

Adaptivity of the program is then defined as:

$$Adapt(M,V,G,i_1,i_2) := \max_{vt_1, vt_2 \in Vt} \{\mathsf{Weight}(p(vt_1,vt_2),Wt)\},$$

where $p(k,l)$ is the path in graph $G_{ssa}(M,V,G,i_1,i_2)$ starting from k to l, and $\mathsf{Weight}(p(vt_1,vt_2),Wt)$ is the total sum of weights of nodes along the path $p(vt_1,vt_2)$.

15.4 SOUNDNESS OF ADAPT

We would like to show that the query-based dependency graph generated from the trace of the execution of the target SSA program is a subgraph of the variable-based dependency graph generated by $ADAPT$, and the total number of queries asked in the program implementing an adaptive data analysis is also bounded by our algorithm.

We first define when a query-based dependency graph whose nodes are queries is a subgraph of a variable-based dependency graph whose nodes are variables. Intuitively, this happens when there is a mapping of nodes from the first graph to the second.

Definition 10 (Subgraph)

Given two graphs $G_s = (V_1, E_1)$, $G_{ssa} = (V_2, E_2, Wt)$, $G_s \subseteq G_{ssa}$ iff:

$\exists f, g$ so that

1. for every $v \in V_1$, $f(v) \in V_2 \wedge f$ is injective, and $Wt(f(v)) = 1$.

2. $\forall e = (v_i, v_j) \in E_1$, there exists a path $g(e)$ from $f(v_i)$ to $f(v_2)$ in G_{ssa}.

We also show that there exists a well defined mapping $f_{D,m} : \mathscr{AQ} \rightarrow \mathscr{LV}$ that maps every distinct query node to distinct variable node between the two graphs.

Lemma 18 ($f_{D,m}$ is well defined)**.** *Given* $\Gamma \vdash^{(i_1,i_2)}_{M,V} \mathbf{c}$*, for any global list* G *and a loop maps* w*, such that* $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V)$ *and* $\Gamma \vdash i_1$*. For any database* D *and memory* $\mathbf{m}$*, the function* $f_{D,m} : \mathscr{AQ} \rightarrow \mathscr{LV}$ *maps all the nodes in query-based graph* $(V_1, E_1) = G_s(\mathbf{c}, D, \mathbf{m}, w)$ *to the unique node in the variable-based graph* $(V_2, E_2, Wt) = G_{ssa}(M, V, G, i_1, i_2)$*, defined as :*

$f_{D,m}(q^{l,w}) = \mathbf{x}^{(l,w)}$ *where* $q^{(l,w)} \in V_1$ *and* $\mathbf{x}^{(l,w)} \in V_2$ *is injective*

and $\forall q(v_q)^{(l,w)} \in V_1, wt(f(q(v_q)^{(l,w)})) = 1$.

The proof can be found in Appendix C, Section C.3.3.1.

We now show the soundness of ADAPT. We use some new definitions. $G \vDash M, V$ means that G matches M, V in size. For example, if the cardinality of G is N, a matched matrix M is of size $N \times N$ and a good vector V has the same size as G. The assumption $G; w \vDash (\mathbf{c}, i_1, i_2)$ checks that the variables assigned in $\mathbf{c}$ estimated by VE match the variables in G from index i_1 to i_2.

Theorem 19 (Soundness of ADAPT)**.** *Given* $\mathsf{GG}(\Gamma; \mathbf{c}; i_1) \rightarrow (M; V; i_2)$*, for any global list* G*, loop maps* w *such that* $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V)$*, let* K *be the number of queries made during the execution of the piece of program* $\mathbf{c}$ *and* $|V|$ *be the number of non-zeros in* V*. Then,*

$$K \leq |V| \wedge \forall D, \mathbf{m}. G_s(\mathbf{c}, D, \mathbf{m}, w) \subseteq G_{ssa}(M, V, G, i_1, i_2)$$

Proof. By induction on the judgment $\mathsf{GG}(\Gamma; \mathbf{c}; i_1) \rightarrow (M; V; i_2)$.

$$\textbf{Case:}\ \frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e}, i) + \Gamma)}{\mathsf{GG}(\Gamma; [\mathbf{x} \leftarrow \mathbf{e}]^l; i) \rightarrow (M; V_\emptyset; i+1)}\ \textbf{ad-asgn}$$

Given a memory m, database D, trace t, loop maps w, then from the following operational semantics:

$$\frac{}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{e}]^l, t, w\rangle \rightarrow^* \langle \mathbf{m}, [\mathbf{x} \leftarrow v]^l, t, w\rangle \rightarrow \langle \mathbf{m}[\mathbf{x} \mapsto v], [\texttt{skip}]^l, t, w\rangle}$$

We need to show:

1. $G_s(\mathbf{x} \leftarrow \mathbf{e}, D, m, w) \subseteq G_{ssa}(M, V_\emptyset, i, i+1)$.
2. $K \leq |V_\emptyset|$.

The first goal is shown because the new generated trace t' is empty, based on the definition of G_s, we know that G_s has no vertices.
The second goal is proved because $K = 0$.

$$\textbf{Case:}\ \frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e_q}, i) + \Gamma) \qquad V = \mathsf{L}(i)}{\mathsf{GG}(\Gamma; [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l; i) \rightarrow (M; V; i+1)}\ \textbf{ad-query}$$

Given a memory m, database D, trace t, loop maps w, then from the following operational semantics:

$$\frac{q(v_q)(D) = v}{\langle \mathbf{m}, [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l, t, w\rangle \rightarrow \langle \mathbf{m}[v/\mathbf{x}], \texttt{skip}, t \mathbin{+\!\!+} [(q(v_q)^{(l,w)})], w\rangle}$$

We need to show:

1. $G_s([\mathbf{x} \leftarrow q(\mathbf{e_q})]^l, D, m, w) \subseteq G_{ssa}(M, V, i, i+1)$.
2. $K \leq |V|$.

The first goal is shown as follows: because the new generated trace $t' = [q(v)^{(l,w)}]$ only has one element. By the definition of $G_s = (V_s, E_s)$, we know there is only one vertex vx in V and no edge in E, i.e., $V_s = \{q(v)^{(l,w)}\}$ and $E_s = \{\}$.

From Lemma 91,we know that there exists a function f so that $f(vx)$ (equivalent to $q(v)^{(l,w)}$ in G_s) exists in the vertices V_{ssa} of $G_{ssa}(M, v, i, i+1) = (V_{ssa}, E_{ssa})$. Since $V_s \subseteq V_{ssa}$ and $E_s \subseteq E_{ssa}$, it is proved that $G_s([\mathbf{x} \leftarrow q(\mathbf{e_q})]^l, D, m, w) \subseteq G_{ssa}(M, V, i, i+1)$. So we have the first goal proved.

The second goal is proved because $V = \mathsf{L}(i)$ is not empty so that $|V| \geq 1$, and $K = 1$.

Case:

$$\frac{\begin{array}{c}\mathsf{GG}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_1}; i_1) \rightarrow (M_1; V_1; i_2) \\ \mathsf{GG}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_2}; i_2) \rightarrow (M_2; V_2; i_3) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i_3) \rightarrow (M_x; V_\emptyset; i_3 + |\bar{\mathbf{x}}|) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]; i_3 + |\bar{\mathbf{x}}|) \rightarrow (M_y; V_\emptyset; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \rightarrow (M_y; V_\emptyset; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}| + |\bar{\mathbf{z}}|) \\ M = (M_1 + M_2) + M_x + M_y + M_z\end{array}}{GG\Gamma; \mathbf{if}\ ([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}); i_1 \rightarrow (M; V_1 \uplus V_2; i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)}\ \textbf{ad-if}$$

Given a memory $\mathbf{m}$, database D, trace t, loop maps w and a global list G. We assume that $G \vDash M, V_1 \uplus V_2, G; w \vDash (\text{if } (\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}), i_1, i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)$. There are two possible executions. We choose one branch and the other will be similar.

From the assumption, denote $G' = G[0, \ldots, (i_1 - 1)]$ we know that

$$\frac{\begin{array}{c}G'; w; \mathbf{c_1} \rightarrow G_1; w_1 \\ G_1; w; \mathbf{c_2} \rightarrow G_2; w_2 \qquad G_3 = G_2 + +[\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w})}] + +[\bar{\mathbf{y}}^{(\mathbf{l},\mathbf{w})}] + +[\bar{\mathbf{z}}^{(\mathbf{l},\mathbf{w})}]\end{array}}{G'; w; \text{if } ([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}) \rightarrow G_3; w}\ \textbf{if}$$

$\mathbf{c_1}$ is the sub term of if $(\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})$. so we have: $G \vDash M_1, V_1$ and

$G; w \vDash (\mathbf{c_1}, i_1, i_2)$, and $\Gamma + \mathsf{R}(b, i_1) \vDash i_1$.

By induction hypothesis on the first premise $\Gamma + \mathsf{R}(b, i_1) \vdash^{(i_1,i_2)}_{M,V} \mathbf{c_1}$, we can conclude that :

$$K_{\mathbf{c_1}} \leq |V_1| \wedge G_s(\mathbf{c_1}, D, m, w) \subseteq G_{ssa}(M_1, V_1, i_1, i_2).$$

By the definition of K, we have $K_{\text{if } (\mathbf{b},[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}],[\bar{\mathbf{y}},\bar{\mathbf{y_1}},\bar{\mathbf{y_2}}],[\bar{\mathbf{z}},\bar{\mathbf{z_1}},\bar{\mathbf{z_2}}],\mathbf{c_1},\mathbf{c_2})} = K_{\mathbf{c_1}}$ when $b = \text{true}$.

Next, we want to show : $G_s(\text{if } ([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}), D, \mathbf{m}, w)$ $\subseteq G_{ssa}(M, V_1 \uplus V_2, i_1, i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)$. By the definition of G_s, we know that $G_s(\text{if } ([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}), D, \mathbf{m}, w)$ is the same graph as $G_s(\mathbf{c_1}, D, \mathbf{m}, w)$. We can also show that $G_{ssa}(M_1, V_1, G, i_1, i_2)$ is a subgraph of $G_{ssa}(M, V_1 \uplus V_2, i_1, i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)$ because all the vertices in the former also appears in the later one ($[i_1, i_2]$ is a smaller range of $[i_1, i_3 + i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|]$) and same for the edges(M contains M_1).

□

The details can be found in Appendix C, Section C.3.3.2.

Corollary 19.1

Given $\mathsf{GG}(\Gamma; \mathbf{c}; i_1) \rightarrow (M; V; i_2)$, for any global list G, loop map w such that $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V)$,

$$K \leq |V| \wedge A_s(\mathbf{c}, D, \mathbf{m}, w) \leq Adapt(M, V, G, i_1, i_2)$$

We have already shown in Lemma 17 that for every transformation of a program c to $\mathbf{c}$, the adaptivity of c and $\mathbf{c}$ is equal. It follows that the bound estimated by ADAPT on $\mathbf{c}$ is an upper bound on the adaptivity of c.

CHAPTER 16

Examples of ADAPT

16.1 TWO ROUND ODD ELEMENTS

We present a variant of the previous two round example in Figure 16.1. In this odd example, only the data at odd index of the database is used. This algorithm only touches the odd part of the database, by adding an extra if statement to checking the index j in Figure 16.1(a). The extra complexity is added to handle the newly generated variables in the loop and if statement in the SSA version in Figure 16.1(b). The query-based dependency graph does not change a lot compared to the previous two rounds example in Figure 12.2(c), but the node does change according to the trace. We assume $a = n$ in the final memory is the result of the sum of previous query results in the loop. We give the trace $t = [q(\chi[1])^{5,[3:1]}, q(\chi[1])^{6,[3:2]}, q(\chi[3])^{5,[3:3]}, q(n * \chi[3])^{9,\emptyset}]$ and use q_1 for $q(\chi[1])$, q_3 representing for $q(\chi[3])$, q_4 for $q(n * \chi[3])$. The query-based dependency graph based on this trace is shown in Figure 16.2(a). We show the red path, which is a sequence of adaptively chosen queries of length 2. So among the total 4 queries, we have 2-round adaptive queries. According to Theorem 13 and Theorem 14, we will have a tighter up-

$[a \leftarrow 0]^1;$
$[j \leftarrow 0]^2;$
`loop` $[3]^3$ `do`
  if $([(j\%2 == 0)]^4$
    $, [x \leftarrow q(\chi[j+1])]^5$
    $, [x \leftarrow q(\chi[j])]^6);$
  $[a \leftarrow a + x]^7;$
  $[j \leftarrow j + 1]^8;$
$[l \leftarrow q(a * \chi[3])]^9$

(a)

$[a_1 \leftarrow 0]^1;$
$[j_1 \leftarrow 0]^2;$
`loop` $[3]^3, 0,$ `do` $[(j_3, j_1, j_2), (a_3, a_1, a_2)],$
  if $([(j_3\%2 == 0)]^4, [x_3, x_1, x_2], [], []$
    $, [x_1 \leftarrow q(\chi[j_3+1])]^5$
    $, [x_2 \leftarrow q(\chi[j_3])]^6);$
  $[a_2 \leftarrow a_3 + x_3]^7;$
  $[j_2 \leftarrow j_3 + 1]^8;$
$[l_1 \leftarrow q(a_3 * \chi[3])]^9$

(b)

Figure 16.1: (a) The two round odd algorithm in labeled `Loop` language, (b) The SSA program for the same example.

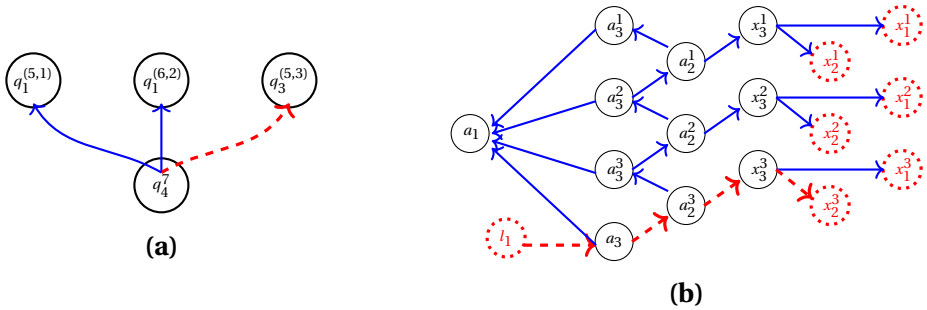


Figure 16.2: (a) The query-based dependency graph for odd example (b) The SSA variable-based weighted dependency graph for the same example, the node in red dashed circle is weighted.

per bound on the generalization error if we know the adaptivity 2, obtained from the red path in Figure 16.2(a).

Our algorithm ADAPT gives us the upper bound on the aforementioned adaptivity 2. We construct the variable-based weighted dependency graph in Figure 16.2(b). The weighted nodes are in the red dashed circle and the red dashed paths show the most weighted path in the graph, with the weight 2. So, for this two rounds odd algorithm, our system gives a tight upper bound 2, which can be used to get a better generalization error bound.

16.2 MULTIPLE ROUND ALGORITHM

The two round strategy works well in our framework, we explore further to look at an advanced adaptive data analysis algorithm - multiple round algorithm. The multiple rounds algorithm starts from an initialized empty tracking list I, two scores called Nscore $ns = 0$ and Cscore $cs = 0$, initialzied to 0. It goes k rounds and at every round, the two scores ns and cs are updated by the result a of a query $q(f(I))$. The function $f(I)$ specifies a complex linear query using the updated tracking list I. The tracking list I is updated by the two scores via a function $update(I, ns, cs)$ at every round. This update function mainly compares ns and cs, when $ns \geq cs$, certain elements are

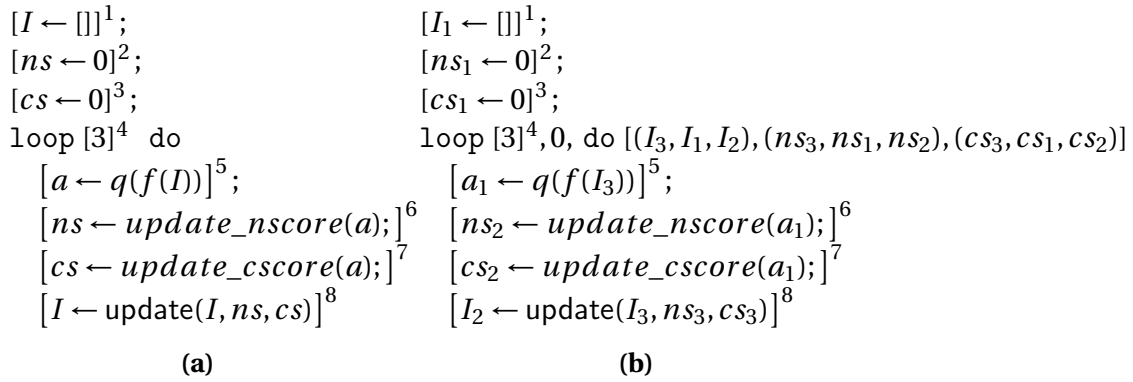


Figure 16.3: (a) The labeled program implementing the multiple round algorithm (b)The same program in the SSA version.

added to the tracking list I. An implementation of the algorithm is presented in Figure 16.3(a), in which the round number k are set to 3, and we use $update\_cscore(a)$ and $update\_nscore(a)$ to simplify the complex update on Cscore and Nscore respectively, for the sake of simplicity.

One complexity of the multiple rounds algorithm in comparison with the two rounds one, is that the query asked in each iteration is not independent(non-adaptive) anymore. For example, the query q^j at iteration j now may depend on the tracking list I, which comes from the previous iteration $j-1$. Additionally, this list I at iteration $j-1$ is updated by the query result q^{j-1} at the same iteration. Intuitively, we can see the connection between queries from different iterations, which means these queries are adaptively chosen according to our Theorem 14.

We first show its query-based dependency graph in Figure 16.4(a), the execution trace t_{mr} is generated as follows: $t_{mr} = \{q(f(I_a))^{5,[4:1]}, q(f(I_b))^{5,[4:2]}, q(f(I_c))^{5,[4:3]}\}$. For a better presentation of the graph, we add some notations: q_a for $q(f(I_a))^{5,[4:1]}$, similar for q_b, q_c for $q(f(I_b))^{5,[4:2]}$, $q(f(I_c))^{5,[4:3]}$. We can see the red dashed path from $q_c \rightarrow q_b \rightarrow q_a$ is the round of adpativity our theorem wants, as the longest path in the dependency graph. Since $k = 3$, the multiple rounds algorithm takes in total $k = 3$ queries from an

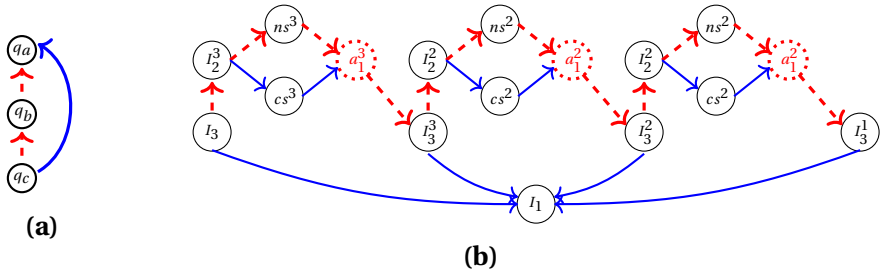


Figure 16.4: (a) The query-based dependency graph for multiple round example (b) The SSA variable-based weighted dependency graph for the same example, the node in red dashed circle is weighted.

data analyst, answers the queries. And from the graph, we know that there are 3-round adaptive queries in these 3 input queries(fully adaptive), since the red path has length 3.

Next, we show our algorithm providing the estimated upper bound for this multiple rounds example through constructing a variable-based weighted dependency graph in Figure16.4(b). We use a short in the graph, such as a_1^3 for $a_1^{(5,[4:3])}$ and so on. We show the most weighted path in the graph, which is the red dashed path as usual. Along the red dashed path, 3 weighted nodes a_1^3, a_1^2, a_1^1, correspond to our queries q_c, q_b and q_a respectively. This is our intuition to estimate one graph in Figure 16.4(b), to upper bound another graph(Figure 16.4)(a). Here, we simplify the estimated graph by omitting some variables such ns_1, cs_1 in Figure 16.4(b). Every query node in the query-based dependency graph has a corresponding node(variable the query is associated) in the variable-based dependency graph generated by our analysis algorithm ADAPT.

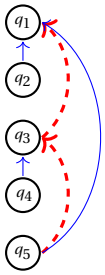
16.3 OVER-APPROXIMATION ALGORITHM

Our algorithm comes across an over-approximation on the estimation due to its path-insensitive nature. It occurs when the control flow can be decided in a particular way in front of conditional branches, while the static analysis fails to witness.

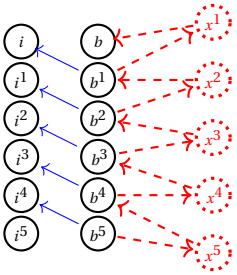
We use one example to show the over-approximation, Figure 16.5(a). This example

$[b \leftarrow 0]^1$;
$[i \leftarrow 1]^1$;
`loop` $[5]^3$ `do`
  $\left[x \leftarrow q(b + \chi[i-1])\right]^4$;
  if $([(i\%2 == 1)]^5$
    , $[b \leftarrow b + x]^6$
    , $\left[\texttt{skip}\right]^7$);
  $[i \leftarrow i + 1]^8$

(a)



(b)



(c)

Figure 16.5: (a) The labeled program implementing multiple round odd iteration example (b) The query-based dependency graph for the example (c) The SSA variable-based weighted dependency graph for the same example.

is the variant of the multiple rounds strategy, we call it a multiple rounds odd iteration algorithm. In this algorithm, at every iteration, a query $q(b + \chi[i])$ based on previous query results stored in b is asked by the analyst like in the multiple rounds strategy. The difference is that only the query answers from odd iterations ($i = 1,3,5$) are added to b. Because the execution trace only updates b using the query answers at odd iterations, so the answers from even iterations do not affect the queries at odd iterations. From the query-based dependency graph in Figure 16.5(b), we can see that there is no edge from queries at odd iterations (such as q_1, q_3, q_5) to queries at even iteration(such as q_2, q_4). The longest path is dashed with a length 3. However, ADAPT fails to realize that odd iteration will always execute then branch and even iteration means else branch, so its dependency graph considers both branches for every iteration. In this sense, the dependency graph by ADAPT is similar to the one in the multiple rounds strategy. We show the estimated graph in Figure 16.5(c). The estimated upper bound is then, 5, instead of 3.

CHAPTER 17

Related Work of ADAPT

In terms of techniques, our work relies on ideas from both static analysis and dynamic analysis. We discuss closely related work in both areas.

17.1 STATIC PROGRAM ANALYSIS

Our program analysis algorithm in Chapter 15 is influenced by many areas of static program analysis such as effect systems, control-flow analysis, and data-flow analysis Ryder & Paull (1988). The idea of statically estimating a sound upper bound for the adaptivity from the semantics is indirectly inspired from prior work on cost analysis via effect systems Çiçek et al. (2017); Radicek et al. (2018b); Qu et al. (2019). The idea of using an adjacency matrix to reason about data flows as a resource has been also studied as an instance of graded Hoare logic Gaboardi et al. (2021). One of the most important ingredients of our work is the estimation of the variable-based dependency graph. There are many ways to construct a dependency graph statically. Some of the most related work focuses on the testing of graphical user interfaces (GUIs), using an event graph. For example, Memon (2007) proposes an event-flow model using an algorithm to construct an event-flow graph, representing all the possible event interactions. This event-flow graph has a vertex for every GUI event such as click-to-paste and an edge between pairs of events that can be performed immediately one after the other. Our variable-based dependency graph uses the edge to track the may-dependence of one variable with respect to another variable. The main difference is in the way the graph is constructed. ARel relies on the structure of the target program, while the event-flow model only considers the event type. Another work Arlt et al. (2012) constructs a weighted event-dependency graph, capturing data dependencies between events by analyzing

bytecode. Every weighted edge indicates a dependency between two events, meaning one event possibly reads data written by the other event, with the weight showing the intensity of the dependency (the quantity of data involved). Our approach of generating the variable-based dependency graph shares the idea of tracking data dependency via static analysis on the source code. However, because of the different domains, we care about assigned variables, and we use the weight in a different way.

17.2 DYNAMIC PROGRAM ANALYSIS

Our framework constructs a dependency graph based on the execution trace of a program. We define semantic dependence on this graph by considering (intraprocedural) data and control dependency Bilardi & Pingali (1996); Cytron et al. (1991); Pollock & Soffa (1989). One related work Austin & Sohi (1992) presents a methodology to construct a dynamic dependency graph (DDG) based on the dynamic execution of a program in an imperative language, where edges represent dependency between instructions. Data dependency, control dependency, storage dependency, and resource dependency between instructions are all considered. Our query-based dependency graph only needs data dependency and control dependency between query requests. Critical path length analysis on DDGs is useful for understanding the scope for parallelization, while we use the length of the longest path to define adaptivity. DDGs have been used in many other domains. Nagar & Jagannathan (2018) use DDGs to find serializability violations. Hammer et al. (2006) use similar *program dependency graphs* Ferrante et al. (1987) for dynamic program slicing. They use a combination of static and dynamic dependency graphs but in a manner that is different from how we use the two. Their slicing uses both static and dynamic dependency graphs, while we use the dynamic dependency graph as the basis of a definition, which is then soundly approximated by an analysis based on the static

dependency graph.

17.3 GENERALIZATION IN ADAPTIVE DATA ANALYSIS

Starting from the works by Dwork et al. (2015c) and Hardt & Ullman (2014), several works have designed methods that ensure generalization for adaptive data analyses. Some examples are: Dwork et al. (2015b,a); Bassily et al. (2016); Ullman et al. (2018); Feldman & Steinke (2017); Jung et al. (2020); Steinke & Zakynthinou (2020); Rogers et al. (2020). Several of these works drew inspiration from the idea of using methods designed to ensure differential privacy, a notion of formal data privacy, to guarantee generalization for adaptive data analyses. By limiting the influence that an individual can have on the result of data analysis, even in adaptive settings, differential privacy can also be used to limit the influence that a specific data sample can have on the statistical validity of data analysis. This connection is actually in two directions, as discussed for example by Yeom et al. (2018).

Considering this connection between generalization and privacy, it is not surprising that some of the works on programming language techniques for privacy-preserving data analysis are related to our work. Adaptive Fuzz Winograd-Cort et al. (2017) is a programming framework for differential privacy that is designed around the concept of adaptivity. This framework is based on a typed functional language that distinguishes between several forms of adaptive and non-adaptive composition theorem with the goal of achieving better upper bounds on the privacy cost. Adaptive Fuzz uses a type system and some partial evaluation to guarantee that the programs respect differential privacy. However, it does not include any technique to bound the number of rounds of adaptivity. Lobo-Vesga et al. (2021) propose a language for differential privacy where one can reason about the accuracy of programs in terms of confidence intervals on the error

that the use of differential privacy can generate. These are akin to bounds on the generalization error. This language is based on a static analysis which however cannot handle adaptivity. The way we formalize the access to the data mediated by a mechanism is a reminiscence of how the interaction with an oracle is modeled in the verification of security properties. As an example, the recent works by Barbosa et al. (2021) and Aguirre et al. (2021) use different techniques to track the number of accesses to an oracle. However, reasoning about the number of accesses is easier than estimating the adaptivity of these calls, as we do instead here.

IV EPILOGUE

I review the works on studying the quantitative properties in this dissertation in the next part, and the future directions are also discussed.

CHAPTER 18

Conclusion

In this chapter, we conclude this dissertation, mainly studying two quantitative properties of programs, the relative cost of two programs, and the adaptivity of adaptive data analysis programs. The relative cost can be reasoned about by relational cost analysis, and we will review the work of ARel and its implementation BiARel. Adaptivity is another topic, the work ADAPT focuses on it. We review its key novelties and contributions discuss several directions on the future researches.

18.1 RELATIVE COST

In the dissertation, we first present ARel, a relational type-and-effect system that can be used to reason about the relative cost of functional-imperative programs with mutable arrays.

The contribution of ARel can be summarized as:

1. The support of array-based operations is useful for relational cost analysis.

2. The key contribution of ARel is a set of lightweight relational refinements allowing one to establish different relations between pairs of state-affecting computations, including upper bounds on cost difference.

3. The soundness proof of ARel via a novel Kriple logical relation model gives the theoretical foundation of the combination of type and effect systems and relational refinements for state-affecting computations.

4. The implementation of ARel shows the practical utility.

On the other hand, there are also limitations when we introduce the mutable arrays into relational cost analysis and what we can do in future work.

1. ARel It now only supports arrays whose elements are of the ground type, due to our choice of the lightweight monadic types for the array-based operations. As a reminder, the elements of arrays are restricted to base type in ARel. One of the directions to do in the future is to study other imperative features, finding out if there are some connections with our mutable arrays. The support of more complicated but common data types such as matrix(arrays of arrays) is now missing in ARel.

2. The other limitation comes from the implementation mentioned in Section 10.2.5, the efficiency of our type checker remains to be improved in future work.

3. One of the future works is to formalize the metatheory of ARel via a proof assistant.

Another work we presented, which is related to ARel is the implementation BiARel. Since ARel is not suitable for implementation due to its non-determinism nature coming from its typing and subtyping, it is not straightforward to have a concrete implementation. To this end, we presented a theoretical study and a concrete implementation of bidirectional type checking in a setting that combines relational refinements, comonadic types, and relational effects, as well as the monadic types for arrays in ARel. This rich setting poses unique challenges: The typing rules are not syntax-directed due to relational refinements; subtyping for (relational) comonads poses additional problems, as do the relational effects. We resolve these challenges through a process of elaboration and subtyping-elimination in the theory and using example-guided heuristics in the implementation. We validate experimentally that this approach is practical—it

works for many different kinds of programs in ARel, has little annotation burden and type checking is acceptable. We believe that our work, along with the prior work Çiçek (2018) will help future designers implement other relational type and effect systems as well.

18.2 ADAPTIVITY

Another quantitative property is the adaptivity of programs. We presented ADAPT, a program analysis algorithm that is useful to provide an upper bound on the adaptivity of data analysis under a specific data analysis model. This estimation can help data analysts to control the generalization errors of their analyses by choosing different algorithmic techniques based on adaptivity. Besides, a key contribution of our works is the formalization of the notion of adaptivity for adaptive data analysis.

There are some limitations of this work.

1. One limitation is that our algorithm may over-estimate the adaptivity of a program, as shown in Section 16.3, due to its path-insensitive nature when analyzing if conditions. The future direction is to use the constraint to track the necessary information and help to predict the path when the control flow diverges.

2. Another one is that our algorithm now can only analyze concrete bound for loops. The direction of future work is to support dynamic or unbounded loops.

List of Figures in Appendices

APPENDIX A

Appendix for ARel

A.1 SYNTAX OF AREL

Unary types	$A ::= \mathsf{int} \mid \mathsf{int}[I] \mid \mathsf{unit} \mid \{P\}\ \exists \vec{\gamma}.A\ \{Q\}^{\mathsf{exec}(L,U)} \mid \forall i{::}S.\ A^{\mathsf{exec}(L,U)} \mid A \longrightarrow A'^{\mathsf{exec}(L,U)} \mid A_1 \times A_2$ $\mathsf{Array}_\gamma[I]\ A \mid \mathsf{list}[I]\ A \mid A_1 + A_2 \mid C \& A \mid C \supset A \mid \exists i :: S.\ A$
Relational types	$\tau ::= \mathsf{int} \mid \mathsf{unit} \mid \mathsf{int}[I] \mid \{P\}\ \exists \vec{\gamma}.\tau\ \{Q\}^{\mathsf{diff}(D)} \mid \forall i{::}S.\ \tau^{\mathsf{diff}(D)} \mid \tau \longrightarrow \tau'^{\mathsf{diff}(D)} \mid \mathsf{Array}_\gamma[I]\ \tau$ $\mid U\ (A_1, A_2) \mid \tau_1 \times \tau_2\ \mathsf{list}^\alpha[I]\ \tau \mid \square\tau \mid \tau_1 + \tau_2 \mid C \& \tau \mid C \supset \tau \mid \exists i :: S.\ \tau$
Index terms	$I, L, U, D, \alpha, \beta ::= i \mid b \mid n \mid r \mid I_1 + I_2 \mid I_1 * I_2 \mid I_1 - I_2 \mid max(I_1, I_2)$ $\mid min(I_1, I_2) \mid \frac{I_1}{I_2} \mid \sum_{i=I_1}^{I_n} I \mid log_2(I) \mid \lfloor I \rfloor \mid \lceil I \rceil \mid \{I_i\}_{i \in K} \mid \beta \cup \beta \mid \beta \setminus \beta \mid \beta \cap \beta$
Sort	$S ::= \mathbb{R} \mid \mathbb{N} \mid \mathbb{B} \mid \mathbb{P} \mid \mathbb{L}$
Terms	$t ::= x \mid n \mid r \mid () \mid \lambda x.t \mid \mathsf{fix}\, f(x).t \mid t_1\ t_2 \mid \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \mid \langle t_1, t_2 \rangle \mid \mathsf{inl}\ t$ $\mid \mathsf{inr}\ t \mid \mathsf{case}\ (t, x.t_1, y.t_2) \mid \Lambda.t \mid t\,[] \mid \mathsf{pack}\ t \mid \mathsf{unpack}\ t_1\ \mathsf{as}\ x\ \mathsf{in}\ t_2$
|celim t	$\mathsf{nil} \mid \mathsf{cons}(t_1, t_2) \mid \mathsf{case}\ t\ \mathsf{of\ nil} \rightarrow t_1 \mid h :: tl \rightarrow t_2 \mid \pi_1(t) \mid \pi_2(t) \mid$ $\mid \mathsf{return}\ t \mid \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \mid \mathsf{alloc}\ t_1\ t_2 \mid \mathsf{read}\ t_1\ t_2 \mid \mathsf{updt}\ t_1\ t_2\ t_3$
Values	$v ::= n \mid l \mid r \mid () \mid \lambda x.t \mid \mathsf{fix}\, f(x).t \mid \mathsf{inl}\ v \mid \mathsf{inr}\ v \mid \Lambda.t \mid \langle v_1, v_2 \rangle \mid \mathsf{pack}\ v$ $\mid \mathsf{return}\ t \mid \mathsf{alloc}\ t_1\ t_2 \mid \mathsf{updt}\ t_1\ t_2\ t_3 \mid \mathsf{read}\ t_1\ t_2 \mid \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \mid \mathsf{nil}$ $n \in \mathbb{N},\ r \in \mathbb{R},\ x \in Var,\ i \in iVar,\ \gamma \in iLoc,\ l \in dLoc$

Figure A.1: Syntax of values & expressions of ARel.

Unary Type Env.	$\Omega ::= \emptyset \mid \Omega, x : A$
Relational Type Env.	$\Gamma ::= \emptyset \mid \Gamma, x : \tau$
Sort Env.	$\Delta ::= \emptyset \mid \Delta, i :: S$
Loc Env.	$\Sigma ::= \emptyset \mid \Sigma, \gamma :: \mathbb{L}$
Constraint	$C ::= I_1 = I_2 \mid I_1 < I_2 \mid \neg C \mid I_1 \in I_2$
Constraint Env.	$\Phi_a ::= \top \mid C \wedge \Phi_a$
Assertions	$P, Q ::= \mathsf{empty} \mid \gamma \rightarrow \beta \mid P \star Q \mid P \cup Q$
Unary Heap	$H ::= [] \mid [l \rightarrow z] \mid H_1 \uplus H_2$
array	$z ::= [v_1, \ldots, v_m]$

Figure A.2: Notations of ARel.

A.2 OPERATIONAL SEMANTICS OF AREL

$\boxed{t; H \Downarrow_p^{c,k} v; H_1}$

$$\frac{t \Downarrow^{c_1,k_1} v' \qquad v'; H \Downarrow_f^{c_2,k_2} v; H'}{t; H \Downarrow_p^{c_1+c_2,k_1+k_2} v; H'}\ \text{E-P}$$

$\boxed{t \Downarrow^{c,k} v}$

$$\frac{}{v \Downarrow^{0,0} v}\ \text{E-V} \qquad \frac{t \Downarrow^{c,k} v}{\text{inl}\ t \Downarrow^{c,k} \text{inl}\ v}\ \text{E-INL} \qquad \frac{t \Downarrow^{c,k} v}{\text{inr}\ \ t \Downarrow^{c,k} \text{inr}\ \ v}\ \text{E-INR}$$

$$\frac{t \Downarrow^{c_1,k_1} \text{inl}\ v \qquad t_1[v/x] \Downarrow^{c_2,k_2} v_r}{\text{case}\ (t, x.t_1, y.t_2) \Downarrow^{c_1+c_2+c_{\text{case}},k_1+k_2+1} v_r}\ \text{E-CASEL}$$

$$\frac{t \Downarrow^{c_1,k_1} \text{inr}\ \ v \qquad t_2[v/y] \Downarrow^{c_2,k_2} v_r}{\text{case}\ (t, x.t_1, y.t_2) \Downarrow^{c_1+c_2+c_{\text{case}},k_1+k_2+1} v_r}\ \text{E-CASER}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} \lambda x.t' \qquad t_2 \Downarrow^{c_2,k_2} v \qquad t'[v/x] \Downarrow^{c_3,k_3} v_1}{t_1\ t_2 \Downarrow^{c_1+c_2+c_3+c_{\text{app}},k_1+k_2+k_3+1} v_1}\ \text{E-A}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} v \qquad t_2/[v/x] \Downarrow^{c_2,k_2} v_1}{\text{let}\ x = t_1\ \text{in}\ t_2 \Downarrow^{c_1+c_2+c_{\text{lt}},k_1+k_2+1} v_1}\ \text{E-LT}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} \text{fix}\, f\, x.t' \qquad t_2 \Downarrow^{c_2,k_2} v \qquad t'[\text{fix}\, f\, x.t'/f][v/x] \Downarrow^{c_3,k_3} v_1}{t_1\ t_2 \Downarrow^{c_1+c_2+c_3+c_{\text{app}},k_1+k_2+k_3+1} v_1}\ \text{E-F}$$

Figure A.3: Standard operational semantics of ARel, part 1.

$$\frac{t \Downarrow^{c,k} \Lambda t_b \qquad t_b \Downarrow^{(c',k')} v_r}{t\ [] \Downarrow^{c+c',k+k'} v_r}\ \text{E-IAPP} \qquad \frac{t \Downarrow^{c,k} v}{\mathsf{pack}\ t \Downarrow^{c,k} \mathsf{pack}\ v}\ \text{E-PACK}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} \mathsf{pack}\ v \qquad t_2[v/x] \Downarrow^{(c_2,k_2)} v_r}{\mathsf{unpack}\ t_1\ \mathsf{as}\ x\ \mathsf{in}\ t_2 \Downarrow^{c_1+c_2,k_1+k_2} v_r}\ \text{E-UNPACK}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} \Lambda t_b \qquad t_2 \Downarrow^{(c_2,k_2)} v_2}{\mathsf{cons}(t_1,t_2) \Downarrow^{c_1+c_2,k_1+k_2} \mathsf{cons}(v_1,v_2)}\ \text{E-CONS}$$

$$\frac{t \Downarrow^{c,k} \mathsf{nil} \qquad t_1 \Downarrow^{(c_r,k_r)} v_r}{\text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2 \Downarrow^{c+c_r+c_{caseL},k+k_r+1} v_r}\ \text{E-CASEL-NIL}$$

$$\frac{t \Downarrow^{c,k} \mathsf{cons}(v_1,v_2) \qquad t_2[v_1/h, v_2/tl] \Downarrow^{(c_r,k_r)} v_r}{\text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2 \Downarrow^{c+c_r+c_{caseL},k+k_r+1} v_r}\ \text{E-CASEL-CONS}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} v_1 \qquad t_2 \Downarrow^{(c_2,k_2)} v_2}{\langle t_1,t_2\rangle \Downarrow^{c_1+c_2,k_1+k_2} \langle v_1,v_2\rangle}\ \text{E-PROD} \qquad \frac{t \Downarrow^{c,k} \langle v_1,v_2\rangle}{\pi_1(t) \Downarrow^{c+c_{proj},k+1} v_1}\ \text{E-PROJ1}$$

$$\frac{t \Downarrow^{c,k} \langle v_1,v_2\rangle}{\pi_2(t) \Downarrow^{c+c_{proj},k+1} v_2}\ \text{E-PROJ2} \qquad \frac{t_1 \Downarrow^{c_1,k_1} v_1 \qquad t_2[v_1/x] \Downarrow^{c_r,k_r} v_r}{\mathsf{clet}\ t_1\ \mathsf{as}\ x\ \mathsf{in}\ t_2 \Downarrow^{c_1+c_r,k+k_r} v_r}\ \text{E-CLET}$$

Figure A.4: Standard operational semantics of ARel, part 2.

$\boxed{t; H \Downarrow_f^{c,k} v; H_1}$

$$\frac{t \Downarrow^{c,k} v}{\mathsf{return}\ t; H \Downarrow_f^{c+c_{\mathrm{ret}},k+1} v; H}\ \text{F-T}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} v \qquad v; H \Downarrow_f^{c_2,k_2} v_1; H_1 \qquad t_2[v_1/x] \Downarrow^{c_3,k_3} v_2 \qquad v_2; H_1 \Downarrow_f^{c_4,k_4} v_3; H_2}{\mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2; H \Downarrow_f^{c_1+c_2+c_3+c_4+c_{\mathrm{let}},k_1+k_2+k_3+k_4+1} v_3; H_2}\ \text{F-E}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} l \qquad t_2 \Downarrow^{c_2,k_2} n \qquad t_3 \Downarrow^{c_3,k_3} v}{\mathsf{updt}\ t_1\ t_2\ t_3; H \Downarrow_f^{c_1+c_2+c_3+c_{\mathrm{update}},k_1+k_2+k_3+1} (); H(l)[n] \leftarrow v}\ \text{F-U}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} l \qquad t_2 \Downarrow^{c_2,k_2} n_2 \qquad H(l)[n] = v}{\mathsf{read}\ t_1\ t_2; H \Downarrow_f^{c_1+c_2+c_{\mathrm{read}},k_1+k_2+1} v; H}\ \text{F-R}$$

$$\frac{t_1 \Downarrow^{c_1,k_1} n \qquad t_2 \Downarrow^{c_2,k_2} v \qquad z = [\overbrace{v,\ldots,v}^{n}] \qquad l\ \text{fresh}}{\mathsf{alloc}\ t_1\ t_2; H \Downarrow_f^{c_1+c_2+c_{\mathrm{alloc}},k_1+k_2+1} l; H, l \rightarrow z}\ \text{F-L}$$

Figure A.5: Operational semantics of ARel: forcing evaluation.

A.3 TYPING OF AREL

$$
\begin{array}{rcl}
|.|_{i\in\{1,2\}} & : & \mathsf{Binarytype} \to \mathsf{Unarytype} \\
|\mathrm{int}|_i & = & \mathrm{int} \\
|\mathrm{unit}|_i & = & \mathrm{unit} \\
|\ \{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma} : \tau\ \{Q\}\ |_i & = & \{|P|_i\}\ \exists^{\mathrm{exec}(0,\infty)}\vec{\gamma} : |\tau|_i\ \{|Q|_i\} \\
|\ \tau \overset{\mathrm{diff}(D)}{\longrightarrow} \tau'\ |_i & = & |\tau|_i \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\tau'|_i \\
|\ \mathsf{Array}_\gamma[I]\ \tau\ |_i & = & \mathsf{Array}_\gamma[I]\ |\tau|_i \\
|U\ A|_i & = & A_i \\
|\Box\tau|_i & = & |\tau|_i \\
|\ \mathsf{list}^\alpha[I]\ \tau\ |_i & = & \mathsf{list}[I]\ |\tau|_i \\
|\tau_1 + \tau_2|_i & = & |\tau_1|_i + |\tau_2|_i \\
|C\&\tau|_i & = & C\&|\tau|_i \\
|C \supset \tau|_i & = & C \supset |\tau|_i \\
|\emptyset|_i & = & \emptyset \\
|\Gamma, x:\tau|_i & = & |\Gamma|_i, x:|\tau|_i \\
|\Gamma, x:\tau, x:U(A_1,A_2)|_i & = & |\Gamma|_i, x:A_i \\
|\mathsf{empty}|_i & = & \mathsf{empty} \\
|\gamma_n \to \beta_n|_i & = & \gamma_n \to \mathbb{N}
\end{array}
$$

Figure A.6: Binary to unary type erasure of ARel.

A.3.1 Well-formedness

$\boxed{\vdash \Delta}$

$$\frac{}{\vdash \emptyset}\ \textsc{senv-empty} \qquad \frac{\vdash \Delta \qquad i \in \mathsf{iVar} \qquad S \in \mathsf{Sort}}{\vdash \Delta, i :: S}\ \textsc{senv-iv}$$

Figure A.7: Sort environment well formedness of ARel.

$\boxed{\vdash \Sigma}$

$$\frac{}{\vdash \emptyset}\ \textsc{lenv-empty} \qquad \frac{\vdash \Sigma \qquad \gamma \in \mathsf{iLoc}}{\vdash \Sigma, \gamma :: \mathbb{L}}\ \textsc{lenv-lv}$$

Figure A.8: Location environment well-formedness of ARel.

$\boxed{\Sigma; \Delta \vdash P\ \mathsf{wf}}$

$$\frac{}{\Sigma; \Delta \vdash \mathrm{empty}\ \mathsf{wf}}\ \textsc{wf-as-empty} \qquad \frac{\Sigma; \Delta \vdash \gamma :: \mathbb{L} \qquad \Sigma; \Delta \vdash \beta :: \mathbb{P}}{\Sigma; \Delta \vdash \gamma \to \beta\ \mathsf{wf}}\ \textsc{wf-as-referTo}$$

$$\frac{\Sigma; \Delta \vdash P\ \mathsf{wf} \qquad \Sigma; \Delta \vdash Q\ \mathsf{wf}}{\Sigma; \Delta \vdash P * Q\ \mathsf{wf}}\ \textsc{wf-as-star}$$

Figure A.9: Assertion well-formedness of ARel.

$\boxed{\Delta \vdash C\ \mathsf{wf}}$

$$\frac{\Delta \vdash I_1 :: S \qquad \Delta \vdash I_2 :: S \qquad S \in \{\mathbb{N}, \mathbb{R}\}}{\Delta \vdash I_1 = I_2\ \mathsf{wf}}\ \textsc{wf-cs-eq}$$

$$\frac{\Delta \vdash I_1 :: S \qquad \Delta \vdash I_2 :: S \qquad S \in \{\mathbb{N}, \mathbb{R}\}}{\Delta \vdash I_1 < I_2\ \mathsf{wf}}\ \textsc{wf-cs-lt} \qquad \frac{\Delta \vdash C\ \mathsf{wf}}{\Delta \vdash \neg C\ \mathsf{wf}}\ \textsc{wf-cs-neg}$$

$$\frac{\Delta \vdash I_1 :: S \qquad \Delta \vdash I_2 :: S \qquad S \in \{\mathbb{N}, \mathbb{R}\}}{\Delta \vdash I_1 \in I_2\ \mathsf{wf}}\ \textsc{wf-cs-in}$$

Figure A.10: Constraint well-formedness of ARel.

$\boxed{\Sigma;\Delta \vdash I :: S}$

$$\frac{\Delta(i) = S}{\Sigma;\Delta \vdash i :: S}\ \text{I-VAR} \qquad \frac{\Sigma(\gamma) = \mathbb{L}}{\Sigma;\Delta \vdash \gamma :: \mathbb{L}}\ \text{I-LOC} \qquad \frac{}{\Sigma;\Delta \vdash b :: \mathbb{B}}\ \text{I-BOOL}$$

$$\frac{}{\Sigma;\Delta \vdash n :: \mathbb{N}}\ \text{I-NAT} \qquad\qquad \frac{}{\Sigma;\Delta \vdash r :: \mathbb{R}}\ \text{I-REAL}$$

$$\frac{\Sigma;\Delta \vdash I_1 :: \mathbb{N} \qquad \Sigma;\Delta \vdash I_2 :: \mathbb{N} \qquad \diamond \in \{min, max, +, -, *, \}}{\Sigma;\Delta \vdash (I_1 \diamond I_2) :: \mathbb{N}}\ \text{I-BIN-N}$$

$$\frac{\Sigma;\Delta \vdash I_1 :: \mathbb{R} \qquad \Sigma;\Delta \vdash I_2 :: \mathbb{R} \qquad \diamond \in \{min, max, +, -, *, \}}{\Sigma;\Delta \vdash (I_1 \diamond I_2) :: \mathbb{R}}\ \text{I-BIN-R}$$

$$\frac{\Sigma;\Delta \vdash I :: \mathbb{R} \qquad \circ \in \{\lfloor\ \rfloor, \lceil\ \rceil\}}{\Sigma;\Delta \vdash (\circ I) :: \mathbb{N}}\ \text{I-UNARY-R}$$

$$\frac{\Sigma;\Delta \vdash I :: \mathbb{R}}{\Sigma;\Delta \vdash log_2(I) :: \mathbb{R}}\ \text{I-LOG}$$

$$\frac{\Sigma;\Delta \vdash I_i :: \mathbf{N} \quad \forall i \in K \subseteq \mathbb{N}}{\Sigma;\Delta \vdash \{I_i\}_{i \in K} :: \mathbf{P}}\ \text{I-NATSET}$$

$$\frac{\Sigma;\Delta \vdash \beta_1 :: \mathbf{P} \qquad \Sigma;\Delta \vdash \beta_2 :: \mathbf{P} \quad \diamond \in \{\cup, \cap, /\}}{\Sigma;\Delta \vdash \beta_1 \diamond \beta_2 :: \mathbf{P}}\ \text{I-SET-OP}$$

$$\frac{\Sigma;\Delta \vdash I_1 :: \mathbb{N} \qquad \Sigma;\Delta \vdash I_n :: \mathbb{N} \qquad \Sigma;\Delta, i :: \mathbf{N} \vdash I :: \mathbb{S} \qquad S \in \{\mathbf{N}, \mathbb{R}\}}{\Sigma;\Delta \vdash \sum_{i=I_1}^{I_n} I :: \mathbb{S}}\ \text{I-SUM}$$

Figure A.11: Sorting rules of ARel.

$\boxed{\Sigma;\Delta;\Phi_a \vdash A\ \mathsf{wf}}$

$$\frac{}{\Sigma;\Delta;\Phi_a \vdash \emptyset\ \mathsf{wf}}\ \textsc{wf-a-omega-emptyset}$$

$$\frac{\Sigma;\Delta;\Phi_a \vdash \Omega\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \vdash A\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash \Omega, x : A\ \mathsf{wf}}\ \textsc{wf-a-omega} \qquad \frac{}{\Sigma;\Delta;\Phi_a \vdash unit\ \mathsf{wf}}\ \textsc{wf-a-unit}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \vdash Int\ \mathsf{wf}}\ \textsc{wf-a-int} \qquad \frac{\Sigma;\Delta \vdash I :: \mathbb{N}}{\Sigma;\Delta;\Phi_a \vdash \mathrm{int}[I]\ \mathsf{wf}}\ \textsc{wf-a-int-i}$$

$$\frac{\Sigma;\Delta;\Phi_a \vdash A_1\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \vdash A_2\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash A_1 + A_2\ \mathsf{wf}}\ \textsc{wf-a-sum}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a \vdash A\ \mathsf{wf} \qquad \Sigma;\Delta \vdash \gamma :: \mathbb{L} \\ \Sigma;\Delta \vdash L :: \mathbb{R} \qquad \Sigma;\Delta \vdash U :: \mathbb{R} \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Sigma, \vec{\gamma} : \vec{\mathbb{L}};\Delta \vdash Q\ \mathsf{wf}\end{array}}{\begin{array}{c}\mathrm{exec}(L,U) \\ \Sigma;\Delta;\Phi_a \vdash \{P\}\ \exists \vec{\gamma} : A\ \{Q\}\ \mathsf{wf}\end{array}}\ \textsc{wf-a-monad}$$

$$\frac{\Sigma;\Delta;\Phi_a \vdash A\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \vdash A'\ \mathsf{wf} \qquad \Sigma;\Delta \vdash L :: \mathbb{R} \qquad \Sigma;\Delta \vdash U :: \mathbb{R}}{\begin{array}{c}\mathrm{exec}(L,U) \\ \Sigma;\Delta;\Phi_a \vdash A \longrightarrow A'\ \mathsf{wf}\end{array}}\ \textsc{wf-a-arrow}$$

$$\frac{\Sigma;\Delta \vdash \gamma :: \mathbb{L} \qquad \Sigma;\Delta \vdash I :: \mathbb{N} \qquad \Sigma;\Delta;\Phi_a \vdash A\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash \mathsf{Array}_\gamma[I]\ A\ \mathsf{wf}}\ \textsc{wf-a-array}$$

$$\frac{\Sigma; i :: S, \Delta;\Phi_a \vdash A\ \mathsf{wf} \qquad \Sigma; i :: S, \Delta \vdash L :: \mathbb{R} \qquad \Sigma; i :: S, \Delta \vdash U :: \mathbb{R}}{\begin{array}{c}\mathrm{exec}(L,U) \\ \Sigma;\Delta;\Phi_a \vdash \forall i{::}S.\ A\ \mathsf{wf}\end{array}}\ \textsc{wf-a-forall}$$

$$\frac{\Sigma; i :: S, \Delta;\Phi_a \vdash A\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash \exists i :: S.\ A\ \ \mathsf{wf}}\ \textsc{wf-a-exist} \qquad \frac{\Delta \vdash I :: \mathbb{N} \qquad \Sigma;\Delta;\Phi_a \vdash A\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash \mathsf{list}[I]\ A\ \mathsf{wf}}\ \textsc{wf-a-list}$$

$$\frac{\Delta \vdash C\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \wedge C\ \vdash A\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash C \& A\ \mathsf{wf}}\ \textsc{wf-a-c-and}$$

$$\frac{\Delta \vdash C\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \wedge C \vdash A\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash C \supset A\ \mathsf{wf}}\ \textsc{wf-a-c-imply}$$

Figure A.12: Well-formedness of unary types of ARel.

$\boxed{\Sigma;\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}$

$$\frac{}{\Sigma;\Delta;\Phi_a \vdash \emptyset\ \mathsf{wf}}\ \textsc{wf-r-gamma-emptyset}$$

$$\frac{\Sigma;\Delta;\Phi_a \vdash \Gamma\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash \Gamma, x:\tau\ \mathsf{wf}}\ \textsc{wf-r-gamma} \qquad \frac{}{\Sigma;\Delta;\Phi_a \vdash \mathrm{unit}\ \mathsf{wf}}\ \textsc{wf-r-unit}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \vdash \mathrm{int}\ \mathsf{wf}}\ \textsc{wf-r-int} \qquad \frac{\Sigma;\Delta;\Phi_a \vdash \tau_1\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \vdash \tau_2\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash \tau_1 + \tau_2\ \mathsf{wf}}\ \textsc{wf-r-sum}$$

$$\frac{\Sigma;\Delta;\Phi_a \vdash \tau_1\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \vdash \tau_2\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash \tau_1 \times \tau_2\ \mathsf{wf}}\ \textsc{wf-r-prod}$$

$$\frac{\Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \vdash \tau\ \mathsf{wf} \qquad \Sigma;\Delta,\vec{\gamma}:\vec{\mathbb{L}} \vdash Q\ \mathsf{wf} \qquad \Sigma;\Delta \vdash D :: \mathbb{R}}{\Sigma;\Delta;\Phi_a \vdash \{P\}\ \overset{\mathrm{diff}(D)}{\exists \vec{\gamma}:\tau}\ \{Q\}\ \mathsf{wf}}\ \textsc{wf-r-monad}$$

$$\frac{\Sigma;\Delta;\Phi_a \vdash \tau\ \mathsf{wf} \qquad \Sigma;\Delta;\Phi_a \vdash \tau'\ \mathsf{wf} \qquad \Sigma;\Delta \vdash D :: \mathbb{R}}{\Sigma;\Delta;\Phi_a \vdash \tau \overset{\mathrm{diff}(D)}{\longrightarrow} \tau'\ \mathsf{wf}}\ \textsc{wf-r-arrow}$$

$$\frac{\begin{array}{c}\Sigma; i :: S, \Delta;\Phi_a \vdash \tau\ \mathsf{wf} \\ \Sigma; i :: S, \Delta \vdash D :: \mathbb{R}\end{array}}{\Sigma;\Delta;\Phi_a \vdash \forall i{::}S.\ \tau\ \mathsf{wf}}\ \textsc{wf-r-forall} \qquad \frac{\Sigma; i :: S, \Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a \vdash \exists i :: S.\ \tau\ \ \mathsf{wf}}\ \textsc{wf-r-exist}$$

Figure A.13: Well-formedness of relational types of ARel, part 1.

$$\frac{\Sigma;\Delta \vdash \gamma :: \mathbb{L} \qquad \Sigma;\Delta \vdash I :: \mathbb{N} \qquad \Sigma;\Delta;\Phi_a \vdash \tau \text{ wf}}{\Sigma;\Delta;\Phi_a \vdash \mathsf{Array}_\gamma[I]\ \tau \text{ wf}}\ \text{WF-R-ARRAY}$$

$$\frac{\Sigma;\Delta;\Phi_a \vdash A_1 \text{ wf} \qquad \Sigma;\Delta;\Phi_a \vdash A_2 \text{ wf}}{\Sigma;\Delta;\Phi_a \vdash U\ (A_1, A_2) \text{ wf}}\ \text{WF-R-UA}$$

$$\frac{\Sigma;\Delta \vdash \alpha :: \mathbb{N} \qquad \Sigma;\Delta \vdash I :: \mathbb{N} \qquad \Sigma;\Delta;\Phi_a \vdash \tau \text{ wf}}{\Sigma;\Delta;\Phi_a \vdash \mathsf{list}^\alpha[I]\ \tau \text{ wf}}\ \text{WF-R-LIST}$$

$$\frac{\Delta \vdash C \text{ wf} \qquad \Sigma;\Delta;\Phi_a \wedge C \vdash \tau \text{ wf}}{\Sigma;\Delta;\Phi_a \vdash C \& \tau \text{ wf}}\ \text{WF-R-C-AND}$$

$$\frac{\Delta \vdash C \text{ wf} \qquad \Sigma;\Delta;\Phi_a \wedge C \vdash \tau \text{ wf}}{\Sigma;\Delta;\Phi_a \vdash C \supset \tau \text{ wf}}\ \text{WF-R-C-IMPLY} \qquad \frac{\Sigma;\Delta;\Phi_a \vdash \tau \text{ wf}}{\Sigma;\Delta;\Phi_a \vdash \square\tau \text{ wf}}\ \text{WF-R-SQUARE}$$

Figure A.14: Well-formedness of relational types of ARel, part 2.

$\boxed{\Omega \vdash H \text{ wf}}$

$$\frac{}{\Omega \vdash empty \text{ wf}}\ \text{WF-H-EMPTY} \qquad \frac{\Omega \vdash H \text{ wf} \qquad x \notin dom(H)}{\Omega \vdash (H, x \to k) \text{ wf}}\ \text{WF-H-UPDATE}$$

Figure A.15: Heap well-formedness of ARel.

A.3.2 Unary Typing Rules

$\boxed{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A}$

$$\frac{}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} n : \mathrm{int}}\ \text{U-INT} \qquad \frac{}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} n : \mathrm{int}[n]}\ \text{U-I}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega, x : A \vdash^{U}_{L} t : B}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} \lambda x.t : A \xrightarrow{\mathrm{exec}(L,U)} B}\ \text{U-ABS} \qquad \frac{\Omega(x) = A}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} x : A}\ \text{U-V}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : A \xrightarrow{\mathrm{exec}(L,U)} B \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : A}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+U+U_{app}}_{L_1+L_2+L+L_{app}} t_1\ t_2 : B}\ \text{U-A}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : A \qquad \Sigma;\Delta;\Phi_a;\Omega, x : A \vdash^{U_2}_{L_2} t_2 : B}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+U_{lt}}_{L_1+L_2+L_{lt}} \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 : B}\ \text{U-LT}$$

$$\frac{}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} () : \mathrm{unit}}\ \text{U-UNIT} \qquad \frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : A_2}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \langle t_1, t_2 \rangle : A_1 \times A_2}\ \text{U-PROD}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A_1 \times A_2}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U+U_{proj}}_{L+L_{proj}} \pi_1(t) : A_1}\ \text{U-PROJ1} \qquad \frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A_1 \times A_2}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U+U_{proj}}_{L+L_{proj}} \pi_2(t) : A_2}\ \text{U-PROJ2}$$

Figure A.16: Unary typing judgment of ARel, part 1.

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_L^U t : A \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L,U)} \mathsf{return}\ t : \{P\}\ \exists\gamma.A\ \{P\}}\ \text{U-T}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \{P\}\ \exists\vec{\gamma}_1 : A\ \{P'\}^{\mathrm{exec}(L,U)} \qquad \Sigma;\Delta,\vec{\gamma}_1;\Phi_a;\Omega,x:A \vdash_{L_2}^{U_2} t_2 : \{P'\}\ \exists\vec{\gamma}_2 : B\ \{Q\}^{\mathrm{exec}(L',U')}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L+L'+L_1+L_2+L_l,U+U'+U_1+U_2+U_l)} \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 : \{P\}\ \exists\vec{\gamma}_1,\vec{\gamma}_2 : B\ \{Q\}}\ \text{U-E}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \mathrm{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : A \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L_1+L_2+L_a,U_1+U_2+U_a)} \mathsf{alloc}\ t_1\ t_2 : \{P\}\ \exists\gamma : \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \to \mathbb{N}\}}\ \text{U-L}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \mathsf{Array}_\gamma[I]\ A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : \mathrm{int}[I'] \\ \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_3}^{U_3} t_3 : A \qquad \Delta;\Phi_a \vDash I' \le I \qquad \Delta;\Phi_a \vDash I' \in \beta \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L_1+L_2+L_3+L_u,U_1+U_2+U_3+U_u)} \mathsf{updt}\ t_1\ t_2\ t_3 : \{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta\}}\ \text{U-U}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \mathsf{Array}_\gamma[I]\ A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : \mathrm{int}[I'] \\ \Delta;\Phi_a \vDash I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L_1+L_2+L_r,U_1+U_2+U_r)} \mathsf{read}\ t_1\ t_2 : \{P\}\ \exists\_ : A\ \{P\}}\ \text{U-R}$$

Figure A.17: Unary typing judgment of ARel, part 2.

$$\frac{\Sigma;\Delta;\Phi_a;x:A, f:A \overset{\mathsf{exec}(L,U)}{\longrightarrow} B, \Omega \vdash^{U}_{L} t : B}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{fix}\, f(x).t : A \overset{\mathsf{exec}(L,U)}{\longrightarrow} B}\ \text{U-F}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A_1 \qquad \Delta;\Sigma;\Phi_a \vdash A_2\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} \mathsf{inl}\ t : A_1 + A_2}\ \text{U-INL}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A_2 \qquad \Delta;\Sigma;\Phi_a \vdash A_1\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} \mathsf{inr}\ t : A_1 + A_2}\ \text{U-INR}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t : A_1 + A_2 \\ \Sigma;\Delta;\Phi_a;x:\ A_1, \Omega \vdash^{U_2}_{L_2} t_1 : A \qquad \Sigma;\Delta;\Phi_a;y:\ A_2, \Omega \vdash^{U_2}_{L_2} t_2 : A\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+U_c}_{L_1+L_2+L_c} \mathsf{case}\ (t, x.t_1, y.t_2) : A}\ \text{U-CASE}$$

$$\frac{}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{nill} : \mathsf{list}[0]\ A}\ \text{U-NILL}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : \mathsf{list}[I]\ A}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \mathsf{cons}(t_1, t_2) : \mathsf{list}[I+1]\ A}\ \text{U-CONS}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : \mathsf{list}[n]\ A \qquad \Sigma;\Delta;\Phi_a \wedge n = 0;\Omega \vdash^{U'}_{L'} t_1 : A' \\ \Sigma;i,\Delta;\Phi_a \wedge n = i+1;h:A, tl : \mathsf{list}[i]\ A, \Omega \vdash^{U'}_{L'} t_2 : A'\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \mathsf{case}\ t\ \mathsf{of\ nil} \to t_1 \mid h :: tl \to t_2 : A'}\ \text{U-CASEL}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A \qquad \Sigma;\Delta \vDash U \le U' \qquad \Sigma;\Delta \vDash L' \le L}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t : A}\ \text{COST-TRANS}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A \qquad \Sigma;\Delta;\Phi_a \vDash A \sqsubseteq A' \qquad \Sigma;\Delta \vDash U \le U' \qquad \Sigma;\Delta \vDash L' \le L}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t : A'}\ \text{U-X}$$

Figure A.18: Unary typing judgment of ARel, part 3.

$$\frac{\Sigma; i :: S, \Delta; \Phi_a; \Omega \vdash_L^U t : A \qquad i \notin FIV(\Phi_a; \Omega)}{\Sigma; \Delta; \Phi_a; \Omega \vdash_0^{\overset{\text{exec}(L,U)}{0}} \Lambda.t : \forall i{::}S.\ A}\ \text{U-LAM}$$

$$\frac{\Sigma; \Delta; \Phi_a; \Omega \vdash_{L_1}^{U_1} t : \overset{\text{exec}(L_2,U_2)}{\forall i{::}S.\ A} \qquad \Sigma; \Delta \vdash I :: S}{\Sigma; \Delta; \Phi_a; \Omega \vdash_{L_1+L_2}^{U_1+U_2} t\,[] : A\{I/i\}}\ \text{U-IAPP}$$

$$\frac{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U t : A\{I/i\} \qquad \Sigma; \Delta \vdash I :: S}{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U \mathsf{pack}\, t : \exists i :: S.\ A}\ \text{U-PACK}$$

$$\frac{\Sigma; \Delta; \Phi_a; \Omega \vdash_{L_1}^{U_1} t_1 : \exists i :: S.\ A \qquad \Sigma; i :: S, \Delta; \Phi_a; \Omega \vdash_{L_2}^{U_2} t_2 : A_2 \qquad i \notin FV(\Phi_a; \Omega; A_2, L_2, U_2)}{\Sigma; \Delta; \Phi_a; \Omega \vdash_{L_1+L_2}^{U_1+U_2} \mathsf{unpack}\, t_1\, \mathsf{as}\, x\, \mathsf{in}\ t_2 : A_2}\ \text{U-UNPACK}$$

$$\frac{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U t : C \supset A \qquad \Sigma; \Delta; \Phi_a \vDash C}{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U \mathsf{celim}\ t : A}\ \text{U-CELIM} \qquad \frac{\Sigma; \Delta; \Phi_a \wedge C; \Omega \vdash_L^U t : A}{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U t : C \supset A}\ \text{U-CIMPL}$$

$$\frac{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U t : C \& A_1 \qquad \Sigma; \Delta; \Phi_a \wedge C; x : A_1, \Omega \vdash_{L'}^{U'} t : A_2}{\Sigma; \Delta; \Phi_a; \Gamma \vdash_{L+L'}^{U+U'} \mathsf{clet}\ t\ \mathsf{as}\ x\ \mathsf{in}\ t' : A_2}\ \text{U-ANDE}$$

$$\frac{\Sigma; \Delta; \Phi_a \wedge C; \Omega \vdash_L^U t : A \qquad \Sigma; \Delta; \Phi_a \wedge \neg C; \Omega \vdash_L^U t : A}{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U t : A}\ \text{U-SPLIT}$$

$$\frac{\Sigma; \Delta; \Phi_a \vDash \bot \qquad \vdash \Sigma; \Delta; \Phi_a \vDash \Omega\ \mathsf{wf}}{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U t : C \& A}\ \text{U-CONTRA}$$

$$\frac{\Sigma; \Delta; \Phi_a \wedge C; \Omega \vdash_L^U t : A \qquad \Sigma; \Delta; \Phi_a \vDash C}{\Sigma; \Delta; \Phi_a; \Omega \vdash_L^U t : C \& A}\ \text{U-ANDI}$$

Figure A.19: Unary typing judgment of ARel, part 4.

A.3.3 Relational Typing Rules

$\boxed{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim c : \tau}$

$$\frac{}{\Sigma;\Delta;\Phi_a;\Gamma \vdash n \ominus n \lesssim 0 : \text{int}}\ \text{R-INT} \qquad \frac{}{\Sigma;\Delta;\Phi_a;\Gamma \vdash n \ominus n \lesssim 0 : \text{int}[n]}\ \text{R-I}$$

$$\frac{}{\Sigma;\Delta;\Phi_a;\Gamma \vdash () \ominus () \lesssim 0 : \text{unit}}\ \text{R-UNIT} \qquad \frac{\Gamma(x) = \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash x \ominus x \lesssim 0 : \tau}\ \text{R-V}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma, x:\tau \vdash t_1 \ominus t_2 \lesssim D : \sigma}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \lambda x.t_1 \ominus \lambda x.t_2 \lesssim 0 : \tau \xrightarrow{\text{diff}\,D} \sigma}\ \text{R-ABS}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau \xrightarrow{\text{diff}\,D} \sigma \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \lesssim D + D_1 + D_2 : \sigma}\ \text{R-APP}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau_1 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x:\tau_1 \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \ominus \mathsf{let}\ x = t_1'\ \mathsf{in}\ t_2' \lesssim D_1 + D_2 : \tau_2}\ \text{R-LT}$$

$$\frac{\Sigma;\Delta;\Phi_a;x:\tau, f\ :\tau \xrightarrow{\text{diff}\,D} \sigma, \Gamma \vdash t_1 \ominus t_2 \lesssim D : \sigma}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{fix}\, f(x).t_1 \ominus \mathsf{fix}\, f(x).t_2 \lesssim 0 : \tau \xrightarrow{\text{diff}\,D} \sigma}\ \text{R-FIX}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_1 \qquad \Sigma;\Delta;\Phi_a \vdash \tau_2\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \text{inl}\ t \ominus \text{inl}\ t' \lesssim D : \tau_1 + \tau_2}\ \text{R-INL}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_2 \qquad \Sigma;\Delta;\Phi_a \vdash \tau_1\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \text{inr}\ t \ominus \text{inr}\ t' \lesssim D : \tau_1 + \tau_2}\ \text{R-INR}$$

Figure A.20: Relational typing judgment of ARel, part 1.

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D_1 : \tau_1 + \tau_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x:\tau_1 \vdash t_1 \ominus t_1' \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma, y:\tau_2 \vdash t_2 \ominus t_2' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \text{ case } (t, x.t_1, y.t_2) \ominus \text{ case } (t', x.t_1', y.t_2') \lesssim D_1 + D_2 : \tau} \text{ R-CASE}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash_{L_1}^{U_1} t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash_{L_2}^{U_2} t_2 : A_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim U_1 - L_2 : U(A_1, A_2)} \text{ R-S}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash t \ominus t \lesssim D : \tau \qquad \Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma;\Delta;\Phi_a \models D \leq D'}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t \lesssim D' : \tau'} \text{ R-X}$$

$$\frac{\Sigma;\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \Sigma;\Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau} \text{ R-P}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \forall x \in dom(\Gamma).\Sigma;\Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim 0 : \Box\tau} \text{ R-NC}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau\{I/i\} \qquad \Sigma;\Delta \vdash I :: S}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{pack}\ t_1 \ominus \mathsf{pack}\ t_2 \lesssim D : \exists i :: S.\tau} \text{ R-PACK}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \exists i :: S.\tau_1 \qquad \Sigma;i :: S,\Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_2 \ominus t_2' \lesssim D_1 + D_2 : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma;\tau_2)}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{unpack}\, t_1\, \mathsf{as}\, x\, \mathsf{in}\ t_2 \ominus \mathsf{unpack}\, t_1'\, \mathsf{as}\, x\, \mathsf{in}\ t_2' \lesssim D_1 + D_2 : \tau_2} \text{ R-UNPACK}$$

Figure A.21: Relational typing judgment of ARel, part 2.

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad i \notin FV(\Phi_a;\Gamma)}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \Lambda\ t_1 \ominus \Lambda\ t_2 \lesssim 0 : \forall i{::}S.\ \tau}\ \text{R-ILAM}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \forall i{::}S.\ \tau \qquad \Sigma;\Delta \vdash I :: S}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1\ [] \ominus t_2\ [] \lesssim D + D'[I/i] : \tau\{I/i\}}\ \text{R-IAPP}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : C \supset \tau \qquad \Sigma;\Delta;\Phi_a \vDash C}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{celim}\ t_1 \ominus \mathsf{celim}\ t_2 \lesssim 0 : \tau}\ \text{R-CELIM}$$

$$\frac{\Sigma;\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim 0 : C \supset \tau}\ \text{R-CIMPL}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : C\&\tau_1 \qquad \Sigma;\Delta;\Phi_a \wedge C;x:\tau_1,\Gamma \vdash t_1' \ominus t_2' \lesssim D' : \tau_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{clet}\ t_1\ \mathsf{as}\ x\ \mathsf{in}\ t_1' \ominus \mathsf{clet}\ t_2\ \mathsf{as}\ x\ \mathsf{in}\ t_2' \lesssim D + D' : \tau_2}\ \text{R-ANDE}$$

$$\frac{\Sigma;\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \Sigma;\Delta;\Phi_a \vDash C}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : C\&\tau}\ \text{R-ANDI}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;x:\tau_1, f\ :\tau_1 \xrightarrow{\mathrm{diff}\,D} \tau_2,\Gamma, f\ : U(A_1,A_2) \vdash t_1 \ominus t_2 \lesssim D : \tau_2 \\ \Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash_0^0 \mathsf{Fix}\, f(x).t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash_0^0 \mathsf{Fix}\, f(x).t_2 : A_2\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{Fix}\, f(x).t_1 \ominus \mathsf{Fix}\, f(x).t_2 \lesssim 0 : \tau_1 \xrightarrow{\mathrm{diff}\,D} \tau_2}\ \text{R-FIX-EXT}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau_1 \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \langle t_1, t_2\rangle \ominus \langle t_1', t_2'\rangle \lesssim D_1 + D_2 : \tau_1 \times \tau_2}\ \text{R-PROD}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_1 \times \tau_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \pi_1(t) \ominus \pi_1(t_1') \lesssim D : \tau_1}\ \text{R-PROJ1}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_1 \times \tau_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \pi_2(t) \ominus \pi_2(t_1') \lesssim D : \tau_2}\ \text{R-PROJ2}$$

Figure A.22: Relational typing judgment of ARel, part 3.

$$\frac{\Sigma;\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{nil} \ominus \mathsf{nil} \lesssim D : \mathsf{list}^{\alpha}[0]\ \tau}\ \text{R-NIL}$$

$$\frac{\Sigma;\Delta;\Phi_a\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau \qquad \Sigma;\Delta;\Phi_a\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathsf{list}^{\alpha}[n]\ \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{cons}(t_1,t_2) \ominus \mathsf{cons}(t_1',t_2') \lesssim D_1 + D_2 : \mathsf{list}^{\alpha+1}[n+1]\ \tau}\ \text{R-CONS1}$$

$$\frac{\Sigma;\Delta;\Phi_a\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \Box\tau \qquad \Sigma;\Delta;\Phi_a\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathsf{list}^{\alpha}[n]\ \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{cons}(t_1,t_2) \ominus \mathsf{cons}(t_1',t_2') \lesssim D_1 + D_2 : \mathsf{list}^{\alpha}[n+1]\ \tau}\ \text{R-CONS2}$$

$$\text{R-CASEL}\ \frac{\begin{array}{c}\Sigma;\Delta;\Phi_a\Gamma \vdash t \ominus t' \lesssim D : \mathsf{list}^{\alpha}[n]\ \tau \qquad \Sigma;\Delta;\Phi_a \wedge n = 0\Gamma \vdash t_1 \ominus t_1' \lesssim D' : \tau' \\ \Sigma;i,\Delta;\Phi_a \wedge n = i+1\Gamma, h:\Box\tau, tl:\mathsf{list}^{\alpha}[i]\ \tau \vdash t_2 \ominus t_2' \lesssim D' : \tau' \\ \Sigma;i,B\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = B+1;\Gamma, h:\tau, tl:\mathsf{list}^{\beta}[i]\ \tau \vdash t_2 \ominus t_2' \lesssim D' : \tau'\end{array}}{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \\ \vdash \text{ case } t' \text{ of nil } \to t_1' \mid h :: tl \to t_2' \ominus \text{ case } t' \text{ of nil } \to t_1' \mid h :: tl \to t_2' \lesssim D + D' : \tau'\end{array}}$$

Figure A.23: Relational typing judgment of ARel, part 4.

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ t_1 \ominus \mathsf{return}\ t_2 \overset{\mathrm{diff}(D)}{\lesssim} 0 : \{P\}\ \exists \_.\tau\ \{P\}}\ \text{R-T}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \overset{\mathrm{diff}(D)}{\lesssim} D_1 : \{P\}\ \exists \vec{\gamma}_1.\tau\ \{P'\} \\ \Sigma;\Delta,\vec{\gamma}_1 : \vec{\mathbb{L}};\Phi_a;\Gamma, x : \tau \vdash t_2 \ominus t_2' \overset{\mathrm{diff}(D')}{\lesssim} D_2 : \{P'\}\ \exists \vec{\gamma}_2.\sigma\ \{Q\}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \overset{\mathrm{diff}(D+D'+D_1+D_2)}{\lesssim} 0 : \{P\}\ \exists \vec{\gamma}_1 \vec{\gamma}_2 : \sigma\ \{Q\}}\ \text{R-LET}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathrm{int}[I] \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau \qquad \gamma\ \mathsf{fresh} \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 : \{P\}\ \exists \gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}}\ \text{R-ALLOC}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathrm{int}[I] \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \square\tau \qquad \gamma\ \mathsf{fresh} \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 : \{P\}\ \exists \gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \emptyset\}}\ \text{R-ALLOC-BOX}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 : \{P \star \gamma \to \beta\}\ \exists \_.\tau\ \{P \star \gamma \to \beta\}}\ \text{R-R}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \\ \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta;\Phi_a \models I' \notin \beta \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 : \{P \star \gamma \to \beta\}\ \exists \_.\square\tau\ \{P \star \gamma \to \beta\}}\ \text{R-RB}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 : \tau \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \overset{\mathrm{diff}(D_1+D_2+D_3)}{\lesssim} 0 : \{P \star \gamma \to \beta\}\ \exists \_ : \mathrm{unit}\ \{P \star \gamma \to \beta \cup \{I'\}\}}\ \text{R-U}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 : \square\tau \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \overset{\mathrm{diff}(D_1+D_2+D_3)}{\lesssim} 0 : \{P \star \gamma \to \beta\}\ \exists \_ : \mathrm{unit}\ \{P \star \gamma \to \beta \setminus \{I'\}\}}\ \text{R-UB}$$

Figure A.24: Relational typing judgment of ARel, part 5, impure rules.

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t_2 \ominus t_2' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \ominus t_2' \lesssim U_1 + D_2 + c_{lt} : \tau}\ \text{R-LT-T}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t_1' : A_1' \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1', A_1') \vdash t_2 \ominus t_2' \lesssim D_2 : \tau'}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus \mathsf{let}\ x = t_1'\ \mathsf{in}\ t_2' \lesssim D_2 - L_1 - c_{lt} : \tau'}\ \text{R-T-LT}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \overset{\mathrm{exec}(L,U)}{\longrightarrow} A_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : U(A_1, A_2')}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_2' \lesssim U_1 + U + D_2 + c_{app} : U(A_2, A_2')}\ \text{R-APP-T}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t_1' : A_1' \overset{\mathrm{exec}(L,U)}{\longrightarrow} A_2' \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : U(A_2, A_1')}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_1'\ t_2' \lesssim D_2 - L_1 - L - c_{app} : U(A_2, A_2')}\ \text{R-T-APP}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t : A_1 + A_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t_1 \ominus t' \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma, y : U(A_2, A_2) \vdash t_2 \ominus t' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \ \mathsf{case}\ (t, x.t_1, y.t_2) \ominus t' \lesssim U_1 + D_2 + c_{case} : \tau}\ \text{R-CASE-T}$$

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1'}_{L_1'} t' : A_1 + A_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t \ominus t_1' \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma, y : U(A_2, A_2) \vdash t \ominus t_2' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t \ominus \ \mathsf{case}\ (t', x.t_1', y.t_2') \lesssim D_2 - L_1' - c_{case} : \tau}\ \text{R-T-CASE}$$

Figure A.25: Relational typing judgment of ARel, part 6, Asychonization pure rules.

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash_{L_1}^{U_1} t_1 : \{P_1\}\ \overset{\text{exec}(L,U)}{\exists\vec{\gamma}_1 : A_1}\ \{Q_1\} \\ dom(P) = dom(P_1) \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash_{L_2}^{U_2} t_2' : \{P_2\}\ \overset{\text{exec}(L',U')}{\exists\vec{\gamma}_1 : A_1'}\ \{Q_2\} \\ \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t_2 \ominus t_2' \lesssim D_2 : \{P \sqcup P_1\}\ \overset{\text{diff}(D')}{\exists\vec{\gamma}_1.\tau}\ \{Q\}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus t_2' \lesssim -L_2 : \overset{\text{diff}(U_1+U+(D_2+U_2)+D'+c_{let})}{\{P\}\ \exists\vec{\gamma}_1.\tau}\ \{Q\}}\ \text{R-LET-T}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash_{L_1}^{U_1} t_1' : \{P_1\}\ \overset{\text{exec}(L,U)}{\exists\vec{\gamma}_1 : A_1'}\ \{Q_1\} \\ dom(P) = dom(P_1) \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash_{L_2}^{U_2} t_2 : \{P_2\}\ \overset{\text{exec}(L',U')}{\exists\vec{\gamma}_1 : A_1}\ \{Q_2\} \\ \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1', A_1') \vdash t_2 \ominus t_2' \lesssim D_2 : \{P \sqcup P_1\}\ \overset{\text{diff}(D')}{\exists\vec{\gamma}_1.\tau'}\ \{Q\}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \lesssim U_2 : \overset{\text{diff}(D'+(D_2-L_2)-L_1-L-c_{\text{let}})}{\{P\}\ \exists\vec{\gamma}_1.\tau'}\ \{Q\}}\ \text{R-T-LET}$$

Figure A.26: Relational typing judgment of ARel, part 7, asychonization impure rules.

A.3.4 Subtyping

$\boxed{\Sigma;\Delta;\Phi_a \models A_1 \sqsubseteq A_2}$

$$\frac{\Sigma;\Delta;\Phi_a \models A_1' \sqsubseteq A_1 \qquad \Sigma;\Delta;\Phi_a \models A_2 \sqsubseteq A_2' \qquad \Sigma;\Delta;\Phi_a \models L' \leq L \qquad \Sigma;\Delta;\Phi_a \models U \leq U'}{\Sigma;\Delta;\Phi_a \models A_1 \overset{\text{exec}(L,U)}{\longrightarrow} A_2 \sqsubseteq A_1' \overset{\text{exec}(L',U')}{\longrightarrow} A_2'}\ \text{S-UA}$$

$$\frac{\Sigma; i :: S,\Delta;\Phi_a \models A \sqsubseteq A' \qquad \Sigma; i :: S,\Delta;\Phi_a \models L' \leq L \qquad \Sigma;\Delta;\Phi_a \models U \leq U' \qquad i \notin FV(\Phi_a)}{\Sigma;\Delta;\Phi_a \models \forall i{::}S.\ A \sqsubseteq \forall i{::}S.\ A'}\ \text{S-A-FORALL}$$

$$\frac{\Sigma;\Delta;\Phi_a \models A \sqsubseteq A' \qquad \Sigma;\Delta;\Phi_a \models L' \leq L \qquad \Sigma;\Delta;\Phi_a \models U \leq U' \qquad \Sigma;\Delta;\Phi_a \models P \sqsubseteq P' \qquad \Sigma,\vec{\gamma}_1;\Delta;\Phi_a \models Q' \sqsubseteq Q \qquad \Sigma;\Delta;\Phi_a \models \vec{\gamma}_1 \subseteq \vec{\gamma}_2}{\Sigma;\Delta;\Phi_a \models \{P\}\ \exists \vec{\gamma}_1 : A\ \{Q\} \sqsubseteq \{P'\}\ \exists \vec{\gamma}_2 : A'\ \{Q'\}}\ \text{S-UM}$$

$$\frac{\Sigma;\Delta;\Phi_a \models A_1 \sqsubseteq A_1' \qquad \Sigma;\Delta;\Phi_a \models A_2 \sqsubseteq A_2'}{\Sigma;\Delta;\Phi_a \models A_1 \times A_2 \sqsubseteq A_1' \times A_2'}\ \text{S-A-PROD}$$

$$\frac{\Sigma;\Delta;\Phi_a \models A_1 \sqsubseteq A_1' \qquad \Sigma;\Delta;\Phi_a \models A_2 \sqsubseteq A_2'}{\Sigma;\Delta;\Phi_a \models A_1 + A_2 \sqsubseteq A_1' + A_2'}\ \text{S-A-SUM}$$

Figure A.27: Unary subtyping rules of ARel, part 1.

$$\frac{\Sigma;\Delta;\Phi_a \models A \sqsubseteq A' \qquad \Sigma;\Delta;\Phi_a \models I{=}I'}{\Sigma;\Delta;\Phi_a \models \mathsf{Array}_\gamma[I]\ A \sqsubseteq \mathsf{Array}_\gamma[I']\ A'}\ \text{S-A-ARRAY}$$

$$\frac{\Sigma;\Delta;\Phi_a \models A \sqsubseteq A' \qquad \Sigma;\Delta;\Phi_a \models I{=}I'}{\Sigma;\Delta;\Phi_a \models \mathsf{list}[I]\ A \sqsubseteq \mathsf{list}[I']\ A'}\ \text{S-A-LIST}$$

$$\frac{\Sigma; i :: S, \Delta;\Phi_a \models A \sqsubseteq A' \qquad i \notin FV(\Phi_a)}{\Sigma;\Delta;\Phi_a \models \exists i :: S.\ A \sqsubseteq \exists i :: S.\ A'}\ \text{S-A-EXIST}$$

$$\frac{\Sigma; f\Delta;\Phi_a \models A \sqsubseteq A' \qquad \Sigma;\Delta;\Phi_a \wedge C \models C'}{\Sigma;\Delta;\Phi_a \models C\&A \sqsubseteq C'\&A'}\ \text{S-A-CAND}$$

$$\frac{\Sigma;\Delta;\Phi_a \models A \sqsubseteq A' \qquad \Sigma;\Delta;\Phi_a \wedge C' \models C}{\Sigma;\Delta;\Phi_a \models C \supset A \sqsubseteq C' \supset A'}\ \text{S-A-CIMPL} \qquad \frac{}{\Sigma;\Delta;\Phi_a \models A \sqsubseteq A}\ \text{S-A-REFL}$$

$$\frac{\Sigma;\Delta;\Phi_a \models A_1 \sqsubseteq A_2 \qquad \Sigma;\Delta;\Phi_a \models A_2 \sqsubseteq A_3}{\Sigma;\Delta;\Phi_a \models A_1 \sqsubseteq A_3}\ \text{S-A-TRAN}$$

Figure A.28: Unary subtyping rules of ARel, part 2.

$\boxed{\Sigma;\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2}$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \text{int} \sqsubseteq \Box U(\text{int},\text{int})}\ \text{S-R-INT-BOX}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \text{int}[I] \sqsubseteq \Box U(\text{int}[I],\text{int}[I])}\ \text{S-R-INT-I-BOX}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box U(\text{int},\text{int}) \sqsubseteq \Box\text{int}}\ \text{S-R-UINT-BOX}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \text{unit} \sqsubseteq \Box\text{unit}}\ \text{S-R-UNIT-BOX}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau_1' \sqsubseteq \tau_1 \qquad \Sigma;\Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2' \qquad \Sigma;\Delta;\Phi_a \models c \leq c'}{\Sigma;\Delta;\Phi_a \models \tau_1 \overset{\text{diff}\,c}{\longrightarrow} \tau_2 \sqsubseteq \tau_1' \overset{\text{diff}\,c'}{\longrightarrow} \tau_2'}\ \text{S-R-DIFF}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box(\tau_1 \overset{\text{diff}\,c}{\longrightarrow} \tau_2) \sqsubseteq \Box\tau_1 \overset{\text{diff}\,0}{\longrightarrow} \Box\tau_2}\ \text{S-R-BD}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models U(A_1 \overset{\text{exec}(L,U)}{\longrightarrow} A_2, A_1' \overset{\text{exec}(L',U')}{\longrightarrow} A_2') \sqsubseteq U(A_1, A_1') \overset{\text{diff}\,U-L'}{\longrightarrow} U(A_2, A_2')}\ \text{S-R-EXECDIFF}$$

$$\frac{\Sigma; i :: S,\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma; i :: S,\Delta;\Phi_a \models c \leq c' \qquad i \notin FV(\Phi_a)}{\Sigma;\Delta;\Phi_a \models \forall i{::}S.\ \tau \sqsubseteq \forall i{::}S.\ \tau_1'}\ \text{S-R-FORALL-DIFF}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box(\forall i{::}S.\ \tau) \sqsubseteq \forall i{::}S.\ \Box\tau}\ \text{S-R-FORALL-BOX}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models U(\text{unit},\text{unit}) \sqsubseteq \text{unit}}\ \text{S-R-UNIT}$$

Figure A.29: Relational subtyping rules of ARel, part 1.

$$\frac{}{\Sigma;\Delta;\Phi_a \models U(\forall i{::}S.\ A, \forall i{::}S.\ A') \sqsubseteq \forall i{::}S.\ U(A, A')}\ \text{S-R-FORALL-U}$$

$$\text{(exec}(L,U)\text{, exec}(L',U')\text{, diff}(U-L')$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_1' \qquad \Sigma;\Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2'}{\Sigma;\Delta;\Phi_a \models \tau_1 \times \tau_2 \sqsubseteq \tau_1' \times \tau_2'}\ \text{S-R-PROD}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box\tau_1 \times \Box\tau_2 \sqsubseteq \Box(\tau_1 \times \tau_2)}\ \text{S-R-PROD-BOX}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models U(A_1 \times A_2, A_1' \times A_2') \sqsubseteq U(A_1, A_1') \times U(A_2, A_2')}\ \text{S-R-PROD-U}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_1' \qquad \Sigma;\Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2'}{\Sigma;\Delta;\Phi_a \models \tau_1 + \tau_2 \sqsubseteq \tau_1' + \tau_2'}\ \text{S-R-SUM}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box\tau_1 + \Box\tau_2 \sqsubseteq \Box(\tau_1 + \tau_2)}\ \text{S-R-SUM-BOX}$$

$$\frac{\Sigma; i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad i \notin FV(\Phi_a)}{\Sigma;\Delta;\Phi_a \models \exists i :: S.\ \tau \sqsubseteq \exists i :: S.\ \tau'}\ \text{S-R-EXIST}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \exists i :: S.\ \Box\tau \sqsubseteq \Box(\exists i :: S.\ \tau)}\ \text{S-R-EXIST-BOX}$$

Figure A.30: Relational subtyping rules of ARel, part 2.

$$\frac{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma;\Delta;\Phi_a \models I{=}I' \qquad \Sigma;\Delta;\Phi_a \models \gamma{=}\gamma'}{\Sigma;\Delta;\Phi_a \models \mathsf{Array}_{\gamma}[I]\ \tau \sqsubseteq \mathsf{Array}_{\gamma'}[I']\ \tau'}\ \text{S-R-ARRAY}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma;\Delta;\Phi_a \models I{=}I'\ \Sigma;\Delta;\Phi_a \models \alpha{\leq}\alpha'}{\Sigma;\Delta;\Phi_a \models \mathsf{list}^{\alpha}[I]\ \tau \sqsubseteq \mathsf{list}^{\alpha'}[I']\ \tau'}\ \text{S-R-LIST}$$

$$\frac{\Sigma;\Delta;\Phi_a \models I{=}0}{\Sigma;\Delta;\Phi_a \models \mathsf{list}^{\alpha}[I]\ \tau \sqsubseteq \mathsf{list}^{\alpha}[I]\ \Box\tau}\ \text{S-R-LIST2}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \mathsf{list}^{\alpha}[I]\ \Box\tau \sqsubseteq \Box(\mathsf{list}^{\alpha}[I]\ \tau)}\ \text{S-R-LISTB}$$

$$\frac{\Sigma;\Delta;\Phi_a \models A_1 \sqsubseteq A_1' \qquad \Sigma;\Delta;\Phi_a \models A_2 \sqsubseteq A_2'}{\Sigma;\Delta;\Phi_a \models U\ (A_1, A_2) \sqsubseteq U\ (A_1', A_2')}\ \text{S-R-UA}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau_1' \sqsubseteq \tau_1 \qquad \Sigma;\Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta;\Phi_a \models D{\leq}D'}{\Sigma;\Delta;\Phi_a \models \Box(\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2) \sqsubseteq \Box\tau_1' \xrightarrow{\mathrm{diff}(D')} \Box\tau_2'}\ \text{S-R-A}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma;\Delta;\Phi_a \wedge C \models C'}{\Sigma;\Delta;\Phi_a \models C\&\tau \sqsubseteq C'\&\tau'}\ \text{S-R-C-AND}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models C\&\Box\tau \sqsubseteq \Box(C\&\tau)}\ \text{S-R-C-AND-BOX}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models U(C \supset A, C \supset A') \sqsubseteq C \supset U(A, A')}\ \text{S-R-CIMPL-U}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Sigma;\Delta;\Phi_a \wedge C' \models C}{\Sigma;\Delta;\Phi_a \models C \supset \tau \sqsubseteq C' \supset \tau'}\ \text{S-R-C-IMPL}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box(C \supset \tau) \sqsubseteq C \supset \Box\tau}\ \text{S-R-C-IMPL-BOX}$$

Figure A.31: Relational subtyping rules of ARel, part 3.

$$\frac{}{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau}\ \text{S-R-REFL} \qquad \frac{}{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq U(|\tau|_1,|\tau|_2)}\ \text{S-R-W}$$

$$\frac{}{\Sigma;\Delta;\Phi_a \models \Box\tau \sqsubseteq \tau}\ \text{S-R-T} \qquad \frac{}{\Sigma;\Delta;\Phi_a \models \Box\tau \sqsubseteq \Box\Box\tau}\ \text{S-R-D}$$

$$\frac{\Sigma;\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2}{\Sigma;\Delta;\Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2}\ \text{S-R-BB} \qquad \frac{\Sigma;\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2 \qquad \Sigma;\Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_3}{\Sigma;\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_3}\ \text{S-R-TRAN}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \\ \Delta;\Phi_a \models c \leq c' \qquad \Delta;\Phi_a \models P' \subseteq P \qquad \Delta;\Phi_a \models \vec{\gamma}_1 \subseteq \vec{\gamma}_2 \qquad \Sigma,\vec{\gamma}_1 : \vec{\mathbb{L}};\Delta;\Phi_a \models Q \subseteq Q'\end{array}}{\Sigma;\Delta;\Phi_a \models \{P\}\ \overset{\text{diff}(c)}{\exists\vec{\gamma}_1 : \tau}\ \{Q\} \sqsubseteq \{P'\}\ \overset{\text{diff}(c')}{\exists\vec{\gamma}_2 : \tau'}\ \{Q'\}}\ \text{S-RM}$$

$$\frac{}{\begin{array}{l}\Sigma;\Delta;\Phi_a \models U\ (\{\gamma_i \to T_i\}\ \overset{\text{exec}(L,U)}{\exists\vec{\gamma}'_1 : A_1}\ \{Q_1\}, \{\gamma_i \to T'_i\}\ \overset{\text{exec}(L',U')}{\exists\vec{\gamma}'_2 : A_1}\ \{Q_2\}) \sqsubseteq \\ \qquad \{\gamma_i \to \beta_i\}\ \overset{\text{diff}(U-L')}{\exists\vec{\gamma}_1,\vec{\gamma}_2 : U(A_1,A_2)}\ \{\gamma_i \to \beta_i \cup T_i \cup T'_i\}\end{array}}\ \text{S-RUM}$$

$$\begin{array}{rcl} P \subseteq P' & \Leftrightarrow & P = \{\gamma_1 \to \beta_1, \gamma_2 \to \beta_2, \ldots, \gamma_n \to \beta_n\} \\ & & P' = \{\gamma_1 \to \beta'_1, \gamma_2 \to \beta'_2, \ldots, \gamma_n \to \beta'_n\} \\ & & \forall i \in \{1,\ldots,n\}.\beta_i \subseteq \beta'_i \end{array}$$

Figure A.32: Relational subtyping rules of ARel, part 4.

A.4 INTERPRETATION OF TYPES

$$
\begin{array}{ll}
[\![\mathrm{int}]\!]_{g,k} & = \{\, n \mid n \in \mathbb{N} \,\} \\
[\![\mathrm{unit}]\!]_{g,k} & = \{\, () \,\} \\
[\![\mathrm{int}[I]]\!]_{g,k} & = \{\, n \mid I = n \,\} \\
[\![\mathsf{Array}_\gamma[I]\ A]\!]_{g,k} & = \{\, l \mid I = n \ \wedge\ g(\gamma) = (l, A, n) \,\} \\
\overset{\mathrm{exec}(L,U)}{[\![A \longrightarrow A']\!]_{g,k}} & = \left\{ \mathsf{fix}\, f(x).t \;\middle|\; \begin{array}{l} \forall k' < k,\ g' \supseteq g.\ \forall v.\ v \in [\![A]\!]_{g',k'} \implies \\ \qquad t[v/x][\mathsf{fix}\, f(x).t/f] \in [\![A']\!]^{e}_{g',k',(L,U)} \end{array} \right\} \\
\overset{\mathrm{exec}(L,U)}{[\![\{P\}\ \exists\vec{\gamma}.A\ \{Q\}]\!]_{g,k}} & = \left\{ v \;\middle|\; \begin{array}{l} \forall g_1 \supseteq g.\ \forall k_1 \le k,\ k_2 < k_1.\ \forall c.\ \forall H. \\ H \vDash_{g_1,k_1} P \wedge\ v; H \Downarrow_f^{c,k_2} \implies \\ \qquad \exists g_2 \supseteq g_1.\ \exists H_1,\ v_1,\ \vec{\gamma}.\ v; H \Downarrow_f^{c,k_2} v_1; H_1\ \wedge \\ L \le c \le U\ \wedge\ H_1 \vDash_{g_2,k_1-k_2} Q\ \wedge\ v_1 \in [\![A]\!]_{g_2,k_1-k_2}\ \wedge \\ \Big(\exists n.\ P = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\}\ \wedge \\ \quad \forall i \in [1,n].\ g(\gamma_i) = (l_i, A, m) \implies \\ \quad \forall j. H[l_i][j] \neq H_1[l_i][j] \implies j \in T_i\Big) \end{array} \right\} \\
\overset{\mathrm{exec}(L,U)}{[\![\forall i{::}S.\ A]\!]_{g,k}} & = \{\, \Lambda.t \mid \forall I.\ \vdash I :: S\ .\ \wedge\ t \in [\![A[I/i]]\!]^{E,(L[I/i],U[I/i])}_{g,k-1} \,\} \\
[\![\exists i :: S.\ A\ ]\!]_{g,k} & = \{\, \mathsf{pack}\, v \mid \exists I.\ \vdash I :: S\ \wedge\ v \in [\![A[I/i]]\!]_{g,k-1} \,\} \\
[\![A_1 + A_2]\!]_{g,k} & = \{\, \mathrm{inl}\ v \mid v \in [\![A_1]\!]_{g,k} \,\} \cup \{\, \mathrm{inr}\ v \mid v \in [\![A_2]\!]_{g,k} \,\} \\
[\![C \supset A]\!]_{g,k} & = \{\, v \mid \nvDash C\ \vee\ v \in [\![A]\!]_{g,k-1} \,\} \\
[\![C \& A]\!]_{g,k} & = \{\, v \mid \vDash C\ \wedge\ v \in [\![A]\!]_{g,k-1} \,\} \\
[\![A]\!]^{e}_{g,k,(L,U)} & = \{\, t \mid \forall v,\ k',,\ k' \le k\ \wedge\ t \Downarrow^{c,k'} v \Rightarrow\ v \in [\![A]\!]_{g,k-k'}\ \wedge\ L \le c \le U \,\} \\
[\![\cdot]\!]_{g,k} & = \{\, \emptyset \,\} \\
[\![\Omega, x : A]\!]_{g,k} & = \{\, (\sigma[v/x]) \mid \sigma \in [\![\Omega]\!]_{g,k}\ \wedge\ v \in [\![A]\!]_{g,k} \,\}
\end{array}
$$

Figure A.33: Unary interpretation of types.

$$
\begin{aligned}
(\!|\mathsf{int}|\!)_{G,k} &= \{\, (n_1, n_2) \mid n_1 = n_2 \,\} \\
(\!|\mathsf{unit}|\!)_{G,k} &= \{\, ((), ()) \,\} \\
(\!|\mathsf{int}[I]|\!)_{G,k} &= \{\, (n, n) \mid I = n \,\} \\
(\!|\mathsf{Array}_\gamma[I]\ \tau|\!)_{G,k} &= \{\, (l_1, l_2) \mid I = n \ \wedge\ G(\gamma) = (l_1, l_2, \tau, n) \,\} \\
(\!|\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2|\!)_{G,k} &= \left\{ \begin{array}{c} (\ \mathsf{fix}\, f(x).\, t_1, \\ \mathsf{fix}\, f(x).\, t_2\ ) \end{array} \;\middle|\; \begin{array}{l} \forall G' \supseteq G.\ \forall k' \le k-1.\ \forall v_1,\ v_2. \\ (v_1, v_2) \in (\!|\tau_1|\!)_{G',k'} \implies \\ \qquad (t_1[v_1/x][\mathsf{fix}\, f(x).\, t_1/f], \\ t_2[v_2/x][\mathsf{fix}\, f(x).\, t_2/f]) \in (\!|\tau_2|\!)^e_{G',k',D} \ \wedge \\ \forall j.\ \Big(\mathsf{fix}\, f(x).\, t_1 \in [\![|\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\tau_2|_1]\!]_{G|_1,j} \ \wedge \\ \mathsf{fix}\, f(x).\, t_2 \in [\![|\tau_1|_2 \xrightarrow{\mathrm{exec}(0,\infty)} |\tau_2|_2]\!]_{G|_2,j}\Big) \end{array} \right\} \\
(\!|\{P\}\ \exists \vec{\gamma}.\tau\ \{Q\}|\!)^{\mathrm{diff}(D)}_{G,k} &= \left\{ (v_1, v_2) \;\middle|\; \begin{array}{l} \forall G_1 \supseteq G.\ \forall k_1 \le k,\ k_2 < k_1,\ k_3,\ c_1,\ c_2,\ H_1,\ H_2. \\ (H_1, H_2) \models_{G_1,k_1} P \ \wedge\ v_1; H_1 \Downarrow_f^{c_1,k_2} \ \wedge\ v_2; H_2 \Downarrow_f^{c_2,k_3} \\ \implies \qquad \exists G_2 \supseteq G_1,\ H_1',\ H_2',\ v_1',\ v_2'. \\ \Big(v_1; H_1 \Downarrow_f^{c_1,k_2} v_1'; H_1' \ \wedge\ v_2; H_2 \Downarrow_f^{c_2,k_3} v_2'; H_2' \ \wedge \\ \quad (H_1', H_2') \models_{G_2,k_1-k_2} Q \ \wedge\ (v_1', v_2') \in (\!|\tau|\!)_{G_2,k_1-k_2} \ \wedge \\ \quad c_1 - c_2 \le D \Big) \end{array} \right\} \\
(\!|U(A_1, A_2)|\!)_{G,k} &= \{\, (v_1, v_2) \mid \forall k'.\ v_1 \in [\![A_1]\!]_{G|_1,k'} \ \wedge\ v_2 \in [\![A_2]\!]_{G|_2,k'} \,\} \\
(\!|\forall i{::}S.\ \tau|\!)^{\mathrm{diff}(D)}_{G,k} &= \left\{ (\Lambda.t_1, \Lambda.t_2) \;\middle|\; \begin{array}{l} \forall I.\ \vdash I :: S \ \wedge\ (t_1, t_2) \in (\!|\tau[I/i]|\!)^{E,D[I/i]}_{G,k-1} \ \wedge \\ \forall j.\ t_1 \in [\![|\tau|_1]\!]^{E,(0,\infty)}_{G|_1,j} \ \wedge\ t_2 \in [\![|\tau|_2]\!]^{E,(0,\infty)}_{G|_2,j} \end{array} \right\} \\
(\!|\exists i :: S.\ \tau\ |\!)_{G,k} &= \{\, (\mathsf{pack}\, v_1, \mathsf{pack}\, v_2) \mid \exists I.\ \vdash I :: S \ \wedge\ (v_1, v_2) \in (\!|\tau[I/i]|\!)_{G,k-1} \,\} \\
(\!|\tau_1 + \tau_2|\!)_{G,k} &= \{\, (\mathsf{inl}\ v_1, \mathsf{inl}\ v_2) \mid (v_1, v_2) \in (\!|\tau_1|\!)_{G,k-1} \ \vee\ (v_1, v_2) \in (\!|\tau_2|\!)_{G,k-1} \} \\
(\!|\Box\tau|\!)_{G,k} &= \{\, (v, v) \mid (v, v) \in (\!|\tau|\!)_{G,k} \,\} \\
(\!|C \supset \tau|\!)_{G,k} &= \{\, (v_1, v_2) \mid \not\models C \ \vee\ (v_1, v_2) \in (\!|\tau|\!)_{G,k-1} \,\} \\
(\!|C \& \tau|\!)_{G,k} &= \{\, (v_1, v_2) \mid \models C \ \wedge\ (v_1, v_2) \in (\!|\tau|\!)_{G,k-1} \,\} \\
(\!|\tau|\!)^e_{G,k,D} &= \left\{ (t_1, t_2) \;\middle|\; \begin{array}{l} \forall k_1 \le k.\ \forall k_2,\ v_1,\ v_2,\ c_1,\ c_2. \\ (t_1 \Downarrow^{c_1,k_1} v_1 \ \wedge\ t_2 \Downarrow^{c_2,k_2} v_2) \\ \implies \qquad (v_1, v_2) \in (\!|\tau|\!)_{G,k-k_1} \ \wedge\ c_1 - c_2 \le D \end{array} \right\} \\
(\!|\cdot|\!)_{G,k} &= \{\, \emptyset \,\} \\
(\!|\Gamma|\!)_{G,k} &= \{\, (\sigma_1, \sigma_2) \mid \forall x \in dom(\Gamma).\ \forall \tau.\ x : \tau \in \Gamma.\ (\sigma_1(x), \sigma_2(x)) \in (\!|\tau|\!)_{G,k} \,\}
\end{aligned}
$$

Figure A.34: Relational interpretation of types.

$$
\begin{array}{rll}
H \vDash_{g,k} empty & \text{iff} & \text{true} \\
H \vDash_{g,k} (\gamma \rightarrow \beta) & \text{iff} & \exists l, A, n : g(\gamma) = (l, A, n) \\
 & & \wedge \left(\forall i \leq n. \left(H[l][i]\right) \in (\!|A|\!)_{g,k-1}\right) \\
H \vDash_{g,k} (P * Q) & \text{iff} & \exists H', H'', g', g'' : \\
 & & \left(H = H' \uplus H''\right) \wedge \left(g = g' \uplus g''\right) \\
 & & \wedge \left(H' \vDash_{g',k} P\right) \wedge \left(H'' \vDash_{g'',k} Q\right) \\
v; H \Downarrow_f^{c,k} & \triangleq & \exists v', H'. v; H \Downarrow_f^{c,k} v'; H' \\
(H_1, H_2) \vDash_{G,k} empty & \text{iff} & \text{true} \\
(H_1, H_2) \vDash_{G,k} (\gamma \mapsto \beta) & \text{iff} & \exists l_1, l_2, \tau, n : G(\gamma) = (l_1, l_2, \tau, n) \\
 & & \wedge \left(\forall i \leq n. \left(H_1[l_1][i], H_2[l_2][i]\right) \in (\!|\tau|\!)_{G,k-1}\right) \\
 & & \wedge \left(\forall i \leq n. \left(H_1[l_1][i] \neq H_2[l_2][i] \Rightarrow i \in \beta\right)\right) \\
(H_1, H_2) \vDash_{G,k} (P * Q) & \text{iff} & \exists H_1', H_1'', H_2', H_2'', G', G'' : \\
 & & \left(H_1 = H_1' \uplus H_1''\right) \wedge \left(H_2 = H_2' \uplus H_2''\right) \wedge \left(G = G' \uplus G''\right) \\
 & & \wedge \left((H_1', H_2') \vDash_{G',k} P\right) \wedge \left((H_1'', H_2'') \vDash_{G'',k} Q\right) \\
G(\gamma) = (l_1, l_2, \tau, n), & & G|_1(\gamma) = (l_1, |\tau|_1, n), G|_2(\gamma) = (l_2, |\tau|_2, n)
\end{array}
$$

Figure A.35: Interpretation of heap assertion.

A.5 METATHEORY OF AREL

We use $\lambda x.t$ as syntatic sugar for $\mathrm{fix}\, f(x).t$ when f does not show in t.

Lemma 1 (Monotonicity)**.**

1. *If $k' \le k$ and $G \subseteq G'$, then $(\!|\tau|\!)_{G,k} \subseteq (\!|\tau|\!)_{G',k'}$.*

2. *If $k' \le k$ and $g \subseteq g'$, then $[\![A]\!]_{g,k} \subseteq [\![A]\!]_{g',k'}$.*

3. *If $k' \le k$ and $G \subseteq G'$, then $(\!|\tau|\!)^{E,D}_{G,k} \subseteq (\!|\tau|\!)^{E,D}_{G',k'}$.*

4. *If $k' \le k$ and $g \subseteq g'$, then $[\![A]\!]^{E,(L,U)}_{g,k} \subseteq [\![A]\!]^{E,(L,U)}_{g',k'}$.*

5. *If $k' \le k$ and $G \subseteq G'$, then $(\!|\Gamma|\!)_{G,k} \subseteq (\!|\Gamma|\!)_{G',k'}$*

6. *If $k' \le k$ and $g \subseteq g'$, then $[\![\Omega]\!]_{g,k} \subseteq [\![\Omega]\!]_{g',k'}$*

Proof. (1,3) and (2,4) are proved simultaneously by induction on τ and induction hypothesis,(5,6) follows from (1,2). □

Lemma 2 (Heap extension)**.** *If $(H_1, H_2) \vDash_{G,k} P$, then for any H_1', H_2', $(H_1 \uplus H_1', H_2 \uplus H_2') \vDash_{G,k} P$.*

Proof. It is simply proved by case analysis on P. □

Lemma 3 (Evaluation cost soundness)**.**

1. *If $\vdash^U_L t : A$ and $t \Downarrow^c v$, then $L \le c \le U$.*
2. *If $\vdash^U_L t : \{P\}\ \overset{\mathrm{exec}(L',U')}{\exists \gamma : A}\ \{Q\}$, and $t \Downarrow^c v$, and there exists H such that $H \vDash_{g,k} P$ and $v; H \Downarrow^{c'}_f v', H'$, then $L' \le c' \le U'$.*

3. *If* $\vdash t_1 \ominus t_2 \lesssim D : \tau$ *and* $t_1 \Downarrow^{c_1} v_1$ *and* $t_2 \Downarrow^{c_2} v_2$, *then* $c_1 - c_2 \leq D$.

4. *If* $\vdash t_1 \ominus t_2 \lesssim D : \{P\}\ \overset{\mathrm{diff}(D')}{\exists \gamma : \tau}\ \{Q\}$, *and* $t_1 \Downarrow^{c_1} v_1$ *and* $t_2 \Downarrow^{c_2} v_2$, *and there exist heaps* H_1, H_2 *such that* $H_1, H_2 \vDash_{G,k} P$, *and* $v_1; H_1 \Downarrow_f^{c_1'} v_1'; H_1'$, $v_2; H_2 \Downarrow_f^{c_2'} v_2'; H_2'$, *then* $c_1' - c_2' \leq D'$.

Proof. (1) is directly proved from the definition of unary expression interpretation. If $\vdash_L^U t : A$, then $t \in [\![A]\!]_{G,k}^{E,(L,U)}$, unfold the definition, we get : $L \leq c \leq U$ and $t \Downarrow^c v$.
(2) $t \in [\![\{P\}\ \overset{\mathrm{exec}(L',U')}{\exists \gamma : A}\ \{Q\}]\!]_{G,k}^{E,(L,U)}$, $t \Downarrow^{c_1} v'$ and $v' \Downarrow_f^{c_2} v$ unfold the definition, we know $L' \leq c_2 \leq U'$.
(3) From the fundamental theory, If $\vdash t_1 \ominus t_2 \lesssim D : \tau$, we know closed terms (t_1, t_2) in the interpretation of relational type τ, $(\!|\tau|\!)_{G,k}^{E,D}$, unfold the definition, $c_1 - c_2 \leq D$ is proved.
(4) From the fundamental theory, If $\vdash t_1 \ominus t_2 \lesssim D : \{P\}\ \overset{\mathrm{diff}(D')}{\exists \gamma : \tau}\ \{Q\}$, we know closed terms (t_1, t_2) in the interpretation $(\!|\{P\}\ \overset{\mathrm{diff}(D')}{\exists \gamma : \tau}\ \{Q\}|\!)_{G,k}^{E,D}$, unfold it, $c_1' - c_2' \leq D'$ is proved.

□

Lemma 4 (Heap evaluation extension)**.** *If* $H; t \Downarrow_f^{c,k} H_1; v$, *then for any* H', $H \uplus H'; t \Downarrow_f^{c,k} H_1 \uplus H'; v$.

Proof. The proof is by induction on evaluation derivation on operation semantics. □

Lemma 5 (Well-formedness)**.**

1. *If* $\Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$, *then* $FV(t_1) \subseteq dom(\Gamma)$ *and* $FV(t_2) \subseteq dom(\Gamma)$.

2. *If* $\Delta; \Phi_a; \Omega \vdash_L^U t : A$, *then* $FV(t) \subseteq dom(\Omega)$.

Proof. The proof is by induction on the typing derivations. □

Lemma 6 (Value Projection)**.**

1. *If* $(v_1, v_2) \in (\!|\tau|\!)_{G,k}$, *then* $\forall k. v_1 \in [\![|\tau|_1]\!]_{G|_1,k}$ *and* $v_2 \in [\![|\tau|_2]\!]_{G|_2,k}$.

2. *If* $(\sigma_1, \sigma_2) \in (\!|\Gamma|\!)_{G,k}$, *then* $\forall k.\sigma_1 \in (\!|\,|\Gamma|_1\,|\!)_{G|_1,k}$ *and* $\sigma_2 \in (\!|\,|\Gamma|_2\,|\!)_{G|_2,k}$.

Proof. Proof of Statement (1) is by induction on $(\!|\tau|\!)_{G,k}$. Proof of Statement (2) follows by proof of (1). □

Lemma 7 (Heap monotonicity)**.**

1. *If* $k' \leq k$ *and* $G' \supseteq G$, *then* $(H_1, H_2) \vDash_{G,k} P \Rightarrow (H_1, H_2) \vDash_{G',k'} P$

2. *If* $k' \leq k$ *and* $g' \supseteq g$, *then* $H \vDash_{g,k} P \Rightarrow H \vDash_{g',k'} P$

Proof. **proof of statement (1)**

Assume $k' \leq k$ and $G' \supseteq G$, and we have $H_1, H_2 \vDash_{G,k} P.$, TS: $H_1, H_2 \vDash_{G',k'} P$
Unfold the definition of $H \vDash_{G',k'} P$, By case analysis on P, there are three cases:

Case 1: P=empty It is vacuously true that $H \vDash_{g',k'}$ empty.

Case 2: P= $\gamma \to \beta$, From $H_1, H_2 \vDash_{G,k} \gamma \to \beta$, we know:

$$G(\gamma) = (l_1, l_2, \tau, n) \tag{1}$$

$$\forall i \leq n.(H_1(l_1)[i], H_2(l_2)[i]) \in (\!|\tau|\!)_{G,k-1} \tag{2}$$

$$\forall i \leq n.(H_1(l_1)[i] \neq H_2(l_2)[i]) \Rightarrow i \in \beta \tag{3}$$

We conclude that $G'(\gamma) = (l_1, l_2, \tau, n)$ from assumption $G' \supseteq G$ and $\forall i \leq n.H(l)[i] \in (\!|\tau|\!)_{G',k'}$ by Lemma 1 on (2).

Case 3: P= $P_1 \star P_2$. Suppose $\exists n.P = \{\gamma_i \to \beta_i\}$ for $i \in [1, \ldots, n]$.

Unfold the definition of $H_1, H_2 \vDash_{G,k} \{\gamma_i \to \beta_i\}$ for $i \in [1, \ldots, n]$.

We know there exists $H_1 = (h_1^1) \uplus (h_1^2) \uplus \cdots \uplus (h_1^n)$ and $H_2 = (h_2^1) \uplus (h_2^2) \uplus \cdots \uplus (h_2^n)$ and

$G = G_1 \uplus G_2 \uplus \cdots \uplus G_n$

so that $h_1^i, h_2^i \vDash_{G_i,k} \gamma_i \to \beta_i$ for any i.

Since $G' \supseteq G$, we know there exists $G'_i \supseteq G_i$ s.t. $G' = G'_1 \uplus G'_2 \uplus \cdots \uplus G'_n \wedge \forall i. G'_i \supseteq G_i$

Use the conclusion of case 2, we know:

$\forall i \in [1,n]. h_1^i, h_2^i \vDash_{G'_i,k'} \gamma_i \to \beta_i$.

proof of statement (2)

Assume $k' \leq k$ and $g' \supseteq g$, and we have $H \vDash_{g,k} P$. TS: $H \vDash_{g',k'} P$

Unfold the definition of $H \vDash_{g',k'} P$, by case analysis on P, there are three cases:

Case 1: P=empty, It is vacuously true that $H \vDash_{g',k'}$ empty.

Case 2: P= $\gamma \to \beta$. From $H \vDash_{g,k} \gamma$, we know:

$$g(\gamma) = (l, A, n) \tag{1}$$

$$\forall i \leq n. H(l)[i] \in [\![A]\!]_{g,k-1} \tag{2}$$

We conclude that $g'(\gamma) = (l, A, n)$ from assumption $g' \supseteq g$ and $\forall i \leq n. H(l)[i] \in [\![A]\!]_{g',k'}$ by Lemma 1 on (2).

Case 3: P = $P * Q$

Assume $p = \{\gamma_i\}$ for $i \in [1, \ldots, n]$.

From the definition of $H \vDash_{g,k} \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$, we know

$\exists H = (h_1^1) \uplus (h_1^2) \uplus \cdots \uplus (h_1^n)$ and $g = g_1 \uplus g_2 \uplus \cdots \uplus g_n$ s.t. $h_1^i \vDash_{g_i,k} \gamma_i \to \beta_i$ for any i.

Since $g' \supseteq g$, we know: $\exists g'_1, g'_2, \ldots, g'_n. g' = g'_1 \uplus g'_2 \uplus \cdots \uplus g'_n \wedge \forall i. g'_i \supseteq g_i$.

By the conclusion of case 2. we know: $h_1^i \vDash_{g'_i,k'} \gamma_i$ for any i.

This completes the proof of statement (2). □

Lemma 8 (Value interpretation containment)**.**

1. *If* $(v_1, v_2) \in (\!|\tau|\!)_{G,k}$*,then* $(v_1, v_2) \in (\!|\tau|\!)^{E,0}_{G,k}$

2. *If* $v \in [\![A]\!]_{g,k}$*, then* $\forall\, t \geq 0. v \in [\![A]\!]^{E,(0,t)}_{g,k}$

Proof. (1) Assume $(v_1, v_2) \in (\!|\tau|\!)_{G,k}$(*)

TS: $(v_1, v_2) \in (\!|\tau|\!)^{E,0}_{G,k}$

Following the definition of $(\!|\tau|\!)^{E,0}_{G,k}$, we get: $v_1 \Downarrow^0 v_1$ and $v_1 \Downarrow^0 v_1$ and $0-0 \leq 0$ and $(v_1, v_2) \in (\!|\tau|\!)_{G,k-0}$. This completes the proof. □

Proof. Assume $v \in [\![A]\!]_{g,k}$(*)

TS: $\forall\, t \geq 0. v \in [\![A]\!]^{E,(0,t)}_{g,k}$ Following the definition of $[\![A]\!]^{E,(0,t)}_{g,k}$. We know $v \Downarrow^0 v$ and $v \in [\![A]\!]_{g,k}$. This completes the proof. □

Lemma 9 (Value evaluation)**.** $v \Downarrow^0 v$

Proof. By induction on the value term v. □

Lemma 10 (heap projection)**.** *If* $(H_1, H_2) \vDash_{G,k} \gamma_i \to \beta_i, i \in (1, n)$*,then* $\forall\, T, T'. H_1 \vDash_{G|_1,k} \gamma_i \to T_i$ *and* $H_2 \vDash_{G|_2,k} \gamma_i \to T'_i$

Proof. Suppose $H_1 = h_1 \uplus h_2 \ldots h_n$ and $H_2 = h'_1 \uplus h'_2 \ldots h'_n$

Unfold $(h_i, h'_i) \vDash_{G,k} \gamma_i \to \beta_i$, we know:

$$G(\gamma) = (l_1, l_2, \tau, n) \tag{1}$$

$$\forall\, i \leq n. (h_i(l_1)[i], h'_i(l_2)[i]) \in (\!|\tau|\!)_{G,k-1} \tag{2}$$

From (1), we know:

$$G|_1(\gamma) = (l_1, \gamma, n) \tag{3}$$

$$G|_2(\gamma) = (l_2, \gamma, n) \tag{4}$$

From(2) and Lemma (Value projection) and ,we know:

$$\forall i \leq n.(H_1(l_1)[i]) \in [\![|\tau|_1]\!]_{G|_1,k-1} \tag{5}$$

$$\forall i \leq n.(H_2(l_1)[i]) \in [\![|\tau|_2]\!]_{G|_2,k-1} \tag{6}$$

Using (3)(5), (4)(6), this lemma is proved.

□

Lemma 11 (heap subtyping)**.** *If* $\vdash \delta : \Delta$ *and* $\vDash \delta\Phi$ *and* $\Delta;\Phi \vDash P \subseteq P'$ *and* $H_1, H_2 \vDash_{G,k} \delta P$*, then* $H_1, H_2 \vDash_{G,k} \delta P'$.

Proof. Assume we have: $H_1, H_2 \vDash_{G,k} \delta P$.

We want to show : $H_1, H_2 \vDash_{G,k} \delta P'$.

Suppose $\exists n.\delta P = \{\gamma_i \to \beta_i\}$ for $i \in [1,\ldots,n]$.

Unfold the definition of $H_1, H_2 \vDash_{G,k} \{\gamma_i \to \beta_i\}$ for $i \in [1,\ldots,n]$.

We know there exists $H_1 = (h_1^1) \uplus (h_1^2) \uplus \cdots \uplus (h_1^n)$ and $H_2 = (h_2^1) \uplus (h_2^2) \uplus \cdots \uplus (h_2^n)$ and $G = G_1 \uplus G_2 \uplus \cdots \uplus G_n$

so that $h_1^i, h_2^i \vDash_{G_i,k} \gamma_i \to \beta_i$ for any i.

Unfold the definition of $\Delta;\Phi \vDash \delta P \subseteq \delta P'$, we know: $\delta P' = \{\gamma_i \to \beta_i'\}, i \in [1,\ldots,n]$.

STS: $\forall i \in [1,\ldots,n].\exists x_1^i, x_2^i \vDash_{G_i,k]} \gamma_i \to \beta_i'$ where $H_1 = (x_1^1) \uplus (x_1^2) \uplus \cdots \uplus (x_1^n)$ and $H_2 = (x_2^1) \uplus (x_2^2) \uplus \cdots \uplus (x_2^n)$

Pick i.Choose $x_1^i = h_1^i$ and $x_2^i = h_2^i$, by unfolding the definition of $h_1^i, h_2^i \vDash_{G_i,k} \gamma_i \to \beta_i$.

we have: $G_i(\gamma_i) = (l_1^i, l_2^i, \tau^i, n)$ and $\forall m \leq n.(h_1^i(l_1^i[m]) \neq h_2^i(l_2^i)[m]) \Rightarrow m \in \beta_i$

Since we know $\beta_i \subseteq \beta_i'$, we get: $\forall i \in [1,\ldots,n].G_i(\gamma_i) = (l_1^i, l_2^i, \tau_i, n) \wedge \forall m \leq n.(x_1^i(m) \neq x_2^i(m)) \Rightarrow m \in \beta_i'$.

For any $i \in [1,\ldots,n]$, choose $x_1^i = h_1^i$ and $x_2^i = h_2^i$, we know: $x_1^i, x_2^i \vDash_{G_i,k} \gamma_i \to \beta_i'$

We also know $H_1 = (x_1^1) \uplus (x_1^2) \uplus \cdots \uplus (x_1^n)$ and $H_2 = (x_2^1) \uplus (x_2^2) \uplus \cdots \uplus (x_2^n)$ from assumption. This completes the proof of the lemma. □

Lemma 12 (Subtyping soundness)**.** *1. If* $\Delta; \Phi_a \models \tau \sqsubseteq \tau'$ *and* $\vdash \delta : \Delta$ *and* $\models \Phi\delta$ *and* $(v, v') \in (\!|\delta\tau|\!)_{G,k}$*, then* $(v, v') \in (\!|\delta\tau'|\!)_{G,k}$.

2. *If* $\Delta; \Phi_a \models A \sqsubseteq A'$ *and* $\vdash \delta : \Delta$ *and* $\models \Phi\delta$ *and* $(v) \in [\![\delta A]\!]_{g,k}$*, then* $(v) \in [\![\delta A']\!]_{g,k}$.

3. *If* $\Delta; \Phi_a \models \tau \sqsubseteq \tau'$ *and* $\vdash \delta : \Delta$ *and* $\models \Phi\delta$ *and* $(t, t') \in (\!|\delta\tau|\!)^{E,D}_{G,k}$ *and* $D \leq D'$*, then* $(t, t') \in (\!|\delta\tau'|\!)^{E,D'}_{G,k}$.

4. *If* $\Delta; \Phi_a \models A \sqsubseteq A'$ *and* $\vdash \delta : \Delta$ *and* $\models \Phi\delta$ *and* $t \in [\![\delta A]\!]^{E,(L,U)}_{g,k}$ *and* $L' \leq L$ *and* $U \leq U'$*, then* $t \in [\![\delta A']\!]^{E,(L',U')}_{g,k}$.

5. *If* $\Delta; \Phi_a \models \tau \sqsubseteq \tau'$ *and* $\vdash \delta : \Delta$ *and* $\models \Phi\delta$ *and* $\forall i \in \{1,2\}$. $(v) \in [\![|\delta\tau|_i]\!]_{G|_i,k}$*, then* $v \in [\![|\delta\tau'|_i]\!]_{G|_i,k}$.

6. *If* $\Delta; \Phi_a \models \tau \sqsubseteq \tau'$ *and* $\vdash \delta : \Delta$ *and* $\models \Phi\delta$ *and* $\forall i \in \{1,2\}$. $(e) \in [\![|\delta\tau|_i]\!]^{E,(L,U)}_{G|_i,k}$ *and* $L' \leq L$ *and* $U \leq U'$*, then* $e \in [\![|\delta\tau'|_i]\!]^{E,(L,U)}_{G|_i,k}$.

Proof Statements (1) and (2) (5) are by proven simultaneously by induction on the subtyping derivation. (3),(4),(6) will be proved first.

Proof. **Proof of statement (3)** Assume that $\Delta; \Phi_a \models \tau \sqsubseteq \tau'$ and $\vdash \delta : \Delta$ and $\models \Phi\delta$ and $(t, t') \in (\!|\delta\tau|\!)^{E,D}_{G,k}$ and $D \leq D'$.
TS: $(t, t') \in (\!|\delta\tau'|\!)^{E,D'}_{G,k}$.
Assume that 1) $t \Downarrow^{s,c} v$, 2) $t' \Downarrow^{s',c'} v'$, 3) $s < k$.
Unfold the assumption $(t, t') \in (\!|\delta\tau|\!)^{E,D}_{G,k}$ using assumption (1)(2)(3), we know that $c - c' \leq D(a)$, $(v, v') \in (\!|\delta\tau|\!)_{G,k-s}(b)$.
we conclude: $c - c' \leq D'$ from assumption $D \leq D'$. and by IH1 on (b), we know $(v, v') \in$

$(\!|\delta\tau'|\!)_{G,k-s}$.

Proof of statement (4) Assume $\Delta;\Phi_a \models A \sqsubseteq A'$ and $\vdash \delta : \Delta$ and $\models \Phi\delta$ and $e \in [\![\delta A]\!]^{E,(L,U)}_{g,k}$ and $L' \leq L$ and $U \leq U'$. We want to show: $t \in [\![\delta A]\!]^{E,(L,U)}_{g,k}$

Assume: $t \Downarrow^{s,c}$, $s < k$.

Unfold the assumption $t \in [\![\delta A]\!]^{E,(L,U)}_{g,k}$, we know: $L \leq c \leq U(a)$, $v \in [\![\delta A]\!]_{g,k-s}(b)$.

Then, we conclude: $L' \leq c \leq U'$ from assumption. By IH2 with the first premise on (b), we get:$v \in [\![\delta A']\!]_{g,k-s}$

.

Proof of statement (6) Assume $\Delta;\Phi_a \models \tau \sqsubseteq \tau'$ and $\vdash \delta : \Delta$ and $\models \Phi\delta$ and $\forall i \in \{1,2\}$. $(e) \in [\![|\delta\tau|_i]\!]^{E,(L,U)}_{G|_i,k}$ and $L' \leq L$ and $U \leq U'$

TS: $t \in [\![|\delta\tau'|_i]\!]^{E,(L,U)}_{G|_i,k}$

Assume: $t \Downarrow^{s,c}$, $s < k$. Unfold the assumption $t \in [\![|\delta\tau'|_i]\!]^{E,(L,U)}_{G|_i,k}$, we know: $L \leq c \leq U(a)$, $v \in [\![|\delta\tau'|_i]\!]_{G|_i,k-s}(b)$.

We conclude: $L' \leq c \leq U'$ from assumption, by IH5 with the first premise on (b), we get:$v \in [\![|\delta\tau'|_i]\!]_{G|_i,k-s}$.

Proof of statement (1)

The proof is by induction on the subtyping derivation, we will detail some of the most interesting cases.

Case

$$\frac{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Delta;\Phi_a \models D \leq D' \qquad \Delta;\Phi_a \models P' \sqsubseteq P \qquad \Sigma, \vec{\gamma}_1 : \vec{\mathbb{L}};\Delta;\Phi_a \models Q \sqsubseteq Q' \qquad \Sigma \models \vec{\gamma}_1 \sqsubseteq \vec{\gamma}_2}{\Sigma;\Delta;\Phi_a \models \{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma}_1 : \tau\ \{Q\} \sqsubseteq \{P'\}\ \exists^{\mathrm{diff}(D')}\vec{\gamma}_2 : \tau'\ \{Q'\}}\ \text{S-RM}$$

Assume δ such that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$. Assume also that we have G, k, v_1, v_2 such that

$$(v_1, v_2) \in (\!|\{\delta P\}\ \exists \vec{\gamma}_1 : \delta\tau\ \{\delta Q\}|\!)^{\mathrm{diff}(\delta D)}_{\delta G, k} \tag{1}$$

We want to show

$$(v_1, v_2) \in (\!|\{\delta P'\}\ \exists \vec{\gamma}_2 : \delta\tau'\ \{\delta Q'\}|\!)^{\mathrm{diff}(\delta D')}_{\delta G, k} \tag{2}$$

By definition, consider $G_1 \supseteq \delta G, k_1 \leq k, k_2 < k, H_1, H_2$ and assume

$(H_1, H_2) \vDash_{G_1,k_1} \delta P' \wedge H_1; v_1 \downarrow_f^{k_2} \wedge H_2; v_2 \not\downarrow^f$. Since $\Delta; \Phi \vDash P' \subseteq P$ we also have

$(H_1, H_2) \vDash_{G_1,k_1} \delta P$.

Thus, by Equation (1) we have that there exist $G_2 \supseteq G_1, H_1', H_2', v_1', v_2', c_1, c_2$ such that

$v_1; H_1 \Downarrow_f^{c_1,k_2} v_1'; H_1' \wedge v_2; H_2 \Downarrow_f^{c_2,k_2} v_2'; H_2' \wedge$

$(H_1', H_2') \vDash_{G_2,k_1-k_2} \delta Q \wedge (v_1', v_2') \in (\!|\delta\tau|\!)_{G_2,k_1-k_2} \wedge c_1 - c_2 \leq \delta D$.

Since $\Sigma, \vec{\gamma}_1 : \vec{\mathbb{L}}; \Delta; \Phi_a \vDash Q \subseteq Q'$ we also have $(H_1', H_2') \vDash_{G_2,k_1-k_2} \delta Q'$ by using Lemma 11 on

$(H_1', H_2') \vDash_{G_2,k_1-k_2} \delta Q$.

Additionally, from $\Delta; \Phi_a \vDash \tau \sqsubseteq \tau'$ and $\Delta; \Phi_a \vDash D \leq D'$

By IH on the first premise, we also get

$(v_1', v_2') \in (\!|\delta\tau'|\!)_{G_2,k_1-k_2}$ and $c_1 - c_2 \leq \delta D'$ respectively.

So we can derive Equation (2), and this completes the proof of case-sub-r-monad.

Case

S-RUM

$$\frac{}{\Sigma; \Delta; \Phi_a \vdash U\ (\{\gamma_i \to T_i\}\ \exists \vec{\gamma_1'} : A_1\ \{Q_1\}, \{\gamma_i \to T_i'\}\ \exists \vec{\gamma_2'} : A_1\ \{Q_2\}) \sqsubseteq \{\gamma_i \to \beta_i\}\ \exists \vec{\gamma}_1, \vec{\gamma}_2 : U(A_1, A_2)\ \{\gamma_i \to \beta_i \cup T_i \cup T_i'\}}$$

$\mathrm{exec}(L,U)$ $\mathrm{exec}(L',U')$ $\mathrm{diff}(U-L')$

Pick G,k,v,v', we assume $\vdash \delta : \Delta$ and $\vDash \Phi\delta$, we have:

$$(v, v') \in (\!|\delta U\ (\{\gamma_i \to \overset{\text{exec}(L,U)}{T_i}\}\ \exists \vec{\gamma}_1 : A_1\ \{Q_1\}, \{\gamma_i \to \overset{\text{exec}(L',U')}{T'_i}\}\ \exists \vec{\gamma}_1 : A_2\ \{Q_2\})|\!)_{G,k} \tag{a}$$

TS: $(v, v') \in (\!|\delta\ \{\gamma_i \to \beta_i\}\ \exists \vec{\gamma}_1, \vec{\gamma}_2 : \overset{\text{diff}(U-L')}{U}(A_1, A_2)\ \{\gamma_i \to \beta_i \cup T_i \cup T'_i\}|\!)_{G,k}$

Unfold the definition of $(\!|\delta\ \{\gamma_i \to \beta_i\}\ \exists \vec{\gamma}_1, \vec{\gamma}_2 : \overset{\text{diff}(U-L')}{U}(A_1, A_2)\ \{\gamma_i \to \beta_i \cup T_i \cup T'_i\}|\!)_{G,k}$.

Pick $G' \supseteq G, k' \le k, k'' < k, H_1, H_2$,

we assume $(H_1, H_2) \vDash_{G',k'} \gamma_i \to \beta_i(1)$, $H_1; v \downarrow_f^{k''} \wedge H_2; v' \not\downarrow^f$ (2).

Still To Show:

$\exists G'' \supseteq G, H'_1, H'_2, v_1, v'_1,\ H_1; v \Downarrow_f^{c_1,k''} H'_1; v_1(3),\ H_2; v' \Downarrow_f^{c_2} H'_2; v'_1(4)\ H'_1, H'_2 \vDash_{G'',k'-k''} \gamma_i \to \beta_i \cup T_i \cup T'_i(5)$, $(v_1, v'_1) \in (\!|U(A_1, A_2)|\!)_{G'',k'-k''}(6)$.

From (a), unfold its definition of $(\!|\delta U\ (\{\gamma_i \to \overset{\text{exec}(L,U)}{T_i}\}\ \exists \vec{\gamma}_1 : A_1\ \{Q_1\}, \{\gamma_i \to \overset{\text{exec}(L',U')}{T'_i}\}\ \exists \vec{\gamma}_1 : A_2\ \{Q_2\})|\!)_{G,k}$

$$\forall k', v \in [\![\{\gamma_i \to \overset{\text{exec}(L,U)}{T_i}\}\ \exists \vec{\gamma}_1 : A_1\ \{Q_1\}]\!]_{G|_1,k'} \tag{b}$$

$$\forall k', v' \in [\![\{\gamma_i \to \overset{\text{exec}(L',U')}{T'_i}\}\ \exists \vec{\gamma}_2 : A_2\ \{Q_2\}]\!]_{G|_2,k'} \tag{c}$$

Pick k' as k, unfold its definition in (b), we already got: $G'|_1 \supseteq G|_1$ and $k' \le k$ and $k'' < k'$ and $H_1; v \downarrow_f^{k''}$ and $H_1 \vDash_{G'|_1,k'} \gamma_i \to T_i$ holds by using Lemma (heap projection) on (1).

Derive from the definition, we know: $\exists g''_1 \supseteq G'|_1, H'_1, v_1., H_1; v \Downarrow_f^{c_1,k''} H'_1; v_1(d)$,

$H'_1 \vDash_{g''_1,k'-k''} Q_1(e)$, $v_1 \in [\![A_1]\!]_{g''_1,k'-k''}(f)$ $\forall i. g'_1(\gamma_i) = (l_i, A, m) \Rightarrow H_1(l)[n] \neq H'_1(l)[n] \Rightarrow n \in T_i(*)$.

Unfold its definition in (c), we already got: $G'|_2 \supseteq G|_2$ and $k' \le k$ and $k'' < k'$ and $H_2; v \downarrow_f$ and $H_2 \vDash_{G'|_2,k'} \gamma_i \to T'_i$ holds by using Lemma 10 (heap projection) on (1).

There are two cases for $H_2; v \downarrow_f \Rightarrow H_2; v \downarrow_f^{k''}$:

1. $H_2; v \downarrow_f^{k_1}$ and $k_1 > k'$. Using Lemma 1, we get $H_2; v \downarrow_f^{k''}$
2. $H_2; v \downarrow_f^{k_1}$ and $k_1 \le k'$. We choose $k'' = k_1$. we get $H_2; v \downarrow_f^{k''}$

Derive from the definition, we know:

$\exists g_2'' \supseteq G'|_2, H_2', v_1'., H_2; v \Downarrow_f^{c_2,k''} H_2'; v_1'(g)$,

$H_2' \vDash_{g_2'',k'-k''} Q_2(h)$,

$v_1' \in [\![A_2]\!]_{g_2'',k'-k''}(i)$.

$\forall i. g_2'(\gamma_i) = (l_i, A, m) \Rightarrow H_2(l)[n] \neq H_2'(l)[n] \Rightarrow n \in T_i'(**)$.

(3),(4) already proved. STS1: $H_1', H_2' \vDash_{G'',k'-k''} \gamma_i \rightarrow \beta_i \cup T_i \cup T_i'$

From (e), we assume: $H_1' = h_1 \uplus h_2 \uplus \ldots h_n(7)$, $H_2' = h_1' \uplus h_2' \uplus \ldots h_n'(8)$.

From (*),(**), we know: $\forall i, n.(H_1(l_i)[n] = (H_2(l_i)[n]) \Rightarrow n \notin \beta_i$, otherwise, $n \in \beta_i$.

for all the n not in β_i, if $H_1(l_i)[n] \neq H_1'(l_i)[n]$, then $n \in T_i$. Similarly, if $H_2(l_i)[n] \neq H_2'(l_i)[n]$, then $n \in T_i'$.

Unfold the definition of $H_1', H_2' \vDash_{G,k} \gamma_i \rightarrow \beta_i \cup T_i \cup T_i'$. $\forall n. H_1'(l_i)[n] \neq H_2'(l_i)[n] \Rightarrow n \in \beta_i \cup T_i \cup T_i'$ because (1) at position n, values differs in H_1, H_2 means n in β_i (2) at position n, value same in H_1, H_2. and differ in H_1', H_2' means n must in T_i or T_i' or both.

This completes the proof of case sub-r-u-monad

Proof of statement (2). Proof is by induction on the subtyping derivation.

Case

$$\frac{\Delta;\Phi_a \vDash U \leq U' \qquad \Delta;\Phi_a \vDash P \subseteq P' \qquad \Sigma, \vec{\gamma}_1; \Delta;\Phi_a \vDash Q' \subseteq Q \qquad \vec{\gamma}_1 \subseteq \vec{\gamma}_2 \qquad \Delta;\Phi_a \vDash A \sqsubseteq A' \qquad \Delta;\Phi_a \vDash L' \leq L}{\Sigma;\Delta;\Phi_a \vDash \{P\}\ \exists \vec{\gamma}_1 : A\ \{Q\}^{\mathrm{exec}(L,U)} \sqsubseteq \{P'\}\ \exists \vec{\gamma}_2 : A'\ \{Q'\}^{\mathrm{exec}(L',U')}}\ \textbf{S-A-MONAD}$$

Pick g,k,v. We assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$, and then we know : $v \in \{P\}\ \exists \vec{\gamma}_1 : A\ \{Q\}^{\mathrm{exec}(L,U)}\ (a)$.
To Show: $v \in \{P'\}\ \exists \vec{\gamma}_2 : A'\ \{Q'\}^{\mathrm{exec}(L',U')}\ (b)$.

By unfolding the definition of $\{P\}\ \exists\vec{\gamma}_1 : A\ \{Q\}^{\text{exec}(L,U)}$, we pick $g' \supseteq g, k' \leq k, k'' < k', H$.

Assume $H \vDash_{g',k'} P(1)$, $H; v \downarrow_f^{k''}$ (2).

We know: $\exists g'' \supseteq g, H', v_1\ ,\ H; v \Downarrow_f^{c_1,k''} H'; v_1(3)$, $H' \vDash_{g'',k'-k''} Q(4)$, $v_1 \in [\![A]\!]_{g'',k'-k''}(5)$, $P = \gamma_i \to T_i \Rightarrow \forall i. g''(\gamma_i) = (l_i, A, m) \Rightarrow H(l_i)[n] \neq H'(l_i)[n] \Rightarrow n \in T_i(*)$.

Pick g'', H', v_1. By IH 2 on the first premise $\Delta; \Phi_a \vDash A \sqsubseteq A'$ with (5), we know: $v_1 \in [\![A']\!]_{g'',k'-k''}(6)$.

TS: $v \in \{P'\}\ \exists\vec{\gamma}_2 : A'\ \{Q'\}^{\text{exec}(L',U')}$, unfold its definition, we know: $H \vDash_{g',k'} P'$ from (1) and $P \supseteq P'$ because P and P' have the same γ_i. together with (2)

Still To Show : $H; v \Downarrow_f^{c_1,k''} H'; v_1(c)$, $H' \vDash_{g'',k'-k''} Q'(d)$, $v_1 \in [\![A']\!]_{g'',k'-k''}(e)$, $P' = \gamma_i \to T_i' \Rightarrow \forall i. g_1'(\gamma_i) = (l_i, A, m) \Rightarrow H(l_i)[n] \neq H'(l_i)[n] \Rightarrow n \in T_i'(**)$.

(c) is proved by assumption (3), (e) is proved by (6), (d) is proved by (4) and $Q' \supseteq Q$, (**) is proved by (*). Because $P \supseteq P' \Rightarrow \forall i. T_i \supseteq T_i'$. For all the n in T_i, it is also in T_i'. This completes the proof of sub-A-monad.‘

Proof of statement (5). Proof is by induction on the subtyping derivation. We will prove i=1, the proof of i=2 is similar.

Case

S-RUM

$$\frac{}{\Sigma;\Delta;\Phi_a \vDash U\ (\{\gamma_i \to T_i\}\ \exists\vec{\gamma_1'} : A_1\ \{Q_1\}^{\text{exec}(L,U)}, \{\gamma_i \to T_i'\}\ \exists\vec{\gamma_2'} : A_1\ \{Q_2\}^{\text{exec}(L',U')}) \sqsubseteq \{\gamma_i \to \beta_i\}\ \exists\vec{\gamma}_1, \vec{\gamma}_2 : U(A_1, A_2)\ \{\gamma_i \to \beta_i \cup T_i \cup T_i'\}^{\text{diff}(U-L')}}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$. we have

$$v \in [\![|\delta U\ (\{\gamma_i \to T_i\}\ \exists\vec{\gamma}_1 : A_1\ \{Q_1\}^{\text{exec}(L,U)}, \{\gamma_i \to T_i'\}\ \exists\vec{\gamma}_1 : A_2\ \{Q_2\}^{\text{exec}(L',U')})|_i]\!]_{|G|_i,k} \quad \text{(a)}$$

TS: $v \in [\![|\delta\ \{\gamma_i \to \beta_i\}\ \exists\vec{\gamma}_1, \vec{\gamma}_2 : U(A_1, A_2)\ \{\gamma_i \to \beta_i \cup T_i \cup T_i'\}^{\text{diff}(U-L')}\ |_i]\!]_{|G|_i,k}(b)$ Choose i=1. From (a), we know: $v \in [\![\delta\ \{\gamma_i \to T_i\}\ \exists\vec{\gamma}_1 : A_1\ \{Q_1\}^{\text{exec}(L,U)}]\!]_{|G|_1,k}(c)$.

From (b), Still To Show:$v \in [\![|\delta\ \{\gamma_i \to \beta_i\}\ \exists \vec{\gamma}_1, \vec{\gamma}_2 : A_1\ \{\gamma_i \to \beta_i \cup T_i \cup T_i'\}]\!]_{|G|_1,k}^{\mathrm{exec}(0,\infty)}(d)$.

Assume $g' \supseteq G|_1, k' \leq k, k'' < k'$. $H \vDash_{g',k'} \gamma_i \to \mathbb{N}$ (1) , $H; v \downarrow_f^{k''}$ (2).

STS: $\exists g'' \supseteq g', H', v',\ H; v \Downarrow_f^{k'',c_1} H'; v'$(3), $H' \vDash_{g'',k'-k''} \gamma_i \to \mathbb{N}$ (4), $v' \in [\![A_1]\!]_{g'',k'-k''}$ (5),

$P' = \gamma_i \to \mathbb{N} \Rightarrow \forall i. g_1'(\gamma_i) = (l_i, A, m) \Rightarrow H(l_i)[n] \neq H'(l_i)[n] \Rightarrow n \in N$ $(**)$.

From (c), unfold its definition, we know (2) and $H \vDash_{g',k'} \gamma_i \to T_i$ holds from (1).

Then we get: $\exists g'' \supseteq g', H', v', H; v \Downarrow_f^{k'',c_1} H'; v'$ (e), $H' \vDash_{g'',k'-k''} Q_1(f)$, $v' \in [\![A_1]\!]_{g'',k'-k''}$ (g),

$P = \gamma_i \to T_i \Rightarrow \forall i. g_1'(\gamma_i) = (l_i, A, m) \Rightarrow H(l_i)[n] \neq H'(l_i)[n] \Rightarrow n \in T_i$ $(**)$.

(3) is proved by (e), (4) is proved by (f) because Q_1 contains at least as many γ_i as its precondition $\gamma_i \to T_i$, (*) is proved by (**) because $T_i \supseteq \mathbb{N}$.

Case

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a \vDash \tau \sqsubseteq \tau' \\ \Delta;\Phi_a \vDash D \leq D' \qquad \Delta;\Phi_a \vDash P' \subseteq P \qquad \Sigma, \vec{\gamma}_1 : \vec{\mathbb{L}}; \Delta; \Phi_a \vDash Q \subseteq Q' \qquad \Sigma \vDash \vec{\gamma}_1 \sqsubseteq \vec{\gamma}_2\end{array}}{\Sigma;\Delta;\Phi_a \vDash \{P\}\ \exists \vec{\gamma}_1 : \tau^{\mathrm{diff}(D)}\ \{Q\} \sqsubseteq \{P'\}\ \exists \vec{\gamma}_2 : \tau'^{\mathrm{diff}(D')}\ \{Q'\}}\ \text{S-RM}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$, we have $v \in [\![|\ \{P\}\ \exists \vec{\gamma}_1 : \tau^{\mathrm{diff}((c))}\ \{Q\}\ |_i]\!]_{|G|_i,k}$ (a).

TS:$v \in [\![|\delta\ \{P'\}\ \exists \vec{\gamma}_2 : \tau'^{\mathrm{diff}((c'))}\ \{Q'\}\ |_i]\!]_{|G|_i,k}(b)$.

Choose i=1. From (a), we know: $v \in [\![\delta\ \{|P|_1\}\ \exists \vec{\gamma}_1 : |\tau|_1^{\mathrm{exec}(0,\infty)}\ \{|Q|_1\}]\!]_{|G|_1,k}(c)$.

From (c),Still To Show: $v \in [\![\delta\ \{|P'|_1\}\ \exists \vec{\gamma}_2 : |\tau'|_1^{\mathrm{exec}(0,\infty)}\ \{|Q'|_1\}]\!]_{|G|_1,k}$ (d)

From $P' \supseteq P$ and $Q \supseteq Q'$, we know $|P|_1 = |P'|_1 \wedge |Q|_1 = |Q'|_1$, it is trivially proved.

This completes the proof of case sub-r-monad.

□

A.5.0.1 Proof of Theorem

Theorem A.5.1 (Fundamental Theorem)**.**

1. *If* $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ *and* $\vdash \delta : \Delta$ *and* $\vDash \delta\Phi$ *and* $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$*, then*

$(\delta\sigma_1 t_1, \delta\sigma_2 t_2) \in (\!|\delta\tau|\!)^{E,(\delta D)}_{G,k}$.

2. *If* $\Sigma;\Delta;\Phi_a;\Omega \vdash^U_L t : A$ *and* $\vdash \delta : \Delta$ *and* $\vDash \delta\Phi$ *and there exists* Ω'*. s.t.* $FV(t) \subseteq dom(\Omega')$ *and* $\Omega' \subseteq \Omega$ *and* $\sigma \in (\!|\delta\Omega'|\!)_{g,k}$*, then* $(\delta\sigma\, t) \in [\![\delta A]\!]^{E,(\delta L,\delta U)}_{g,k}$.

3. *If* $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ *and* $\vdash \delta : \Delta$ *and* $\vDash \delta\Phi$*, then for* $i \in \{1,2\}$*. if there exists* Γ'_i *s.t.* $FV(t_i) \subseteq dom(\Gamma'_i)$ *and* $\Gamma'_i \subseteq \Gamma$ *and* $\sigma_i \in [\![|\delta\Gamma'_i|_i]\!]_{G|_i,k}$*, then* $\delta\sigma_i\, t_i \in [\![|\delta\tau_i|]\!]^{E,(0,\infty)}_{G|_i,k}$.

Corollary A.5.1.1. *If* $\forall k.\bullet \vdash t_1 \ominus t_2 \lesssim n : \tau$*, then* $(t_1, t_2) \in (\!|\tau|\!)^{E,\delta(n)}_{G,k}$.

Proof of statement (1). if $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ and $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$ then $(t_1\delta\sigma_1, t_2\delta\sigma_2) \in (\!|\delta\tau|\!)^{E,\delta(D)}_{G,k}$.

Proof by induction on typing derivation:

$$\frac{\Sigma;\Delta;\Phi_a;x:\tau_1, f\ :\tau_1 \xrightarrow{\text{diff}\,D} \tau_2, \Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{Fix}\, f(x).t_1 \ominus \mathsf{Fix}\, f(x).t_2 \lesssim 0 : \tau_1 \xrightarrow{\text{diff}\,D} \tau_2}\ \text{R-FIX}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1 \mathsf{fix}\, f(x).t_1, \delta\sigma_2 \mathsf{fix}\, f(x).t_2) \in (\!|\delta(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2)|\!)^{E,0}_{G,k}$

By unfolding the definition and the fact that $\mathsf{fix}\, f(x).t_1$ and $\mathsf{fix}\, f(x).t_2$ are values

STS: $(\delta\sigma_1 \mathsf{fix}\, f(x).t_1, \delta\sigma_2 \mathsf{fix}\, f(x).t_2) \in (\!|\delta\tau_1 \xrightarrow{\text{diff}(\delta D)} \delta\tau_2|\!)_{G,k}$

Set $F = \delta\sigma_1 \mathsf{fix}\, f(x).t_1, F' = \delta\sigma_2 \mathsf{fix}\, f(x).t_2$, We want to prove a general statement: $\forall m \leq k.(F, F') \in (\!|\delta\tau_1 \xrightarrow{\text{diff}\delta D} \delta\tau_2|\!)_{G,m}$. By sub-induction on m, there are two cases to show.

Subcase 1: m=0, we unfold the definition of function type interpretation, there are two parts to show:

Subsubcase 1: $\forall k' < m.(v_1, v_2) \in (\!|\tau_1|\!)_{G,k'} \Rightarrow \ldots$. It is vacuous by definition to show $k' \leq -1$.

Subsubcase 2: STS: $\forall j.F \in [\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,j} \wedge F' \in [\![|\delta\tau_1|_2 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_2]\!]_{G|_2,j}$.

Pick j.

1. STS 1: $F \in [\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,j}$, we prove the more general statement.

$$\forall m' \leq j.F \in [\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m'} \tag{A.1}$$

   By subinduction on m', there are two csases:

   1.1. m'=0, we unfold the definition of $[\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m'}$, there is no non-negative $k' < m'(0)$, it is vacuously true.

   1.2. m'=m"+1. By sub-IH on m"

$$F \in [\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m''} \tag{A.2}$$

   STS: $F \in [\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m'}$

   Unfold the definition of $[\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m''+1}$.

   Pick $j'' < m''+1, g' \supseteq G|_1$ and assume that $v \in [\![|\delta\tau_1|_1]\!]_{g',j''}$.

   STS: $\delta\sigma_1 t_1[v/x][\text{fix}\, f(x).t_1/f] \in [\![|\tau_2|_1]\!]^{E,(0,\infty)}_{g',j''}$

   This follows by IH 3 on the premise instantiated with

   $\sigma_1[v/x][\text{fix}\, f(x).t_1/f] \in [\![\delta(x:|\tau_1|_1, f:|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1, |\Gamma_1|)]\!]_{g',j''}$

   which holds because

   i. $FV(t_1) \subseteq dom(x:\tau_1, f:\tau_1 \overset{\text{diff}\,D}{\longrightarrow} \tau_2, \Gamma)$ using Lemma 5.

   ii. $\sigma_1 \in [\![|\delta\Gamma|_1]\!]_{g',j''}$ by Lemma 6 and then Lemma 1.

   iii. $v \in [\![|\delta\tau_1|_1]\!]_{g',j''}$ from the assumption.

   iv. $F \in [\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{g',j''}$ by Lemma 1 on (3.2).

2. STS 2: $F' \in [\![|\delta\tau_1|_2 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_2]\!]_{G|_2,j}$

Its proof is similar with STS 1.

Subcase 2: $m = m' + 1 \leq k$

By the definition of function types, there are two parts to show:

Subsubcase 1: $\forall k' < m.(v_1, v_2) \in (\!|\tau_1|\!)_{G,k} \Rightarrow \ldots$

Assume $G' \supseteq G, k' \leq (m-1)$. Pick v_1, v_2 s.t. $(v_1, v_2) \in (\!|\delta\tau_1|\!)_{G',k'}$.

STS: $(t_1[v_1/x][\mathsf{fix}\, f(x).t_1/f], t_2[v_2/x][\mathsf{fix}\, f(x).t_2/f]) \in (\!|\delta\tau_2|\!)^{E,(\delta D)}_{G',k'}$

By IH on premise instantiated with

$\sigma_1' = \sigma_1[v_1/x][\mathsf{fix}\, f(x).t_1/f], \sigma_2' = \sigma_2[v_2/x][\mathsf{fix}\, f(x).t_2/f]$

and $(\sigma_1', \sigma_2') \in (\!|(\Gamma, x:\tau, f\ :\tau_1 \xrightarrow{\mathrm{diff}\, D} \tau_2)\delta|\!)_{G',k'}$ which holds because

1. $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$ by assumption and Lemma 1.
2. $(v_1, v_2) \in (\!|\tau|\!)_{G',k'}$ from assumption
3. $(F_1, F_2) \in (\!|\delta(\tau_1 \to \tau_2)|\!)_{G',k'}$ from sub-IH $(F_1, F_2) \in (\!|\delta(\tau_1 \xrightarrow{\mathrm{diff}\, D} \tau_2)|\!)_{G,m'}$ and Lemma 1 $(k' < m', G' \supseteq G)$

Subsubcase 2: $\forall j.F \in [\![|\delta\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\delta\tau_2|_1]\!]_{G|_1,j} \wedge F' \in [\![|\delta\tau_1|_2 \xrightarrow{\mathrm{exec}(0,\infty)} |\delta\tau_2|_2]\!]_{G|_2,j}$

which is already proved above

This completes the proof of this case.

Case

$$\frac{\Sigma;\Delta;\Phi_a;x:\tau_1, f\ :\tau_1 \xrightarrow{\mathrm{diff}\, D} \tau_2, \Gamma, f\ : U(A_1, A_2) \vdash t_1 \ominus t_2 \lesssim D : \tau_2 \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^0_0 \mathsf{Fix}\, f(x).t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^0_0 \mathsf{Fix}\, f(x).t_2 : A_2}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{Fix}\, f(x).t_1 \ominus \mathsf{Fix}\, f(x).t_2 \lesssim 0 : \tau_1 \xrightarrow{\mathrm{diff}\, D} \tau_2}\ \text{R-FIX-EXT}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$.

To Show $(\mathsf{fix}\, f(x).t_1\delta\sigma_1, \mathsf{fix}\, f(x).t_2\delta\sigma_2) \in (\!|\delta(\tau_1 \xrightarrow{\mathrm{diff}\, D} \tau_2)|\!)^{E,0}_{G,k}$.

By unfolding the definition and the fact that $\mathsf{fix}\, f(x).t_1$ and $\mathsf{fix}\, f(x).t_2$ are values

STS: $(\mathsf{fix}\, f(x).t_1\delta\sigma_1, \mathsf{fix}\, f(x).t_2\delta\sigma_2) \in (\!|\delta\tau_1 \overset{\mathrm{diff}\,\delta D}{\longrightarrow} \delta\tau_2|\!)_{G,k}$,

we set $F = \mathsf{fix}\, f(x).t_1\delta\sigma_1, F' = \mathsf{fix}\, f(x).t_2\delta\sigma_2$, We want to prove a general statement.

$\forall m \le k.(F,F') \in (\!|\delta\tau_1 \overset{\mathrm{diff}\,(\delta D)}{\longrightarrow} \delta\tau_2|\!)_{G,m}$

By sub-induction on m, there are two cases to show.

Subcase 1: m=0. Unfold the definition of function type interpretation, there are two parts to show:

Subsubcase 1: $\forall k' < m.(v_1,v_2) \in (\!|\tau_1|\!)_{G,k'} \Rightarrow \ldots$. It is vacuous by definition to show $k' \le -1$.

Subsubcase 2: STS: $\forall j.F \in [\![|\delta\tau_1|_1 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,j} \wedge F' \in [\![|\delta\tau_1|_2 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_2]\!]_{G|_2,j}$.

Pick j.

1. STS 1: $F \in [\![|\delta\tau_1|_1 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,j}$. We prove the more general statement

$$\forall m' \le j.F \in [\![|\delta\tau_1|_1 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m'} \tag{A.3}$$

By subinduction on m', there are two csases:

1.1. m'=0. Unfold the definition of $[\![|\delta\tau_1|_1 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m'}$, there is no non-negative $k' < 0(m')$, it is vacuously true.

1.2. m'=m"+1. By sub-IH on m'.'

$$F \in [\![|\delta\tau_1|_1 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m''} \tag{A.4}$$

STS: $F \in [\![|\delta\tau_1|_1 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m'}$

Unfold the definition of $[\![|\delta\tau_1|_1 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,m''+1}$.

Pick $j'' < m''+1, g' \supseteq G|_1$ and assume that $v \in [\![|\delta\tau_1|_1]\!]_{g',j''}$.

STS: $\delta\sigma_1 t_1[v/x][\text{fix}\, f(x).t_1/f] \in [\![|\tau_2|_1]\!]^{E,(0,\infty)}_{g',j''}$

This follows by IH 3 on the premise instantiated with

$\sigma_1[v/x, \text{fix}\, f(x).t_1/f] \in [\![\delta|(x:\tau_1, f:\tau_1 \xrightarrow{\text{diff}(D)} \tau_2, f:U(A_1,A_2),\Gamma_1)|_1]\!]_{g',j''}$

$\Rightarrow \sigma_1[v/x][\text{fix}\, f(x).t_1/f] \in [\![\delta(x:|\tau_1|_1, f:A_1, |\Gamma_1|_1)]\!]_{g',j''}$ which holds because

i. $FV(t_1) \subseteq dom(x:\tau_1, f:\tau_1 \xrightarrow{\text{diff}(D)} \tau_2, f:U(A_1,A_2),\Gamma)$ using Lemma 5.

ii. $\sigma_1 \in [\![|\delta\Gamma|_1]\!]_{g',j''}$ by Lemma 6 and then Lemma 1.

iii. $v \in [\![|\delta\tau_1|_1]\!]_{g',j''}$ from assumption.

iv. $F \in [\![\delta A_1]\!]_{g',j''}$ by IH2 on the second premise of FIX-EXT instantiated with $\Omega' = |\Gamma|_1$ to show $F \in [\![\delta A_1]\!]_{G|_1,k}$. Then use Lemma 1.

2. STS 2: $F' \in [\![|\delta\tau_1|_2 \xrightarrow{\text{exec}(0,\infty)} |\delta\tau_2|_2]\!]_{G|_2,j}$

   Its proof is similar with STS 1.

Subcase 2: $m = m' + 1 \leq k$. By the definition of function types, there are two parts to show:

Subsubcase 1: $\forall k' < m.(v_1, v_2) \in (\!|\tau_1|\!)_{G,k} \Rightarrow \ldots$

Assume $G' \supseteq G, k' \leq (m-1)$. Pick v_1, v_2 s.t. $(v_1, v_2) \in (\!|\delta\tau_1|\!)_{G',k'}$.

STS: $(t_1[v_1/x][\text{fix}\, f(x).t_1/f], t_2[v_2/x][\text{fix}\, f(x).t_2/f]) \in (\!|\delta\tau_2|\!)^{E,(\delta D)}_{G',k'}$

By IH on premise instantiated with

$\sigma_1' = \sigma_1[v_1/x][\text{fix}\, f(x).t_1/f], \sigma_2' = \sigma_2[v_2/x][\text{fix}\, f(x).t_2/f]$

and $(\sigma_1', \sigma_2') \in (\!|(\Gamma, x:\tau, f:\tau_1 \xrightarrow{\text{diff}\, D} \tau_2, f:U(A_1,A_2))\delta|\!)_{G',k'}$ because

1. $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$ by assumption and Lemma 1.
2. $(v_1, v_2) \in (\!|\tau|\!)_{G',k'}$ from assumption
3. $(F_1, F_2) \in (\!|\delta(\tau_1 \to \tau_2)|\!)_{G',k'}$ from sub-IH $(F_1, F_2) \in (\!|\delta(\tau_1 \xrightarrow{\text{diff}\, D} \tau_2)|\!)_{G,m'}$ and Lemma 1 $(k' < m', G' \supseteq G)$

4. $(F_1, F_2) \in (\!|U(A_1, A_2)|\!)_{G',k}$. STS: $\forall n, i \in \{1,2\}.F_i \in (\!|A_i|\!)_{G|_i,n}$. By Lemma 6, we know $\forall k.\sigma_i \in (\!|\delta|\Gamma|_i|\!)_{g,k}$, it is proved by IH2 on the second and third premises respectively.

Subsubcase 2: $\forall j.F \in [\![|\delta\tau_1|_1 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_1]\!]_{G|_1,j} \wedge F' \in [\![|\delta\tau_1|_2 \overset{\text{exec}(0,\infty)}{\longrightarrow} |\delta\tau_2|_2]\!]_{G|_2,j}$

which is already proved above

This proof is the same as case Fix.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \text{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \text{alloc}\, t_1\, t_2 \ominus \text{alloc}\, t_1'\, t_2' \overset{\text{diff}(D_1+D_2)}{\lesssim} 0 : \{P\}\ \exists\gamma.\ \text{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}}\ \text{R-ALLOC}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $\left(\delta\sigma_1(\text{alloc}\, t_1\, t_2), (\delta\sigma_2(\text{alloc}\, t_1'\, t_2')\right) \in (\!|\delta(\{P\}\ \exists\vec{\gamma}.\ \text{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\})|\!)^{E,0}_{G,k}$,

unfolding the definition of $(\!|\tau|\!)^E_{G,k}$.

STS: $\left(\delta\sigma_1(\text{alloc}\, t_1\, t_2), (\delta\sigma_2(\text{alloc}\, t_1'\, t_2')\right) \in (\!|\delta(\{P\}\ \exists\gamma.\ \text{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\})|\!)_{G,k}$

Unfold the definition of $(\!|\delta(\{P\}\ \exists\gamma.\ \text{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\})|\!)_{G,k}$.

Pick $G' \supseteq G$, H_1, H_2,$k' \leq k, k'' < k', k'''$. Assume $\delta\sigma_1(\text{alloc}\, t_1\, t_2); H_1 \Downarrow_f^{k''}$ (1), $\delta\sigma_2(\text{alloc}\, t_1'\, t_2'); H_2 \Downarrow_f^{k'''}$ (2).

Because t_1, t_2 are sub terms of $\delta\sigma$ $\text{alloc}\, t_1\, t_2$, t_1', t_2' are sub terms of $\delta\sigma$ $\text{alloc}\, t_1'\, t_2'$, from (1)(2), we get: $\delta\sigma_1 t_1 \Downarrow_f^{k_1} \wedge \delta\sigma_1 t_2 \Downarrow_f^{k_2}$ (3), $\delta\sigma_2 t_1' \Downarrow_f \wedge \delta\sigma_2 t_2' \Downarrow_f$ (4).

From (3),(4), we get:

$\exists v_1, v_2, k_1, k_2.\delta\sigma_1 t_1 \Downarrow^{k_1,c_1} v_1 \wedge \delta\sigma_1 t_2 \Downarrow^{k_2,c_2} v_2 \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_2 + c_{alloc}$ (a),

$\exists v_1', v_2'.\, k_1', k_2'.\delta\sigma_2 t_1' \Downarrow^{k_1',c_1'} v_1' \wedge \delta\sigma_2 t_2' \Downarrow^{k_2',c_2'} v_2' \wedge k''' = k_1' + k_2' + 1 \wedge c' = c_1' + c_2' + c_{alloc}$ (b).

From (a),(b) and the evaluation rules E-alloc we get:

$$\exists l.(\mathsf{alloc}\ t_1\ t_2)\delta\sigma; H_1 \Downarrow_f^{k'',c} l; H_1, l \to [v_2, \ldots v_2] \tag{c}$$

$$\exists l'.(\mathsf{alloc}\ t_1'\ t_2')\delta\sigma; H_2 \Downarrow_f^{K''',c'} l'; H_2, l' \to [v_2', \ldots v_2'] \tag{d}$$

By IH on the first premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k''}$ using lemma 1, we get:

$$(\delta\sigma_1\, t_1, \delta\sigma_2\, t_1') \in (\!|\mathrm{int}[I]|\!)^{E,\delta(D_1)}_{G',k''} \tag{*}$$

Unfold the definition of $(\!|\mathrm{int}[I]|\!)^{E}_{G',k''}$ and using (a) and $k_1 \le k''$, we know:

$$(v_1, v_1') \in (\!|\mathrm{int}[I]|\!)_{G',k''-k_1} \Rightarrow v_1 = v_1' = I \wedge c_1 - c_1' \le \delta D_1 \tag{ih1}$$

By IH on the second premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'-k_1}$ using lemma 1, we get:

$$(\delta\sigma_1\, t_2, \delta\sigma_2\, t_2') \in (\!|\tau|\!)^{E,\delta D_2}_{G',k'-k_1} \tag{**}$$

Unfold the definition of (**) and using (a), (b) and $k_2 \le k' - k_1$, we know

$$(v_2, v_2') \in (\!|\tau|\!)_{G',k'-k_1-k_2} = (\!|\tau|\!)_{G',k'-k''+1} \wedge c_2 - c_2' \le \delta D_2 \tag{ih2}$$

Let us assume: $G'' = G'[r \to (l, l', \tau, I)]$ (e), $H_1' = H_1, l \to [v_2, v_2, \ldots, v_2]$ (f), $H_2' = H_2, l' \to [v_2', v_2', \ldots, v_2']$ (g).

we want to show 3 cases.

TS1: $(l, l') \in (\!|\mathsf{Array}_\gamma[I]\ \tau|\!)_{G'',k'-k''}$

it is proved by unfolding the definition and using the assumption (e).

TS2: $(H_1', H_2') \vDash_{G'',k'-k''} P \star \gamma \to \mathbb{N}$

$G''(\gamma) = (l, l', \tau, I)$, $H_1'' = l \to [v_2, v_2, \ldots, v_2]$, $H_2'' = l' \to [v_2', v_2', \ldots, v_2']$

STS1: $\forall i \le I, (H_1''(l)[i], H_2''(l')[i]) \in (\!|\tau|\!)_{G'', k'-k''-1}$.

It is proved by using ih2 and Lemma 1.

STS2: $\forall i \le I, H_1'(l)[i] \neq H_2''(l')[i] \to i \in \beta = \mathbb{N}$. It is trivial.

TS3: $c - c' \le \delta(D_1 + D_2)$ proved from ih1 and ih2.

This proof is complete.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathrm{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \Box\tau \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\, t_1\, t_2 \ominus \mathsf{alloc}\, t_1'\, t_2' \lesssim 0 : \{P\}\ \exists\gamma.\ \overset{\mathrm{diff}(D_1+D_2)}{\mathsf{Array}_\gamma}[I]\ \Box\tau\ \{P \star \gamma \to \emptyset\}}\ \text{R-ALLOC-BOX}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $\left(\delta\sigma_1(\mathsf{alloc}\, t_1\, t_2), (\delta\sigma_2(\mathsf{alloc}\, t_1'\, t_2')\right) \in (\!|\delta(\{P\}\ \exists\vec{\gamma}.\ \overset{\mathrm{diff}(D_1+D_2)}{\mathsf{Array}_\gamma}[I]\ \tau\ \{P \star \gamma \to \emptyset\})|\!)^{E,0}_{G,k}$

By unfolding the definition of $(\!|\tau|\!)^E_{G,k}$

STS: $\left(\delta\sigma_1(\mathsf{alloc}\, t_1\, t_2), (\delta\sigma_2(\mathsf{alloc}\, t_1'\, t_2')\right) \in (\!|\delta(\{P\}\ \exists\gamma.\ \overset{\mathrm{diff}(D_1+D_2)}{\mathsf{Array}_\gamma}[I]\ \tau\ \{P \star \gamma \to \emptyset\})|\!)_{G,k}$

Unfold the definition of $(\!|\delta\ \{P\}\ \exists\gamma.\ \overset{\mathrm{diff}(D_1+D_2)}{\mathsf{Array}_\gamma}[I]\ \tau\ \{P \star \gamma \to \emptyset\}|\!)_{G,k}$.

Pick $G^{'} \supseteq G$, H_1, H_2,$k' \le k, k'' < k', k'''$. Assume $\delta\sigma_1(\mathsf{alloc}\, t_1\, t_2); H_1 \Downarrow_f^{k''}$ (1), $\delta\sigma_2(\mathsf{alloc}\, t_1'\, t_2'); H_2 \Downarrow_f^{k'''}$ (2).

Because t_1, t_2 are sub terms of $\delta\sigma\ \mathsf{alloc}\, t_1\, t_2$, t_1', t_2' are sub terms of $\delta\sigma\ \mathsf{alloc}\, t_1'\, t_2'$. From (1)(2), we get:

$\delta\sigma_1 t_1 \Downarrow_f^{k_1} \wedge \delta\sigma_1 t_2 \Downarrow_f^{k_2}$ (3), $\delta\sigma_2 t_1' \Downarrow_f \wedge \delta\sigma_2 t_2' \Downarrow_f$ (4).

From (3),(4), we get: $\exists v_1, v_2, k_1, k_2. \delta\sigma_1 t_1 \Downarrow^{k_1, c_1} v_1 \wedge \delta\sigma_1 t_2 \Downarrow^{k_2, c_2} v_2 \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_2 + c_{alloc}$ (a)

$\exists v_1', v_2'. k_1', k_2'. \delta\sigma_2 t_1' \Downarrow^{k_1', c_1'} v_1' \wedge \delta\sigma_2 t_2' \Downarrow^{k_2', c_2'} v_2' \wedge k''' = k_1' + k_2' + 1 \wedge c' = c_1' + c_2' + c_{alloc}$ (b)

From (a),(b) and evaluation rules we get:

$$\exists l.(\text{alloc } t_1\ t_2)\delta\sigma; H_1 \Downarrow_f^{k'',c} l; H_1, l \to [v_2, \ldots v_2] \tag{c}$$

$$\exists l'.(\text{alloc } t_1'\ t_2')\delta\sigma; H_2 \Downarrow_f^{c'} l'; H_2, l' \to [v_2', \ldots v_2'] \tag{d}$$

By IH on the first premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k''}$ using lemma 1, we get:

$$(\delta\sigma_1 t_1, \delta\sigma_2 t_1') \in (\!|\text{int}[I]|\!)_{G',k''}^{E,\delta D_1} \tag{*}$$

Unfold the definition of $(\!|\text{int}[I]|\!)_{G',k''}^{E}$ and using (a) and $k_1 \le k''$, we know:

$$(v_1, v_1') \in (\!|\text{int}[I]|\!)_{G',k''-k_1} \Rightarrow v_1 = v_1' = I \wedge c_1 - c_1' \le \delta D_1 \qquad \text{ih1}$$

By IH on the second premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'-k_1}$ using lemma 1, we get:

$$(\delta\sigma_1 t_2, \delta\sigma_2 t_2') \in (\!|\tau|\!)_{G',k'-k_1}^{E,\delta D_2} \tag{**}$$

Unfold the definition of (**) and using (a), (b) and $k_2 \le k' - k_1$, we know

$$(v_2, v_2') \in (\!|\Box\tau|\!)_{G',k'-k_1-k_2} = (\!|\Box\tau|\!)_{G',k'-k''+1} \wedge c_2 - c_2' \le \delta D_2 \Rightarrow v_2 = v_2' \qquad \text{ih2}$$

Let us assume: $G'' = G'[r \to (l, l', \tau, I)]$ (e), $H_1' = H_1, l \to [v_2, v_2, \ldots, v_2](f)$, $H_2' = H_2, l' \to [v_2', v_2', \ldots, v_2'](g)$.

we want to show 3 cases.

TS1: $(l, l') \in (\!|\text{Array}_\gamma[I]\ \tau|\!)_{G'',k'-k''}$

it is proved by unfolding the definition and using the assumption (e).

TS2: $(H_1', H_2') \vDash_{G'',k'-k''} P \star \gamma \to \emptyset$

$G''(\gamma) = (l, l', \tau, I)$, $H_1'' = l \to [v_2, v_2, \ldots, v_2]$, $H_2'' = l' \to [v_2', v_2', \ldots, v_2']$

STS1: $\forall i \le I, (H_1''(l)[i], H_2''(l')[i]) \in (\!|\tau|\!)_{G'', k'-k''-1}$

It is proved by using ih2 and Lemma 1.

STS2: $\forall i \le I, H_1'(l)[i] \neq H_2''(l')[i] \to i \in \emptyset$

We know that $\forall i \le I. H_1(l)[i] = v_2, H_1'(l')[i] = v_2'$ and $v_2 = v_2'$ from ih2. We can not find any i that make $H_1'(l)[i] \neq H_2''(l')[i]$ hold. It is proved.

TS3: $c - c' \le \delta(D_1 + D_2)$ from ih1 and ih2.

This proof is complete.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D_1 : \tau_1 \xrightarrow{\text{diff}\,D} \tau_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash u_1 \ominus u_2 \lesssim D_2 : \tau_1}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1\, u_1 \ominus t_2\, u_2 \lesssim D + D_1 + D_2 : \tau_2}\ \text{R-APP}$$

The first one talks about relational application. When we assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$. We would like to show, after the suitable substitution of index variables and the typing context, two applications related at type τ_1 are in the expression interpretation of type τ_2.

$$(\delta\sigma_1(t_1\, u_1), (\delta\sigma_2(t_2\, u_2)) \in (\!|\delta\tau_2|\!)^{E,\delta(D+D_1+D_2)}_{G,k}.$$

We unfold the definiton of expression interpretation, and first pick $k' \le k$, v_1, v_2, then assume that $\delta\sigma_1(t_1\, u_1) \Downarrow^{c,k'} v_1 \wedge \delta\sigma_2(t_2\, u_2) \Downarrow^{c'} v_2(a)$. We still need to show that: (v_1, v_2) $\in (\!|\delta\tau_2|\!)_{G,k-k'} \wedge c - c' \le \delta(D + D_1 + D_2)$.

From the evaluation (a) and the evaluation rule R-App, we are able to know

$\exists \ldots . (\delta\sigma_1\, t_1) \Downarrow^{k_1,c_1} \text{fix}\, f(x).t_1' \wedge (\delta\sigma_1\, u_1) \Downarrow^{k_1',c_1'} v_1' \wedge t_1'[\text{fix}\, f(x).t_1'/f][v_1'/x] \Downarrow^{k_1'',c_1''} v_1 \wedge k' = k_1 +$

$k_1' + k_1'' + 1 \wedge c = c_1 + c_1' + c_1'' + c_{app}(1)$. and $\exists \ldots . (\delta\sigma_2 t_2) \Downarrow^{c_2} \mathsf{fix} f(x).t_2' \wedge (\delta\sigma_2 u_2) \Downarrow^{c_2'} v_2' \wedge t_2'[\mathsf{fix} f(x).t_2'/f][v_2'/x] \Downarrow^{c_2''} v_2 \wedge c = c_2 + c_2' + c_2'' + c_{app}(2)$.

Induction Hypothesis on the first premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$ tells us that:

$$(\delta\sigma_1 t_1, \delta\sigma_2 t_2) \in (\!|\delta\tau_1 \overset{\mathrm{diff}\,\delta(D)}{\longrightarrow} \delta\tau_2|\!)^{E,\delta(D_1)}_{G,k} \tag{b}$$

Similarly, by IH on the second premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$, we know the following:

$$(\delta\sigma_1 u_1, \delta\sigma_2 u_2) \in (\!|\delta\tau_1|\!)^{E,\delta(D_2)}_{G,k} \tag{c}$$

Now we unfold the expression interpretation definition of (b), using the inequality $k_1 < k$ known from the statement (1). We can conclude the following conclustion about the head of the application.

$$(\mathsf{fix} f(x).t_1', \mathsf{fix} f(x).t_2') \in (\!|\delta\tau_1 \overset{\mathrm{diff}\,\delta(D)}{\longrightarrow} \delta\tau_2|\!)_{G,k-k_1} \wedge c_1 - c_2 \leq \delta(D_1) \tag{3}$$

Unfold the definition of (c), because $k_1' < k$, from (2), we get:

$$(v_1', v_2') \in (\!|\delta\tau_1|\!)_{G,k-k_1'} \wedge c_1' - c_2' \leq \delta(D_2) \tag{4}$$

We want to show $(v_1, v_2) \in (\!|\delta\tau_2|\!)_{G,k-k',D}$ and $c_1'' - c_2'' \leq \delta(D)$.

Considering both the third statements in (1),(2), it is suffice to show:

$(t_1'[\mathsf{fix} f(x).t_1'/f][v_1'/x], t_2'[\mathsf{fix} f(x).t_2'/f][v_2'/x]) \in (\!|\delta(\tau_2)|\!)^{E,\delta(D)}_{G,k-k'+k_1''}$

Use Lemma 1 on (3) with $k - k' + k_1'' \leq k - k_1$. we get:

$$(\mathsf{fix} f(x).t_1', \mathsf{fix} f(x).t_2') \in (\!|\delta\tau_1 \overset{\mathrm{diff}\,\delta(D)}{\longrightarrow} \delta\tau_2|\!)_{G,k-k'+k_1''} \tag{5}$$

Unfold (5). Because $G \supseteq G$ and $(k - k' + k_1'') \leq k - k_1 - 1$

By using Lemma 1 on (4) and $(k - k' + k_1'') \leq k - k_1'$, we get :

$$(v_1', v_2') \in (\!|\delta\tau_1|\!)_{G,k-k'+k_1''} \tag{6}$$

Using (6), we know:

$$(t_1'[\text{fix}\, f(x).t_1'/f][v_1'/x], t_2'[\text{fix}\, f(x).t_2'/f][v_2'/x]) \in (\!|\delta\tau_2|\!)^{E,\delta(D)}_{G,k-k'+k_1''} \tag{7}$$

Unfold (7) and we know: $c_1'' - c_2'' \leq \delta(D)$.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \text{int}[I'] \qquad \Delta \models I' \leq I \qquad \Sigma;\Delta \vdash P\ \text{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \text{read}\ t_1\ t_2 \ominus \text{read}\ t_1'\ t_2' \lesssim 0 : \{P \star \gamma \to \beta\}\ \exists\_.\tau\ \{P \star \gamma \to \beta\}}\ \text{R-R}$$

Assume that $\vdash \delta : \Delta$ and $\models \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(\text{read}\ t_1\ t_2), \delta\sigma_2(\text{read}\ t_1'\ t_2')) \in (\!|\delta\ \{P \star \gamma \to \beta\}\ \exists\_.\tau\ \{P \star \gamma \to \beta\}|\!)^{E,0}_{G,k}$

Because read $t_1\ t_2$ is a value.

STS: $(\delta\sigma_1(\text{read}\ t_1\ t_2), \delta\sigma_2(\text{read}\ t_1'\ t_2')) \in (\!|\delta\ \{P \star \gamma \to \beta\}\ \exists\_.\tau\ \{P \star \gamma \to \beta\}|\!)_{G,k}$

Unfold the definition of $(\!|\delta\ \{P \star \gamma \to \beta\}\ \exists\_.\tau\ \{P \star \gamma \to \beta\}|\!)_{G,k}$.

Pick $G^{'} \supseteq G$, H_1, H_2, $k' \leq k$, $k'' < k'$, k'''. Assume $(H_1, H_2) \models_{G',k'} P \star \gamma \to \beta$ (1), $\delta\sigma_1(\text{read}\ t_1\ t_2); H_1 \Downarrow_f^{k''}$ (2), $\delta\sigma_2(\text{read}\ t_1'\ t_2'); H_2 \Downarrow_f^{k''}$ (3).

Because t_1, t_2 are sub terms of read $t_1\ t_2$, t_1', t_2' are sub terms of read $t_1'\ t_2'$. From (2)(3), we get: $\delta\sigma_1 t_1 \Downarrow_f^{k_1} \wedge \delta\sigma_1 t_2 \Downarrow_f^{k_2}$ (4), $\delta\sigma_2 t_1' \Downarrow_f^{k_1'} \wedge \delta\sigma_2 t_2' \Downarrow_f^{k_2'}$ (5).

From (4),(5), we get: $\exists l, n, k_1, k_2.\delta\sigma_1 t_1 \Downarrow^{k_1,c_1} l \wedge \delta\sigma_1 t_2 \Downarrow^{k_2,c_1'} n \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_1' + c_{read}$ (a).

$\exists l', n', k_1', k_2'.\delta\sigma_2 t_1' \Downarrow^{k_1',c_2} l' \wedge \delta\sigma_2 t_2' \Downarrow^{k_2',c_2'} n' \wedge k''' = k_1' + k_2' + 1 \wedge c' = c_2 + c_2' + c_{read}$ (b)

From (a),(b) and the evaluation rule we get:

$\exists v.\delta\sigma_1(\text{read } t_1\ t_2); H_1 \Downarrow_f^{k'',c} v; H_1 \wedge H_1(l)[n] = v$ (c)

$\exists v'.\delta\sigma_2(\text{read } t_1'\ t_2'); H_2 \Downarrow_f^{k''',c'} v'; H_2 \wedge H_2(l')[n'] = v'$ (d)

By IH on the first premise instantiated with $(\sigma_1,\sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$ by lemma 1. we get:

$$(\delta\sigma_1 t_1, \delta\sigma_2 t_1') \in (\!|\mathsf{Array}_\gamma[I]\ \delta\tau|\!)^{E,D_1}_{G',k'} \tag{6}$$

Unfold (6), since $k_1 \le k'$ and $\delta\sigma_1 t_1 \Downarrow^{k_1} l$ and $\delta\sigma_2 t_1' \Downarrow l'$, we know

$$(l,l') \in (\!|\mathsf{Array}_\gamma[I]\ \delta\tau|\!)_{G',k'-k_1} \wedge c_1 - c_2 \le \delta D_1 \tag{e}$$

From (e), we know : $G'(\gamma) = (l,l',\tau,I)$ (7).

By IH on the second premise instantiated with $(\sigma_1,\sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$ by lemma 1. we get:

$$(\delta\sigma_1 t_2, \delta\sigma_2 t_2') \in (\!|\delta\text{int}[I']|\!)^{E}_{G',k'} \tag{8}$$

Unfold (8), since $k_2 \le k'$,we know $(n,n') \in (\!|\text{int}[I']|\!)_{G',k'-k_2} \Rightarrow n = n' = I' \wedge c_1' - c_2' \le \delta D_2(f)$.

Let us assume: $G' = G'$ (g).

STS1: $(H_1,H_2) \vDash_{G',k'-k''} P \star \gamma \to \beta$. By Lemma 7 and (1),this is proved.

STS2: $(v,v') \in (\!|\tau|\!)_{G',k'-k''}$. From (c),(d). we know $H_1(l)[n] = v$ and $H_2(l')[n'] = v'$. based on (e) and (f), unfold (1), we know: $\forall i \le n, (H_1(l)(i), H_2(l')(i)) \in (\!|\tau|\!)_{G',k'-1}$ s.t we know $(H_1(l)[n], H_2(l')[n']) \in (\!|\tau|\!)_{G',k'-1}$. Because $k'-k'' \le k'-1$, By Lemma 1, We get: $(v,v') \in (\!|\tau|\!)_{G',k'-k''}$

STS3: $c - c' \le \delta(D_1 + D_2)$, which is proved by (e),(f).

This completes the proof of case read.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Sigma;\Delta \models I' \le I \qquad \Sigma;\Delta;\Phi_a \models I' \notin \beta \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \lesssim 0 : \{P \star \gamma \to \beta\}\ \exists\_.\Box\tau^{\mathrm{diff}(D_1+D_2)}\ \{P \star \gamma \to \beta\}}\ \text{R-RB}$$

Assume that $\vdash \delta : \Delta$ and $\models \Phi\delta$ and $(\sigma_1,\sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(\mathsf{read}\ t_1\ t_2), \delta\sigma_2(\mathsf{read}\ t_1'\ t_2')) \in (\!|\delta\ \{P \star \gamma \to \beta\}^{\mathrm{diff}(D_1+D_2)}\ \exists\_.\tau\ \{P \star \gamma \to \beta\}|\!)^{E,0}_{G,k,}$

Because $\mathsf{read}\ t_1\ t_2$ is a value.

STS: $(\delta\sigma_1(\mathsf{read}\ t_1\ t_2), \delta\sigma_2(\mathsf{read}\ t_1'\ t_2')) \in (\!|\delta\ \{P \star \gamma \to \beta\}^{\mathrm{diff}(D_1+D_2)}\ \exists\_.\tau\ \{P \star \gamma \to \beta\}|\!)_{G,k}$

Unfold the definition of $(\!|\delta\ \{P \star \gamma \to \beta\}^{\mathrm{diff}(D_1+D_2)}\ \exists\_.\tau\ \{P \star \gamma \to \beta\}|\!)_{G,k}$.

Pick $G^{'} \supseteq G$, H_1, H_2, $k' \le k$, $k'' < k'$, k'''. Assume $(H_1,H_2) \models_{G',k'} P \star \gamma \to \beta$ (1), $\delta\sigma_1(\mathsf{read}\ t_1\ t_2); H_1 \Downarrow_f^{k''}$ (2), $\delta\sigma_2(\mathsf{read}\ t_1'\ t_2'); H_2 \Downarrow_f^{k''}$ (3).

Because t_1, t_2 are sub terms of $\mathsf{read}\ t_1\ t_2$, t_1', t_2' are sub terms of $\mathsf{read}\ t_1'\ t_2'$. From (2)(3), we get $\delta\sigma_1 t_1 \Downarrow_f^{k_1} \wedge \delta\sigma_1 t_2 \Downarrow_f^{k_2}$ (4), $\delta\sigma_2 t_1' \Downarrow_f^{k_1'} \wedge \delta\sigma_2 t_2' \Downarrow_f^{k_2'}$ (5).

From (4),(5), we get: $\exists l, n, k_1, k_2.\, \delta\sigma_1 t_1 \Downarrow^{k_1,c_1} l \wedge \delta\sigma_1 t_2 \Downarrow^{k_2,c_1'} n \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_1' + c_{read}$ (a),

$\exists l', n', k_1', k_2'.\, \delta\sigma_2 t_1' \Downarrow^{k_1',c_2} l' \wedge \delta\sigma_2 t_2' \Downarrow^{k_2',c_2'} n' \wedge k''' = k_1' + k_2' + 1 \wedge c' = c_2 + c_2' + c_{read}$ (b).

From (a),(b) and evaluation rules we get:

$$\exists v.\, H_1; \delta\sigma_1(\mathsf{read}\ t_1\ t_2) \Downarrow_f^{k'',c} v; H_1 \wedge H_1(l)[n] = v \tag{c}$$

$$\exists v'.\, H_2; \delta\sigma_2(\mathsf{read}\ t_1'\ t_2') \Downarrow_f^{k''',c'} v'; H_2 \wedge H_2(l')[n'] = v' \tag{d}$$

By IH on the first premise instantiated with $(\sigma_1,\sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$ by lemma 1. we get: $(\delta\sigma_1 t_1, \delta\sigma_2 t_1') \in (\!|\mathsf{Array}_\gamma[I]\ \delta\tau|\!)^{E,D_1}_{G',k',}$ (6). Unfold (6), since $k_1 \le k'$ and $\delta\sigma_1 t_1 \Downarrow^{k_1} l$ and

$\delta\sigma_2 t_1' \Downarrow l'$, we know:

$$(l, l') \in (\!|\mathsf{Array}_\gamma[I]\ \delta\tau|\!)_{G', k'-k_1} \wedge c_1 - c_2 \leq \delta D_1 \tag{e}$$

From (e), we know : $G'(\gamma) = (l, l', \tau, I)$ (7). By IH on the second premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$ by lemma 1. we get: $(\delta\sigma_1 t_2, \delta\sigma_2 t_2') \in (\!|\delta\mathrm{int}[I']|\!)^E_{G',k'}$ (8). Unfold (8), since $k_2 \leq k'$,we know that $(n, n') \in (\!|\mathrm{int}[I']|\!)_{G',k'-k_2} \Rightarrow n = n' = I' \wedge c_1' - c_2' \leq \delta D_2$ (f).
Let us assume: $G' = G'$ (g).

STS1: $(H_1, H_2) \vDash_{G',k'-k''} P \star \gamma \to \beta$. By Lemma 7 and (1),this is proved.

STS2: $(v, v') \in (\!|\Box\tau|\!)_{G',k'-k''}$. from (c),(d). we know $H_1(l)[n] = v$ and $H_2(l')[n'] = v'$, based on (e) and (f),unfold (1), we know: $\forall i \leq n, (H_1(l)(i), H_2(l')(i)) \in (\!|\tau|\!)_{G',k'-1}$. s.t we know that $(H_1(l)[n], H_2(l')[n']) \in (\!|\tau|\!)_{G',k'-1}$. Because $k' - k'' \leq k' - 1$, By Lemma 1, We get: $(v, v') \in (\!|\tau|\!)_{G',k'-k''}$,consider the premise $i \notin \beta$, we know $v = v'$, so that we prove $(v, v') \in (\!|\Box\tau|\!)_{G',k'-k''}$.

STS3: $c - c' \leq \delta(D_1 + D_2)$, which is proved by (e),(f).

This completes the proof of case read-box.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 : \tau \qquad \Delta;\Phi_a \vDash I' \leq I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \lesssim^{\mathrm{diff}(D_1+D_2+D_3)} 0 : \{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta \cup \{I'\}\}}\ \text{R-U}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$
TS: $(\delta\sigma_1(\mathsf{updt}\ t_1\ t_2\ t_3), \delta\sigma_2(\mathsf{updt}\ t_1'\ t_2'\ t_3')) \in (\!|\delta(\{\gamma \to \beta\}\ \exists\_.^{\mathrm{diff}(D_1+D_2)}\mathrm{unit}\ \{\gamma \to \beta \cup \{I'\}\})|\!)^{E,0}_{G,k}$

Because updt $t_1\ t_2\ t_3$ is value.

STS: $(\delta\sigma_1(\text{updt } t_1\ t_2\ t_3), \delta\sigma_2(\text{updt } t_1'\ t_2'\ t_3')) \in (\!|\delta(\{\gamma \to \beta\}\ \exists\_.\text{unit}^{\text{diff}(D_1+D_2)}\ \{\gamma \to \beta \cup \{I'\}\})|\!)_{G,k}$

Unfold the definition of $(\!|\delta(\{\gamma \to \beta\}\ \exists\_.\text{unit}^{\text{diff}(D_1+D_2)}\ \{\gamma \to \beta \cup \{I'\}\})|\!)_{G,k}$.

Pick $G^{'} \supseteq G$, H_1, H_2, $k' \le k$, $k'' < k'$, k'''. Assume $(H_1, H_2) \vDash_{G',k'} P \star \gamma \to \beta \wedge H_1 = H_{p1} \uplus H_{l1} \wedge H_{l1} = l \to [v, \ldots, v] \wedge H_2 = H_{p2} \uplus H_{l2} \wedge H_{l2} = l' \to [v', \ldots, v']$(1) ,$\delta\sigma_1(\text{updt } t_1\ t_2\ t_3); H_1 \Downarrow_f^{k''}$ (2), $\delta\sigma_2(\text{updt } t_1'\ t_2'\ t_3'); H_2 \Downarrow_f^{k'''}$ (3).

Because t_1, t_2, t_3 are sub terms of updt $t_1\ t_2\ t_3$, t_1', t_2', t_3' are sub terms of updt $t_1'\ t_2'\ t_3'$. From (2)(3), we get $\delta\sigma_1 t_1 \Downarrow_f^{k_1} \wedge \delta\sigma_1 t_2 \Downarrow_f^{k_2} \wedge \delta\sigma_1 t_3 \Downarrow_f^{k_3}$ (4), $\delta\sigma_2 t_1' \Downarrow_f^{k_1'} \wedge \delta\sigma_2 t_2' \Downarrow_f^{k_2'} \wedge \delta\sigma_2 t_3' \Downarrow_f^{k_3'}$ (5).

From (4),(5), we get:$\exists l, n, v, k_1, k_2, k_3. \delta\sigma_1 t_1 \Downarrow^{k_1, c_1} l \wedge \delta\sigma_1 t_2 \Downarrow^{k_2, c_1'} n \wedge \delta\sigma_1 t_3 \Downarrow^{k_3, c_1''} v \wedge k'' = k_1 + k_2 + k_3 + 1 \wedge c = c_1 + c_1' + c_1'' + c_{update}(a)$. and $\exists l', n', v', k_1', k_2', k_3'. \delta\sigma_2 t_1' \Downarrow^{k_1', c_2} l' \wedge \delta\sigma_2 t_2' \Downarrow^{k_2', c_2'} n' \wedge \delta\sigma_2 t_3' \Downarrow^{k_3', c_2''} v' \wedge c' = c_2 + c_2' + c_2'' + c_{update}(b)$.

From evaluation rule: E-Update-F and (a)(b), we know:

$$\text{updt } t_1\ t_2\ t_3; H_1 \Downarrow^{k'', c} (); H_1(l)[n] \leftarrow v \tag{c}$$

$$\text{updt } t_1'\ t_2'\ t_3'; H_2 \Downarrow^{k''', c'} (); H_2(l')[n'] \leftarrow v' \tag{d}$$

By IH on first premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$ by lemma 1. We get:

$$(\delta\sigma_1 t_1, \delta\sigma_2 t_1') \in (\!|\text{Array}_\gamma[I]\ \delta\tau|\!)_{G',k'}^{E,\delta D_1} \tag{6}$$

Similarly, By IH on the second premise and third premise, we get:

$(\delta\sigma_1 t_2, \delta\sigma_2 t_2') \in (\!|\text{int}[I]|\!)_{G',k'}^{E,\delta D_2}$(7), $(\delta\sigma_1 t_3, \delta\sigma_2 t_3') \in (\!|\tau|\!)_{G',k'}^{E,\delta D_3}$(8).

From (6),(7),(8), we know:

$$(l, l') \in (\!|\text{Array}_\gamma[I]\ \tau|\!)_{G',k'-k_1} \Rightarrow G'(\gamma) = (l, l', \tau, I) \wedge c_1 - c_2 \le D_1 \tag{e}$$

$$(n, n') \in (\!|\text{int}[I]|\!)_{G,k'-k_2} \Rightarrow n = n' = I \wedge c_1' - c_2' \le D_2 \tag{f}$$

$$(v, v') \in (\!|\tau|\!)_{G',k'-k_3} \wedge c_1'' - c_2'' \leq D_3 \qquad \text{(g)}$$

Let us assume: $G'' = G'(9)$, $H_1' = H_1(l)[n] \leftarrow v \wedge H_2' = H_2(l')[n'] \leftarrow v'(10)$

STS1: $(H_1', H_2') \vDash_{G',k'-k''} P \star \gamma \rightarrow \beta \cup \{I'\}$

Unfold (1) and consider (e), we know: $\forall i \leq I.(H_{l1}(l)[i], H_{l2}(l')[i]) \in (\!|\tau|\!)_{G',k'-1}(11)$, $\forall i \leq I.(H_{l1}(l)[i] \neq H_{l2}(l')[i]) \Rightarrow i \in \beta(12)$. Since (e), we need to prove two sub cases.

subgoal 1: $\forall i \leq I.(H_{l1}'(l)[i], H_{l2}'(l')[i]) \in (\!|\tau|\!)_{G',k'-k''}$

when i=n, we prove $(H_{l1}'(l)[i], H_{l2}'(l')[i]) \in (\!|\tau|\!)_{G',k'-k_3}$ from (g) and (10), and then use lemma 1.

when $i \neq n$, proved by using lemma 1 on (11)

subgoal 2: $\forall i \leq I.(H_{l1}'(l)[i] \neq H_{l2}'(l')[i]) \Rightarrow i \in \beta \cup \{I'\}$

when i=n, $n \in \{I'\}$ from (f) $\Rightarrow n \in \beta \cup \{I'\}$

when $i \neq n$, proved by using lemma 1 on (12)

STS2: $((), ()) \in (\!|\text{unit}|\!)_{G',k'-k''}$. It is proved by unfolding the definition.

STS3 $c - c' \leq \delta(D_1 + D_2 + D_3)$. It is proved by (e),(f),(g).

This completes the proof of case update.

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \text{int}[I'] \qquad \Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 : \square\tau \qquad \Delta;\Phi_a \vDash I' \leq I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \lesssim_{\text{diff}(D_1+D_2+D_3)} 0 : \{P \star \gamma \rightarrow \beta\}\ \exists\_ : \text{unit}\ \{P \star \gamma \rightarrow \beta \setminus \{I'\}\}}\ \text{R-UB}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(\mathsf{updt}\ t_1\ t_2\ t_3), \delta\sigma_2(\mathsf{updt}\ t_1'\ t_2'\ t_3')) \in (\!|\overset{\mathrm{diff}(D_1+D_2+D_3)}{\delta(\{P\star\gamma\to\beta\}\ \exists\_.\mathrm{unit}\ \{P\star\gamma\to\beta\setminus\{I'\}\})}|\!)^{E,0}_{G,k}$

Because $\mathsf{updt}\ t_1\ t_2\ t_3$ is value.

STS: $(\delta\sigma_1(\mathsf{updt}\ t_1\ t_2\ t_3), \delta\sigma_2(\mathsf{updt}\ t_1'\ t_2'\ t_3')) \in (\!|\overset{\mathrm{diff}(D_1+D_2+D_3)}{\delta(\{P\star\gamma\to\beta\}\ \exists\_.\mathrm{unit}\ \{P\star\gamma\to\beta/\{I'\}\})}|\!)_{G,k}$

Unfold the definition of $(\!|\overset{\mathrm{diff}(D_1+D_2+D_3)}{\delta(\{P\star\gamma\to\beta\}\ \exists\_.\mathrm{unit}\ \{P\star\gamma\to\beta\setminus\{I'\}\})}|\!)_{G,k}$.

Pick $G^{'}\supseteq G$, H_1, H_2, $k'\le k$, $k''<k'$, k'''. Assume $(H_1,H_2)\vDash_{G',k'} P\star\gamma\to\beta\wedge H_1 = H_{p1}\uplus H_{l1}\wedge H_{l1} = l\to[v,\ldots,v]\wedge H_2 = H_{p2}\uplus H_{l2}\wedge H_{l2} = l'\to[v',\ldots,v']$(1) $\delta\sigma_1(\mathsf{updt}\ t_1\ t_2\ t_3); H_1 \Downarrow_f^{k''}$ (2) $\delta\sigma_2(\mathsf{updt}\ t_1'\ t_2'\ t_3'); H_2 \Downarrow_f^{k'''}$ (3).

Because t_1,t_2,t_3 are sub terms of $\mathsf{updt}\ t_1\ t_2\ t_3$, t_1',t_2',t_3' are sub terms of $\mathsf{updt}\ t_1'\ t_2'\ t_3'$ From (2)(3), we get: $\delta\sigma_1 t_1 \Downarrow_f^{k_1}\wedge\delta\sigma_1 t_2\Downarrow_f^{k_2}\wedge\delta\sigma_1 t_3\Downarrow_f^{k_3}$ (4), $\delta\sigma_2 t_1'\Downarrow_f^{k_1'}\wedge\delta\sigma_2 t_2'\Downarrow_f^{k_2'}\wedge\delta\sigma_2 t_3'\Downarrow_f^{k_3'}$ (5).

From (4),(5), we get: $\exists l,n,v,k_1,k_2,k_3.\delta\sigma_1 t_1\Downarrow^{k_1,c_1} l\wedge\delta\sigma_1 t_2\Downarrow^{k_2,c_1'} n\wedge\delta\sigma_1 t_3\Downarrow^{k_3,c_1''} v\wedge k''=k_1+k_2+k_3+1\wedge c=c_1+c_1'+c_1''+c_{update}(a)$

$\exists l',n',v',k_1',k_2',k_3'.\delta\sigma_2 t_1'\Downarrow^{k_1',c_2} l'\wedge\delta\sigma_2 t_2'\Downarrow^{k_2',c_2'} n'\wedge\delta\sigma_2 t_3'\Downarrow^{k_3',c_2''} v'\wedge c' = c_2+c_2'+c_2''+c_{update}(b)$.

From evaluation rule: E-Update-F and (a)(b), we know:

$$\mathsf{updt}\ t_1\ t_2\ t_3; H_1\Downarrow^{k'',c}();H_1(l)[n]\leftarrow v \tag{c}$$

$$\mathsf{updt}\ t_1'\ t_2'\ t_3'; H_2\Downarrow^{k''',c'}();H_2(l')[n']\leftarrow v' \tag{d}$$

By IH on first premise instantiated with $(\sigma_1,\sigma_2)\in(\!|\delta\Gamma|\!)_{G',k'}$ by lemma 1. We get:

$$(\delta\sigma_1 t_1,\delta\sigma_2 t_1')\in(\!|\mathsf{Array}_\gamma[I]\ \delta\tau|\!)^{E,\delta D_1}_{G',k'} \tag{6}$$

Similarly, By IH on the second premise and third premise, we get

$$(\delta\sigma_1 t_2,\delta\sigma_2 t_2')\in(\!|\delta\mathrm{int}[I]|\!)^{E,\delta D_2}_{G',k'} \tag{7}$$

$$(\delta\sigma_1 t_3,\delta\sigma_2 t_3')\in(\!|\square\delta\tau|\!)^{E,\delta D_3}_{G',k'} \tag{8}$$

From (6),(7),(8), we know:

$$(l,l') \in (\!|\mathsf{Array}_\gamma[I]\ \tau|\!)_{G',k'-k_1} \Rightarrow G'(\gamma) = (l,l',\tau,I) \wedge c_1 - c_2 \leq \delta D_1 \quad \text{(e)}$$

$$(n,n') \in (\!|\mathrm{int}[I]|\!)_{G,k'-k_2} \Rightarrow n = n' = I \wedge c_1' - c_2' \leq \delta D_2 \quad \text{(f)}$$

$$v = v' \wedge (v,v') \in (\!|\tau|\!)_{G',k'-k_3} c_1'' - c_2'' \leq \delta D_3 \quad \text{(g)}$$

Let us assume: $G'' = G'(9)$, $H_1' = H_1(l)[n] \leftarrow v \wedge H_2' = H_2(l')[n'] \leftarrow v'(10)$.

STS1: $(H_1', H_2') \vDash_{G',k'-k''} P \star \gamma \rightarrow \beta / \{I'\}$

Unfold (1) and consider (e), we know $\forall i \leq I.(H_{l1}(l)[i], H_{l2}(l')[i]) \in (\!|\delta\tau|\!)_{G',k'-1}(11)$ $\forall i \leq I.(H_{l1}(l)[i] \neq H_{l2}(l')[i]) \Rightarrow i \in \beta(12)$. Since (e), we need to prove two sub cases.

subgoal 1: $\forall i \leq I.(H_{l1}'(l)[i], H_{l2}'(l')[i]) \in (\!|\tau|\!)_{G',k'-k''}$

when i=n, we prove $(H_{l1}'(l)[i], H_{l2}'(l')[i]) \in (\!|\tau|\!)_{G',k'-k_3}$ from (g) and (10), and then use lemma 1.

when $i \neq n$, proved by using lemma 1 on (11)

subgoal 2: $\forall i \leq I.(H_{l1}'(l)[i] \neq H_{l2}'(l')[i]) \Rightarrow i \in \beta / \{I'\}$

when i=n, $H_1'(l)[i] = H_2'(l)[i]$, it is trivially proved. when $i \neq n$, proved by using lemma 1 on (12)

STS2: $((),()) \in (\!|\mathrm{unit}|\!)_{G',k'-k''}$ It is proved by unfolding the definition.

STS3 $c - c' \leq \delta(D_1 + D_2 + D_3)$. It is proved by (e),(f),(g).

This completes the proof of case update-box.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ t_1 \ominus \mathsf{return}\ t_2 \lesssim 0 : \{P\}\ \exists\_.\tau\ \{P\}}\ \overset{\mathrm{diff}(D)}{\text{R-T}}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \Phi\delta$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$ TS: $(\delta\sigma_1(\text{return } t_1), \delta\sigma_2(\text{return } t_2)) \in (\!|\delta\ \{P\}\ \overset{\text{diff}(D)}{\exists\_.\tau}\ \{P\}|\!)^{E,0}_{G,k}$

Because return t is value, STS: $(\delta\sigma_1(\text{return } t_1), \delta\sigma_2(\text{return } t_2)) \in (\!|\delta\ \{P\}\ \overset{\text{diff}(D)}{\exists\_.\tau}\ \{P\}|\!)_{G,k}$

Unfold the definition of $(\!|\delta\ \{P\}\ \overset{\text{diff}(D)}{\exists\_.\tau}\ \{P\}|\!)_{G,k}$. Pick $G^{'} \supseteq G$, H_1, H_2,$k' \leq k, k'' < k', k'''$. Assume $(H_1, H_2) \vDash_{G',k'} P$ (1), $\delta\sigma_1(\text{return } t_1); H_1 \Downarrow^{k''}_f$ (2), $\delta\sigma_2(\text{return } t_2); H_2 \Downarrow^{k'''}_f$ (3).

Because t_1 is the subterm of return t_1 . From (2)(3), We know: $\delta\sigma_1 t_1 \Downarrow^{k_1}$ (4), $\delta\sigma_2 t_2 \Downarrow^{k'''}_f$ (5).

From (4),(5), we know:

$$\exists v_1, k_1, \delta\sigma_1 t_1 \Downarrow^{k_1, c_1} v_1 \wedge k'' = k_1 + 1 \wedge c = c_1 + c_{ret} \tag{a}$$

$$\exists v_2, k_1', \delta\sigma_2 t_2 \Downarrow^{k_1', c_2} v_2 \wedge k''' = k_1' + 1 \wedge c' = c_2 + c_{ret} \tag{b}$$

From (a),(b) and the evaluation rule R-ret: $H_1; \text{return } t_1 \Downarrow^{k'',c}_f v_1; H_1(c)$, $H_2; \text{return } t_2 \Downarrow^{k''',c'}_f v_2; H_2(d)$.

By IH on the premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$ by lemma 1.

We get: $(\delta\sigma_1 t_1, \delta\sigma_2 t_2) \in (\!|\delta\tau|\!)^{E,\delta D}_{G',k'}$ (6). Unfold (6), we get: $(v_1, v_2) \in (\!|\delta\tau|\!)_{G',k'-k_1}$ (7).

Let us assume: $G'' = G'$(8), $H_1' = H_1' \wedge H_2' = H_2' \leftarrow v'$(9).

STS1: $(H_1', H_2') \vDash_{G',k'-k''} P$. It is trivially true.

STS2: $\left(v_1, v_2\right) \in (\!|\tau|\!)_{G',k'-k''}$. It is proved by (7) using Lemma 1.

STS3: $c - c' \leq \delta D$ which is proved from (a),(b).

This completes the proof.

Case

$$\frac{\begin{array}{c} P = P_1 \star P_2 \qquad \Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D_1 : \{P_1\}\ \overset{\text{diff}(D)}{\exists \vec{\gamma}_1.\tau_1}\ \{Q_1 \star Q_2\} \\ \Delta, \vec{\gamma}_1 : \vec{\mathbb{L}}; \Phi_a; \Gamma, x : \tau_1 \vdash t_1' \ominus t_2' \lesssim D_2 : \{Q_1 \star P_2\}\ \overset{\text{diff}(D')}{\exists \vec{\gamma}_2.\tau_2}\ \{Q\} \end{array}}{\Delta; \Phi_a; \Gamma \vdash \text{let}\{x\} = t_1 \text{ in } t_1' \ominus \text{let}\{x\} = t_2 \text{ in } t_2' \lesssim 0 : \{P\}\ \overset{\text{diff}(D_1+D_2+D+D')}{\exists \vec{\gamma}_1 \vec{\gamma}_2 : \tau_2}\ \{Q \star Q_2\}}\ \text{R-LET}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(\mathsf{let}\{x\} = t_1 \ \mathsf{in} \ t_1'), \delta\sigma_2(\mathsf{let}\{x\} = t_2 \ \mathsf{in} \ t_2')) \in (\!|\delta \ \{P\} \ \exists \vec{\gamma}_1 \vec{\gamma}_2 : \tau_2 \ \{Q \star Q_2\}|\!)^{E,0}_{G,k}$ $^{\mathrm{diff}(D+D'+D_1+D_2)}$

Because $\mathsf{let}\{x\} = t_1 \ \mathsf{in} \ t_1'$ is value.

STS: $(\delta\sigma_1(\mathsf{let}\{x\} = t_1 \ \mathsf{in} \ t_1'), \delta\sigma_2(\mathsf{let}\{x\} = t_2 \ \mathsf{in} \ t_2')) \in (\!|\delta \ \{P\} \ \exists \vec{\gamma}_1 \vec{\gamma}_2 : \tau_2 \ \{Q \star Q_2\}|\!)_{G,k}$ $^{\mathrm{diff}(D+D'+D_1+D_2)}$

Unfold the definition of $(\!|\delta \ \{P\} \ \exists \vec{\gamma}_1 \vec{\gamma}_2 : \tau_2 \ \{Q \star Q_2\}|\!)_{G,k}$ $^{\mathrm{diff}(D+D'+D_1+D_2)}$.

Pick $G^{'} \supseteq G$, H_1, $H_2, k' \leq k, k'' < k', k'''$. Assume $(H_1, H_2) \vDash_{G',k'} P(1)$, $\delta\sigma_1(\mathsf{let}\{x\} = t_1 \ \mathsf{in} \ t_1'$ $); H_1 \Downarrow_f^{k''} (2)$, $\delta\sigma_2(\mathsf{let}\{x\} = t_2 \ \mathsf{in} \ t_2'); H_2 \Downarrow_f^{k'''} (3)$.

Because t_1 is the subterm of $\mathsf{let}\{x\} = t_1 \ \mathsf{in} \ t_1'$. From (2) ,we know:

$\exists v_1, v_1', v_1'', v_1''', k_1, k_2, k_3, k_4, H_1', H_1''. \delta\sigma_1 t_1 \Downarrow^{k_1, c_1} v_1 \wedge v_1; H_1 \Downarrow_f^{k_2, c_1'} v_1'; H_1' \delta\sigma_1 t_1'[v_1'/x] \Downarrow^{k_3, c_1''}$
$v_1'' \wedge v_1''; H_1' \Downarrow_f^{k_4, c_1'''} v_1'''; H_1'' \wedge k'' = k_1 + k_2 + k_3 + k_4 + 1 \wedge c = c_1 + c_1' + c_1'' + c_1''' + c_{let}(a).$

Similarly, from (3) , we get:

$\exists v_2, v_2', v_2'', v_2''', k_1', k_2', k_3', k_4', H_2', H_2''.$
$\delta\sigma_1 t_2 \Downarrow^{k_1', c_2} v_2 \wedge v_2; H_2 \Downarrow_f^{k_2', c_2'} v_2'; H_2' \wedge \delta\sigma_1 t_2'[v_2'/x] \Downarrow^{k_3', c_2''} v_2'' \wedge v_2''; H_2' \Downarrow_f^{k_4', c_2'''} v_2'''; H_2'' \wedge k'' =$
$k_1' + k_2' + k_3' + k_4' + 1 \wedge c' = c_2 + c_2' + c_2'' + c_2''' + c_{let}(b).$

From evaluation rule: E-Let-F and (a)(b), we know:

$$\exists H_a, H_c, H_a', H1_{Q1}, H1_{Q2}, H1_Q, s.t. H_1 = H_a \uplus H_c \wedge v1; H_a \Downarrow_f v_1'; H_a' \wedge H_a' = H1_{Q1} \uplus H1_{Q2}$$

$$\wedge v_1''; H1_{Q1} \uplus H_c \Downarrow_f v_1'''; H1_Q \wedge H_1'' = H1_Q \uplus H1_{Q2} \tag{c}$$

$$\exists H_a, H_c, H_a', H1_{Q1}, H1_{Q2}, H1_Q, s.t. H_1 = H_a \uplus H_c \wedge v1; H_a \Downarrow_f v_1'; H_a' \wedge H_a' = H1_{Q1} \uplus H1_{Q2}$$

$$\wedge v_1''; H1_{Q1} \uplus H_c \Downarrow_f v_1'''; H1_Q \wedge H_1'' = H1_Q \uplus H1_{Q2} \tag{d}$$

From (1) : $(H_1, H_2) \vDash_{G',k'} P_1 \star P_2$, we assume that

$$\exists G_a, G_b : (H_1 = H_a \uplus H_c) \quad \text{and} \quad (H_2 = H_b \uplus H_d)$$

$$(G' = G_a \uplus G_c) \quad \text{and} \quad (H_a, H_b) \vDash_{G_a,k'} P_1 \quad \text{and} \quad (H_c, H_d) \vDash_{G_c,k'} P_2 \qquad (*)$$

STS1: $(H_1'', H_2'') \vDash_{G'',k'-k''} Q \star Q_2$. STS2: $(v_1''', v_2''') \in (\!|\tau_2|\!)_{G'',k'-k''}$. STS3: $c - c' \leq \delta(D_1 + D_2 + D + D')$.

By IH on the second premise instantiated with $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G',k'}$, we know: $(\delta\sigma_1 t_1, \delta\sigma_2 t_2) \in (\!|\{P_1\}\ \exists\vec{\gamma}_1.\tau\ \{Q_1 \star Q_2\}|\!)^{E,D_1}_{G',k'}(4)$. Unfold the definition of $(\!|\{P_1\}\ \exists\vec{\gamma}_1.\tau\ \{Q_1 \star Q_2\}|\!)^{E,D_1}_{G',k'}$, we use (a), we get: $(v_1, v_2) \in (\!|\{P_1\}\ \exists\vec{\gamma}_1.\tau_1\ \{Q_1 \star Q_2\}|\!)_{G',k'-k_1} \wedge c_1 - c_2 \leq D_1(e)$.

To unfold (e), we choose: $k_f \leq k' - k_1$ and $k_f' < k_f$ and $G_f \supseteq G'$ and $H_a \uplus H_c, H_b \uplus H_d$ and (1),(a),(b). We have $(H_a, H_b) \vDash_{G_f,k_f} P_1 \wedge v_1; H_a \Downarrow_f \wedge v_2; H_b \Downarrow_f$.

Unfold the definition, we get: $v_1; H_a \Downarrow_f^{c_1'} v_1'; H_a' \wedge v_2; H_b \Downarrow_f^{c_2'} v_2'; H_b' \wedge H_a', H_b' \vDash Q_1 \star Q_2$.

$$(v_1', v_2') \in (\!|\tau_1|\!)_{G',k'-k_1} \wedge c_1' - c_2' \leq \delta D \qquad (k)$$

By IH on the third premise, we know: $(\delta\sigma_1 v_1'', \delta\sigma_2 v_2'') \in (\!|\{Q_1 \star P_2\}\ \exists\vec{\gamma}_1.\tau\ \{Q\}|\!)^{E,D_2}_{G',k'}(5)$.
Unfold the definition of $(\!|\{Q_1 \star P_2\}\ \exists\vec{\gamma}_1.\tau_2\ \{Q\}|\!)^{E,D_2}_{G',k'}$,use (a), we get:
$(v_1, v_2) \in (\!|\{Q_1 \star P_2\}\ \exists\vec{\gamma}_1.\tau_2\ \{Q\}|\!)_{G',k'-k_1} \wedge c_1'' - c_2'' \leq D_2(f)$ We have $(H_a', H_b') \vDash_{G_f,k_f} Q_1 \star Q_2 \Rightarrow H1_{Q1}, H2_{Q1} \vDash Q_1 \wedge H1_{Q2}, H2_{Q2} \vDash Q_2 \wedge H_c, H_d \vDash P_2,\ H1_{Q1} \uplus H_c; v_1''' \Downarrow_f \wedge H2_{Q1} \uplus H_d; v_2''' \Downarrow_f$.// Unfold the definition of (f), we get: $H1_{Q1} \uplus H_c; v_1''' \Downarrow_f^{c_1'''} H1_Q; v_1'''' \wedge H2_{Q1} \uplus H_d; v_2''' \Downarrow_f^{c_2'''} H2_Q; v_2''' \wedge H1_Q, H2_Q \vDash Q$.

$$(v_1''', v_2''') \in (\!|\tau_2|\!)_{G',k'-k_1} \wedge c_1''' - c_2''' \leq \delta D' \qquad (g)$$

STS1: Use Lemma 2, we know that $H1_Q \uplus H1_{Q2}, H2_Q \uplus H2_{Q2} \vDash Q \star Q_2$.

STS2: it is proved from (g).

STS3: it is proved from (e),(k),(f),(g).

Case

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \qquad \Delta;\Phi_a;|\Gamma|_2 \vdash^{U_2}_{L_2} t_2 : A_2}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim U_1 - L_2 : U(A_1, A_2)}\ \text{R-S}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$.

TS: $(\delta\sigma_1(t_1), \delta\sigma_2(t_2)) \in (\!|\delta U(A_1, A_2)|\!)^{E,(\delta(U_1-L_2)}_{G,k,\delta(U_1-L_2)}$

Unfold the definition. Assume: $\delta\sigma_1 t_1 \Downarrow^{k_1,c} v_1(a)$, $\delta\sigma_2 t_2 \Downarrow^{k_2,c'} v_2(b)$, $k_1 < k(c)$.

TS 1: $(v_1, v_2) \in (\!|U(\delta A_1, \delta A_2)|\!)_{G,k-k_1}$. TS 2: $c - c' \leq \delta(U_1 - L_2)$

We prove TS 1 first. Unfold its definition, STS: $\forall k'. v_1 \in (\!|\delta A_1|\!)_{G|_1,k'}$ and $v_2 \in (\!|\delta A_2|\!)_{G|_2,k'}$

Pick k'. By IH 2 on the first premise using $FV(t_1) \subseteq dom(\delta|\Gamma|_1)$ using Lemma 5. $\vDash \delta\Phi$, $\forall k. \sigma_1 \in (\!|\delta|\Gamma|_1|\!)_{G|_1,k}$.

We get: $\forall k. \delta\sigma_1 t_1 \in [\![\delta A_1]\!]^{E,(\delta L_1,\delta U_1)}_{G|_1,k}$(1). Unfold the definition of (1), choose $k = k_1 + k'$. we know $v_1 \in [\![\delta A_1]\!]_{G|_1,k'} \wedge \delta L_1 \leq c \leq \delta U_1(d)$.

By IH 2 on the second premise using $\forall k. \sigma_2 \in (\!|\delta|\Gamma|_2|\!)_{G|_2,k}$.

We get: $\forall k. \delta\sigma_2 t_2 \in [\![\delta A_2]\!]^{E,(\delta L_2,\delta U_2)}_{G|_2,k}$ (2). Unfold the definition of (1), choose $k = k_2 + k'$. we know $v_2 \in [\![\delta A_2]\!]_{G|_2,k'} \wedge \delta L_2 \leq c' \leq \delta U_2(e)$.

(d),(e) finished the proof of STS1. (d),(e) finished the proof of STS2.

Case

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : U(A_1, A_2')}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_2' \lesssim U_1 + U + D_2 + c_{app} : U(A_2, A_2')}\ \text{R-APP-T}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(t_1\ t_2), \delta\sigma_2(t_2')) \in (\!|\delta U(A_2, A_2')|\!)^{E,\delta(U_1+U+D_2+c_{app})}_{G,k}$.

Following the definition of $(\!|\delta U(A_2, A_2')|\!)^{E,\delta(U_1+U+D_2+c_{app})}_{G,k}$, assume that:

$$\frac{\delta\sigma_1 t_1 \Downarrow^{c_1,k_1} \mathsf{fix} f\, x.t' \ (\star) \qquad \delta\sigma_1 t_2 \Downarrow^{c_2,k_2} v_2 \ (\diamond) \qquad t'[\mathsf{fix} f\, x.t'/f][v_2/x] \Downarrow^{c_3,k_3} v_r \ (\spadesuit)}{(\delta\sigma_1 t_1)\,(\delta\sigma_1 t_2) \Downarrow^{c_1+c_2+c_3+c_{\text{app}},k_1+k_2+k_3+1} v_r}\ \text{E-F}$$

and $\delta\sigma_2(t_2') \Downarrow^{c',k'} v'$ $(\diamond\diamond)$ and $k_1 + k_2 + k_3 + 1 < k$.

TS1: $c_1 + c_2 + c_3 + c_{\text{app}} - c' \leq \delta(U_1 + U + D_2 + c_{app})$.

TS2: $(v_r, v') \in (\!|\delta U(A_2, A_2')|\!)_{G,k-(k_1+k_2+k_3+1)}$.

Frist show the second one. By unrolling the definition of $(\!|\delta U(A_2, A_2')|\!)_{G,k-(k_1+k_2+k_3+1)}$,

STS: $\forall j. v_r \in [\![\delta A_2]\!]_{|G|_1,j} \wedge v' \in [\![\delta A_2']\!]_{|G|_2,j}$.

Pick j. By IH 2 on the first premise using $FV(t_1) \subseteq dom(\delta|\Gamma|_1)$ using Lemma 5. $\vDash \delta\Phi$, $\forall k.\sigma_1 \in (\!|\delta|\Gamma|_1|\!)_{G|_1,k}$. we get $\forall k.\delta\sigma_1 t_1 \in [\![\delta(A_1 \xrightarrow{\text{exec}(L,U)} A_2)]\!]^{E,(\delta L_1,\delta U_1)}_{G|_1,k}$ (1).

Instantiating (1) with $X = j + k_1 + k_3 + 1$, we get $\delta\sigma_1 t_1 \in [\![\delta(A_1 \xrightarrow{\text{exec}(L,U)} A_2)]\!]^{E,(\delta L_1,\delta U_1)}_{G|_1,X}$ (2).

Following the definition, we get:

$\delta\sigma_1 t_1 \Downarrow^{c_1,k_1} \mathsf{fix} f\, x.t'$ and $\mathsf{fix} f\, x.t' \in [\![\delta(A_1 \xrightarrow{\text{exec}(L,U)} A_2)]\!]_{G|_1,X-k_1}$ (a) and $\delta L_1 < c_1 < \delta U_1$ (e).

By IH on the second premise, we get: $(\delta\sigma_1 t_2, \delta\sigma_2 t_2') \in (\!|\delta U(A_1, A_2')|\!)^{E,\delta(D_2)}_{G,k}$ Unfold the defintion, by $(\diamond)$ and $(\diamond\diamond)$, we get : $c_2 - c' \leq \delta D_2$ (f) and $(v_2, v') \in (\!|\delta U(A_1, A_2')|\!)_{G,k-k_2}$ (b).

From (b), we know : $\forall k. v_2 \in [\![\delta A_1]\!]_{G|_1,k} \wedge v' \in [\![\delta A_2]\!]_{G|_2,k}$. Instantiate k with $X - k_1 - 1$, we get: $v_2 \in [\![\delta A_1]\!]_{G|_1,X-k_1-1}$ (3), $v' \in [\![\delta A_2]\!]_{G|_2,X-k_1-1}$ (4).

Unfold (a), we get: $t'[\mathsf{fix} f\, x.t'/f][v_2/x] \in [\![\delta A_2]\!]^{E,\delta(L,U)}_{G|_1,X-k_1-1}$ (c).

Unfold (c) using $(\spadesuit)$, we get : $v_r \in [\![\delta A_2]\!]_{G|_1,X-k_1-k_3-1}$ (h), $\delta L \leq c_3 \leq \delta U$ (g).

STS1 is proved by $(e), (f), (g)$. STS2 is proved by (h) and (3) using Lemma 1.

Case

$$\frac{\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t_1' : A_1' \xrightarrow{\text{exec}(L,U)} A_2' \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : U(A_2, A_1')}{\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_1'\ t_2' \lesssim D_2 - L_1 - L - c_{app} : U(A_2, A_2')}\ \text{R-T-APP}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(t_2), \delta\sigma_2(t_1'\ t_2')) \in (\!|\delta U(A_2, A_2')|\!)^{E,\delta(D_2-L_1-L-c_{app})}_{G,k}$.

Following the definition of $(\!|\delta U(A_2, A_2')|\!)^{E,\delta(D_2-L_1-L-c_{app})}_{G,k}$, assume that:

$$\frac{\delta\sigma_2 t_1' \Downarrow^{c_1',k_1'} \text{fix}\, f\, x.t' \ (\star) \qquad \delta\sigma_2 t_2' \Downarrow^{c_2',k_2'} v_2' \ (\diamond) \qquad t'[\text{fix}\, f\, x.t'/f][v_2'/x] \Downarrow^{c_3',k_3'} v_r' \ (\spadesuit)}{(\delta\sigma_2 t_1')\,(\delta\sigma_2 t_2') \Downarrow^{c_1'+c_2'+c_3'+c_{\text{app}},k_1'+k_2'+k_3'+1} v_r'}\ \text{E-F}$$

and $\delta\sigma_1 t_2 \Downarrow^{c_1,k_1} v_r$ $(\diamond\diamond)$ and $k_1 < k$.

TS1: $c_1 - (c_1' + c_2' + c_3' + c_{\text{app}}) \leq \delta(D_2 - U_1 - U - c_{app})$.

TS2: $(v_r, v_r') \in (\!|\delta U(A_2, A_2')|\!)_{G,k-k_1}$.

Frist show the second statement. By unrolling the definition of $(\!|\delta U(A_2, A_2')|\!)_{G,k-k_1}$,

STS: $\forall j.v_r \in [\![\delta A_2]\!]_{|G|_1,j} \wedge v_r' \in [\![\delta A_2']\!]_{|G|_2,j}$.

Pick j. By IH 2 on the first premise using $FV(t_1') \subseteq dom(\delta|\Gamma|_2)$ using Lemma 5. $\vDash \delta\Phi$, $\forall k.\sigma_2 \in (\!|\delta|\Gamma|_2|\!)_{G|_2,k}$.

we get $\forall k.\delta\sigma_2 t_1' \in [\![\delta(A_1' \xrightarrow{\text{exec}(L,U)} A_2')]\!]^{E,(\delta L_1,\delta U_1)}_{G|_2,k}$ (1). Instantiating (1) with $X = j + k_1' + k_3' + 1$, we get $\delta\sigma_2 t_1' \in [\![\delta(A_1' \xrightarrow{\text{exec}(L,U)} A_2')]\!]^{E,(\delta L_1,\delta U_1)}_{G|_2,X}$ (2).

Following the definition, we get:

$\delta\sigma_2 t_1' \Downarrow^{c_1',k_1'} \text{fix}\, f\, x.t'$ and $\text{fix}\, f\, x.t' \in [\![\delta(A_1' \xrightarrow{\text{exec}(L,U)} A_2')]\!]_{G|_2,X-k_1'}$ (a) and $\delta L_1 < c_1' < \delta U_1$ (e).

By IH on the second premise, we get: $(\delta\sigma_1 t_2, \delta\sigma_2 t_2') \in (\!|\delta U(A_2, A_1')|\!)^{E,\delta(D_2)}_{G,k}$ Unfold the defintion, by $(\diamond)$ and $(\diamond\diamond)$, we get :

$c_1 - c_2' \le \delta D_2$ (f) and $(v_r, v_2') \in (\!|\delta U(A_2, A_1')|\!)_{G,k-k_1}$ (b). From (b), we know : $\forall k. v_r \in [\![\delta A_2]\!]_{G|_1,k} \wedge v_2' \in [\![\delta A_1']\!]_{G|_2,k}$.

Instantiate k with $X - k_1' - 1$, we get: $v_r \in [\![\delta A_2]\!]_{G|_1,X-k_1'-1}$ (3), $v_2' \in [\![\delta A_1']\!]_{G|_2,X-k_1'-1}$ (4).

Unfold (a), we get: $t'[\mathsf{fix}\, f\, x.t'/f][v_2'/x] \in [\![\delta A_2']\!]^{E,\delta(L,U)}_{G|_2,X-k_1'-1}$ (c).

Unfold (c) using $(\spadesuit)$, we get : $v_r' \in [\![\delta A_2']\!]_{G|_2,X-k_1'-k_3'-1}$ (h), $\delta L \le c_3' \le \delta U$ (g).

STS1 is proved by $(e), (f), (g)$. STS2 is proved by (h) and (4) using Lemma 1.

Case

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t_2 \ominus t_2' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \ominus t_2' \lesssim U_1 + D_2 + c_{lt} : \tau}\ \text{R-LT-T}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2), \delta\sigma_2(t_2')) \in (\!|\delta\tau|\!)^{E,\delta(D_2+U_1+c_{lt})}_{G,k}$.

Following the definition of $(\!|\delta\tau|\!)^{E,\delta(D_2+U_1+c_{lt})}_{G,k}$, assume that:

$$\frac{\delta\sigma_1 t_1 \Downarrow^{c_1,k_1} v_1\ (\star) \qquad \delta\sigma_1 t_2[v_1/x] \Downarrow^{c_r,k_r} v_r\ (\spadesuit)}{\mathsf{let}\ x = \delta\sigma_1 t_1\ \mathsf{in}\ \delta\sigma_1 t_2 \Downarrow^{c_1+c_r+c_{\mathrm{lt}},k_1+k_r+1} v_r}\ \text{E-LET}$$

and $\delta\sigma_2 t_2' \Downarrow^{c',k'} v'$ $(\diamond\diamond)$. STS1: $(v_r, v') \in (\!|\delta\tau|\!)_{G,k-(k_1+k_r+1)}$.

STS2: $c_1 + c_r + c_{lt} - c' \le \delta(D_2 + U_1 + c_{lt}$.

To prove : $\forall k. v_1 \in [\![\delta A_1]\!]_{|G|_1,k})$.

Proof. Pick k. IH2 on the first premise.

1. $FV(t_1) \subseteq |\delta\Gamma|_1$ using Lemma 5.

2. $\sigma_1 \in (\!||\delta\Gamma|_1|\!)_{|G|_1,k+\delta U_1+1}$ by Lemma 6 using assumptions.

We get: $\sigma_1 t_1 \in [\![\delta A_1]\!]^{E,(\delta L_1,\delta U_1)}_{|G|_1,k+\delta U_1+1}$. Unfold the defintion, we know: $\delta L_1 \le c_1 \le \delta U_1$ (a) and

$v_1 \in [\![\delta A_1]\!]_{|G|_1,k+\delta u_1+1-c_1}$.

So, we know: $v_1 \in [\![\delta A_1]\!]_{|G|_1,k}$ by Lemma 1. ■

By IH on the second premise using

1. $(\sigma_1[v_1/x], \sigma_2[v_1/x]) \in (\!|\delta\Gamma, x : U(\delta A_1, \delta A_1)|\!)_{G,k}$ using

   1.1. $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

   1.2. $(v_1, v_1) \in (\!|U(\delta A_1, \delta A_1)|\!)_{G,k}$

we get: $(\delta\sigma_1[v_1/x]\, t_2, \delta\sigma_2[v_1/x]\, t_2') \in (\!|\delta\tau|\!)^{E,\delta D_2}_{G,k}$.

Since x does not occur free in t_2', we have : $(\delta\sigma_1[v_1/x]\, t_2, \delta\sigma_2\, t_2') \in (\!|\delta\tau|\!)^{E,\delta D_2}_{G,k}$. Unfold the defintion, we have: $c_r - c_1' \leq \delta D_2$ (b), $(v_r, v') \in (\!|\delta\tau|\!)_{G,k-k_r}$ (c).

STS1 is proved by using Lemma 1 on (c) because $k-(k_1+k_r+1) < k-k_r$. STS2 is proved by (a) and (b).

Case

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t : A_1 + A_2 \qquad \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1,A_1) \vdash t_1 \ominus t' \lesssim D_2 : \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma, y : U(A_2,A_2) \vdash t_2 \ominus t' \lesssim D_2 : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \text{ case } (t, x.t_1, y.t_2) \ominus t' \lesssim U_1 + D_2 + c_{case} : \tau}\ \text{R-CASE-T}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(\text{ case } (t, x.t_1, y.t_2)), \delta\sigma_2(t')) \in (\!|\delta\tau|\!)^{E,\delta(D_2+U_1+c_{case})}_{G,k}$.

Following the definition of $(\!|\delta\tau|\!)^{E,\delta(D_2+U_1+c_{case})}_{G,k}$, assume that:

$\delta\sigma_1$ case $(t, x.t_1, y.t_2) \Downarrow^{C,K} v_r$ and $\delta\sigma_2\, t' \Downarrow^{c',k'} v'$ $(\diamond\diamond)$ and $C < k$.

STS1: $(v_r, v') \in (\!|\delta\tau|\!)_{G,k-K}$. STS2: $C - c' \leq \delta(D_2 + U_1 + c_{case})$.

To prove : $\forall k. v_1 \in [\![\delta A_1]\!]_{|G|_1,k}$. Depending on what $\delta\sigma_1\, t$ evaluates to, there are two cases.

Subcase1:

$$\frac{\delta\sigma_1\, t \Downarrow^{c_1,k_1} \mathsf{inl}\, v_1 \;\; (\star) \qquad \delta\sigma_1\, t_1[v_1/x] \Downarrow^{c_r,k_r} v_r \;\; (\spadesuit)}{\delta\sigma_1\ \mathsf{case}\ (t, x.t_1, y.t_2) \Downarrow^{c_1+c_r+c_{\mathsf{case}},k_1+k_r+1} v_r}\ \text{CASE-INL}$$

To prove: $\forall k.(\mathsf{inl}\, v_1) \in (\!|\delta A_1 + \delta A_2|\!)_{G,k}$.

Proof. Pick k. IH2 on the first premise.

1. $FV(t) \subseteq |\delta\Gamma|_1$ using Lemma 5.
2. $\sigma_1 \in (\!|\,|\delta\Gamma|_1|\!)_{|G|_1,k+\delta U_1+1}$ by Lemma 6 using assumptions.

We get: $\sigma_1\, t \in [\![\delta A_1 + \delta A_2]\!]^{E,(\delta L_1,\delta U_1)}_{|G|_1,k+\delta U_1+1}$. Unfold the defintion, we know:
$\delta\sigma_1\, t \Downarrow^{c_1,k_1} \mathsf{inl}\, v_1$ and $\delta L_1 \le c_1 \le \delta U_1$ (a) and $\mathsf{inl}\, v_1 \in [\![\delta A_1 + \delta A_2]\!]_{|G|_1,k+\delta u_1+1-c_1}$.
So, we know: $\mathsf{inl}\, v_1 \in [\![\delta A_1 + \delta A_2]\!]_{|G|_1,k}$ by Lemma 1. ■

By IH on the second premise using

1. $(\sigma_1[v_1/x], \sigma_2[v_1/x]) \in (\!|\delta\Gamma, x : U(\delta A_1, \delta A_1)|\!)_{G,k}$ using
   1.1. $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$
   1.2. $(v_1, v_1) \in (\!|U(\delta A_1, \delta A_1)|\!)_{G,k}$

we get: $(\delta\sigma_1[v_1/x]\, t_1, \delta\sigma_2[v_1/x]\, t') \in (\!|\delta\tau|\!)^{E,\delta D_2}_{G,k}$.
Since x does not occur free in t', we have : $(\delta\sigma_1[v_1/x]\, t_1, \delta\sigma_2\, t') \in (\!|\delta\tau|\!)^{E,\delta D_2}_{G,k}$.
Unfold its defintion, we get: $c_r - c_1' \le \delta D_2$ (b). $(v_r, v') \in (\!|\delta\tau|\!)_{G,k-k_r}$ (c).
STS1 is proved by using Lemma 1 on (c) because $k - (k_1 + k_r + 1) < k - k_r$. STS2 is proved by (a) and (b).

Subcase 2:

$$\frac{\delta\sigma_1\, t \Downarrow^{c_1,k_1} \mathsf{inr}\, v_1 \;\; (\star) \qquad \delta\sigma_1\, t_2[v_1/x] \Downarrow^{c_r,k_r} v_r \;\; (\spadesuit)}{\delta\sigma_1\ \mathsf{case}\ (t, x.t_1, y.t_2) \Downarrow^{c_1+c_r+c_{\mathsf{case}},k_1+k_r+1} v_r}\ \text{CASE-INR}$$

Prove similarily : $\forall k.(\mathsf{inr}\, v_1) \in (\!|\delta A_1 + \delta A_2|\!)_{G,k}$.

IH on the third premise using

1. $(\sigma_1[v_1/x], \sigma_2[v_1/x]) \in (\!|\delta\Gamma, x : U(\delta A_1, \delta A_1)|\!)_{G,k}$ using

   1.1. $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

   1.2. $(v_1, v_1) \in (\!|U(\delta A_1, \delta A_1)|\!)_{G,k}$

we get: $(\delta\sigma_1[v_1/x]t_2, \delta\sigma_2[v_1/x]t') \in (\!|\delta\tau|\!)^{E,\delta D_2}_{G,k}$. Since x does not occur free in t', we have : $(\delta\sigma_1[v_1/x]t_2, \delta\sigma_2 t') \in (\!|\delta\tau|\!)^{E,\delta D_2}_{G,k}$.

Unfold its defintion, we get: $c_r - c_1' \leq \delta D_2$ (b) ,$(v_r, v') \in (\!|\delta\tau|\!)_{G,k-k_r}$ (c).

STS1 is proved by using Lemma 1 on (c) because $k - (k_1 + k_r + 1) < k - k_r$. STS2 is proved by (a) and (b).

Case

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : \{P_1\}\ \overset{\mathrm{exec}(L,U)}{\exists\vec{\gamma}_1 : A_1}\ \{Q_1\} \\ \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_2}_{L_2} t_2' : \{P_2\}\ \overset{\mathrm{exec}(L',U')}{\exists\vec{\gamma}_1 : A_1'}\ \{Q_2\} \\ \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1, A_1) \vdash t_2 \ominus t_2' \lesssim D_2 : \{P \sqcup P_1\}\ \overset{\mathrm{diff}(D')}{\exists\vec{\gamma}_1.\tau}\ \{Q\} \\ dom(P) = dom(P_1)\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus t_2' \lesssim -L_2 : \overset{\mathrm{diff}(U_1+U+(D_2+U_2)+D'+c_{let})}{\{P\}\ \exists\vec{\gamma}_1.\tau\ \{Q\}}}\ \text{R-LET-T}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(\mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2), \delta\sigma_2(t_2')) \in (\!|\delta(\{P\}\ \overset{\mathrm{diff}(-L_2)}{\exists\vec{\gamma}_1.\tau}\ \{Q\})|\!)^{E,\delta(U_1+U+(D_2+U_2)+D'+c_{let})}_{G,k}$.

Assume $\mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \Downarrow^{0,0} \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2$ and $\delta\sigma_2 t_2' \Downarrow^{c_1',k_1'} v_1'$.

Unfold the definition, STS1: $0 - c_1' \leq -(\delta L_2)$. By IH2 on the second premise, we get: $\delta L_2 \leq c_1' \leq \delta U_2$, which proves STS1.

STS2: $(\delta\sigma_1(\mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2), v_1') \in (\!|\delta(\overset{\mathrm{diff}(U_1+U+(D_2+U_2)+D'+c_{let})}{\{P\}\ \exists\vec{\gamma}_1.\tau\ \{Q\}})|\!)_{G,k}$.

Unfold its defintion, assume : $G \supseteq G, k' < k, k_2 < k'. H_1, H_1' \vDash_{G,k'} P$

and $\mathsf{let}\{x\} = \delta\sigma_1 t_1 \text{ in } \delta\sigma_1 t_2; H_1 \Downarrow_f^{c_2,k_2} v_{let}; H_2$ (a) and $v_1'; H_1' \Downarrow_f^{c_2',k_2'} v_2'; H_2'$ (b).

STS1: $(H_2, H_2') \vDash_{G,k'-k_2} Q$. STS2: $(v_{let}, v_2') \in (\!|\delta\tau|\!)_{G,k'-k_2}$. STS3: $c_2 - c_2' \leq \delta((D_2 + U_2) + D' + U + U_1 + c_{let})$.

By IH2 on the first premise, we get: $\delta\sigma_1 t_1 \in (\!|\delta(\{P_1\} \overset{\text{exec}(L,U)}{\exists\vec{\gamma}_1 : A_1} \{Q_1\})|\!)_{|G|_1,k}^{E,\delta(L_1,U_1)}$ (c).

Unfold (c), we assume $\delta\sigma_1 t_1 \Downarrow^{c_a,k_a} v$, we get : $v \in (\!|\delta(\{P_1\} \overset{\text{exec}(L,U)}{\exists\vec{\gamma}_1 : A_1} \{Q_1\})|\!)_{|G|_1,k-k_a}$ (d) and $\delta L_1 \leq c_a \leq \delta U_1$ (e). Unfold (d), pick $k_1 \leq k - k_a$. assume $H_1 \vDash_{|G|_1,k_1} P_1$,

we know: $v; H_1 \Downarrow_f^{c_3,k_3} v_2; H_3$ and $H_3 \vDash_{|G|_1,k_1-k_3} Q_1$ and $v_a \in (\!|A|\!)_{|G|_1,k_1-k_3} \wedge \delta L \leq c_3 \leq \delta U$ (f)

and $\exists n. P_1 = \{\gamma_1 \to T_1, \ldots \gamma_n \to T_n\} \wedge \forall i \in [1,n].|G|_1(\gamma_i) = (l_i, A, m) \implies \forall j. H_1[l_i][j] \neq H_3[l_i][j] \implies j \in T_i$ (z).

From (f), we know : $(\sigma_1[v_a/x], \sigma_2[v_a/x]) \in (\!|\delta(\Gamma, x : U(A,A))|\!)_{G,k}$ (h).

IH on the third premise instantiated with (h), we get:

$$(\delta\sigma_1[v_a/x](t_2), \delta\sigma_2[v_a/x](t_2')) \in (\!|\delta(\{P \sqcup P_1\} \overset{\text{diff}(D')}{\exists\vec{\gamma}_1.\tau} \{Q\})|\!)_{G,k}^{E,\delta(D_2)} \ (i)$$

Unfold (i), assume : $\delta\sigma_1[v_a/x]t_2 \Downarrow^{c_4,k_4} v_4$ and $\delta\sigma_2[v_a/x]t_2' \Downarrow^{c_1',k_1'} v_1'$.

we get: $(v_4, v_1') \in (\!|\delta(\{P \sqcup P_1\} \overset{\text{diff}(D')}{\exists\vec{\gamma}_1.\tau} \{Q\})|\!)_{G,k}$ (j) and $c_4 - c_1' \leq \delta D_2$ (k).

Unfold (j), we assume :Pick $k_6 = k' - ka - k_3 - k_4 - 1 \leq k - k_4$.

We first show the conclusion: $(H_3, H_1') \vDash_{G,k_6} P \sqcup P_1$ $(\clubsuit)$.

Proof. We have the assuption (z) and the assumption $H_1, H_1' \vDash_{G,k} P$.

Unfold $(\clubsuit)$, we need to prove :

$\forall \gamma \in dom(P) \cup dom(P_1). \exists l_1, l_2, \tau, n. G(\gamma) = (l_1, l_2, \tau, n) \wedge \forall i < n. H_3[\gamma][i] \neq H_1'[\gamma][i] \implies i \in (P \sqcup P_1)[\gamma]$. $(\star)$

Unfold the definition of $H_1, H_1' \vDash_{G,k} P$, we know:

$\forall \gamma \in dom(P). \exists l_1, l_2, \tau, n. G(\gamma) = (l_1, l_2, \tau, n) \wedge \forall i < n. H_1[\gamma][i] \neq H_1'[\gamma][i] \implies i \in (P)[\gamma]$. $(\heartsuit)$

From the premise $dom(P) = dom(P_1)$, there is only one case:

1. $\gamma \in dom(P), \gamma \in dom(P_1)$

   From (♡), we know

   $\forall \gamma \in dom(P) \cap dom(P_1).\exists l_1, l_2, \tau, n.G(\gamma) = (l_1, l_2, \tau, n)$

   $\wedge \forall i < n.H_1[\gamma][i] \neq H_1'[\gamma][i] \implies i \in (P)[\gamma]$,

   which means H_1 and H_1' differ at most at indices in $P[\gamma]$, also means $\forall i < n.i \notin P[\gamma] \implies H_1[\gamma][i] = H_1'[\gamma][i]$.

   From (z), we also know that $\forall \gamma \in dom(P_1), \forall i < n.H_1[\gamma][i] \neq H_3[\gamma][i] \implies i \in P_1[\gamma]$, which means H_1 and H_3 differ at most at all the indices in $P_1[\gamma]$.

   So we know H_3 and H_1' differ at most at indices appear in P or P_1, which is $\forall \gamma \in (dom(P) \cap dom(P_1)).\exists l_1, l_2, \tau, n.G(\gamma) = (l_1, l_2, \tau, n) \wedge \forall i < n.H_1[\gamma][i] \neq H_1'[\gamma][i] \implies i \in P[\gamma] \vee i \in P_1[\gamma]$.

   This subcase is proved because $i \in P[\gamma] \vee i \in P_1[\gamma] \implies i \in P[\gamma] \cup P_1[\gamma] \implies (P \sqcup P_1)[\gamma]$.

■

Also assume $v_4; H_3 \Downarrow_f^{c_5,k_5} v_5; H_2$ and $v_1'; H_1' \Downarrow_f^{c_2',k_2'} v_2'; H_2'$.

We get: $H_2, H_2' \vDash_{G,k_6-k_5} Q\ (l)$ and $(v_5, v_2') \in (\!|\delta\tau|\!)_{G,k_6-k_5}\ (m)$ and $c_5 - c_2' \leq \delta D'\ (n)$.

From the forcing evaluation rule

$$\frac{\delta t_1 \Downarrow^{c_a,k_a} v \qquad v; H_1 \Downarrow_f^{c_3,k_3} v_a; H_3 \qquad t_2[v_a/x] \Downarrow^{c_4,k_4} v_4 \qquad v_4; H_3 \Downarrow_f^{c_5,k_5} v_5; H_2}{\mathsf{let}\{x\} = t_1 \text{ in } t_2; H_1 \Downarrow_f^{c_a+c_3+c_4+c_5+c_{\text{let}},k_a+k_3+k_4+k_5+1} v_5; H_2}\ \text{F-E}$$

STS1 is proved by (l). STS2 is proved by (m) using Lemma 1.

STS3 : $c_2 - c_2' = c_a + c_3 + c_4 + c_5 + c_{\text{let}} - c_2' \leq \delta(U + U_1 + D' + (D_2 + U_2) + c_{\text{let}})$ by (e),(k),(n), (f).

Case

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_1}_{L_1} t_1' : \{P_1\}\ \overset{\mathrm{exec}(L,U)}{\exists \vec{\gamma}_1} : A_1'\ \{Q_1\} \\ \Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_2}_{L_2} t_2 : \{P_2\}\ \overset{\mathrm{exec}(L',U')}{\exists \vec{\gamma}_1} : A_1\ \{Q_2\} \\ \Sigma;\Delta;\Phi_a;\Gamma, x : U(A_1', A_1') \vdash t_2 \ominus t_2' \lesssim D_2 : \{P \sqcup P_1\}\ \overset{\mathrm{diff}(D')}{\exists \vec{\gamma}_1}.\tau'\ \{Q\} \\ dom(P) = dom(P_1)\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus \mathsf{let}\{x\} = t_1' \ \mathsf{in}\ t_2' \lesssim U_2 : \quad \overset{\mathrm{diff}(D'+(D_2-L_2)-L_1-L-c_{\mathrm{let}})}{\{P\}\ \exists \vec{\gamma}_1.\tau'\ \{Q\}}}\ \text{R-T-LET}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and $(\sigma_1, \sigma_2) \in (\!|\delta\Gamma|\!)_{G,k}$

TS: $(\delta\sigma_1(t_2), \delta\sigma_2(\mathsf{let}\{x\} = t_1' \ \mathsf{in}\ t_2')) \in (\!|\delta(\quad \overset{\mathrm{diff}(D'+(D_2-L_2)-L_1-L-c_{\mathrm{let}})}{\{P\}\ \exists \vec{\gamma}_1.\tau'\ \{Q\}} \quad)|\!)^{E,\delta(U_2)}_{G,k}$.

Assume $\delta\sigma_2(\mathsf{let}\{x\} = t_1' \ \mathsf{in}\ t_2') \Downarrow^{0,0} \mathsf{let}\{x\} = \delta\sigma_2 t_1' \ \mathsf{in}\ \delta\sigma_2 t_2'$ and $\delta\sigma_1 t_2 \Downarrow^{c_1,k_1} v_1$.

Unfold the definition, STS1: $c_1 - 0 \le \delta U_2$. By IH2 on the second premise, we get: $\delta L_2 \le c_1 \le \delta U_2$, which proves STS1.

STS2: $(v_1, \delta\sigma_2(\mathsf{let}\{x\} = t_1' \ \mathsf{in}\ t_2')) \in (\!|\delta(\quad \overset{\mathrm{diff}(D'+(D_2-L_2)-L_1-L-c_{\mathrm{let}})}{\{P\}\ \exists \vec{\gamma}_1.\tau\ \{Q\}} \quad)|\!)_{G,k}$.

Unfold its defintion, assume : $G \supseteq G, k' < k, k_2 < k'.H_1, H_1' \vDash_{G,k'} P$

and $\mathsf{let}\{x\} = \delta\sigma_2 t_1' \ \mathsf{in}\ \delta\sigma_2 t_2'; H_1' \Downarrow^{c_2',k_2'}_f v'_{let}; H_2'$ (a) and $v_1; H_1 \Downarrow^{c_2,k_2}_f v_2; H_2$ (b).

STS1: $(H_2, H_2') \vDash_{G,k'-k_2} Q$. STS2: $(v_2, v'_{let}) \in (\!|\delta\tau|\!)_{G,k'-k_2}$. STS3: $c_2 - c_2' \le \delta(D' + (D_2 - L_2) - L_1 - L - c_{\mathrm{let}})$.

By IH2 on the first premise, we get: $\delta\sigma_2 t_1' \in (\!|\delta(\{P_1\}\ \overset{\mathrm{exec}(L,U)}{\exists \vec{\gamma}_1} : A_1'\ \{Q_1\})|\!)^{E,\delta(L_1,U_1)}_{|G|_2,k}$ (c).

Unfold (c), we assume $\delta\sigma_1 t_1' \Downarrow^{c'_a,k'_a} v'$, we get : $v' \in (\!|\delta(\{P_1\}\ \overset{\mathrm{exec}(L,U)}{\exists \vec{\gamma}_1} : A_1'\ \{Q_1\})|\!)_{|G|_2,k-k'_a}$ (d) and $\delta L_1 \le c'_a \le \delta U_1$ (e).

Unfold (d), assume : pick $k_1 \le k - k'_a$. $H_1' \vDash_{|G|_2,k_1} P_1$ and $v'; H_1' \Downarrow^{c_3',k_3'}_f v'_a; H_3'$

and $H_3' \vDash_{|G|_2,k_1-k_3'} Q_1$ and $v'_a \in (\!|A_1'|\!)_{|G|_2,k_1-k_3'} \wedge \delta L \le c_3' \le \delta U$ (f).

From (f), we know : $(\sigma_1[v'_a/x], \sigma_2[v'_a/x]) \in (\!|\delta(\Gamma, x : U(A_1', A_1'))|\!)_{G,k}$ (h).

IH on the third premise instantiated with (h), we get:

$$(\delta\sigma_1[v'_a/x](t_2), \delta\sigma_2[v'_a/x](t_2')) \in (\!|\delta(\{P \cup P_1\}\ \overset{\mathrm{diff}(D')}{\exists \vec{\gamma}_1.\tau'\ \{Q\}})|\!)^{E,\delta(D_2)}_{G,k}\ (i)$$

. Unfold (i), assume : $\delta\sigma_1[v'_a/x]\,t_2 \Downarrow^{c_1,k_1} v_1$ and $\delta\sigma_2[v'_a/x]\,t'_2 \Downarrow^{c'_4,k'_4} v'_4$.
we get: $(v_1, v'_4) \in (\!|\delta(\{P \cup P_1\} \overset{\text{diff}(D')}{\exists\vec{\gamma}_1}.\tau'\ \{Q\})|\!)_{G,k}$ (j) and $c_1 - c'_4 \le \delta D_2$ (k).

Unfold (j), we assume :Pick $k_6 = k' - k_4 - 1 \le k - k'_4$..

We prove $(H_1, H'_3) \vDash_{G,k_6} P \sqcup P_1$ in a similar way in the previous case.

Also assume $v'_4; H'_3 \Downarrow_f^{c'_5,k'_5} v'_5; H'_2$ and $v_1; H_1 \Downarrow_f^{c_2,k_2} v_2; H_2$.

We get: $H_2, H'_2 \vDash_{G,k_6-k_5} Q$ (l) and $(v_2, v'_5) \in (\!|\delta\tau'|\!)_{G,k_6-k_5}$ (m) and $c_2 - c'_5 \le \delta D'$ (n).

From the forcing evaluation rule

$$\frac{\delta t'_1 \Downarrow^{c'_a,k'_a} v' \qquad v'; H'_1 \Downarrow_f^{c'_3,k'_3} v'_a; H'_3 \qquad t'_2[v'_a/x] \Downarrow^{c'_4,k'_4} v'_4 \qquad v'_4; H'_3 \Downarrow_f^{c'_5,k'_5} v'_5; H'_2}{\mathsf{let}\{x\} = t'_1 \ \mathsf{in}\ t'_2; H'_1 \Downarrow_f^{c'_a+c'_3+c'_4+c'_5+c_{\text{let}},k'_a+k'_3+k'_4+k'_5+1} v'_5; H'_2}\ \text{F-E}$$

STS1 is proved by (l).

STS2 is proved by (m) using Lemma 1.

STS3 : $c_2 - c'_2 = c_2 - c'_a - c'_3 - c'_4 - c'_5 - c_{\text{let}} \le \delta(D' + (D_2 - L_2) - L_1 - L - c_{\text{let}})$ by (e),(k),(n), (f).

Proof of statement (2) if $\Delta;\Phi_a;\Omega \vdash_L^U t : A$ and $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and there exists Ω'. s.t. $FV(t) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(\sigma) \in (\!|\delta\Omega'|\!)_{g,k}$. Then, $(\delta\sigma\, t) \in (\!|A|\!)_{g,k}^{E,(\delta L,\delta U)}$

Proof by induction on typing derivation:

Case

$$\frac{\xi;\Delta;\Phi_a;x:A,f:\overset{\text{exec}(L,U)}{A \longrightarrow} B,\Omega \vdash_L^U t : B}{\Delta;\Phi_a;\Omega \vdash_0^0 \text{fix}\, f(x).t : \overset{\text{exec}(L,U)}{A \longrightarrow} B}\ \text{U-F}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and there exists Ω'. s.t. $FV(t) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(\sigma) \in (\!|\delta\Omega'|\!)_{g,k}$

TS: $(\delta\sigma \text{fix}\, f(x).t) \in [\![\delta\ \overset{\text{exec}(L,U)}{A \longrightarrow} B]\!]_{g,k}^{E,(0,0)}$. Because $\text{fix}\, f(x).e$ is value.

STS: $(\delta\sigma \text{fix}\, f(x).t) \in [\![\delta\ \overset{\text{exec}(L,U)}{A \longrightarrow} B]\!]_{g,k}$

We prove the more general statement $\forall k' \le k.\delta\sigma(\text{fix}\, f(x).t) \in [\![\delta A \overset{\text{exec}(L,U)}{\longrightarrow} \delta B]\!]_{g,k}(1)$.

By sub-induction on k'.There are two cases:

Subcase 1: k'=0. there is no non-negative k such that k ≤ 0, the goal is vacuously true.

Subcase 2: $k' = k'' + 1 \le k$

By sub-IH, we know: $\delta\sigma(\text{fix}\, f(x).t) \in [\![\delta A \overset{\text{exec}(L,U)}{\longrightarrow} \delta B]\!]_{g,k''}(2)$.

STS: $(\text{fix}\, f(x).\delta\sigma e) \in [\![\delta A \overset{\text{exec}(L,U)}{\longrightarrow} \delta B]\!]_{g,k''+1}$. Pick j < k"+1 and assume that $v \in [\![\delta A]\!]_{g,j}$.

STS: $\sigma[v/x][\text{fix}\, f(x).e/f]) \in [\![\delta B]\!]_{g,j}^{E,(\delta L,\delta U)}$

This follows by IH on the premise instantiated with

1. FV(e) $\subseteq dom(x:A,f:\overset{\text{exec}(L,U)}{A \longrightarrow} B,\Omega')$ from Lemma (well-formness) and $(x:A,f:\overset{\text{exec}(L,U)}{A \longrightarrow} B,\Omega') \subseteq (x:A,f:\overset{\text{exec}(L,U)}{A \longrightarrow} B,\Omega)$
2. $(\sigma[v/x][\text{fix}\, f(x).t/f]) \in [\![\delta(x:A,f:\overset{\text{exec}(L,U)}{A \longrightarrow} B,\Omega')]\!]_{g,j}$ which holds because
   - 2.1. $\sigma \in [\![\delta\Omega']\!]_{g,j}$ by Lemma downward closure and $j < k'' + 1 \le k$
   - 2.2. $v \in [\![A]\!]_{g,j}$ from assumption
   - 2.3. $(\delta\sigma \text{fix}\, f(x).e) \in [\![\delta A \overset{\text{exec}(L,U)}{\longrightarrow} \delta B]\!]_{g,j}$ from Lemma 1 on (2).

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \text{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : A \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^0 \text{alloc}\, t_1\, t_2 : \{P\}\ \exists\gamma : \text{Array}_\gamma[I]\ A\ ^{\text{exec}(L_1+L_2+L_a,U_1+U_2+U_a)}\ \{P \star \gamma \to \mathbb{N}\}}\ \text{U-ALLOC}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and there exists Ω'. s.t. $FV(t) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(\sigma) \in (\!|\delta\Omega'|\!)_{g,k}$

TS: $(\delta\sigma\ \text{alloc}\, t_1\, t_2) \in [\![\{P\}\ \exists\gamma : \text{Array}_\gamma[I]\ A\ ^{\text{exec}(L_1+L_2+L_a,U_1+U_2+U_a)}\ \{P\gamma \to \mathbb{N}\}]\!]^{E,(0,0)}_{g,k}$

Because $\text{alloc}\, t_1\, t_2$ is value. STS: $(\delta\sigma\ \text{alloc}\, t_1\, t_2) \in [\![\{P\}\ \exists\gamma : \text{Array}_\gamma[I]\ A\ ^{\text{exec}(L_1+L_2+L_a,U_1+U_2+U_a)}\ \{P\gamma \to \mathbb{N}\}]\!]_{g,k}$

Unfold the definition of $[\![\{P\}\ \exists\gamma : \text{Array}_\gamma[I]\ A\ ^{\text{exec}(L_1+L_2+L_a,U_1+U_2+U_a)}\ \{P\gamma \to \mathbb{N}\}]\!]_{g,k}$. Pick $g' \supseteq g$, $k' \leq k$, $k'' < k'$ and H. Assume $H \vDash_{g',k'} P$ and $H; \text{alloc}\, t_1\, t_2 \Downarrow^{k''}$.

From $H; \text{alloc}\, t_1\, t_2 \downarrow^{k''}$, because t_1, t_2 are sub terms of $\delta\sigma\ \text{alloc}\, t_1\, t_2$,we get :

$\delta\sigma\, t_1 \Downarrow_f^{k_1} \wedge \delta\sigma\, t_2 \Downarrow_f^{k_2}$ (1). From (1) , we get:

$\exists v_1, v_2, k_1, k_2. \delta\sigma\, t_1 \Downarrow^{k_1,c_1} v_1 \wedge \delta\sigma\, t_2 \Downarrow^{k_2,c_1'} v_2 \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_1' + c_{alloc}(a)$.

From (a) and evaluation rules we get: $\exists l.H, (\text{alloc}\, t_1\, t_2)\delta\sigma \Downarrow_f^{k'',c} l; H[l \to [v_2, \ldots v_2](b)$.

By IH on the first premise instantiated with $\sigma \in [\![\delta\Omega']\!]_{g,k''}$using lemma 1 and $FV(t_1) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$, we get $(\delta\sigma\, t_1) \in [\![\text{int}[I]]\!]^{E,(\delta L_1,\delta U_1)}_{g',k''}(*)$.

Unfold the definition of $[\![\text{int}[I]]\!]^{E,(\delta L_1,\delta U_1)}_{g',k''}$ and using (a) and $k_1 \leq k''$, we know:

$$(v_1) \in [\![\text{int}[I]]\!]_{G',k''-k_1} \Rightarrow v_1 = I \wedge \delta L_1 \leq c_1 \leq \delta U_1 \qquad \text{IH1}$$

By IH on the second premise instantiated with $(\sigma) \in [\![\delta\Gamma]\!]_{g',k'-k_1}$using lemma 1 and $FV(t_2) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$, we get: $(\delta\sigma\, t_2) \in [\![\delta A]\!]^{E,(\delta L_2,\delta U_2)}_{g',k'-k_1}(**)$.

Unfold the definition of (**) and using (a), (b) and $k_2 \leq k' - k_1$, we know

$$(v_2) \in [\![\delta A]\!]_{g',k'-k_1-k_2} = [\![\delta A]\!]_{g',k'-k''+1} \wedge \delta L_2 \leq c_1' \leq \delta U_2 \qquad \text{IH2}$$

Let us assume: $g'' = g'[r \to (l, A, I)](c)$, $H' = H, \left[l \to [v_2, v_2, \ldots, v_2]\right](d)$. We want to show 4 cases.

TS1 $(l) \in [\![\mathsf{Array}_\gamma[I]\ A]\!]_{g'', k'-k''}$

it is proved by unfolding the definition and using the assumption (c).

TS2 $(H') \vDash_{g'', k'-k''} p \star \gamma \to \mathbb{N}$

we know that $g''(\gamma) = (l, A, I)$, $H' = H \uplus H_l \wedge H_l = l \to [v_2, v_2, \ldots, v_2]$

STS1 $\forall i \le I, (H_l(l)[i] \in [\![A]\!]_{g'', k'-k''-1}$. It is proved by using IH2 and Lemma 1.

TS3 $\delta(L_1 + L_2 + L_a) \le c \le \delta(U_1 + U_2 + U_a)$

It is proved by IH1 and IH2 and $\delta L_a \le c_{alloc} \le \delta U_a$.

TS4 $\exists n.P = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \wedge \forall i \in [1, n].g(\gamma_i) = (l_i, A, m)$
$\Rightarrow \forall j.(H[l_i][j]) \neq H'[l_i][j] \Rightarrow j \in T_i \Big\}$

Because new allocated γ is not in P, so for all the γ_i, it is not changed.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : \mathsf{Array}_\gamma[I]\ A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : \mathrm{int}[I'] \qquad \vDash I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{read}\ t_1\ t_2 : \overset{\mathrm{exec}(L_1+L_2+L_r, U_1+U_2+U_r)}{\{P\}\ \exists\_ : A\ \{P\}}}\ \text{U-R}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and there exists Ω'. s.t. $FV(t) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(\sigma) \in (\!|\delta\Omega'|\!)_{g,k}$

TS: $(\delta\sigma\ \mathsf{read}\ t_1\ t_2) \in [\![\delta \overset{\mathrm{exec}(L_1+L_2+L_r, U_1+U_2+U_r)}{\{P\}\ \exists\_ : A\ \{P\}}]\!]^{E,(0,0)}_{g,k}$

Because $\mathsf{read}\ t_1\ t_2$ is value. STS: $(\delta\sigma\ \mathsf{read}\ t_1\ t_2) \in [\![\delta \overset{\mathrm{exec}(L_1+L_2+L_r, U_1+U_2+U_r)}{\{P\}\ \exists\_ : A\ \{P\}}]\!]_{g,k}$

Unfold the definition of $[\![\delta \overset{\mathrm{exec}(L_1+L_2+L_r, U_1+U_2+U_r)}{\{P\}\ \exists\_ : A\ \{P\}}]\!]_{g,k}$.

Pick $g' \supseteq g$, $k' \le k$, $k'' < k'$ and H. Assume $H \vDash_{g', k'} \gamma$ and $H; \mathsf{read}\ t_1\ t_2 \Downarrow^{k''}$

From $H; \mathsf{read}\ t_1\ t_2 \Downarrow^{k''}$, because t_1, t_2 are sub terms of $\delta\sigma\ \mathsf{read}\ t_1\ t_2$,we get:

$\delta\sigma\, t_1 \Downarrow_f^{k_1} \wedge \delta\sigma\, t_2 \Downarrow_f^{k_2}$ (1).

From (1) , we get: $\exists l, n, v, k_1, k_2. \delta\sigma\, t_1 \Downarrow^{k_1, c_1} l \wedge \delta\sigma\, t_2 \Downarrow^{k_2, c_1'} n \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_1' + c_{read} \wedge H(l)[n] = v(a)$. From (a) and evaluation rules we get: $H; \delta\sigma(\text{read } t_1\ t_2) \Downarrow_f^{k'', c} H; v(b)$.

By IH on the first premise instantiated with $(\sigma) \in [\![\delta\Omega']\!]_{g', k'}$ by lemma 1 and $FV(t_1) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$. we get: $(\delta\sigma\, t_1) \in [\![\mathsf{Array}_\gamma[I]\ \delta A]\!]_{g', k'}^{E, \delta(L_1, U_1)}$(6).

Unfold (6), since $k_1 \leq k'$ and $\delta\sigma_1\, t_1 \Downarrow^{k_1, c_1} l$, we know $l \in [\![\mathsf{Array}_\gamma[I]\ \delta A]\!]_{g', k'-k_1} \wedge \delta L_1 \leq c_1 \leq \delta U_1(e)$. From (e), we know :$g'(\gamma) = (l, \delta A, I)$(7).

By IH on the second premise instantiated with $(\sigma) \in [\![\delta\Omega']\!]_{g', k'}$ by lemma 1 and $FV(t_2) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$. we get: $(\delta\sigma_1\, t_2') \in [\![\text{int}[I']]\!]_{g', k'}^{E, \delta(L_2, U_2)}$(8).

Unfold (8), since $k_2 \leq k'$,we know: $n \in [\![\text{int}[I']]\!]_{g', k'-k_2} \Rightarrow n = I' \wedge \delta L_2 \leq c_1' \leq \delta U_2(f)$. Let us assume: $g' = g'(g)$.

STS1: $H \vDash_{g', k'-k''} P$. proved by assumption $H \vDash_{g', k'} P$

STS2: $v \in [\![A]\!]_{g', k'-k''}$. From (a). we know $H_1(l)[n] = v$. From $H \vDash_{g', k'-k''} \gamma$ we know: $\forall i \leq n, (H_1(l)(i)) \in [\![A]\!]_{g', k'-1}$ s.t we know $(H_1(l)[n]) \in [\![A]\!]_{g', k'-1}$

Because $k' - k'' \leq k' - 1$, By Lemma 1, We get: $(v) \in [\![A]\!]_{g', k'-k''}$.

STS3: $\delta(L_1 + L_2 + L_r) \leq c \leq \delta(U_1 + U_2 + U_r)$. By the fact that $\delta L_r \leq c_{read} \leq \delta U_r$, it is proved by (e)(f).

STS4: $\exists n. P = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \wedge \forall i \in [1, n]. g(\gamma_i) = (l_i, A, m) \Rightarrow \forall j. (H[l_i][j]) \neq H'[l_i][j] \Rightarrow j \in T_i$

H is not changed, H=H'. it is trivially true.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \mathsf{Array}[I]\ A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : \mathrm{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash_{L_3}^{U_3} t_3 : A \qquad \Delta;\Phi_a \vDash I' \leq I \qquad \Sigma;\Delta;\vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L_1+L_2+L_3+L_u,U_1+U_2+U_3+U_u)} \mathsf{updt}\ t_1\ t_2\ t_3 : \{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta\}}\ \text{U-U}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and there exists Ω'. s.t. $FV(t) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(\sigma) \in (\!|\delta\Omega'|\!)_{g,k}$

TS: $(\delta\sigma\ \mathsf{updt}\ t_1\ t_2\ t_3) \in [\![\{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta\}]\!]_{g,k}^{E,(0,0)}$

Because $\mathsf{updt}\ t_1\ t_2\ t_3$ is a value.

STS: $(\delta\sigma\ \mathsf{updt}\ t_1\ t_2\ t_3) \in [\![\{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta\}]\!]_{g,k}$

Unfold the definition of $[\![\{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta\}]\!]_{g,k}$.

Pick $g' \supseteq g,\ k' \leq k,\ k'' < k'$ and H. Assume $H \vDash_{g',k'} P \star \gamma \to \beta$ and $H;\mathsf{updt}\ t_1\ t_2\ t_3 \Downarrow^{k''}$

$H = H_p \uplus H_l \wedge H_p \vDash P \wedge H_l \vDash \gamma \to \beta$

From $H;\mathsf{updt}\ t_1\ t_2\ t_3 \Downarrow^{k''}$, because t_1, t_2 are sub terms of $\delta\sigma\ \mathsf{updt}\ t_1\ t_2\ t_3$,we get $\delta\sigma t_1 \Downarrow_f^{k_1}$ $\wedge \delta\sigma t_2 \Downarrow_f^{k_2} \wedge \delta\sigma t_3 \downarrow_f^{k_3}$ (1).

From (1), we get: $\exists l, n, v, k_1, k_2, k_3. \delta\sigma t_1 \Downarrow^{k_1,c_1} l \wedge \delta\sigma t_2 \Downarrow^{k_2,c_1'} n \wedge \delta\sigma t_3 \Downarrow^{k_3,c_1''} v \wedge\ k'' = k_1 + k_2 + k_3 + 1 \wedge c = c_1 + c_1' + c_1'' + c_{update}(a)$. From (a) and evaluation rules we get:

$H;\delta\sigma(\mathsf{updt}\ t_1\ t_2\ t_3) \Downarrow_f^{k'',c} H[l][n] \leftarrow v;()(b)$.

By IH on first premise instantiated with $(\sigma) \in [\![\delta\Omega']\!]_{g',k'}$ by lemma 1 and $FV(t_2) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$. We get:

$(\delta\sigma t_1) \in [\![\mathsf{Array}_\gamma[I]\ A]\!]_{G',k'}^{E,\delta(L_1,U_1)}(2)$.

Similarly, By IH on the second premise and third premise, we get:

$(\delta\sigma t_2) \in [\![\mathrm{int}[I]]\!]_{g',k'}^{E,\delta(L_2,U_2)}(3)$, $(\delta\sigma t_3) \in [\![A]\!]_{g',k'}^{E,\delta(L_3,U_3)}(4)$.

From (2),(3),(4), we know:

$$l \in [\![\mathsf{Array}_\gamma[I]\ A]\!]_{g',k'-k_1} \Rightarrow g'(\gamma) = (l, A, I) \wedge \delta L_1 \leq c_1 \leq \delta U_1 \quad \text{(c)}$$

$$n \in [\![\text{int}[I]]\!]_{g,k'-k_2} \Rightarrow n = I \wedge \delta L_2 \leq c_1' \leq \delta U_2 \quad \text{(d)}$$

$$v \in [\![A]\!]_{g',k'-k_3} \wedge \delta L_3 \leq c_1'' \leq \delta U_3 \quad \text{(e)}$$

Let us assume: $g'' = g'(5)$, $H_1' = H(l)[n] \leftarrow v = H_p + H_l(l)[n] \leftarrow v \wedge H_l' = H_l(l)[n] \leftarrow v(6)$.

STS1: $(H_1') \vDash_{g',k'-k''} P \star \gamma \rightarrow \beta$

Unfold $H \vDash_{g',k'} P \star \gamma \rightarrow \beta$ and consider (c), we know $\forall i \leq I.(H_l(l)[i]) \in (\!|A|\!)_{g',k'-1}(7)$.

We need to prove:

Subgoal 1: $\forall i \leq I.(H_l'(l)[i] \in [\![A]\!]_{g',k'-k''}$

when i=n, we prove $(H_l'(l)[i]) \in [\![A]\!]_{g',k'-k_3}$ from (e) and (6), and then use lemma 1.

when $i \neq n$, proved by using lemma 1 on (7).

STS2: $(()) \in [\![\text{unit}]\!]_{g',k'-k''}$. It is proved by unfolding the definition.

STS3: $\delta(L_1+L_2+L_3+L_u) \leq c \leq \delta(U_1+U_2+U_3+U_u)$. by the fact that $\delta L_u \leq c_{update} \leq \delta U_u$, it is proved by (c)(d)(e).

STS4: $\exists n.P = \{\gamma_1 \rightarrow T_1, \ldots, \gamma_n \rightarrow T_n\} \wedge \forall i \in [1,n].g(\gamma_i) = (l_i, A, m) \Rightarrow \forall j.(H[l_i][j]) \neq H'[l_i][j] \Rightarrow j \in T_i$

Because $H_1' = H(l)[n] \leftarrow v = H_p + H_l(l)[n] \leftarrow v \wedge H_l' = H_l(l)[n] \leftarrow v$, When $l_i \neq l$, $H(l_i) = H'(l_i)$, it is true. When $l_i = l$, only at position n so that $H(l)[n] \neq H(l)[n]$, we need to show $n \notin \beta$, which is the assumption from the rule.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash_L^U t : A \qquad \Delta;\Phi_a;\Omega \vdash p\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L,U)} \mathsf{return}\ t : \{P\}\ \exists\gamma.A\ \{P\}}\ \text{U-T}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and there exists Ω'. s.t. $FV(t) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(\sigma) \in (\!|\delta\Omega'|\!)_{g,k}$

TS: $(\delta\sigma\ \mathsf{return}\ t) \in [\![\delta\ \{P\}\ \exists\gamma.A\ \{P\}]\!]^{E,(0,0)}_{g,k}$

Because return t is value. STS: $(\delta\sigma\ \mathsf{return}\ t) \in [\![\delta\ \{P\}\ \exists\gamma.A\ \{P\}]\!]^{\mathrm{exec}(L,U)}_{g,k}$

Unfold the definition of $[\![\delta\ \{P\}\ \exists\gamma.A\ \{P\}]\!]^{\mathrm{exec}(L,U)}_{g,k}$. Pick $g^{'} \supseteq g$, $H_1, k' \leq k, k'' < k'$. Assume $H_1 \vDash_{g',k'} P(1)$, $H_1;\delta\sigma(\mathsf{return}\ t) \Downarrow_f^{k''} (2)$.

Because t is the subterm of return t , From (2), We know: $\delta\sigma_1 t \Downarrow^{k''}$ (3). From (3), we know:

$$\exists v_1, k_1, \delta\sigma_1 t \Downarrow^{k_1,c_1} v_1 \wedge k'' = k_1 + 1 \wedge c = c_1 + c_{ret} \tag{a}$$

From (a) and the evaluation rule U-ret: $\exists H_1'.H_1;\mathsf{return}\ t \Downarrow_f^{k'',c} v_1; H_1'(b)$.

By IH on the premise instantiated with $(\sigma) \in [\![\delta\Omega']\!]_{g',k'}$ by lemma 1. We get: $(\delta\sigma t_1) \in [\![\delta A]\!]^{E,\delta(L,U)}_{G',k'}(5)$.

Unfold (5), we get: $v_1 \in (\!|\delta A|\!)_{g',k'-k_1} \wedge \delta L_1 \leq c_1 \leq \delta U_1(6)$.

Let us assume: $g'' = g'(7)$, $H_1' = H_1'(8)$.

STS1: $H_1' \vDash_{g',k'-k''} P$. it is from assumption by using lemma 1.

STS2: $v_1 \in (\!|\tau|\!)_{g',k'-k''}$. It is proved by (6) using Lemma 1.

STS3: $\delta L \leq c \leq \delta U$ which is proved from (6)

STS4: $\exists n.P = \{\gamma_1 \rightarrow T_1, \ldots, \gamma_n \rightarrow T_n\} \wedge \forall i \in [1,n].g(\gamma_i) = (l_i, A, m) \Rightarrow \forall j.(H[l_i][j]) \neq H'[l_i][j] \Rightarrow j \in T_i$

H is not changed, it is trivial true.

Case

$$\frac{P = P_1 \star P_2 \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t : \{P_1\}\ \exists \vec{\gamma}_1 : A\ ^{\mathrm{exec}(L,U)}\{Q_1 \star Q_2\} \qquad \Sigma;\Delta,\vec{\gamma}_1;\Phi_a;\Omega,x:A \vdash^{U_2}_{L_2} u : \{Q_1 \star P_2\}\ \exists \vec{\gamma}_2 : B\ ^{\mathrm{exec}(L',U')}\{Q\}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+U_l}_{L_1+L_2+L_l} \mathsf{let}\{x\} = t\ \mathsf{in}\ u : \{P\}\ \exists \vec{\gamma}_1,\vec{\gamma}_2 : B\ ^{\mathrm{exec}(L+L',U+U')}\{Q \star Q_2\}}\ \text{U-E}$$

Assume that $\vdash \delta : \Delta$ and $\vDash \delta\Phi$ and there exists Ω'. s.t. $FV(t) \subseteq dom(\Omega')$ and $\Omega' \subseteq \Omega$ and $(\sigma) \in (\!|\delta\Omega'|\!)_{g,k}$

TS: $(\delta\sigma_1(\mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_1')) \in (\!|\delta\ \{P\}\ \exists \vec{\gamma}_1,\vec{\gamma}_2 : B\ ^{\mathrm{exec}(L+L',U+U')}\{Q \star Q_2\}|\!)^{(E,(U_1+U_2,L_1+L_2))}_{g,k}$

Unfold the definition . Pick $g^{'} \supseteq g$, H_1,$k' \leq k, k'' < k'$.

Assume $H_1 \vDash_{G',k'} P(1)$, $H_1;\delta\sigma_1(\mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_1') \Downarrow^{k''}_f$ (2).

Because t_1 is the subterm of $\mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_1'$, From (2), we know:

$\exists v_1, v_1', v_1'', v_1''', k_1, k_2, k_3, k_4, H_1', H_1''$.

$\delta\sigma_1 t_1 \Downarrow^{k_1,c_1} v_1 \wedge H_1; v_1 \Downarrow^{k_2,c_2}_f H_1'; v_1' \wedge \delta\sigma_1 t_1'[v_1'/x] \Downarrow^{k_3,c_3} v_1'' \wedge H_1'; v_1'' \Downarrow^{k_4,c_4}_f H_1''; v_1''',\ k'' = k_1 + k_2 + k_3 + k_4 + 1 \wedge c = c_1 + c_2 + c_3 + c_4 + c_{let}(a)$.

From evaluation rule: E-Let-F and (a)(b), we know:

$\exists H_a, H_c, H_a', H1_{Q1}, H1_{Q2}, H1_Q$,

$s.t. H_1 = H_a \uplus H_c \wedge v1; H_a \Downarrow_f v_1'; H_a' \wedge H_a' = H1_{Q1} \uplus H1_{Q2}$

$\wedge v_1''; H1_{Q1} \uplus H_c \Downarrow_f v_1'''; H1_Q \wedge H_1'' = H1_Q \uplus H1_{Q2}(c)$.

From (1) : $H \vDash_{g',k'} P_1 \star P_2$, we assume that $\exists g_a, g_b : (H_1 = H_a \uplus H_c)\ (g' = g_a \uplus g_c)$ and $(H_a) \vDash_{g_a,k'} P_1$ and $(H_c) \vDash_{g_c,k'} P_2(*)$.

STS1: $(H_1'') \vDash_{g'',k'-k''} Q \star Q_2$.

STS2: $(v_1''') \in (\!|\delta B|\!)_{G'',k'-k''}$.

STS3: $\delta(L + L') \leq c_f \leq \delta(U + U') \wedge \delta(L_1 + L_2 + L_{let}) \leq c_p \leq \delta(U_1 + U_2 + U_{let})$.

STS4: $P = \gamma_i \rightarrow \beta_i \forall i. g''(\gamma_i) = (l_i, m, A) \rightarrow H_1(l_i)[n] \neq H_1''(l_i)[n] \Rightarrow n \in \beta_i$.

By IH on the second premise instantiated with $(\sigma_1) \in (\!|\delta\Omega|\!)_{g',k'}$, we know:

$(\delta\sigma_1 t_1) \in (\!|\{P_1\}\ \exists\vec{\gamma}_1 : A\ \overset{\text{exec}\delta(L,U)}{\{Q_1 \star Q_2\}}|\!)^{E,(L_1,U_1)}_{g',k'}$(4). Unfold the definition ,use (a), we get:

$$(v_1) \in (\!|\delta\ \{P_1\}\ \exists\vec{\gamma}_1 : A\ \overset{\text{exec}\delta(L,U)}{\{Q_1 \star Q_2\}}|\!)_{g',k'} \wedge L_1 \le c_1 \le U_1 \wedge P_1 = \gamma_i \rightarrow \beta_i \Rightarrow$$

$$\forall i.g''(\gamma_i) = (l_i, m, A) \rightarrow H_1(l_i)[n] \neq H_1'(l_i)[n] \Rightarrow n \in \beta_i \qquad \text{(e)}$$

To unfold (e), we choose: $k_f \le k' - k_1$ and $k_f' < k_f$ and $g_f \supseteq g'$ and (1),(a).

We have $(H_a \vDash_{g_f,k_f} P_1 \wedge H_a; v_1 \Downarrow_f \wedge H_b; v_2 \Downarrow_f$.

Unfold the definition, we get: $H_a; v_1 \Downarrow_f^{c_2} H_a'; v_1' \wedge H_a' \vDash Q_1 \star Q_2$.

$(v_1') \in (\!|A|\!)_{g',k'-k_1} \wedge \delta L \le c_2 \le \delta U(k)$.

By IH on the third premise, we know: $(\delta\sigma_1 v_1'') \in (\!|\{Q_1 \star P_2\}\ \overset{\text{exec}\delta(L',U')}{\exists\vec{\gamma}_1.\delta B\ \{Q\}}|\!)^{E,\delta D_2}_{G',k'}$(5).

Unfold the definition, we get:

$v_1 \in (\!|\{Q_1 \star P_2\}\ \overset{\text{exec}\delta(L',U')}{\exists\vec{\gamma}_1.\tau_2\ \{Q\}}|\!)_{g',k'-k_1} \wedge l_2 \le c_3 \le U_2 \wedge Q_1 \uplus P_2 = \gamma_i \rightarrow \beta_i$

$\Rightarrow \forall i.g''(\gamma_i) = (l_i, m, A) \rightarrow H_1'(l_i)[n] \neq H_1''(l_i)[n] \Rightarrow n \in \beta_i(f)$.

We have $(H_a') \vDash_{g_f,k_f} Q_1 \star Q_2 \Rightarrow H1_{Q1} \vDash Q_1 \wedge H1_{Q2} \vDash Q_2 \wedge H_c \vDash P_2.\ H1_{Q1} \uplus H_c; v_1''' \Downarrow_f$.

Unfold the definition (f), we get:

$H1_{Q1} \uplus H_c; v_1''' \Downarrow_f^{c_4} H1_Q; v_1'''' \wedge H1_Q \vDash Q$, $(v_1''') \in (\!|\delta B|\!)_{g',k'-k_1} \wedge \delta L' \le c_4 \le \delta U'(g)$.

STS1: Use Lemma 2, we know that $H1_Q \uplus H1_{Q2} \vDash Q \star Q_2$, which proves STS1.

STS2: it is proved from (g).

STS3: it is proved from (e),(k),(f),(g) and $c = c_f + c_p. \wedge c_f = c_2 + c_4$

STS4: from (e),(f), we know all the n in P will appear in P_1 or $Q_1 \uplus P_2$. It is proved.

This completes the proof of Unary let.

Proof of statement (3) if $\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ and $\vdash \delta : \Delta$ and $\vDash \delta\Phi$. Then for $i \in \{1,2\}$. if there exists Γ_i' s.t. $\mathrm{FV}(e_i) \subseteq dom(\Gamma_i'), \Gamma_i' \subseteq \Gamma$ and $(\sigma_i) \in [\![|\delta\Gamma_i'|_i]\!]_{G|_i,k}$, then $\delta\sigma_i e_i \in [\![|\delta\tau|_i]\!]^{E,(0,\infty)}_{G|_i,k}$

Because it is similar for the two case, we only show the case when i=1.

Case

$$\frac{\Delta;\Phi_a;x:\tau_1,f\ :\tau_1 \xrightarrow{\mathrm{diff}\,D} \tau_2,\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \mathsf{Fix}\, f(x).t_1 \ominus \mathsf{Fix}\, f(x).t_2 \lesssim 0 : \tau_1 \xrightarrow{\mathrm{diff}\,D} \tau_2}\ \text{R-FIX}$$

Assume $\vDash \Phi\delta$ and there exists Γ' s.t. $\mathrm{FV}(\mathsf{fix}\, f(x).t_1) \subseteq dom(\Gamma'), \Gamma' \subseteq \Gamma$ and $\sigma \in [\![|\delta\Gamma'|_1]\!]_{G|_1,k}$.
TS: $(\mathsf{fix}\, f(x).\delta\sigma t_1) \in [\![(|\delta\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\delta\tau_2|_1)]\!]^{E,(0,\infty)}_{G|_1,k}$
By Lemma 31,STS: $(\mathsf{fix}\, f(x).\delta\sigma t_1) \in [\![(|\delta\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\delta\tau_2|_1)]\!]_{G|_1,k}$. We prove the more general statement:

$$\forall m' \le k.(\mathsf{fix}\, f(x).\delta\sigma t_1) \in [\![(|\delta\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\delta\tau_2|_1)]\!]_{G|_1,m} \tag{1}$$

By subinduction on m'. There are two cases.

Subcase 1: $m' = 0$. there is no non-negative k' < 0, it is vacuously true.

Subcase 2: $m' = m'' + 1 \le k$

By sub-IH: $(\mathsf{fix}\, f(x).\delta\sigma t_1) \in [\![(|\delta\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\delta\tau_2|_1)]\!]_{G|_1,m''}$.
STS: $(\mathsf{fix}\, f(x).\delta\sigma t_1) \in [\![(|\delta\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\delta\tau_2|_1)]\!]_{G|_1,m''+1}$

Unfold the definition. Pick $j \le m''+1$, $g' \supseteq G|_1$. Assume $v \in [\![|\delta\tau_1|_1]\!]_{g',j}$, STS: $\delta\sigma t_1[v/x][\mathsf{fix}\, f(x).t_1/f] \in [\![|\delta\tau_2|_1]\!]^{E,(0,\infty)}_{g',j}$

By IH 3 on the premise instantiated with
$\sigma[v/x][\mathsf{fix}\, f(x).t_1/f] \in [\![\delta(x:|\tau_1|_1, f:|\delta\tau_1|_1 \xrightarrow{\mathrm{exec}(0,\infty)} |\delta\tau_2|_1, |\Gamma'|)]\!]_{g',j}$ because

1. $\sigma \in [\![|\delta\Gamma'|_1)]\!]_{g',j}$ by Lemma 1.
2. $v \in [\![|\tau_1|_1]\!]_{g',j}$ from assumption.

3. $\mathsf{fix}\, f(x).t_1 \in [\![|\tau_1|_1 \overset{\mathrm{exec}(0,\infty)}{\longrightarrow} |\tau_2|_1]\!]_{g',j}$ by Lemma 1 on the sub-IH conclusion.

and premises: $FV(t_1) \subseteq dom(x:\tau_1, f:\tau_1 \overset{\mathrm{diff}\,D}{\longrightarrow} \tau_2, \Gamma')$ and $x:\tau_1, f:\tau_1 \overset{\mathrm{diff}\,D}{\longrightarrow} \tau_2, \Gamma' \subseteq x:\tau_1, f:\tau_1 \overset{\mathrm{diff}\,D}{\longrightarrow} \tau_2, \Gamma$.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathrm{int}[I] \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau \qquad \gamma \text{ fresh}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\, t_1\, t_2 \ominus \mathsf{alloc}\, t_1'\, t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 : \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}}\ \textsc{alloc}$$

Assume $\vDash \Phi\delta$ and there exists Γ' s.t. $\mathrm{FV}(\mathsf{alloc}\, t_1\, t_2) \subseteq dom(\Gamma'), \Gamma' \subseteq \Gamma$ and $\sigma \in [\![|\delta\Gamma'|_1]\!]_{G|_1,k}$

TS: $\delta\sigma\ \mathsf{alloc}\, t_1\, t_2 \in [\![|\delta(\{|P|_1\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ |\tau|_1\ \ \{|P|_1 \star \gamma \to \mathbb{N}\})|_1]\!]^{E,(0,\infty)}_{G|_1,k}$

STS: $\delta\sigma\ \mathsf{alloc}\, t_1\, t_2 \in [\![|\delta(\{|P|_1\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ |\tau|_1\ \ \{|P|_1 \star \gamma \to \mathbb{N}\})|_1]\!]_{G|_1,k}$

Unfold the definition.

Pick $g' \supseteq G|_1, k' \le k, k'' < k'$ and assume $H \vDash_{g',k'} |P|_1$ and $H; \mathsf{alloc}\, t_1\, t_2 \Downarrow_f^{k''}$

STS: $H; v \Downarrow_f^{k'',c} v', H'$ and $0 \le c \le \infty$ and $H' \vDash_{g'',k'-k''} \gamma$ and $v' \in [\![\mathsf{Array}_\gamma[I]\ |\tau|_1]\!]_{g'',k'-k''}$

From $H; \mathsf{alloc}\, t_1\, t_2 \Downarrow^{k''}$, because t_1, t_2 are sub terms of $\delta\sigma\ \mathsf{alloc}\, t_1\, t_2$,

we get: $\delta\sigma\, t_1 \Downarrow_f^{k_1} \wedge \delta\sigma\, t_2 \Downarrow_f^{k_2}$ (1). From (1), we get:

$$\exists v_1, v_2, k_1, k_2.\, \delta\sigma\, t_1 \Downarrow^{k_1,c_1} v_1 \wedge \delta\sigma\, t_2 \Downarrow^{k_2,c_1'} v_2 \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_1' + c_{alloc} \qquad \text{(a)}$$

From (a) and evaluation rules we get: $\exists l.H, (\mathsf{alloc}\, t_1\, t_2)\delta\sigma \Downarrow_f^{k'',c} l; H[l \to [v_2, \ldots v_2](b)$.

By IH 3 on the first premise instantiated with $\sigma \in [\![|\delta\Gamma'|_1]\!]_{g,k''}$ using lemma 1 and $FV(t_1) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$, we get: $(\delta\sigma\, t_1) \in [\![|\delta\mathrm{int}[I]|_1]\!]^{E,(0,\infty)}_{g',k''}(*)$.

Unfold the definition of $[\![\mathrm{int}[I]]\!]^{E,(0,\infty)}_{g',k''}$ and using (a) and $k_1 \le k''$, we know:

$$(v_1) \in [\![\mathrm{int}[I]]\!]_{G',k''-k_1} \Rightarrow v_1 = I \wedge 0 \le c_1 \le \infty \qquad \text{IH1}$$

By IH 3 on the second premise instantiated with $(\sigma) \in [\![|\delta\Gamma'|_1]\!]_{g',k'-k_1}$ using lemma 1 and $FV(t_2) \subseteq dom(|\Gamma'|_1)$ and $\Gamma' \subseteq \Gamma$, we get: $(\delta\sigma\, t_2) \in [\![|\delta\tau|_1]\!]^{E,(0,\infty)}_{g',k'-k_1}$ $(**)$. Unfold the definition of (**) and using (a), (b) and $k_2 \leq k' - k_1$, we know

$$(v_2) \in [\![|\delta\tau|_1]\!]_{g',k'-k_1-k_2} = [\![|\delta\tau|_1]\!]_{g',k'-k''+1} \wedge 0 \leq c_1' \leq \infty \qquad \text{IH2}$$

Let us assume: $g'' = g'[\gamma \to (l, |\tau|_1, I)](c)$, $H_1' = H_1, [l \to [v_2, v_2, \ldots, v_2]](d)$. We want to show 2 cases.

TS1 $(v' = l) \in [\![\mathsf{Array}_\gamma[I]\ |\tau|_1]\!]_{g'',k'-k''}$. it is proved by unfolding the definition and using the assumption (c).

TS2 $(H_1') \vDash_{g'',k'-k''} |P|_1 \star \gamma \to \mathbb{N}$

we know that $g''(\gamma) = (l, |\tau|_1, I)$, $H_1' = H_1 \uplus H_l \wedge H_l = l \to [v_2, \ldots, v_2]$, $H_1 \vDash |P|_1$ from assumption.

STS1 $\forall i \leq I, (H_l(l)[i] \in [\![A]\!]_{g'',k'-k''-1}$. It is proved by using IH2 and Lemma 1.

TS3 $0 \leq c \leq \infty$. It is proved by IH1 and IH2 and $0 \leq c_{alloc} \leq \infty$.

TS4: $\exists n.|P|_1 = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \wedge \forall i \in [1,n].g(\gamma_i) = (l_i, A, m) \Rightarrow \forall j.(H[l_i][j]) \neq H'[l_i][j] \Rightarrow j \in T_i$

Because γ not in $|P|_1$. For all γ_i, H,H' is the same.

Case

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a \vDash I' \leq I\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \lesssim 0 : \{P\}\ \exists\_.\tau\ \{P\}}\ \text{R-READ}$$

Assume $\vDash \Phi\delta$ and there exists Γ' s.t. FV(read t_1 t_2) $\subseteq dom(\Gamma'), \Gamma' \subseteq \Gamma$ and $\sigma \in [\![|\delta\Gamma'|_1]\!]_{G|_1,k}$

TS: $\delta\sigma$ read t_1 $t_2 \in [\![|\delta(\{|P|_1\}\ \overset{\mathrm{exec}(0,\infty)}{\exists\_.|\tau|_1}\ \{|P|_1\})|_1]\!]^{E,(0,\infty)}_{G|_1,k}$

STS: $\delta\sigma$ read $t_1\ t_2 \in [\![|\delta(\{|P|_1\}\ \overset{\text{exec}(0,\infty)}{\exists\_.}|\tau|_1\ \{|P|_1\})|_1]\!]_{G|_1,k}$

Unfold the definition.

Pick $g' \supseteq G|_1, k' \leq k, k'' < k'$ and assume $H \vDash_{g',k'} |P|_1$ and $H;\text{read}\ t_1\ t_2 \Downarrow_f^{k''}$

STS: $H;\text{read}\ t_1\ t_2 \Downarrow_f^{k'',c}\ v', H'$ and $0 \leq c \leq \infty$ and $H' \vDash_{g'',k'-k''} \gamma$ and $v' \in [\![|\tau|_1]\!]_{g'',k'-k''}$

From $H;\text{read}\ t_1\ t_2 \Downarrow^{k''}$, because t_1, t_2 are sub terms of $\delta\sigma$ read $t_1\ t_2$,

we get:$\delta\sigma\, t_1 \Downarrow_f^{k_1} \wedge \delta\sigma\, t_2 \Downarrow_f^{k_2}$ (1). From (1), we get: $\exists l, n, v, k_1, k_2$.

$\delta\sigma\, t_1 \Downarrow^{k_1,c_1} l \wedge \delta\sigma\, t_2 \Downarrow^{k_2,c_1'} n \wedge k'' = k_1 + k_2 + 1 \wedge c = c_1 + c_1' + c_{read} \wedge H(l)[n] = v(a)$.

From (a) and evaluation rules we get: $H;\delta\sigma(\text{read}\ t_1\ t_2) \Downarrow_f^{k'',c} H; v$ (b).

By IH 3 on the first premise instantiated with $\sigma \in [\![|\delta\Gamma'|_1]\!]_{g,k''}$using lemma 1 and $FV(t_1) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$, we get:$(\delta\sigma\, t_1) \in [\![\text{Array}_\gamma[I]\ |\tau|_1]\!]_{g',k'}^{E,(0,\infty)}$(6).

Unfold (6), since $k_1 \leq k'$ and $\delta\sigma_1\, t_1 \Downarrow^{k_1,c_1} l$, we know $l \in [\![\text{Array}_\gamma[I]\ |\tau|_1]\!]_{g',k'-k_1} \wedge 0 \leq c_1 \leq \infty(e)$.

From (e), we know : $g'(\gamma) = (l, |\tau|_1, I)(7)$.

By IH 3 on the first premise instantiated with $\sigma \in [\![|\delta\Gamma'|_1]\!]_{g,k''}$using lemma 1 and $FV(t_2) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$, we get: $(\delta\sigma_1\, t_2') \in [\![\text{int}[I']]\!]_{g',k'}^{E,(0,\infty)}$(8). Unfold (8), since $k_2 \leq k'$,we know

$$n \in [\![\text{int}[I']]\!]_{g',k'-k_2} \Rightarrow n = I' \wedge 0 \leq c_1' \leq \infty \tag{f}$$

Let us assume: $g' = g'$ (g).

STS1: $H \vDash_{g',k'-k''} |P|_1$. It is proved by Lemma 1 on assumption $H \vDash_{g',k'} |P|_1$

STS2: $(v' = v) \in [\![|\tau|_1]\!]_{g',k'-k''}$

from (a). we know $H_1(l)[n] = v$. From $H \vDash_{g',k'-k''} \gamma$, we know: $\forall i \leq n, (H_1(l)(i)) \in [\![A]\!]_{g',k'-1}$.

s.t we know $(H_1(l)[n]) \in [\![A]\!]_{g',k'-1}$. Because $k' - k'' \leq k' - 1$, By Lemma 1, We get:

$v \in [\![A]\!]_{g',k'-k''}$

STS3: $0 \le c \le \infty$. Proved by the fact that $0 \le c_{read} \le \infty$, it is proved by (e)(f).

STS4: $\exists n.|P|_1 = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \wedge \forall i \in [1,n].g(\gamma_i) = (l_i, A, m) \Rightarrow \forall j.(H[l_i][j]) \neq H'[l_i][j] \Rightarrow j \in T_i$

H is not changed , it is trivially true.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad \Delta;\Phi_a \models I' \in \beta}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \overset{\mathrm{diff}(D_1+D_2)}{\lesssim} 0 : \{\gamma \to \beta\}\ \exists\_.\square\tau\ \{\gamma \to \beta\}}\ \text{R-RB}$$

The proof is same as case read above.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \mathrm{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 : \tau \qquad \Delta;\Phi_a \models I' \le I}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \overset{\mathrm{diff}(D_1+D_2+D_3)}{\lesssim} 0 : \{P \star \gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{P \star \gamma \to \beta \cup \{I'\}\}}\ \text{R-U}$$

Assume that $\models \Phi\delta$ and there exists Γ' s.t. FV(updt $t_1\ t_2\ t_3$) $\subseteq dom(\Gamma'), \Gamma' \subseteq \Gamma$ and $\sigma \in [\![|\delta\Gamma'|_1]\!]_{G|_1,k}$

TS: $\delta\sigma$ updt $t_1\ t_2\ t_3 \in [\![|\delta(\{|P|_1 \star \gamma \to \mathbb{N}\}\ \exists\_.|\mathrm{unit}|_1\ \{|P|_1 \star \gamma \to \mathbb{N}\})|_1]\!]^{E,(0,\infty)}_{G|_1,k}$

STS: $\delta\sigma$ updt $t_1\ t_2\ t_3 \in [\![|\delta(\{|P|_1 \star \gamma \to \mathbb{N}\}\ \exists\_.|\mathrm{unit}|_1\ \{|P|_1 \star \gamma \to \mathbb{N}\})|_1]\!]_{G|_1,k}$

Unfold the definition.

Pick $g' \supseteq G|_1, k' \le k, k'' < k'$ and assume $H \models_{g',k'} |P|_i \star \gamma \to \mathbb{N}$ and H;updt $t_1\ t_2\ t_3 \downarrow^{k''}_f$

STS: H;updt $t_1\ t_2\ t_3 \Downarrow^{k'',c}_f v', H'$ and $0 \le c \le \infty$ and $H' \models_{g'',k'-k''} \gamma$ and $v' \in [\![|\mathrm{unit}|_1]\!]_{g'',k'-k''}$

From $H;\mathsf{updt}\ t_1\ t_2\ t_3 \Downarrow^{k''}$, because t_1, t_2 are sub terms of $\delta\sigma\ \mathsf{updt}\ t_1\ t_2\ t_3$,we get: $\delta\sigma\, t_1 \Downarrow_f^{k_1}$ $\wedge \delta\sigma\, t_2 \Downarrow_f^{k_2} \wedge \delta\sigma\, t_3 \Downarrow_f^{k_3}$ (1). From (1), we get:

$$\exists l, n, v, k_1, k_2, k_3.\, \delta\sigma\, t_1 \Downarrow^{k_1, c_1} l \wedge \delta\sigma\, t_2 \Downarrow^{k_2, c_1'} n \wedge \delta\sigma\, t_3 \Downarrow^{k_3, c_1''} v \wedge$$
$$k'' = k_1 + k_2 + k_3 + 1 \wedge c = c_1 + c_1' + c_1'' + c_{update} \tag{a}$$

From (a) and evaluation rules we get:$H;\delta\sigma(\mathsf{updt}\ t_1\ t_2\ t_3) \Downarrow_f^{k'', c} H[l][n] \leftarrow v;()(b)$.

By IH 3 on the first premise instantiated with $\sigma \in [\![|\delta\Gamma'|_1]\!]_{g,k''}$using lemma 1 and $FV(t_2) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$,

we get:$(\delta\sigma\, t_1) \in [\![\mathsf{Array}_\gamma[I]\ |\tau|_1]\!]^{E,(0,\infty)}_{G',k'}(2)$.

Similarly, By IH 3 on the second premise and third premise,

we get: $(\delta\sigma\, t_2) \in [\![|\mathsf{int}[I]|_1]\!]^{E,(0,\infty)}_{g',k'}(3)$, $(\delta\sigma\, t_3) \in [\![|\tau|_1]\!]^{E,(0,\infty)}_{g',k'}(4)$. From (2),(3),(4), we know:

$$l \in [\![\mathsf{Array}_\gamma[I]\ A]\!]_{g',k'-k_1} \Rightarrow g'(\gamma) = (l, A, I) \wedge 0 \leq c_1 \leq \infty \tag{c}$$
$$n \in [\![\mathsf{int}[I]]\!]_{g,k'-k_2} \Rightarrow n = I \wedge 0 \leq c_1' \leq \infty \tag{d}$$
$$v \in [\![A]\!]_{g',k'-k_3} \wedge 0 \leq c_1'' \leq \infty \tag{e}$$

Let us assume: $g'' = g'(5)$, $H_1' = H(l)[n] \leftarrow v(6)$.

STS1: $(H_1') \vDash_{g',k'-k''} |P|_1 \star \gamma \rightarrow \mathbb{N}$

$H_1 = H_p \uplus H_l$, $H_1' = H_p \uplus H_l(l) \leftarrow v$, From assumption $H_1 \vDash |P|_1 \star \gamma \rightarrow \mathbb{N}$

STS: $H_l' = H_l(l) \leftarrow v \vDash \gamma \rightarrow \mathbb{N}$, which is proved by the assumption that $H_l \vDash \gamma \rightarrow \mathbb{N}$.

STS2: $\big(()\big) \in [\![\mathsf{unit}]\!]_{g',k'-k''}$. It is proved by unfolding the definition.

STS3: $0 \leq c \leq \infty$. by the fact that $\delta L_u \leq c_{update} \leq \delta U_u$, it is proved by (c)(d)(e).

STS4: $\exists n.|P|_i \star \gamma \rightarrow \mathbb{N} = \{\gamma_1 \rightarrow T_1, \ldots, \gamma_n \rightarrow T_n\} \wedge \forall i \in [1, n].g(\gamma_i) = (l_i, A, m)$

$\Rightarrow \forall j.(H[l_i][j]) \neq H'[l_i][j] \Rightarrow j \in T_i$

When $l_i \neq l$, the heap is not changed. it is true;

When $l_i = l$, $H(l)[n] \neq H'(l)[n]$, To show $n \in \beta$, which the premise in the typing rule.

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ t_1 \ominus \mathsf{return}\ t_2 \lesssim 0 : \{P\}\ \overset{\mathrm{diff}(D)}{\exists\_.\tau}\ \{P\}}\ \text{R-T}$$

Assume $\vDash \Phi\delta$ and there exists Γ' s.t. FV(return t_1) $\subseteq dom(\Gamma'), \Gamma' \subseteq \Gamma$ and $\sigma \in [\![|\delta\Gamma'|_1]\!]_{G|_1,k}$

TS: $\delta\sigma$ return $t_1 \in [\![|\delta(\{|P|_1\}\ \overset{\mathrm{exec}(0,\infty)}{\exists\_.|\tau|_1}\ \{|P|_1\})|_1]\!]^{E,(0,\infty)}_{G|_1,k}$

STS: $\delta\sigma$ return $t_1 \in [\![|\delta(\{|P|_1\}\ \overset{\mathrm{exec}(0,\infty)}{\exists\_.|\tau|_1}\ \{|P|_1\})|_1]\!]_{G|_1,k}$

Unfold its definition.

Pick $g' \supseteq G|_1, k' \leq k, k'' < k'$ and assume $H \vDash_{g',k'} |P|_1$ and H;return $t_1 \Downarrow^{k''}_f$

STS: H;return $t_1 \Downarrow^{k'',c}_f v', H'$ and $0 \leq c \leq \infty$ and $H' \vDash_{g'',k'-k''} \gamma$ and $v' \in [\![|\tau|_1]\!]_{g'',k'-k''}$

Because t_1 is the subterm of return t_1 ,we know: $\delta\sigma_1 t_1 \Downarrow^{k''}$ (3). From (3), we know:

$$\exists v_1, k_1, \delta\sigma_1 t_1 \Downarrow^{k_1,c_1} v_1 \wedge k'' = k_1 + 1 \wedge c = c_1 + c_{ret} \tag{a}$$

From (a),(b) and the evaluation rule R-ret: H_1;return $t_1 \Downarrow^{k'',c}_f v_1; H_1(b)$.

By IH 3 on the first premise instantiated with $\sigma \in [\![|\delta\Gamma'|_1]\!]_{g,k''}$using lemma 1 and $FV(t_2) \subseteq dom(\Gamma')$ and $\Gamma' \subseteq \Gamma$, we get: $(\delta\sigma\, t_1) \in [\![|\delta\tau|_1]\!]^{E,(0,\infty)}_{G',k'}$(5).

Unfold (5), we get:$v_1 \in (\!|\delta A|\!)_{g',k'-k_1} \wedge 0 \leq c_1 \leq \infty$(6). Let us assume: $g'' = g'$(7) $H'_1 = H_1$(8).

STS1: $H_1 \vDash_{g',k'-k''} |P|_1$. It is proved by the assumption on Lemma (monotonicity).

STS2: $(v_1) \in (\!||\tau|_1|\!)_{g',k'-k''}$. It is proved by (6) using Lemma 1.

STS3: $0 \leq c \leq \infty$ which is proved from (6)

STS4: $\exists n.|P|_1 = \{\gamma_1 \to T_1, \ldots, \gamma_n \to T_n\} \wedge \forall i \in [1,n].g(\gamma_i) = (l_i, A, m) \Rightarrow \forall j.(H[l_i][j]) \neq$

$H'[l_i][j] \Rightarrow j \in T_i$

H is not changed, it is true.

This completes the proof of case unary return.

A.6 EXAMPLES REVISTED

A.6.1 Cooley Tukey FFT Algorithm

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{sp}\ (\_).\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda x.\lambda n.\lambda i.\lambda y.\lambda pr. \\
\quad \mathsf{if}\ i < (n/2)\ \mathsf{then} \\
\quad\quad \mathsf{let}\ \{a\} = \mathsf{read}\ x\ (pr + 2 * i)\ \ \mathsf{in} \\
\quad\quad \mathsf{let}\ \{b\} = \mathsf{read}\ x\ (pr + 2 * i + 1)\ \ \mathsf{in} \\
\quad\quad \mathsf{let}\ \{\_\} = \mathsf{updt}\ y\ (pr + i)\ a\ \ \mathsf{in} \\
\quad\quad \mathsf{let}\ \{\_\} = \mathsf{updt}\ y\ (pr + i + n/2)\ b\ \ \mathsf{in} \\
\quad\quad\quad \mathsf{celim}(\mathsf{sp}()\ []\ []\ []\ []\ []\ [])\ x\ n\ (i+1)\ y\ pr \\
\quad \mathsf{else} \\
\quad\quad \mathsf{return}\ ()
\end{array}
$$

$$
\begin{array}{lll}
 & & \mathrm{unit} \to \forall \gamma_1, \gamma_2 : \mathbb{L}.\forall M, N, I, PR : \mathbb{N}.(I \le (M/2) \wedge (M + PR) < N) \supset \\
\vdash & \mathsf{sp} & : \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \mathrm{int}[M] \to \mathrm{int}[I] \to \mathsf{Array}_{\gamma_2}[N]\ \mathrm{int} \to \mathrm{int}[PR] \\
 & & \to \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\}\ \overset{\mathrm{exec}(4*(M/2-I),4*(M/2-I))}{\exists\_}.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\}
\end{array}
$$

$$
\begin{array}{lll}
 & & \mathrm{unit} \to \forall \gamma_1 \gamma_2 : \mathbb{L}.\forall M, N, I, PR : \mathbb{N}.\forall \beta_1.(I \le (M/2) \wedge (M + PR) < N) \\
\vdash \mathsf{sp} \ominus \mathsf{sp} \lesssim 0 : & & \supset \mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}, \mathrm{int}) \to \mathrm{int}[M] \to \mathrm{int}[I] \to \mathsf{Array}_{\gamma_2}[N]\ U(\mathrm{int}, \mathrm{int}) \\
 & & \to \mathrm{int}[PR] \to \{\gamma_1 \to \beta_1, \gamma_2 \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists\_}.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{cp}\,(\_).\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda x.\lambda y.\lambda l.\lambda u.\\
\quad \mathsf{if}\ l \le u\ \mathsf{then}\\
\quad\quad \mathsf{let}\ \{a\} = \big(\ \mathsf{read}\ x\ l\ \big)\ \mathsf{in}\\
\quad\quad \mathsf{let}\{\_\} = \big(\ \mathsf{updt}\ y\ l\ a\ \big)\ \mathsf{in}\\
\quad\quad \big(\mathsf{celim}(\mathsf{cp}()\ []\ []\ []\ []\ [])\ x\ y\ (l+1)\ u\big)\\
\quad \mathsf{else}\ \ \mathsf{return}()
\end{array}
$$

$$
\vdash\ \mathsf{cp}\ :\ \begin{array}{l}\mathrm{unit} \to \forall\gamma_1,\gamma_2.\forall L,U,N.(L \le U \le N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \mathsf{Array}_{\gamma_2}[N]\ \mathrm{int} \to \\ \mathrm{int}[L] \to \mathrm{int}[U] \to \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\}\ \overset{\mathrm{exec}(2*(U-L+1),2*(U-L+1))}{\exists\_.\mathrm{unit}}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\}\end{array}
$$

$$
\vdash \mathsf{cp} \ominus \mathsf{cp} \lesssim 0 :\ \begin{array}{l}\mathrm{unit} \to \forall\gamma_1,\gamma_2,\beta_1.\forall L,U,N.(L \le U \le N)\\ \supset \mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int},\mathrm{int}) \to \mathsf{Array}_{\gamma_2}[N]\ U(\mathrm{int},\mathrm{int})\\ \to \mathrm{int}[L] \to \mathrm{int}[U] \to \{\gamma_1 \to \beta_1, \gamma_2 \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists\_.\mathrm{unit}}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\}\end{array}
$$

separate

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{separate}\,(\_).\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda x.\lambda n.\lambda y.\lambda pr.\\
\quad \mathsf{let}\ \_ = \mathsf{celim}(\mathsf{sp}()\ []\ []\ []\ []\ []\ [])\ x\ n\ 0\ y\ pr\ \mathsf{in}\\
\quad\quad \mathsf{celim}(\mathsf{cp}()\ []\ []\ []\ []\ [])\ y\ x\ pr\ (n+pr)
\end{array}
$$

$$\vdash \text{separate} : \begin{array}{l} \text{unit} \to \forall \gamma_1, \gamma_2. \forall M, N, PR. (M + PR < N) \supset \text{Array}_{\gamma_1}[N]\ \text{int} \to \text{int}[M] \\ \to \text{Array}_{\gamma_2}[N]\ \text{int} \to \text{int}[PR] \to \\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\}\ \overset{\text{exec}(4*M, 4*M)}{\exists \_.\text{unit}}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\} \end{array}$$

$$\vdash \text{separate} \ominus \text{seperate} \lesssim 0 : \begin{array}{l} \text{unit} \to \forall \gamma_1, \gamma_2, \beta_1. \forall M, N, PR. (M + PR < N) \\ \supset \text{Array}_{\gamma_1}[N]\ U(\text{int}, \text{int}) \to \\ \text{int}[M] \to \text{Array}_{\gamma_2}[N]\ U(\text{int}, \text{int}) \to \text{int}[PR] \to \\ \{\gamma_1 \to \beta_1, \gamma_2 \to \mathbb{N}\}\ \overset{\text{diff}(0)}{\exists \_.\text{unit}}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\} \end{array}$$

fix loop (_).Λ.Λ.Λ.Λ.Λ.$\lambda k.\lambda n.\lambda x.\lambda pr.$

if $k < (n/2)$ then

let $\{e\}$ =read x $(k + pr)$ in

let $\{o\}$ =read x $(k + pr + n/2)$ in

let $w = e^{-2*PI*k/n}$ in

let $\{\_\}$ =updt x $(k + pr)$ $(e + w * o)$ in

let $\{\_\}$ =updt x $(k + pr + n/2)$ $(e - w * o)$ in

celim(loop() [] [] [] [] []) $(k + 1)$ n x pr

else

return ()

$$\vdash \quad \mathsf{loop} \quad : \quad \begin{array}{l} \mathrm{unit} \to \forall \gamma_1.\forall K, M, N, PR.(PR + M < N) \supset \\ \mathrm{int}[K] \to \mathrm{int}[M] \to \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \\ \to \mathrm{int}[PR] \to \{\gamma_1 \to \mathbb{N}\}\ \exists\_.\mathrm{unit}^{\mathrm{exec}(4*(M-K),4*(M-K))}\ \{\gamma_1 \to \mathbb{N}\} \end{array}$$

$$\vdash \mathsf{loop} \ominus \mathsf{loop} \lesssim 0 : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1, \beta_1.\forall K, M, N, PR.(PR + M < N) \supset \\ \mathrm{int}[K] \to \mathrm{int}[M] \to \\ \mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}, \mathrm{int}) \to \mathrm{int}[PR] \to \{\gamma_1 \to \beta_1\}\ \exists\_.\mathrm{unit}^{\mathrm{diff}(0)}\ \{\gamma_1 \to \mathbb{N}\} \end{array}$$

$\mathsf{fix}\ \mathsf{FFT}\ (\_).\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda x.\lambda y.\lambda n.\lambda pr.$

$\quad \mathsf{if}\ 2 \le n\ \mathsf{then}$

$\quad\quad \mathsf{let}\ \{\_\} = \mathsf{celim}(\mathsf{separate}\ ()\ []\ []\ []\ []\ [])\ x\ n\ y\ pr\ \mathsf{in}$

$\quad\quad \mathsf{let}\ \{\_\} = \mathsf{celim}(\mathsf{FFT}\ ()\ []\ []\ []\ []\ [])\ x\ y\ (n/2)\ pr\ \mathsf{in}$

$\quad\quad \mathsf{let}\ \{\_\} = \mathsf{celim}(\mathsf{FFT}\ ()\ []\ []\ []\ []\ [])\ x\ y\ (n/2)\ (pr + n/2)\ \mathsf{in}$

$\quad\quad\quad \mathsf{celim}(\mathsf{loop}()\ []\ []\ []\ []\ []\ [])\ 0\ n\ x\ pr$

$\quad \mathsf{else}$

$\quad\quad \mathsf{return}\ ()$

$$\vdash \quad \mathsf{FFT} \quad : \quad \begin{array}{l} \mathrm{unit} \to \forall \gamma_1, \gamma_2.\forall M, N, PR.(PR + M < N) \supset \\ \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \mathsf{Array}_{\gamma_2}[N]\ \mathrm{int} \to \\ \mathrm{int}[M] \to \mathrm{int}[PR] \to \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N},\}\ \exists\_.\mathrm{unit}^{\mathrm{exec}(8*M*\log(M),8*M*\log(M))}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N},\} \end{array}$$

$$\vdash \mathsf{FFT} \ominus \mathsf{FFT} \lesssim 0 : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1, \gamma_2, \beta_1. \forall M, N, PR. (PR + M < N) \supset \\ \mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{int}, \mathrm{int}) \\ \to \mathsf{Array}_{\gamma_2}[N]\ U(\mathrm{int}, \mathrm{int}) \to \mathrm{int}[M] \to \mathrm{int}[PR] \to \\ \{\gamma_1 \to \beta_1, \gamma_2 \to \mathbb{N}\}\ \exists^{\mathrm{diff}(0)}\_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}, \gamma_2 \to \mathbb{N}\} \end{array}$$

A.6.2 Naive String Search

Description: Given a long text string S :Array$_{\gamma_1}[N]$ int and a shorter substring W :Array$_{\gamma_2}[M]$ int to be searched in S. There is an int array P :Array$_{\gamma_3}[N]$ int which will store the matching result (1 as match, 0 as mismatch) for the same index as the text string. For example, for index 1, there is a match with W from index 1 in the text string S and $P[1] = 1$.

Helper function

Helper function will do the match process for one index m in s and update $p[m]$, i will start from 0 till the length of w (Q) if the prefix matches.

```
search = fix f (_).Λ.Λ.Λ.Λ.Λ.Λ.Λ.Λ.λs.λw.λm.λi.λl_w.λp.
        let {a} =read s (m + i) in
        let {b} =read w i in
          if (i + 1) == l_w then
            if a == b then...(1)
              updt p m 1
            else
              updt p m 0
          else
            if a == b then...(2)
              celim (f () [] [] [] [] [] [] []) w m (i + 1) l_w p
            else
              updt p m 0
```

$$\vdash \text{search} : \begin{array}{l} \text{unit} \to \\ \forall \gamma_1, \gamma_2, \gamma_3 : \mathbb{L}. \forall I, M, Q, N : \mathbb{N}. (I < Q < N \wedge M + I < N) \supset \text{Array}_{\gamma_1}[N]\ \text{int} \\ \to \text{Array}_{\gamma_2}[Q]\ \text{int} \to \text{int}[M] \to \text{int}[I] \to \text{int}[Q] \to \text{Array}_{\gamma_3}[N]\ \text{int} \to \\ \{\gamma_1 \to \emptyset, \gamma_2 \to \emptyset, \gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u + r, (Q-I) * r + c_u)}{\exists \_.\text{unit}}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \emptyset, \gamma_3 \to \{M\}\} \end{array}$$

Notice, one thing to mention is that in the unary type, in the precondition, $\gamma_3 \to \{M\}$, it means before helper is executed, it can only touch position M in p.

Where r is the cost of two reads and c_u is the cost for update, w has length Q, s,p have length N.

$$\vdash \text{search} \ominus \text{search} \lesssim 0 : \begin{array}{l} \text{unit} \to \forall \gamma_1, \gamma_2, \gamma_3 : \mathbb{L}. \forall I, M, Q, N : \mathbb{N}. \forall \beta_2, \beta_3. \\ (I < Q < N \wedge M + I < N) \subset \text{Array}_{\gamma_1}[N]\ \text{int} \to \text{Array}_{\gamma_2}[Q]\ \text{int} \\ \to \text{int}[M] \to \text{int}[I] \to \text{int}[N] \to \text{int}[Q] \to \text{Array}_{\gamma_3}[N]\ \text{int} \to \\ \{P, \gamma_3 \to \beta_3\}\ \overset{\text{diff}((Q-1-min(MIN(\beta_2 \cap [I,\infty)), Q-1)) * r)}{\exists \_.\text{unit}}\ \{P, \gamma_3 \to \beta_3 \cup \{M\}\} \end{array}$$

where $MIN(beta : set)$ return the minimum element in the set $beta$, $P = \gamma_1 \to \emptyset, \gamma_2 \to \beta_2$

First, we will apply rule R-Fix-Ext.

Set $\sigma = \text{unit} \to \forall \gamma_1, \cdots \{P, \gamma_3 \to \beta_3\}\ \overset{\text{diff}((Q-1-min(MIN(\beta_2 \cap [I,\infty)), Q-1)) * r)}{\exists \_.\text{unit}}\ \{P, \gamma_3 \to \beta_3 \cup \{M\}\}$.

Set $A_1 = A_2 = \text{unit} \to \forall \gamma_1 \gamma_2, \gamma_3 : \mathbb{L}. \forall I, M, Q, N : \mathbb{N}. (I < Q < N \wedge M + I < N)$

$\supset \text{Array}_{\gamma_1}[N]\ \text{int} \to \text{Array}_{\gamma_2}[Q]\ \text{int}$

$\to \text{int}[M] \to \text{int}[I] \to \text{int}[Q] \to \text{Array}_{\gamma_3}[N]\ \text{int} \to \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u, (Q-I-1) * r + c_u)}{\exists \_.\text{unit}}\ \{\gamma_3 \to \{M\}\}$.

Set $\Gamma = \_ : \text{unit}, f : \text{unit} \xrightarrow{\text{diff0}} \sigma, f\ U(A_1, A_2)$

$$\frac{\Gamma \vdash \Lambda.\ldots.\lambda s.\ldots.. \ominus \Lambda.\ldots..\lambda s.\ldots.. \lesssim 0 : \sigma \qquad |\Gamma|_1 \vdash_0^0 \mathsf{Fix}\, f(\_).\Lambda.\ldots..\lambda s.\ldots. : A_1 \qquad |\Gamma|_2 \vdash_0^0 \mathsf{Fix}\, f(\_).\Lambda.\ldots..\lambda s.\ldots. : A_2}{\Gamma \vdash \mathsf{Fix}\, f(\_).\Lambda.\ldots. \ominus \mathsf{Fix}\, f(\_).\Lambda.\ldots. \lesssim 0 : \text{unit} \xrightarrow{\text{diff0}} \sigma}\ \text{R-FIX-EXT}$$

We first show how do we get the unary type A_1.

Set $A_1 = \text{unit} \to B_1$. So $B_1 = \forall \gamma_1 \gamma_2, \gamma_3 : \mathbb{L}.\ldots..$, Set $\Omega = |\Gamma|_1 = \_ : \text{unit}, f : A_1$.

unary type

First, we apply U-fix rule.

$$\frac{\Omega \vdash_0^0 \Lambda.\ldots..\lambda S.\ldots. : B_1}{\Omega \vdash_0^0 \mathsf{Fix}\, f(\_).\Lambda.\ldots. : A_1}\ \text{U-FIX}$$

Then, we apply rule U-ILam several times to introduce new index term into sort environment Δ.

Set $\Delta = \gamma_1 : \mathbb{L}, \gamma_2 : \mathbb{L}, \gamma_2 : \mathbb{L}, I : \mathbb{N}, M : \mathbb{N}, N : \mathbb{N}, Q : \mathbb{N}$.

To introduce the constraint, apply U-CIMPL. $\Phi_a = (I < Q < N \wedge M + I < N)$.

Apply U-Abs several times to eliminate all the lambdas, we introduce variables into the unary context, $\Omega = s : \mathsf{Array}_{\gamma_1}[N]\ \text{int}, w : \mathsf{Array}_{\gamma_2}[Q]\ \text{int}, m : \text{int}[M]$,

$i : \text{int}[I], l_w : \text{int}[Q], p : \mathsf{Array}_{\gamma_3}[N]\ \text{int}$. we will show:

$$\frac{\begin{array}{c}\{\gamma_3 \to \{M\}\} = \{\gamma_3 \to \{M\}\} \star \mathsf{empty} \\ \vdash_0^0 \mathsf{read}\ s\ (m+i) : \{\gamma_3 \to \{M\}\} \xrightarrow{\text{exec}(c_r, c_r)} \exists \_.\text{int}\ \{\gamma_3 \to \{M\}\} \\ \Delta; \Phi_a; a : \text{int}, \Omega \vdash_0^0 \text{let}\ \{b\} = \mathsf{read}\ b\ (i) \cdots : \{\gamma_3 \to \{M\}\} \xrightarrow{\text{exec}(c_r + c_u, c_r + (Q-I-1)*r + c_u)} \exists \_.\text{int}\ \{\gamma_3 \to \{M\}\}\end{array}}{\Delta; \Phi_a; \Omega \vdash \text{let}\ \{a\} = \mathsf{read}\ s\ (m+i) \ldots : \{\gamma_3 \to \{M\}\} \xrightarrow{\text{exec}(c_u + r, (Q-I)*r + c_u)} \exists \_.\text{unit}\ \{\gamma_3 \to \{M\}\}}\ \text{U-LET}$$

We apply U-Let again to introduce b into the unary environment. Set $\Omega' = a : \text{int}, b : \text{int}, \Omega$.

We need to show the unary type of the first if term.

$$\dfrac{\Delta;\Phi_a;\Omega' \vdash_0^0 \text{if } (a == b)(1) : \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \mathbb{N}\} \quad \Delta;\Phi_a;\Omega' \vdash_0^0 \text{if } (a == b)(2) : \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\} \quad \vdash_0^0 i+1 == l_w : \text{bool}}{\Delta;\Phi_a;\Omega' \vdash_0^0 \text{if } (i+1) == l_w \text{ then if } (a == b)(1) \text{ elseif } (a == b)(2) : \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\}}\ \text{U-CASE}$$

For if (2):

$$\dfrac{\Delta;\Phi_a;\Omega' \vdash_0^0 \text{celim } (f\ ()\ []\ []\ []\ []\ []\ []\ [])\ w\ m\ (i+1)\ l_w\ p : \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\} \quad \Delta;\Phi_a;\Omega' \vdash_0^0 \text{updt } p\ m\ 0 : \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\} \quad \vdash_0^0 a == b : \text{bool}}{\Delta;\Phi_a;\Omega' \vdash_0^0 \text{if } (a == b) \text{ then celim } (f\ ()\ []\ []\ []\ []\ []\ []\ [])\ w\ m\ (i+1)\ l_w\ p \text{ else updt } p\ m\ 0 : \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\}}\ \text{U-CASE}$$

By applying U-app, U-Iapp, U-celim, we get the type for $\text{celim } (f\ ()\ []\ []\ []\ []\ []\ []\ [])\ w\ m\ (i+1)\ l_w\ p$ is $\{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\}$. Along with the constraint $I+1 < Q < N \wedge M+I+1 < N$.

For the other branch, the term $\text{updt } p\ m\ 0$. We use U-X and subtyping rule S-A-MONAD.

$$\dfrac{\Delta;\Phi_a;\Omega' \vdash_0^0 \text{updt } p\ m\ 0 : \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\} \quad \Delta;\Phi_a \vDash \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\} \sqsubseteq \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\} \quad \Delta \vDash 0 \le 0 \quad \Delta \vDash 0 \le 0}{\Delta;\Phi_a;\Omega' \vdash_0^0 \text{updt } p\ m\ 0 : \{\gamma_3 \to \{M\}\}\ \overset{\text{exec}(c_u,(Q-I-1)*r+c_u)}{\exists\_.\text{int}}\ \{\gamma_3 \to \{M\}\}}\ \text{U-EXEC}$$

For if (1), it is similar by using U-exec and subtyping rule S-A-MONAD to make its type consistent with if (2).

Relational type

There are two cases we need to consider for relational types. We first apply some simple relational rules R-ILAM and R-ABS. Subscription 1 and 2 are used for the two runs. We apply R-CIMPL to introduce the constraints into the constraint environment Φ_a.

Set Z as the body of the function helper. where $C = (I < Q < N \wedge M + I < N)$ and $\tau = \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}$.

$$\overset{\text{diff}((Q-1-min(MIN(\beta_2 \cap [I,\infty)),Q-1))*r)}{}$$

To show the type of part Z, we will use the rule R-SPLIT first.

$$\frac{\Sigma;\Delta;\Phi_a \wedge C \wedge I \notin \beta_2;\Gamma \vdash Z \ominus Z \lesssim 0 : \tau \ldots (Case\,1) \quad \Sigma;\Delta;\Phi_a \wedge C \wedge I \in \beta_2;\Gamma \vdash Z \ominus Z \lesssim 0 : \tau \ldots (Case\,2)}{\Sigma;\Delta;\Phi_a \wedge C;\Gamma \vdash Z \ominus Z \lesssim 0 : \tau}\ \text{R-SPLIT}$$

Case 1 $I \notin \beta_2$

Because $I \notin \beta_2 \Rightarrow b_1 = b_2$. We know $a_1 = a_2, m_1 = m_2$ and $i_1 = i_2$. The two runs will go to the same path,, in the rest part of the proof, we think $\Phi'_a = \Phi_a \wedge (I < Q < N \wedge M + I < N) \wedge i \notin \beta_2$. After two reads, we will reach a if conditional.

R-READ

$$\frac{\Delta;\Phi'_a;\Gamma \vdash a \ominus a \lesssim 0 : \text{Array}_\gamma[n]\ \tau \qquad \Delta;\Phi'_a;\Gamma \vdash k \ominus k \lesssim 0 : \text{int}[i] \qquad \Delta;\Phi'_a \models i \le n \qquad \Delta;\vdash \text{empty wf}}{\Delta;\Phi'_a;\Gamma \vdash \text{read}\ S\ (m+i) \ominus \text{read}\ S\ (m+i) \lesssim 0 : \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \overset{\text{diff}(0)}{\exists\_.\text{int}}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}}$$

Set R as the outer if if$(i + 1 == l_w)$then (1) else (2).

Set V as let b =read W i in R.

R-LET

$$\Delta;\Phi_a \vdash i+1 == l_w \ominus i+1 == l_w \lesssim 0 : \text{unit} + \text{unit}$$

$$\Delta;\Phi_a;\Gamma \vdash (1) \ominus (1) \lesssim 0 :$$
$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}$$

$$\Delta;\Phi_a;\Gamma \vdash (2) \ominus (2) \lesssim 0 :$$
$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}$$

R-CASE

$$\Delta;\Phi_a;\Gamma \vdash R \ominus R \lesssim 0 :$$
$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}$$

$$\Delta;\Phi_a;\Gamma \vdash \text{read } W\ i \ominus \text{read } W\ i \lesssim 0 : \{\gamma_2 \to \beta_2\}\ \exists\_ :^{\text{diff}(0)} \text{int}\ \{\gamma_2 \to \beta_2\} \qquad P_2 = \text{empty}$$

$$\Delta;\Phi_a;\Gamma \vdash V \ominus V \lesssim 0 :$$
$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}$$

$$\Delta;\Phi_a;\Gamma \vdash \text{read } S\ (m+i) \ominus \text{read } S\ (m+i) \lesssim 0 : \{\gamma_1 \to \emptyset\}\ \exists\_ :^{\text{diff}(0)} \text{int}\ \{\gamma_1 \to \emptyset\}$$

$$P_2 = \text{empty}$$

$$\Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 :$$
$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}$$

(a) The relative cost for (1) is 0, because the cost of update is the same. We use sub-typing rule S-RM to make the types of two branches consistent.

(b)The relative cost of (2) depends on its first branch , which is a recursive call of the f function. We know that the relative cost is $(Q-1-min(MIN(\beta_2 \cap [I,\infty)), Q-1)) * r$ by using relational typing rules R-APP and R-Iapp . Because we already know $i \notin \beta$, it is easy to infer that $MIN(\beta_2 \cap [I,\infty)) = MIN(\beta_2 \cap [I+1,\infty))$.

Case 2 $I \in \beta_2$.

When $i \in \beta_2$, b_1 may not be the same as b_2, which means the two runs may go to different branches when reaching the if conditional. So we will switch to unary typing. From R-S, we know the relative cost of helper is $(Q-I-1)*r$, the type is still $\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}^{\text{diff}((Q-1-min(MIN(\beta_2 \cap [I,\infty)),Q-1))*r)}$. Because $i \in \beta_2$, $(Q-1-min(MIN(\beta_2 \cap [I,\infty)),Q-1)) = (Q-I-1)$.

We get the unary type of function helper from the context $|\Gamma|$ and apply U-IAPP and U-APP to get the unary type of the body:

$$\{\gamma_3 \to \{M\}\}\ \exists\_.\text{unit}\ \{\gamma_3 \to\ \{M\}\}^{\text{exec}(c_u,(Q-I-1)*r+c_u)}$$

By applying unary subtyping rule S-UM, we know the following unary typing used in R-S.

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash_0^0 (1) : \{\gamma_3 \to \{M\}\}\ \exists\_.\text{unit}\ \{\gamma_3 \to \{M\}\}^{\text{exec}(c_u,(Q-I-1)*r+c_u)} \quad \Delta;\Phi_a;|\Gamma|_2 \vdash_0^0 (1) : \{\gamma_3 \to \{M\}\}\ \exists\_.\text{unit}\ \{\gamma_3 \to \{M\}\}^{\text{exec}(c_u,(Q-I-1)*r+c_u)}}{\Delta;\Phi_a;\Gamma \vdash (1) \ominus (1) \lesssim 0 : U(\{\gamma_3 \to \{M\}\}\ \exists\_.\text{unit}\ \{\gamma_3 \to \{M\}\}^{\text{exec}(c_u,(Q-I-1)*r+c_u)}, \{\gamma_3 \to \{M\}\}\ \exists\_.\text{unit}\ \{\gamma_3 \to \{M\}\}^{\text{exec}(c_u,(Q-I-1)*r+c_u)})}\ \text{R-S}$$

$$\models U(\{\gamma_3 \to \{M\}\}\ \exists\_.\text{unit}\ \{\gamma_3 \to \{M\}\}^{\text{exec}(c_u,(Q-I-1)*r+c_u)}, \{\gamma_3 \to \{M\}\}\ \exists\_.\text{unit}\ \{\gamma_3 \to \{M\}\}^{\text{exec}(c_u,(Q-I-1)*r+c_u)}) \sqsubseteq$$

$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\}\ \exists\_.U(\text{unit},\text{unit})\ \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}^{\text{diff}((Q-I-1)*r)}$$

By subtyping rule S-R-UNIT, we get the relational type.

NSS function

$$\mathsf{NSS} = \mathsf{fix}\ F\ (S).\ \lambda W.\lambda m.\lambda l_s.\lambda l_w.\lambda P.$$

$$
\begin{array}{l}
\quad \mathsf{if}\ (m + l_w) \leq l_s\ \mathsf{then} \\
\quad\quad \mathsf{let}\ \{\_\} = (\mathsf{celim\ search}\ S\, W\, m\, 0\, l_s\, l_w\, P)\ \mathsf{in} \\
\quad\quad F(S)\, W\, (m+1)\, l_s\, l_w\, P \\
\quad \mathsf{else} \\
\quad\quad \mathsf{return}\ () \\
\end{array}
$$

$$
\begin{array}{l}
\vdash \mathsf{NSS} \ominus \mathsf{NSS} \lesssim 0 : \\
\end{array}
\begin{array}{l}
\forall M. \forall \gamma_1, \gamma_2, \gamma_3 : \mathbb{L}. \forall Q, N : \mathbb{N}. \\
(Q \leq N \wedge M + Q \leq N) \subset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \\
\rightarrow \mathsf{Array}_{\gamma_2}[Q]\ \mathrm{int} \rightarrow \mathrm{int}[M] \rightarrow \mathrm{int}[N] \rightarrow \mathrm{int}[Q] \rightarrow \mathsf{Array}_{\gamma_3}[N]\ \mathrm{int} \rightarrow \\
\mathrm{diff}((Q-1-min(MIN(\beta_2 \cap [0,\infty), Q-1)) * r * (N-M-Q)) \\
\{P, \gamma_3 \rightarrow \beta_3\}\ \exists \_. \mathrm{unit}\ \{P, \gamma_3 \rightarrow \beta_3 \cup [M, N]\}
\end{array}
$$

where $P = \gamma_1 \rightarrow \emptyset, \gamma_2 \rightarrow \beta_2$. For the type of NSS, we first apply R-FIX. Then we apply R-ILAM and R-ABS multiple times. Then after we introduce the constraints $(Q \leq N \wedge M+Q \leq N)$ into the constraint environment Φ_a, the cost comes from the first branch.

Set K as $(\mathsf{celim\ helper}\ S\, W\, m\, 0\, l_s\, l_w\, P)$, Set Q as $F(S)\, W\, (m+1)\, l_s\, l_w\, P$.

Set X as $\mathsf{let}\ \_ = (\mathsf{celim\ helper}\ S\, W\, m\, 0\, l_s\, l_w\, P)\ \mathsf{in}$

$$F(S)\, W\, (m+1)\, l_s\, l_w\, P.$$

R-LET

$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\} = \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\} * \mathsf{empty}$$

$$\Delta; \Phi_a; \Gamma \vdash K \ominus K \lesssim 0 :$$
$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\} \; \exists\_.\mathrm{unit} \; \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\}^{\mathrm{diff}((Q-1-min(MIN(\beta_2 \cap [0,\infty), Q-1)) * r)}$$

$$\Delta; \Phi_a; \Gamma \vdash Q \ominus Q \lesssim 0 :$$
$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\}\} \; \exists\_.\mathrm{unit} \; \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup \{M\} \cup [M+1, N]\}^{\mathrm{diff}((Q-1-min(MIN(\beta_2 \cap [0,\infty), Q-1)) * r * (N-(M+1)-Q))}$$

$$\Delta; \Phi_a; \Gamma \vdash X \ominus X \lesssim 0 :$$
$$\{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3\} \; \exists\_.\mathrm{unit} \; \{\gamma_1 \to \emptyset, \gamma_2 \to \beta_2, \gamma_3 \to \beta_3 \cup [M, N]\}^{\mathrm{diff}((Q-1-min(MIN(\beta_2 \cap [0,\infty), Q-1)) * r * (N-M-Q))}$$

A.6.3 MergeSort

$$
\begin{array}{l}
\mathsf{copy} = \mathsf{fix}\ cp\,(\_).\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda x.\lambda y.\lambda l.\lambda u. \\
\quad \mathsf{if}\ l \le u\ \mathsf{then} \\
\quad\quad \mathsf{let}\ \{a\} = \big(\,\mathsf{read}\ x\ l\,\big)\,\mathsf{in} \\
\quad\quad \mathsf{let}\{\_\} = \big(\,\mathsf{updt}\ y\ l\ a\,\big)\,\mathsf{in} \\
\quad\quad \big(\mathsf{celim}(cp()\,[]\,[]\,[]\,[]\,[])\ x\ y\ (l+1)\ u\big) \\
\quad \mathsf{else}\ \ \mathsf{return}()
\end{array}
$$

Type 1: copy $S \cap [l,u] = \emptyset$

$$
\begin{array}{rl}
 & \forall n, L, U, \gamma_1, \gamma_2, S, S'.\ L \le U \le n \wedge S' \cap [L,U] = \emptyset \supset \\
\vdash \mathsf{copy} \ominus \mathsf{copy} \lesssim 0 : & \mathsf{Array}_{\gamma_1}[n]\ U(\mathrm{int},\mathrm{int}) \to \mathsf{Array}_{\gamma_2}[n]\ U(\mathrm{int},\mathrm{int}) \to \mathrm{int}[L] \\
 & \to \mathrm{int}[U] \to \{\gamma_1 \to S, \gamma_2 \to S'\}\ \overset{\mathrm{diff}(0)}{\exists}\_.\mathrm{unit}\ \{\gamma_1 \to S, \gamma_2 \to \mathbb{N}\}
\end{array}
$$

Type 2: copy $S \cap [l,u] \neq \emptyset$

$$
\begin{array}{rl}
 & \forall n, L, U, \gamma_1, \gamma_2, S.\ L \le U \le n \supset \mathsf{Array}_{\gamma_1}[n]\ U(\mathrm{int},\mathrm{int}) \\
\vdash \mathsf{copy} \ominus \mathsf{copy} \lesssim 0 : & \to \mathsf{Array}_{\gamma_2}[n]\ U(\mathrm{int},\mathrm{int}) \to \mathrm{int}[L] \to \mathrm{int}[U] \\
 & \to \{\gamma_1 \to S, \gamma_2 \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists}\_.\mathrm{unit}\ \{\gamma_1 \to S \cup [L,U], \gamma_2 \to \mathbb{N}\}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{merge}_{\mathsf{lp}} = \mathsf{fix}\ \mathsf{mg}\,(\_).\lambda.(k : \mathrm{int}[K]).\lambda a : \mathsf{Array}_{\gamma_1}[n]\ \mathrm{int}. \\
\quad \lambda l_s : \mathrm{int}[i], \lambda l_e : \mathrm{int}[j], \lambda r_s : \mathrm{int}[m], \lambda r_e : \mathrm{int}[u].\lambda b : \mathsf{Array}_{\gamma_2}[n]\ \mathrm{int}. \\
\quad \mathsf{if}\ l_s < l_e\ \mathsf{then}
\end{array}
$$

```
if r_s < r_e then
  let {x} = (read a l_s) in
    let {y} = (read a r_s) in
      if x < y then
        (let {_} = (updt b k x) in
          ((celim mg () (k + 1)) a (l_s + 1) l_e r_s r_e b))
      else
        (let {_} = (updt b k y) in
          ((celim mg () (k + 1)) a l_s l_e (r_s + 1) r_e, b))
  else
    (let{x} = (read a l_s) in
      let{_} = (updt b k x) in
        ((celim mg () (k + 1)) a (l_s + 1) l_e r_s r_e b))
else
  if r_s < r_e
    (let{x} = (read a r_s) in
      let{_} = (updt b k x) in
        ((celim mg () (k + 1)) a l_s l_e (r_s + 1) r_e b))
  else (return())
```

$$\begin{array}{lll} & & \forall n, l_s, l_e, r_s, r_e, K, \gamma, \gamma'. l_s \le l_e \le r_s \le r_e \le n \supset \text{unit} \to \text{int}[K] \\ \vdash \quad \mathsf{merge_{lp}} \quad : & & \to \mathsf{Array}_{\gamma}[n]\ \text{int} \to \text{int}[l_s] \to \text{int}[l_e] \to \text{int}[r_s] \to \text{int}[r_e] \\ & & \to \mathsf{Array}_{\gamma'}[n]\ \text{int} \to \{\gamma \to \emptyset, \gamma' \to [K, r_e]\}^{\text{exec}(U,L)} \exists\_.\text{unit}\ \{\gamma, \gamma'\} \end{array}$$

$L = (c_r + c_r + c_u) * min(l_e - l_s, r_e - r_s) + (c_r + c_u) * max(l_e - l_r, r_e - e_s)$

$U = (c_r + c_r + c_u) * ((l_e - l_s) + (r_e - r_s))$

In $\mathsf{merge_{lp}}$, we have 4 positions in the input array, l_s, l_e indicating the left part of the array, r_s, r_e for right part. we need to merge the two part, which means total $(l_e - l_s)$ elements for left part of the input array and $(r_e - r_s)$ elements for right part. In the body of $merge_{lp}$, for every element to be merged and added to the buffer array, there are two possible different branches: in one branch (if $l_s < l_e \wedge r_s < r_e$, which means we still have elements waiting to be merged in both left and right parts), the cost will be $c_r + c_r + c_u$, c_r for the cost of read from the input array, c_u for the cost of updating the buffer array. in the other branch, (either $l_s < l_e$ or $r_s < r_e$, we need to only merge one part into buffer array), the cost is $c_r + c_u$.

For the L, the minimal possible cost is that we run out the shorter part ($min(l_e - l_s, r_e - r_s)$) with the cost of $c_r + c_r + c_u$ first and then merge the rest part with the cost of $c_r + c_u$. For the maximal cost, merge all the elements with the cost of $c_r + c_r + c_u$. If we consider the cost for update as 1 unit cost and cost for read is 1 unit cost, $c_r + c_r + c_u = 3, c_r + c_u = 2$.

Type 1: mergeLp $S \cap [K, r_e] = \emptyset$

$$
\begin{aligned}
& \forall n, l_s, l_e, r_s, r_e, K, \gamma, \gamma'. \\
& l_s \le l_e \le r_s \le r_e \le n \wedge S \cap [K, r_e] = \emptyset \\
\vdash \mathsf{merge_{lp}} \ominus \mathsf{merge_{lp}} \lesssim 0 : \quad & \supset \mathrm{unit} \to \mathrm{int}[K] \to \mathsf{Array}_{\gamma}[n]\ U(\mathrm{int}, \mathrm{int}) \to \mathrm{int}[l_s] \\
& \to \mathrm{int}[l_e] \to \mathrm{int}[r_s] \to \mathrm{int}[r_e] \to \mathsf{Array}_{\gamma'}[n]\ U(\mathrm{int}, \mathrm{int}) \\
& \to \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \overset{\mathrm{diff}(0)}{\exists\_}.\mathrm{unit}\ \{\gamma \to S, \gamma' \to \mathbb{N} \setminus [l, u]\}
\end{aligned}
$$

Because in the range $[K, r_e]$, all the values in the two arrays are exact the same, the two programs will go to the same branch for every step.

In the read rule, i equals to either l_s or r_s.

$$
\frac{\Delta; \Phi_a; \Gamma \vdash a \ominus a \lesssim 0 : \mathsf{Array}_{\gamma}[n]\ \tau \qquad \Delta; \Phi_a; \Gamma \vdash l_s[r_s] \ominus l_s[r_s] \lesssim 0 : \mathrm{int}[i] \qquad \Delta; \Phi_a \models i \le n \qquad \Delta; \Phi_a \vdash \mathsf{empty\ wf}}{\Delta; \Phi_a; \Gamma \vdash \mathsf{read}\ a\ l_s[r_s] \ominus \mathsf{read}\ a\ l_s[r_s] \lesssim 0 : \{\gamma \to S\}\ \overset{\mathrm{diff}(0)}{\exists\_}.\tau\ \{\gamma \to S\}}\ \text{R-R}
$$

and

$$
\frac{\Delta; \Phi_a; \Gamma \vdash b \ominus b \lesssim 0 : \mathsf{Array}_{\gamma'}[n]\ \tau \qquad \Delta; \Phi_a; \Gamma \vdash k \ominus k \lesssim 0 : \mathrm{int}[i] \qquad \Delta; \Phi_a; \Gamma, x : \tau \vdash x \ominus x \lesssim 0 : \mathrm{int} \qquad \Delta; \Phi_a \models i \le n \qquad \Delta; \Phi_a \vdash \mathsf{empty\ wf}}{\Delta; \Phi_a; \Gamma \vdash \mathsf{updt}\ b\ k\ x[y] \ominus \mathsf{updt}\ b\ k\ x[y] \lesssim 0 : \{\gamma' \to N\}\ \overset{\mathrm{diff}(0)}{\exists\_} : \mathrm{unit}\ \{\gamma' \to N \cup [i]\}}\ \text{R-U}
$$

Using R-Case for two times, we have four similar cases, we will have a look at one of them.

$$
\Delta; \Phi_a; \Gamma \vdash (\mathsf{celim}(\mathsf{mg}\ ())\ k + 1\ a\ (l_s + 1)\ l_e\ r_s\ r_s \ominus (\mathsf{celim}(\mathsf{mg}\ ())\ k + 1\ a\ (l_s + 1)\ l_e\ r_s\ r_s \lesssim 0 :
$$

Its type is $\{\gamma \to S, \gamma' \to N\}\ \overset{\mathrm{diff}(0)}{\exists\_} : \mathrm{unit}\ \{\gamma \to S, \gamma' \to N\}$. By applying $R-Celim$ to check the constraint and then applying $R-Iapp$ and $R-App$ for several times.

Type 2: mergeLp $S \cap [K, r_e] \neq \emptyset$

Because if $S \cap [K, r_e] \neq \emptyset$, $merge_{lp}$ in the two runs are likely to go into different branches at any step, We will use R-S to handle this cases because the two runs are likely to go to different branches in every step.

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|_1 \vdash_0^0 \mathsf{merge_{lp}}\ ()\ k\ a\ l_s\ l_e\ r_s\ r_e\ b : \{\gamma \to \emptyset, \gamma' \to \overset{\mathrm{exec}(L,U)}{[K, r_e]}\}\ \exists\_ : \mathrm{unit}\ \{\gamma, \gamma'\} \\ \Delta;\Phi_a;|\Gamma|_2 \vdash_0^0 \mathsf{merge_{lp}}\ ()\ k\ a\ l_s\ l_e\ r_s\ r_e\ b : \{\gamma \to \emptyset, \gamma' \to \overset{\mathrm{exec}(L,U)}{[K, r_e]}\}\ \exists\_ : \mathrm{unit}\ \{\gamma, \gamma'\}\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash \mathsf{merge_{lp}}\ ()\ k\ a\ l_s\ l_e\ r_s\ r_e\ b \ominus \mathsf{merge_{lp}}\ ()\ k\ a\ l_s\ l_e\ r_s\ r_e\ b \lesssim 0 : \\ U(\{\gamma \to \emptyset, \gamma' \to \overset{\mathrm{exec}(L,U)}{[K, r_e]}\}\ \exists\_ : \mathrm{unit}\ \{\gamma, \gamma'\}, \{\gamma \to \emptyset, \gamma' \to \overset{\mathrm{exec}(L,U)}{[K, r_e]}\}\ \exists\_ : \mathrm{unit}\ \{\gamma, \gamma'\})\end{array}}\ \text{R-S}$$

and we know from S-RUM.

$$\models U(\{\gamma, \gamma' \to \overset{\mathrm{exec}(L,U)}{[K, r_e]}\}\ \exists\_ : \mathrm{unit}\ \{\gamma, \gamma'\}, \{\gamma, \gamma' \to \overset{\mathrm{exec}(L,U)}{[L, U]}\}\ \exists\_ : \mathrm{unit}\ \{\gamma, \gamma'\}) \sqsubseteq$$
$$\{\gamma \to S, \gamma' \to S'\}\ \exists\_ : \overset{\mathrm{diff}(U-L)}{\mathrm{unit}}\ \{\gamma \to S, \gamma' \to S' \cup [K, r_e]\}$$

$$\begin{array}{rl}\vdash \mathsf{merge_{lp}} \ominus \mathsf{merge_{lp}} \lesssim 0 : & \forall n, l_s, l_e, r_s, r_e, K, \gamma, \gamma'. l_s \leq l_e \leq r_s \leq r_e \leq n \supset \mathrm{unit} \\ & \to \mathrm{int}[K] \to \mathsf{Array}_\gamma[n]\ U(\mathrm{int}, \mathrm{int}) \to \mathrm{int}[l_s] \\ & \to \mathrm{int}[l_e] \to \mathrm{int}[r_s] \to \mathrm{int}[r_e] \to \mathsf{Array}_{\gamma'}[n]\ U(\mathrm{int}, \mathrm{int}) \\ & \to \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_. \overset{\mathrm{diff}((U-L))}{\mathrm{unit}}\ \{\gamma \to S, \gamma' \to \mathbb{N} \cup [K, r_e]\}\end{array}$$

Function merge

$$\mathsf{merge} = \lambda a.\lambda b.\lambda l.\lambda u.\lambda mid.$$
$$\mathsf{let}\ \{\_\} = \left(celim\ (merge_{lp}\ ()\ l)\ a\ l\ mid\ mid+1\ u\ b\right)\ \mathsf{in}$$

$$(celim\ \text{copy}\ l\ )\ a\ b\ u\Big).$$

Type 1: merge $S \cap [K, r_e] = \emptyset \to S \cap [l, u] = \emptyset$

$$\begin{aligned}\vdash \text{merge} \ominus \text{merge} \lesssim 0 : \quad & \forall n, l, m, u, \gamma, \gamma', S. l \le m \le u \le n \wedge S \cap [l, u] = \emptyset \\ & \supset \text{Array}_{\gamma}[n]\ U(\text{int}, \text{int}) \to \text{Array}_{\gamma'}[n]\ U(\text{int}, \text{int}) \\ & \to \text{int}[l] \to \text{int}[m] \to \text{int}[u] \\ & \to \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \overset{\text{diff}(0)}{\exists}\_.\text{unit}\ \{\gamma \to S, \gamma' \to \mathbb{N}\}\end{aligned}$$

R-LET

$$\frac{\begin{array}{c}\vdash \text{celim merge}_{\text{lp}}\ l\ a\ l\ m\ (m+1)\ u\ b \ominus \text{celim merge}_{\text{lp}}\ l\ a\ l\ m\ (m+1)\ u\ b \lesssim 0 : \\ \overset{\text{diff}(0)}{\{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma \to S, \gamma' \to \mathbb{N} \setminus [l, u]\}} \\ \Delta; \Phi_a; \Gamma \vdash \text{celim copy}\ l\ a\ b\ u \ominus \text{celim copy}\ l\ a\ b\ u \lesssim 0 : \\ \overset{\text{diff}(0)}{\{\gamma \to S, \gamma' \to \mathbb{N} \setminus [l, u]\}\ \exists\_.\text{unit}\ \{\gamma \to S, \gamma' \to \mathbb{N} \setminus [l, u]\}}\end{array}}{\begin{array}{c}\Delta; \Phi_a; \Gamma \vdash \text{merge}\ a\ b\ l\ m\ u \ominus \text{merge}\ a\ b\ l\ m\ u \lesssim 0 : \\ \overset{\text{diff}(0)}{\{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma \to S, \gamma' \to \mathbb{N}\}}\end{array}}$$

we know: $\models \mathbb{N}/[l, u] \sqsubseteq \mathbb{N}$, By S-RM, we can get the post condition of $merge$ to be

$$\{\gamma \to S, \gamma' \to \mathbb{N}\}$$

Type 2:merge $S \cap [K, r_e] \neq \emptyset \to S \cap [l, u] \neq \emptyset$

$$\begin{aligned}\vdash \text{merge} \ominus \text{merge} \lesssim 0 : \quad & \forall n, l, m, u, \gamma, \gamma', S. l \le m \le u \le n \supset \text{Array}_{\gamma}[n]\ U(\text{int}, \text{int}) \\ & \to \text{Array}_{\gamma'}[n]\ U(\text{int}, \text{int}) \to \text{int}[l] \to \text{int}[m] \to \text{int}[u] \\ & \to \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \overset{\text{diff}(max(m-l, u-(m+1)))}{\exists}\_.\text{unit}\ \{\gamma \to S \cup [l, u], \gamma' \to \mathbb{N}\}\end{aligned}$$

R-LET

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \mathsf{celim}\ \mathsf{merge}_{\mathsf{lp}}\ l\ a\ l\ m\ (m+1)\ u\ b \ominus \mathsf{celim}\ \mathsf{merge}_{\mathsf{lp}}\ l\ a\ l\ m\ (m+1)\ u\ b \lesssim 0 : \\ \overset{\mathrm{diff}(max(m-l,u-(m+1))}{\{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_.\mathrm{unit}\ \{\gamma \to S, \gamma' \to \mathbb{N} \cup [l,u]\}} \\ \\ \Delta;\Phi_a;\Gamma \vdash \mathsf{celim}\ copy\ l\ a\ b\ u \ominus \mathsf{celim}\ copy\ l\ a\ b\ u \lesssim 0 : \\ \overset{\mathrm{diff}(0)}{\{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_.\mathrm{unit}\ \{\gamma \to S \cup [l,u], \gamma' \to \mathbb{N}\}}\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash \mathsf{merge}\ a\ b\ l\ m\ u \ominus \mathsf{merge}\ a\ b\ l\ m\ u \lesssim 0 : \\ \overset{\mathrm{diff}(max(m-l,u-(m+1))}{\{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_.\mathrm{unit}\ \{\gamma \to S \cup [l,u], \gamma' \to \mathbb{N}\}}\end{array}}$$

msort

$$\begin{array}{l}\mathsf{msort} = \mathsf{fix}\ \mathsf{sort}\ (a : \mathsf{Array}_{\gamma_1}[n]\ \tau)\ b : \mathsf{Array}_{\gamma_2}[n]\ \tau\ .\lambda l.\lambda u. \\ \quad \mathsf{if}\ (l \geq u)\ \mathsf{then} \\ \quad \big(\mathsf{return}()\big) \\ \quad \mathsf{else}\ \Big(\\ \quad \mathsf{let}\ \_ = \big(celim\ (sort)\ a\ b\ l\ (l + \lfloor (u-l)/2 \rfloor)\big)\ \mathsf{in} \\ \quad \mathsf{let}\ \_ = \big(celim\ (sort)\ a\ b\ (l + \lfloor (u-l)/2 \rfloor + 1)\ u\big)\ \mathsf{in} \\ \quad \big(celim\ merge\ a\ b\ l\ (l + \lfloor (u-l)/2 \rfloor)\ u\big)\Big)\end{array}$$

$$\begin{array}{ll}\vdash \mathsf{msort} : & \forall n, l, u, \gamma, \gamma'. l \leq u \leq n \supset \mathsf{Array}_{\gamma}[n]\ \mathrm{int} \\ & \to \mathrm{int}[l] \to \mathrm{int}[u] \to \mathsf{Array}_{\gamma'}[n]\ \mathrm{int} \to \overset{\mathrm{exec}(0,(u-l)*log(u-l))}{\{\gamma \to [l,u], \gamma' \to [l,u]\}\ \exists\_.\mathrm{unit}\ \{\gamma, \gamma'\}}\end{array}$$

Let us call M the following term: $\mathsf{celim}\ (merge\ a\ b\ l\ (l + \lfloor (u-l)/2 \rfloor)\ u$, and call $S1$ the following term: $\mathsf{celim}\ (sort)\ a\ b\ l\ l + \lfloor (u-l)/2 \rfloor$, and call $S2$ the following term: $\mathsf{celim}\ (sort)\ a\ b\ l + \lfloor (u-l)/2 \rfloor + 1\ u$, and call Z the following term: $\mathsf{let\_} = S1\ \mathsf{in}\ \mathsf{let}\ \_ =$

$S2$ in M, and set V as: if $l = u$, then (return ())else Z .

Type 1: msort $S \cap [l,u] = \emptyset$

$$\vdash \mathsf{msort} \ominus \mathsf{msort} \lesssim 0 : \begin{array}{l} \forall n, l, u, \gamma, \gamma' : \mathbb{N}, S. l \le u \le n \wedge S \cap [l,u] = \emptyset \\ \supset \mathsf{Array}_{\gamma}[n]\ U(\mathrm{int},\mathrm{int}) \rightarrow \mathrm{int}[l] \\ \rightarrow \mathrm{int}[u] \rightarrow \mathsf{Array}_{\gamma'}[n]\ U(\mathrm{int},\mathrm{int}) \\ \rightarrow \{\gamma \rightarrow S, \gamma' \rightarrow \mathbb{N}\}\ \exists\_.\overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \rightarrow S, \gamma' \rightarrow \mathbb{N}\} \end{array}$$

$$\dfrac{\begin{array}{c} \Delta;\Phi_a;\Gamma \vdash S2 \ominus S2 \lesssim 0 : \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\} \\ \Delta;\Phi_a;\Gamma \vdash M \ominus M \lesssim 0 : \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\} \\ P_2 = \mathsf{empty} \end{array}}{\begin{array}{c} \Delta;\Phi_a;\Gamma \vdash let\_ = S2\ in\ M \ominus let\_ = S2\ in M \lesssim 0 : \\ \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\} \end{array}}\ \text{R-LET}$$

$$\dfrac{\begin{array}{c} \Delta;\Phi_a;\Gamma \vdash S1 \ominus S1 \lesssim 0 : \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\} \\ P_2 = \mathsf{empty} \end{array}}{\Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}}\ \text{R-LET}$$

$$\dfrac{\begin{array}{c} \Delta;\Phi_a;\Gamma \vdash l = u \ominus l = u \lesssim 0 : \mathrm{unit} + \mathrm{unit} \\ \Delta;\Phi_a;\Gamma \vdash \ \mathsf{return}\ () \ominus \mathsf{return}\ () \lesssim 0 : \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\} \end{array}}{\Delta;\Phi_a;\Gamma \vdash V \ominus V \lesssim 0 : \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \rightarrow S, \gamma \rightarrow \mathbb{N}\}}$$

Type 2: msort $S \cap [l,u] \neq \emptyset$

$$\vdash \mathsf{msort} \ominus \mathsf{msort} \lesssim 0 : \begin{array}{l} \forall n, l, u, \gamma, \gamma' : \mathbb{N}, S. l \le u \le n \supset \mathsf{Array}_{\gamma}[n]\ U(\mathrm{int},\mathrm{int}) \\ \rightarrow \mathrm{int}[l] \rightarrow \mathrm{int}[u] \rightarrow \mathsf{Array}_{\gamma'}[n]\ U(\mathrm{int},\mathrm{int}) \\ \rightarrow \{\gamma \rightarrow S, \gamma' \rightarrow \mathbb{N}\}\ \exists\_.\overset{\mathrm{diff}(Q(u-l+1,|S\cap[l,u]|))}{\mathrm{unit}}\ \{\gamma \rightarrow S \cup [l,u], \gamma' \rightarrow \mathbb{N}\} \end{array}$$

where $Q(n,\alpha) = \Sigma_{i=0}^{H} h(\lceil 2^{i-1} \rceil).min(\alpha, 2^{H-i})$, $H = \lceil log_2(n) \rceil$

R-CASE
R-EXEC
S-RM

$$\frac{\dfrac{}{\models \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S, \gamma' \to \mathbb{N}\}}\ \mathrm{diff}(0)}{\sqsubseteq \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S \cup [l,u], \gamma' \to \mathbb{N}\}}\ \mathrm{diff}(Q(u-l+1, |S \cap [l,u]|))$$

$$\frac{\dfrac{\dfrac{}{\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ () \ominus \mathsf{return}\ () \lesssim 0 :}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ () \ominus \mathsf{return}\ () \lesssim 0 : \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S \cup [l,u], \gamma' \to \mathbb{N}\}}\ \mathrm{diff}(Q(u-l+1, |S \cap [l,u]|))}$$

$$\frac{\dfrac{\dfrac{}{\Delta;\Phi_a;\Gamma \vdash l = u \ominus l = u \lesssim 0 : \mathrm{unit} + \mathrm{unit}}}{\Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S \cup [l,u], \gamma' \to \mathbb{N}\}}\ \mathrm{diff}(Q(u-l+1, |S \cap [l,u]|))}{\Delta;\Phi_a;\Gamma \vdash V \ominus V \lesssim 0 : \{\gamma \to S, \gamma' \to \mathbb{N}\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S \cup [l,u], \gamma' \to \mathbb{N}\}}\ \mathrm{diff}(Q(u-l+1, |S \cap [l,u]|))$$

R-LET
R-LET

$$\frac{\dfrac{\dfrac{P_2 = \mathsf{empty}}{\Delta;\Phi_a;\Gamma \vdash S2 \ominus S2 \lesssim 0 : \{\gamma \to S \cup [l, \lfloor (u-l)/2 \rfloor]\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S \cup [l,u]\}}\ \mathrm{diff}(Q(u-l-\lfloor (u-l)/2 \rfloor, |S \cap [l+\lfloor (u-l)/2 \rfloor + 1, u]|))}{\Delta;\Phi_a;\Gamma \vdash M \ominus M \lesssim 0 : \{\gamma \to S \cup [l,u]\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S \cup [l,u],\}}\ \mathrm{diff}(max((l+\lfloor (u-l)/2 \rfloor) - l, u - ((l+\lfloor (u-l)/2 \rfloor)+1))}{\Delta;\Phi_a;\Gamma \vdash let\_ = S2\ in\ M \ominus let\_ = S2\ in M \lesssim 0 :}$$

$$\frac{\dfrac{\dfrac{P_2 = \mathsf{empty}}{\Delta;\Phi_a;\Gamma \vdash S1 \ominus S1 \lesssim 0 : \{\gamma \to S, \gamma \to \mathbb{N}\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S \cup [l, \lfloor (u-l)/2 \rfloor], \gamma \to \mathbb{N}\}}\ \mathrm{diff}(Q(\lfloor (u-l)/2 \rfloor + 1, |S \cap [l, l+\lfloor (u-l)/2 \rfloor]|))}{\Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \to S, \gamma \to \mathbb{N}\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to S \cup [l,u], \gamma \to \mathbb{N}\}}\ \mathrm{diff}(Q(u-l+1, |S \cap [l,u]|))$$

If we simplify the relative cost for M, it is $max((\lfloor (u-l)/2 \rfloor), u-l-(\lfloor (u-l)/2 \rfloor)+1) =$

$\lceil \frac{u-l}{2} \rceil$. Let us look at the simplified version of msort:

$$\begin{aligned} msort(in,l,u,buf) &= msort(in,l,l+\lfloor \tfrac{u-l}{2} \rfloor, buf); \\ & \quad msort(in,l+\lfloor \tfrac{u-l}{2} \rfloor + 1,u,buf); \\ & \quad merge(in,l,l+\lfloor \tfrac{u-l}{2} \rfloor + 1,u,buf). \end{aligned}$$

To show : $Q(\lfloor \frac{u-l}{2} \rfloor + 1, |S \cap [l, l+\lfloor \frac{u-l}{2} \rfloor]|) +$

$Q(u-l-\lfloor \frac{u-l}{2} \rfloor, |S \cap [l+\lfloor \frac{u-l}{2} \rfloor + 1, u]|) + \lceil \frac{u-l}{2} \rceil$

$\leq Q(u-l+1, |S \cap [l,u]|)$

Let us simplify the inequality with n=l-u, $\alpha = |S \cap [l, l+\lfloor \frac{u-l}{2} \rfloor$ and $\beta = |S \cap [l+\lfloor \frac{u-l}{2} \rfloor + 1, u]|$:

$\lceil \frac{n}{2} \rceil$ + Q($\lfloor \frac{n}{2} \rfloor$,$\alpha$) + Q($\lceil \frac{n}{2} \rceil$,$\beta$) $\leq$ Q(n, $\alpha + \beta$), it is arithmetically tautology

$\alpha + \beta \neq 0$ because $S \cap [l,u] \neq \emptyset$

A.6.4 Insertion Sort

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{ISort}\ (\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda s.\lambda i.\lambda l_s. \\
\quad \mathsf{if}\ i < l_s\ \mathsf{then} \\
\quad\quad \mathsf{let}\ \{a\} = \mathsf{read}\ s\ i\ \ \mathsf{in} \\
\quad\quad \mathsf{let}\ \{b\} = \mathsf{celim}(insert()\ []\ []\ []\ []\ []\ []\ )\ s\ a\ 0\ i\ \mathsf{in} \\
\quad\quad\quad \mathsf{celim}(\mathsf{ISort}()\ []\ []\ []\ [])s(i+1)l_s \\
\quad \mathsf{else} \\
\quad\quad \mathsf{return}\ () \\
\end{array}
$$

$$
\vdash \quad \mathsf{ISort} \quad : \quad
\begin{array}{l}
\mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall N, I : \mathbb{N}.(I \leq N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \\
\mathrm{int}[I] \to \mathrm{int}[N] \to \ \overset{\mathrm{exec}(\frac{(N+1)*(N+2)-(I+1)*(I+2)}{2}, (2N+1)*(N+1)-(2I+3)*(I+1))}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}\}}
\end{array}
$$

where minimal cost obtained when the array is already sorted in the ascending order and the maximal cost corresponds to a sorted array in the descending order.

$$
\vdash \mathsf{ISort} \ominus \mathsf{ISort} \lesssim 0 : \quad
\begin{array}{l}
\mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall N, I : \mathbb{N}.\forall \beta_1.(I \leq N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \mathrm{int}[I] \\
\to \mathrm{int}[N] \to \{\gamma_1 \to \beta_1\}\ \exists\_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}\}
\end{array}
^{\mathrm{diff}(\frac{N*(N+1)-k*(k+1)}{2})}
$$

where $k = max(I, min(MIN(\beta_1), N))$.

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{insert}\ (\_).\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda s.\lambda a.\lambda i d x.\lambda i. \\
\quad \mathsf{let}\ \{b\} = \mathsf{read}\ s\ i d x\ \ \mathsf{in} \\
\quad\quad \mathsf{if}\ a \geq b\ \mathsf{then}
\end{array}
$$

$$\mathsf{celim}(insert\,()\,[]\,[]\,[]\,[]\,[])\;s\;a\;(idx+1)\;i$$

else

$$\mathsf{let}\ \_ = \mathsf{celim}(\mathsf{shift}\ ()\ []\ []\ []\ [])\;s\;idx\;(i-1)\ \mathsf{in}$$

$$\mathsf{updt}\ s\ idx\ a$$

$$\vdash \ \mathsf{insert}\ : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall N, A, IDX, I : \mathbb{N}.(IDX \le N \wedge I \le N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \\ \mathrm{int} \to \mathrm{int}[IDX] \to \mathrm{int}[I] \to \overset{\mathrm{exec}(I-IDX+1,2*(I-IDX)+2)}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\mathrm{unit}\ \{\gamma_1 \to \mathbb{N}\}} \end{array}$$

where the minimal cost consists of one update operation and (I-IDX) times read operations when the value 'a' inserted is already the biggest element in the sequence ranging from [idx, i]. the maximal cost consists of 2*(I-IDX) coming from shift and one read and update operation in addition when the inserted value 'a' is the smallest element in the sequence.

$$\vdash \mathsf{insert} \ominus \mathsf{insert} \lesssim 0 : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall N, A, IDX, I : \mathbb{N}.\forall \beta_1. \\ (IDX \le N \wedge I \le N \wedge \beta_1 \cap [IDX, I] = \emptyset) \\ \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \mathrm{int}[M] \to \mathrm{int}[N] \to \\ \{\gamma_1 \to \beta_1\}\ \overset{\mathrm{diff}(0)}{\exists\_.\mathrm{unit}\ \{\gamma_1 \to \beta_1\}} \end{array}$$

The relational cost of insert function is 0 when the premise $\beta_1 \cap [IDX, I] = \emptyset$ holds, which means in the range [IDX, I], the elements of arrays on the two runs are exactly the same. Considering we are inserting the same value 'a' (s[I]), it always goes into the

same path and hence incurs no relative cost.

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{shift}\ (\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda s.\lambda idx.\lambda i.\\
\quad \mathsf{if}\ idx \le i\ \mathsf{then}\\
\quad\quad \mathsf{let}\ \{c\} = \mathsf{read}\ s\ i\ \ \mathsf{in}\\
\quad\quad \mathsf{let}\ \{\_\} = \mathsf{updt}\ s\ i+1\ c\ \ \mathsf{in}\\
\quad\quad\quad \mathsf{celim}\ (\mathsf{shift}\ ()\ []\ []\ []\ []\ )\ s\ idx\ (i-1)\\
\quad \mathsf{else}\\
\quad\quad \mathsf{return}\ ()
\end{array}
$$

$$
\vdash \quad \mathsf{shift} \quad : \quad
\begin{array}{l}
\mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall N, IDX, I : \mathbb{N}.(IDX \le I \wedge I < N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \to \\
\mathrm{int}[IDX] \to \mathrm{int}[I] \to \{\gamma_1 \to \mathbb{N}\}\ \exists\_.\mathrm{unit}^{\mathrm{exec}(2*(I-IDX+1),2*(IDX-I+1)}\ \{\gamma_1 \to \mathbb{N}\}
\end{array}
$$

where cost 2 comes from one read and one update operation.

$$
\vdash \mathsf{shift} \ominus \mathsf{shift} \lesssim 0 : \quad
\begin{array}{l}
\mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall N, IDX, I : \mathbb{N}.(IDX \le I \wedge I < N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{int} \\
\to \mathrm{int}[IDX] \to \mathrm{int}[I] \to \{\gamma_1 \to \beta_1\}\ \exists\_.\mathrm{unit}^{\mathrm{diff}(0)}\ \{\gamma_1 \to \beta_1 \cup [ID]\}
\end{array}
$$

Let us look at the derivation of the relational type of ISort.

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{ISort}\ (\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda s.\lambda i.\lambda l_s.\\
\quad \mathsf{if}\ i < l_s\ \mathsf{then}\\
\quad\quad \mathsf{let}\ \{a\} = \mathsf{read}\ s\ i\ \ \mathsf{in}\\
\quad\quad \mathsf{let}\ \{b\} = \mathsf{celim}(insert()\ []\ []\ []\ []\ []\ )\ s\ a\ 0\ i\ \mathsf{in}
\end{array}
$$

celim(ISort() [] [] [] []) $s(i+1)\, l_s$

else

return ()

$$\vdash \quad \mathsf{ISort} \quad : \quad \begin{array}{l} \text{unit} \to \forall \gamma_1 : \mathbb{L}.\forall N, I : \mathbb{N}.(I \le N) \supset \mathsf{Array}_{\gamma_1}[N]\ \text{int} \to \\ \text{int}[I] \to \text{int}[N] \to \overset{\text{exec}(\frac{(N+1)*(N+2)-(I+1)*(I+2)}{2}, (2N+1)*(N+1)-(2I+3)*(I+1))}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}} \end{array}$$

$$\vdash \mathsf{ISort} \ominus \mathsf{ISort} \lesssim 0 : \begin{array}{l} \text{unit} \to \forall \gamma_1 : \mathbb{L}.\forall N, I : \mathbb{N}.\forall \beta_1.(I \le N) \supset \mathsf{Array}_{\gamma_1}[N]\ \text{int} \\ \to \text{int}[I] \to \text{int}[N] \to \{\gamma_1 \to \beta_1\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\} \end{array}$$

where $k = max(I, min(MIN(\beta_1), N))$.

We first apply the fix-ext rule to store the unary information of ISort into the context. After several application of R-ILam and R-Abs rules as well as the rule R-CImpl, we introduce the index variables, variables and the constraint to the left side of the typing judgment. We will focus on the relational type of the body of the function ISort.

Set Z as the body of the function. where $C = (I \le N)$ and
$\tau = \{\gamma_1 \to \beta_1\}^{\text{diff}(\frac{N*(N+1)-k*(k+1)}{2})}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}$.

To show the type of part Z, we will use the rule R-SPLIT first.

$$\frac{\Sigma; \Delta; \Phi_a \wedge C \wedge \beta_1 \cap [0, I] = \emptyset; \Gamma \vdash Z \ominus Z \lesssim 0 : \tau \ldots (Case\,1) \quad \Sigma; \Delta; \Phi_a \wedge C \wedge \beta_1 \cap [0, I] \neq \emptyset; \Gamma \vdash Z \ominus Z \lesssim 0 : \tau \ldots (Case\,2)}{\Sigma; \Delta; \Phi_a \wedge C; \Gamma \vdash Z \ominus Z \lesssim 0 : \tau}\ \text{R-SPLIT}$$

Case 1 $\beta_1 \cap [0, I] = \emptyset$

We know the two runs will go to the same path and we use relational typing rules to obtain the relational type of the body Z. After applying the rule R-Case, we need to show the type of the two branches respectively. Set the first branch as (1), the second (return ()) as (2), Φ' to be the updated constraint environment.

$$\frac{\Delta;\Phi'_a \vdash i < l_s \ominus i < l_s \lesssim 0 : \text{unit} + \text{unit} \qquad \Delta;\Phi'_a;\Gamma \vdash (1) \ominus (1) \lesssim 0 : \tau \qquad \Delta;\Phi'_a;\Gamma \vdash (2) \ominus (2) \lesssim 0 : \tau}{\Delta;\Phi'_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \tau}\ \text{R-CASE}$$

(a) The relative cost for (2) is 0, return () as a unit operation causes no relative cost. We use sub-typing rule S-RM to make the types of two branches consistent.

(b)The relative cost of (1) depends on the insert function as well as the recursive call of ISort.

$$\text{Set (3) as let } b = \text{celim}(insert() [] [] [] [] []) \ s\ a\ 0\ i \text{ in}$$

$$\text{celim}(ISort() [] [] [] [])s(i+1)l_s$$

$$\frac{\Delta;\Phi'_a;\Gamma \vdash insert\ldots \ominus insert\ldots \lesssim 0 : \{\gamma_1 \to \beta_1\}\ \overset{\text{diff(0)}}{\exists}\_.\text{unit}\ \{\gamma_1 \to \beta_1\} \qquad \Delta;\Phi'_a;\Gamma \vdash ISort\ldots \ominus ISort\ldots \lesssim 0 : \tau[k/k']}{\Delta;\Phi'_a;\Gamma \vdash (3) \ominus (3) \lesssim 0 : \tau}\ \text{R-LET}$$

where $k' = max(I+1, min(MIN(\beta_1), N))$.

Because in this case, we know that $\beta_1 \cap [0, I] = \emptyset \Rightarrow k = k'$, and the relative cost of insert is 0 and the relative cost of (3) comes from the recursive call. Because we already know $k' = k$, it is easy to infer that $\tau[k/k'] = \tau$.

Case 2 $\beta_1 \cap [0, I] \neq \emptyset$.

The two runs may go to different branches when inserting one element into the array. So we will switch to unary typing.

We show the unary type of (3) first:

$$\dfrac{\Delta;\Phi_a;\Omega \vdash^{c_1}_{c_1} insert\cdots : \overset{\text{exec}(I+1,2I+2)}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}} \quad \Delta;\Phi_a;\Omega \vdash^{c_2}_{c_2} ISort\cdots : \overset{\text{exec}(\frac{(N+1)*(N+2)-(I+2)*(I+3)}{2},(2N+1)*(N+1)-(2I+5)*(I+2))}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}}}{\Delta;\Phi_a;\Omega \vdash^{c_1+c_2+1}_{c_1+c_2+1} (3) : \overset{\text{exec}(\frac{(N+1)*(N+2)-(I+1)*(I+2)}{2},(2N+1)*(N+1)-(2I+3)*(I+1))}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}}}\ \text{U-LET}$$

From R-S, we know the relative cost of the function ISort is $\frac{N*(N+1)-I*(I+1)}{2}$. Because we know that $\beta_1 \cap [0,I] \neq \emptyset \Rightarrow k = I$,
the type is still $\overset{\text{diff}(\frac{N*(N+1)-k*(k+1)}{2})}{\{\gamma_1 \to \beta_1\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}}$.

By applying unary subtyping rule S-UM, we know the following unary typing used in R-S.

$$\dfrac{\Delta;\Phi_a;|\Gamma|_1 \vdash^{c_1+c_2+1}_{c_1+c_2+1} (3) : \overset{\text{exec}(L',U')}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}} \quad \Delta;\Phi_a;|\Gamma|_2 \vdash^{c_1+c_2+1}_{c_1+c_2+1} (3) : \overset{\text{exec}(L',U')}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}}}{\Delta;\Phi_a;\Gamma \vdash (3) \ominus (3) \lesssim 0 :}\ \text{R-S}$$

$$U(\overset{\text{exec}(L',U')}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}}, \overset{\text{exec}(L',U')}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}})$$

$$\models U(\overset{\text{exec}(L',U')}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}}, \overset{\text{exec}(L',U')}{\{\gamma_1 \to \mathbb{N}\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}}) \sqsubseteq$$

where $L' = \frac{(N+1)*(N+2)-(I+1)*(I+2)}{2}, U' = (2N+1)*(N+1)-(2I+3)*(I+1)$

$$\overset{\text{diff}(\frac{N*(N+1)-I*(I+1)}{2})}{\{\gamma_1 \to \beta_1\}\ \exists\_.\text{unit}\ \{\gamma_1 \to \mathbb{N}\}}$$

A.6.5 Boolean Or

$$
\begin{array}{l}
\mathsf{BoolOr} = \mathsf{fix}\, f\,(\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda s.\ \lambda m.\lambda l_s. \\
\quad \mathsf{if}\ m < l_s\ \mathsf{then} \\
\quad\quad \mathsf{let}\ \{a\} = \mathsf{read}\ s\ m\ \ \mathsf{in} \\
\quad\quad\quad \mathsf{if}\ a\ \mathsf{then} \\
\quad\quad\quad\quad \mathsf{return}\ true \\
\quad\quad\quad \mathsf{else} \\
\quad\quad\quad\quad (\mathsf{celim}\ \mathsf{f}\,()\,[]\,[]\,[]\,[])\ s\,(m+1)\ l_s \\
\quad \mathsf{else} \\
\quad\quad \mathsf{return}\ false
\end{array}
$$

$$
\vdash \quad \mathsf{BoolOr} \quad : \quad
\begin{array}{l}
\mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall M, N : \mathbb{N}.(M \le N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{bool} \to \\
\mathrm{int}[M] \to \mathrm{int}[N] \to \\
\{\gamma_1 \to \emptyset\}\ \overset{\mathrm{exec}(3,(N-M)*3+1)}{\exists \_.\mathrm{bool}}\ \{\gamma_1 \to \emptyset\}
\end{array}
$$

$$
\vdash \mathsf{BoolOr} \ominus \mathsf{BoolOr} \lesssim 0 : \quad
\begin{array}{l}
\mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall M, N : \mathbb{N}.\forall \beta_1.(M \le N) \\
\supset \mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{bool}, \mathrm{bool}) \to \mathrm{int}[M] \to \mathrm{int}[N] \to \\
\{\gamma_1 \to \beta_1\}\ \overset{\mathrm{diff}((N-1-min(MIN(\beta_1 \cap [M,\infty)),N))*3+1)}{\exists \_.U(\mathrm{bool}, \mathrm{bool})}\ \{\gamma_1 \to \beta_1\}
\end{array}
$$

For the relational type, we show the type derivation on case analysis using R-Split rule.

Set Z as the body of the function. where $C = (M \leq N)$ and
$\tau = \{\gamma_1 \to \beta_1\}\ \exists\_.U(\text{bool},\text{bool})\ \{\gamma_1 \to \beta_1\}$.
$^{\text{diff}((N-1-min(MIN(\beta_1 \cap [M,\infty),N))*3+1)}$

To show the type of part Z, we will use the rule R-SPLIT first.

$$\frac{\Sigma;\Delta;\Phi_a \wedge C \wedge M \notin \beta_1;\Gamma \vdash Z \ominus Z \lesssim 0 : \tau \ldots (Case\,1) \quad \Sigma;\Delta;\Phi_a \wedge C \wedge M \in \beta_1;\Gamma \vdash Z \ominus Z \lesssim 0 : \tau \ldots (Case\,2)}{\Sigma;\Delta;\Phi_a \wedge C;\Gamma \vdash Z \ominus Z \lesssim 0 : \tau}\ \text{R-SPLIT}$$

Case 1 $M \notin \beta_1$

Because $M \notin \beta_1$. We know the two runs will go to the same path and we use relational typing rules to obtain the relational type of the body Z. After applying the rule R-Case, we need to show the type of the two branches respectively. Set the first branch as (1), the second (return false) as (2).

$$\frac{\Delta;\Phi_a \vdash a \ominus a \lesssim 0 : \text{unit} + \text{unit} \quad \Delta;\Phi_a;\Gamma \vdash (1) \ominus (1) \lesssim 0 : \{\gamma_1 \to \beta_1\}\ \exists\_.U(\text{bool},\text{bool})\ \{\gamma_1 \to \beta_1\}^{\text{diff}((N-1-min(MIN(\beta_1 \cap [M,\infty)),N))*3+1)} \quad \Delta;\Phi_a;\Gamma \vdash (2) \ominus (2) \lesssim 0 : \{\gamma_1 \to \beta_1\}\ \exists\_.U(\text{bool},\text{bool})\ \{\gamma_1 \to \beta_1\}^{\text{diff}((N-1-min(MIN(\beta_1 \cap [M,\infty)),N))*3+1)}}{\Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma_1 \to \beta_1\}\ \exists\_.U(\text{bool},\text{bool})\ \{\gamma_1 \to \beta_1\}^{\text{diff}((N-1-min(MIN(\beta_1 \cap [M,\infty)),N))*3+1)}}\ \text{R-CASE}$$

(a) The relative cost for (2) is 0, because return false causes no relative cost. We use sub-typing rule S-RM to make the types of two branches consistent.

(b)The relative cost of (1) depends on its second branch , which is a recursive call of the f function. We know the relative cost is $(N - min(MIN(\beta_1 \cap [M,\infty)),N)) * r$ by using relational typing rules R-APP and R-Iapp . Because we already know $M \notin \beta_1$, it is easy to infer that $MIN(\beta_1 \cap [M,\infty)) = MIN(\beta_1 \cap [M+1,\infty))$.

Case 2 $M \in \beta_1$.

When $M \in \beta_1$, the two runs may go to different branches when reaching the if conditional. So we will switch to unary typing. From R-S, we know the relative cost of the function is $(N-M-1)*3+1$,
the type is still $\{\gamma_1 \to \beta_1\}\ \exists\_.U(\text{bool},\text{bool})^{\text{diff}((N-1-min(MIN(\beta_1\cap[M,\infty),N))*3+1)}\ \{\gamma_1 \to \beta_1\}$ for the reason that under the assumption of $M \in \beta_1$, $(N-1-min(MIN(\beta_1 \cap [M,\infty)),N)) = (N-1-M)$.

We get the unary type of function body from the context $|\Gamma|$ and apply U-IAPP and U-APP to get the unary type of the body:

$$\{\gamma_1 \to \emptyset\}\ \exists\_.\text{bool}^{\text{exec}(3,(N-M)*3+1)}\ \{\gamma_1 \to \emptyset\}$$

By applying unary subtyping rule S-UM, we know the following unary typing used in R-S.

$$\frac{\Delta;\Phi_a;|\Gamma|_1 \vdash^0_0 (1) : \{\gamma_1 \to \emptyset\}\ \exists\_.\text{bool}^{\text{exec}(3,(N-M)*3+1)}\ \{\gamma_1 \to \emptyset\} \quad \Delta;\Phi_a;|\Gamma|_2 \vdash^0_0 (1) : \{\gamma_1 \to \emptyset\}\ \exists\_.\text{bool}^{\text{exec}(3,(N-M)*3+1)}\ \{\gamma_1 \to \emptyset\}}{\Delta;\Phi_a;\Gamma \vdash (1) \ominus (1) \lesssim 0 : U(\{\gamma_1 \to \emptyset\}\ \exists\_.\text{bool}^{\text{exec}(3,(N-M)*3+1)}\ \{\gamma_1 \to \emptyset\}, \{\gamma_1 \to \emptyset\}\ \exists\_.\text{bool}^{\text{exec}(3,(N-M)*3+1)}\ \{\gamma_1 \to \emptyset\})}\ \text{R-S}$$

$$\models U(\{\gamma_1 \to \emptyset\}\ \exists\_.\text{bool}^{\text{exec}(3,(N-M)*3+1)}\ \{\gamma_1 \to \emptyset\}, \{\gamma_1 \to \emptyset\}\ \exists\_.\text{bool}^{\text{exec}(3,(N-M)*3+1)}\ \{\gamma_1 \to \emptyset\}) \sqsubseteq$$

$$\{\gamma_1 \to \beta_1\}\ \exists\_.U(\text{bool},\text{bool})^{\text{diff}((N-1-M)*3+1)}\ \{\gamma_1 \to \beta_1\}$$

A.6.6 Boolean Or, Two Implementations

We have two implementaions for the above Boolean or functions.

$$
\begin{array}{l}
\mathsf{fix}\,\mathsf{BoolOr1}\,(\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda s.\ \lambda m.\lambda l_s. \\
\quad \mathsf{if}\ m < l_s\ \mathsf{then} \\
\quad\quad \mathsf{let}\ \{a\} = \mathsf{read}\ s\ m\ \ \mathsf{in} \\
\quad\quad\quad \mathsf{if}\ a\ \mathsf{then} \\
\quad\quad\quad\quad \mathsf{return}\ true \\
\quad\quad\quad \mathsf{else} \\
\quad\quad\quad\quad (\mathsf{celim}\ \mathsf{BoolOr1}\,()\,[]\,[]\,[]\,[])\ s\,(m+1)\ l_s \\
\quad \mathsf{else} \\
\quad\quad \mathsf{return}\ false
\end{array}
$$

The unary type of BoolOr1 as follows.

$$
\vdash \quad \mathsf{BoolOr1} \quad : \quad \begin{array}{l} \mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall M, N : \mathbb{N}.(M \le N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{bool} \to \\ \mathrm{int}[M] \to \mathrm{int}[N] \to \{\gamma_1 \to \emptyset\}\ \overset{\mathrm{exec}(3,(N-M)*3+1)}{\exists \_.\mathrm{bool}}\ \{\gamma_1 \to \emptyset\} \end{array}
$$

The following BoolOr2 is another implenmentation of BoolOr.

$$
\begin{array}{l}
\mathsf{fix}\,\mathsf{BoolOr2}\,(\_).\Lambda.\Lambda.\Lambda.\Lambda.\lambda s.\ \lambda m.\lambda l_s. \\
\quad \mathsf{if}\ m < l_s\ \mathsf{then} \\
\quad\quad \mathsf{let}\ \{a\} = \mathsf{read}\ s\ m\ \ \mathsf{in} \\
\quad\quad\quad \mathsf{if}\ (\mathsf{celim}\ \mathsf{BoolOr2}\,()\,[]\,[]\,[]\,[])\ s\,(m+1)\ l_s)\ \mathsf{then} \\
\quad\quad\quad\quad \mathsf{return}\ true
\end{array}
$$

else

return a

else

return $false$

With its unary type presented below.

$$\vdash \quad \mathsf{BoolOr2} \quad : \quad \begin{array}{l} \mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall M, N : \mathbb{N}.(M \le N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{bool} \to \\ \mathrm{int}[M] \to \mathrm{int}[N] \to \overset{\mathrm{exec}(3*(N-M)+1,3*(N-M)+1)}{\{\gamma_1 \to \emptyset\}\ \exists\_.\mathrm{bool}\ \{\gamma_1 \to \emptyset\}} \end{array}$$

We can obtain the relational type with relative cost 0.

$$\vdash \mathsf{BoolOr1} \ominus \mathsf{BoolOr2} \lesssim 0 : \begin{array}{l} \mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall M, N : \mathbb{N}.\forall \beta_1.(M \le N) \\ \supset U(\mathsf{Array}_{\gamma_1}[N]\ \mathrm{bool}, \mathsf{Array}_{\gamma_1}[N]\ \mathrm{bool}) \\ \to U(\mathrm{int}[M], \mathrm{int}[M]) \to U(\mathrm{int}[N], \mathrm{int}[N]) \to \\ \{\gamma_1 \to \beta_1\}\ \exists\_.\overset{\mathrm{diff}(0)}{U(\mathrm{bool}, \mathrm{bool})}\ \{\gamma_1 \to \beta_1\} \end{array}$$

The derivation mainly relies on the switch rules.

$$\frac{\begin{array}{c} \Delta; \Phi_a; |\Gamma|_1 \vdash^0_0 \mathsf{BoolOr1} : A \to \overset{\mathrm{exec}(3,(N-M)*3+1)}{\{\gamma_1 \to \emptyset\}\ \exists\_.\mathrm{bool}\ \{\gamma_1 \to \emptyset\}} \\ \Delta; \Phi_a; |\Gamma|_2 \vdash^0_0 \mathsf{BoolOr2} : A \to \overset{\mathrm{exec}((N-M)*3+1,(N-M)*3+1)}{\{\gamma_1 \to \emptyset\}\ \exists\_.\mathrm{bool}\ \{\gamma_1 \to \emptyset\}} \end{array}}{\begin{array}{c} \Delta; \Phi_a; \Gamma \vdash \mathsf{BoolOr1} \ominus \mathsf{BoolOr2} \lesssim 0 : \\ U(A \to \overset{\mathrm{exec}(3,(N-M)*3+1)}{\{\gamma_1 \to \emptyset\}\ \exists\_.\mathrm{bool}\ \{\gamma_1 \to \emptyset\}}, A \to \overset{\mathrm{exec}((N-M)*3+1,(N-M)*3+1)}{\{\gamma_1 \to \emptyset\}\ \exists\_.\mathrm{bool}\ \{\gamma_1 \to \emptyset\}}) \end{array}}\ \text{R-S}$$

We set A to be $\mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall M, N : \mathbb{N}.(M \le N) \supset \mathsf{Array}_{\gamma_1}[N]\ \mathrm{bool} \to \mathrm{int}[M] \to \mathrm{int}[N]$.
We set A_1 to be $\overset{\mathrm{exec}(3,(N-M)*3+1)}{\{\gamma_1 \to \emptyset\}\ \exists\_.\mathrm{bool}\ \{\gamma_1 \to \emptyset\}}$. set A_2 to be $\overset{\mathrm{exec}((N-M)*3+1,(N-M)*3+1)}{\{\gamma_1 \to \emptyset\}\ \exists\_.\mathrm{bool}\ \{\gamma_1 \to \emptyset\}}$.

We set τ to be $\mathrm{unit} \to \forall \gamma_1 : \mathbb{L}.\forall M, N : \mathbb{N}.\forall \beta_1.(M \le N) \supset U(\mathsf{Array}_{\gamma_1}[N]\ \mathrm{bool}, \mathsf{Array}_{\gamma_1}[N]\ \mathrm{bool}$

$) \to U(\text{int}[M], \text{int}[M]) \to U(\text{int}[N], \text{int}[N])$.
We set τ' to be $\{\gamma_1 \to \beta_1\}\ \exists\_.U(\overset{\text{diff}(0)}{\text{bool}, \text{bool}})\ \{\gamma_1 \to \beta_1\}$.

Use the subtyping rules S-R-FORALL-U, S-R-CIMPL-U and S-R-EXECDIFF, we can show that

$$\models U(A \to A_1, A \to A_2) \sqsubseteq \tau \to U(A_1, A_2)$$

Next, using the monadic sutyping S-RUM, we can show that

$$\models U(\{\gamma_1 \to \emptyset\}\ \exists\_.\overset{\text{exec}(3,(N-M)*3+1)}{\text{bool}}\ \{\gamma_1 \to \emptyset\}, \{\gamma_1 \to \emptyset\}\ \exists\_.\overset{\text{exec}((N-M)*3+1,(N-M)*3+1)}{\text{bool}}\ \{\gamma_1 \to \emptyset\}) \sqsubseteq$$

$$\{\gamma_1 \to \beta_1\}\ \exists\_.U(\overset{\text{diff}(0)}{\text{bool}, \text{bool}})\ \{\gamma_1 \to \beta_1\}$$

Finally, using the ruele R-EXEC, we can derive the relational type shown above.

A.6.7 InPlaceMap

Example 1: In Place Map

$$\begin{array}{l}\mathsf{mapi} = \mathsf{fix}\ \mathsf{map}\,(f).\lambda a.\lambda k.\lambda n.\\ \ \mathsf{if}\ k \le n\ \mathsf{then}\ \big(\mathsf{let}\ \{x\} = \big(\ \mathsf{read}\ a\ k\ \big)\ \mathsf{in}\\ \qquad \mathsf{let}\{\_\} = \big(\ \mathsf{updt}\ a\ k\ (f\ x)\ \big)\ \mathsf{in}\ \big((\mathsf{celim}\ (\mathsf{map}\,f))\ a\,(k+1)\ n\big)\\ \ \mathsf{else}\ \mathsf{return}\,()\end{array}$$

As a first example we want to prove

$$\vdash\ \mathsf{mapi}\ :\ \begin{array}{l}\forall L.\forall U : \mathbb{N}.(A \overset{\mathrm{exec}(L,U)}{\longrightarrow} A') \longrightarrow \forall n : \mathbb{N}.\ \forall i : \mathbb{N}.\ \forall \gamma : \mathbb{L}.\\ (i \le n) \supset \big(\ \mathsf{Array}_\gamma[n]\ A \longrightarrow \mathrm{int}[i] \longrightarrow \overset{\mathrm{exec}((L+c_r+c_u)*(n-i),(U+c_r+c_u)*(n-i))}{\{\gamma \to \mathbb{N}\}\ \exists\_.\mathrm{unit}\ \{\gamma \to \mathbb{N}\}}\ \big)\end{array}$$

where c_r is the cost for one reading operation and c_u denotes the cost for one updating operation.

$$\vdash \mathsf{mapi} \ominus \mathsf{mapi} \lesssim 0 :\ \begin{array}{l}\forall r.\Box(\tau \overset{\mathrm{diff}\,r}{\longrightarrow} \tau) \longrightarrow \forall n, i : \mathbb{N}.\ \forall \gamma : \mathsf{ref}.\ \forall \beta : \mathbb{P}.\\ (i \le n) \supset \big(\ \mathsf{Array}_\gamma[n]\ \tau \longrightarrow \mathrm{int}[i] \longrightarrow \{\gamma \to \beta\}\ \overset{\mathrm{diff}((|\beta \cap [i,n]| * r))}{\exists\_.\mathrm{unit}\ \{\gamma \to \beta\}}\ \big)\end{array}$$

$$\vdash \mathsf{mapi} \ominus \mathsf{mapi} \lesssim 0 :\ \begin{array}{l}\forall r.(\tau \overset{\mathrm{diff}\,r}{\longrightarrow} \tau) \longrightarrow \forall n, i : \mathbb{N}.\ \forall \gamma : \mathsf{ref}.\ \forall \beta : \mathbb{P}.\ (i \le n) \supset\\ \big(\ \mathsf{Array}_\gamma[n]\ \tau \longrightarrow \mathrm{int}[i] \longrightarrow \{\gamma \to \beta\}\ \overset{\mathrm{diff}(((n-i)*r))}{\exists\_.\mathrm{unit}\ \{\gamma \to \beta \cup [i,n]\}}\ \big)\end{array}$$

Let us call M the following term: $\big(\mathsf{celim}(\mathsf{map}\,f)\ a\,(k+1)\ n\big)$. and call T the following term: $\Big(\mathsf{let}\ \{x\} = \big(\ \mathsf{read}\ a\ k\ \big)\ \mathsf{in}\ \mathsf{let}\{\_\} = \big(\ \mathsf{updt}\ a\ i\ (f\ x)\ \big)\ \mathsf{in}\ (\mathsf{celim}(\mathsf{map}\,f)\ a\,(k+1)\ n)\Big)$.
and call Z the following term: $\mathsf{let}\{\_\} = \big(\ \mathsf{updt}\ a\ i\ (f\ x)\ \big)\ \mathsf{in}\big(\mathsf{celim}(\mathsf{map}\,f)\ a\,(k+1)\ n\big)$.
and set V as: if $k < n$, then T else (return ()).

Before we discussing the two types, we can apply simple rules first to make the relational type easy to read. We apply $R-iLam$ and $R-abs$ several times. Before we start to derivate the relational type of part V, We apply relational rule $R-Cimpl$ to introduce the constraint $i \leq n$ into the constraint environment Φ_a.

Type 1: Same function We have two possible cases, one is that position $I \in \beta$ which means the values in the two arrays of two runs may be different, the other case is $I \notin \beta$, which means in index I, the values are the same in the two arrays:

Before we go to the two cases, we will first apply R-split. We need to show two cases and their types, with different constraint environment $\Phi_a \wedge i \in \beta$ and $\Phi_a \wedge i \notin \beta$ respectively.

Case 1: different values from read, $\Phi_a \wedge i \in \beta$

$$\frac{\Delta;\Phi_a;\Gamma \vdash a \ominus a \lesssim 0 : \mathsf{Array}_\gamma[n]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash k \ominus k \lesssim 0 : \mathrm{int}[i] \qquad \Delta;\Phi_a \models i \leq n \qquad \Delta;\vdash \mathsf{empty\ wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ a\ k \ominus \mathsf{read}\ a\ k \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(0)}{\exists\_.\tau}\ \{\gamma \to \beta\}}\ \text{R-R}$$

and

$$\frac{\Delta;\Phi_a;\Gamma \vdash a \ominus a \lesssim 0 : \mathsf{Array}_\gamma[n]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash k \ominus k \lesssim 0 : \mathrm{int}[i] \qquad \Delta;\Phi_a;\Gamma, x:\tau \vdash f\ x \ominus f\ x \lesssim r : \tau \qquad \Delta;\Phi_a \models i \leq n \qquad \Delta;\vdash \mathsf{empty\ wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ a\ k\ (f\ x) \ominus \mathsf{updt}\ a\ k\ (f\ x) \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(r)}{\exists\_ : \mathrm{unit}}\ \{\gamma \to \beta \cup \{i\}\}}\ \text{R-U}$$

where $\Delta = n:\mathbb{N}, i:\mathbb{N}, r:\mathbb{R}$

$\Sigma = \gamma : \mathbb{L}$, $\Phi_a = i \leq n$,

$\Gamma = a$:$\mathsf{Array}_\gamma[n]\ \tau, k : \mathrm{int}[i], f : \square(\tau \overset{\mathrm{diff}(r)}{\longrightarrow} \tau)$,

$map : \square(\tau \xrightarrow{\text{diff}(r)} \tau) \xrightarrow{\text{diff}(0)} \forall n : \mathbb{N}.\ \forall i : \mathbb{N}.\ \forall \gamma : \mathbb{L}.\ (i \le n) \supset$
$\big(\ \mathsf{Array}_\gamma[n]\ \tau \longrightarrow \text{int}[i] \longrightarrow \text{int}[n] \longrightarrow \{\gamma \to \beta\}\ \text{unit}\ \{\gamma \to \beta\}\ \big).$

Because at position i, $a_1[i] \neq a_2[i]$,s.t. $\beta \cup \{i\} = \beta$.

We will show:

S-RM

$$\frac{\dfrac{}{\Delta;\Phi_a \models \{\gamma \to \beta\}\ \exists\_ : \text{unit}\ \{\gamma \to \beta\} \overset{\text{diff}(0)}{\sqsubseteq} \{\gamma \to \beta\}\ \exists\_ : \text{unit}\ \{\gamma \to \beta\} \overset{\text{diff}(|\beta \cap [i,n]| * r)}{}}}{\Delta;\Phi_a;\Gamma \vdash \ \mathsf{return}\ () \ominus \mathsf{return}\ () \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \text{unit}\ \{\gamma \to \beta\} \overset{\text{diff}(0)}{}}\ \text{R-EXEC}$$
$$\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ () \ominus \mathsf{return}\ () \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \text{unit}\ \{\gamma \to \beta\} \overset{\text{diff}(|\beta \cap [i,n]| * r)}{}$$

$$\frac{\dfrac{\Delta;\Phi_a;\Gamma \vdash k < n \ominus k < n \lesssim 0 : \text{unit} + \text{unit} \qquad \Delta;\Phi_a;\Gamma \vdash\ T \ominus\ T \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \text{unit}\ \{\gamma \to \beta\} \overset{\text{diff}(|\beta \cap [i,n]| * r)}{}}{}}{\Delta;\Phi_a;\Gamma, \{x : \tau\} \vdash V \ominus V \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \text{unit}\ \{\gamma \to \beta\} \overset{\text{diff}(|\beta \cap [i,n]| * r)}{}}\ \text{R-CASE}$$

Using R-Let rules, we get

R-LET

$$\frac{P_2 = \text{empty} \qquad \Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ a\ k \ominus \mathsf{read}\ a\ k \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \text{int}\ \{\gamma \to \beta\} \overset{\text{diff}(0)}{} \qquad \Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \text{unit}\ \{\gamma \to \beta\} \overset{\text{diff}(|\beta \cap [i,n]| * r)}{}}{\Delta;\Phi_a;\Gamma, \{x : \text{int}\} \vdash T \ominus T \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \text{unit}\ \{\gamma \to \beta\} \overset{\text{diff}(|\beta \cap [i,n]| * r)}{}}$$

,

R-LET

$$\frac{P_2 = \mathsf{empty} \qquad \Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ a\ k\ (f\ x) \ominus \mathsf{updt}\ a\ k\ (f\ x) \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to \beta\}^{\mathrm{diff}(r)} \qquad \Delta;\Phi_a;\Gamma \vdash M \ominus M \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to \beta\}^{\mathrm{diff}(|\beta \cap [i+1,n]|*r)}}{\Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to \beta\}^{\mathrm{diff}(|\beta \cap [i+1,n]|*r+r)}}$$

Because i in β, we know: $|\beta \cap [i+1, n]| * r + r = |\beta \cap [i, n]|$ and $\beta \cup \{i\} = \beta$.

We want to show the typing of M. By applying R-IApp (when instantiating $\forall i$, we use i+1) and R-App(when instantiating $\lambda.k$, we use k+1), we can easily get

R-CELIM

$$\frac{\Delta;\Phi_a \vDash i+1 \le n \qquad \Delta;\Phi_a\Gamma \vdash \big((\mathsf{map}\ f)\big) \ominus \big((\mathsf{map}\ f)\big) \lesssim 0 : (i+1 \le n) \supset \big(\mathsf{Array}_\gamma[n]\ \tau \to \mathrm{int}[i+1] \to \mathrm{int}[n] \to \{\gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to \beta\}^{\mathrm{diff}(|\beta \cap [i+1,n]|*r)}\big)}{\Delta;\Phi_a;\Gamma \vdash \big(\mathsf{celim}(\mathsf{map}\ f)\big) \ominus \big(\mathsf{celim}(\mathsf{map}\ f)\big) \lesssim 0 : \mathsf{Array}_\gamma[n]\ \tau \to \mathrm{int}[i+1] \to \mathrm{int}[n] \to \{\gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to \beta\}^{\mathrm{diff}(|\beta \cap [i+1,n]|*r)}}$$

,

$$\Delta;\Phi_a;\Gamma \vdash \big(\mathsf{celim}(\mathsf{map}\ f)\ a\ (k+1)\ n\big) \ominus \big(\mathsf{celim}(\mathsf{map}\ f)\ a\ (k+1)\ n\big) \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \mathrm{unit}\ \{\gamma \to \beta\}^{\mathrm{diff}(|\beta \cap [i+1,n]|*r)}$$

Case 2: the same value from read, $\Phi_a \wedge i \notin \beta$

$$\frac{\Delta;\Phi_a;\Gamma \vdash a \ominus a \lesssim 0 : \mathsf{Array}_\gamma[n]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash k \ominus k \lesssim 0 : \mathrm{int}[i] \qquad \Delta;\Phi_a \vDash i \leq n \qquad \Delta;\Phi_a \vDash i \notin \beta}{\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ a\ k \ominus \mathsf{read}\ a\ k \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(0)}{\exists\_.\Box\tau}\ \{\gamma \to \beta\}}\ \text{R-RB}$$

and

$$\frac{\dfrac{\dfrac{\Delta;\Phi_a;\Gamma \vdash a \ominus a \lesssim 0 : \mathsf{Array}_\gamma[n]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash k \ominus k \lesssim 0 : \mathrm{int}[i]}{\Delta;\Phi_a \vDash \Box(\tau \overset{\mathrm{diff}(r)}{\longrightarrow} \tau) \sqsubseteq \Box\tau \overset{\mathrm{diff}(0)}{\longrightarrow} \Box\tau}\ \text{S-R-BOX-DIFF}}{\Delta;\Phi_a \vdash f \ominus f \lesssim 0 : \Box\tau \overset{\mathrm{diff}(0)}{\longrightarrow} \Box\tau}}{\Delta;\Phi_a;\Gamma\, x : \Box\tau \vdash f\ x \ominus f\ x \lesssim 0 : \Box\tau} \qquad \Delta;\Phi_a \vDash i \leq n}{\Delta;\Phi_a;\Gamma, \vdash \mathsf{updt}\ a\ k\ (f\ x) \ominus \mathsf{updt}\ a\ k\ (f\ x) \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(0)}{\exists\_ : \mathrm{unit}}\ \{\gamma \to \beta / \{i\}\}}$$

R-UB

where $\Delta = n : \mathbb{N}, i : \mathbb{N}, r : \mathbb{R}$

$\Sigma = \gamma : \mathbb{L}$, $\Phi_a = i \leq n$,

$\Gamma = a : \mathsf{Array}_\gamma[n]\ \tau, k : \mathrm{int}[i], f : \Box(\tau \overset{\mathrm{diff}(r)}{\longrightarrow} \tau), map : \Box(\tau \overset{\mathrm{diff}(r)}{\longrightarrow} \tau) \overset{\mathrm{diff}(0)}{\longrightarrow} \forall n, i, \gamma, \beta : \mathbb{N}.$
$(i \leq n) \supset \left(\mathsf{Array}_\gamma[n]\ \tau \longrightarrow \mathrm{int}[i] \longrightarrow \mathrm{int}[n] \longrightarrow \exists \beta.\ \{\gamma \to \beta\}\ \overset{\mathrm{diff}(|\beta \cap [i,n]| * r)}{\mathrm{unit}}\ \{\gamma \to \beta\} \right).$

Because at position i, $a_1[i] \neq a_2[i]$,s.t. $\beta \cup \{i\} = \beta$.

Using R-Let rules, we get

$$\frac{\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ a\ k \ominus \mathsf{read}\ a\ k \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(0)}{\exists\_ : \mathrm{int}}\ \{\gamma \to \beta\} \qquad P_2 = \mathsf{empty} \qquad \Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(|\beta \cap [i,n]| * r)}{\exists\_ : \mathrm{unit}}\ \{\gamma \to \beta\}}{\Delta;\Phi_a;\Gamma, \{x : \mathrm{int}\} \vdash T \ominus T \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(|\beta \cap [i,n]| * r)}{\exists\_ : \mathrm{unit}}\ \{\gamma \to \beta\}}\ \text{R-LET}$$

,

$$\frac{\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ a\ k\ (f\ x) \ominus \mathsf{updt}\ a\ k\ (f\ x) \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(0)}{\exists}\_ : \mathrm{unit}\ \{\gamma \to \beta\} \qquad P_2 = \mathsf{empty} \qquad \Delta;\Phi_a;\Gamma \vdash M \ominus M \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(|\beta \cap [i+1,n]| * r)}{\exists}\_ : \mathrm{unit}\ \{\gamma \to \beta\}}{\Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(|S \cap [i+1,n]| * r + 0)}{\exists}\_ : \mathrm{unit}\ \{\gamma \to \beta\}}\ \text{R-LET}$$

Because i not in β, we know: $|\beta \cap [i+1,n]| * r + 0 = |\beta \cap [i,n]|$.

The type of M is the same as we showed in previous case.

Type 2: Different functions :

$$\vdash \mathsf{mapi} \ominus \mathsf{mapi} \lesssim 0 : \begin{array}{l} \forall r : \mathbb{N}.(\tau \overset{\mathrm{diff}(r)}{\longrightarrow} \tau) \longrightarrow \forall n : \mathbb{N}.\ \forall i : \mathbb{N}.\ \forall \gamma : \mathbb{L}.\ \forall \beta : \mathbb{P}.\ (i \le n) \supset \\ \left(\mathsf{Array}_\gamma[n]\ \tau \longrightarrow \mathrm{int}[i] \longrightarrow \{\gamma \to \beta\}\ \overset{\mathrm{diff}(((n-i)*r))}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}\right) \end{array}$$

Let start with read and update from index k.

$$\frac{\Delta;\Phi_a;\Gamma \vdash a \ominus a \lesssim 0 : \mathsf{Array}_\gamma[n]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash k \ominus k \lesssim 0 : \mathrm{int}[i] \qquad \Delta;\Phi_a \models i \le n \qquad \Delta;\vdash \mathsf{empty}\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ a\ k \ominus \mathsf{read}\ a\ k \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(0)}{\exists}\_.\tau\ \{\gamma \to \beta\}}\ \text{R-R}$$

and

$$\frac{\Delta;\Phi_a;\Gamma \vdash a \ominus a \lesssim 0 : \mathsf{Array}_\gamma[n]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash k \ominus k \lesssim 0 : \mathrm{int}[i] \qquad \Delta;\Phi_a;\Gamma, x : \tau \vdash f\ x \ominus f\ x \lesssim r : \tau \qquad \Delta;\Phi_a \models i \le n \qquad \Delta;\vdash \mathsf{empty}\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ a\ k\ (f\ x) \ominus \mathsf{updt}\ a\ k\ (f\ x) \lesssim 0 : \{\gamma \to \beta\}\ \overset{\mathrm{diff}(r)}{\exists}\_ : \mathrm{unit}\ \{\gamma \to \beta \cup \{i\}\}}\ \text{R-U}$$

where $\Delta = n : \mathbb{N}, i : \mathbb{N}, r : \mathbb{R}$

$\Sigma = \gamma : \mathbb{L}$, $\Phi_a = i \le n$,

$\Gamma = a$:$\mathsf{Array}_\gamma[n]\ \tau, k : \mathrm{int}[i], f : (\tau \xrightarrow{\mathrm{diff}(r)} \tau), map : (\tau \xrightarrow{\mathrm{diff}(r)} \tau) \to \forall n, i, \gamma, \beta.$
$(i \le n) \supset \big(\mathsf{Array}_\gamma[n]\ \tau \to \mathrm{int}[i] \to \mathrm{int}[n]\ \{\gamma \to \beta\}\ \overset{\mathrm{diff}((n-i)*r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}\big).$

We will show:

$$\dfrac{\dfrac{}{\models \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \to \beta\} \sqsubseteq \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(|\beta \cap [i,n]|*r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}} \quad \Delta; \Phi_a; \Gamma \vdash \mathsf{return}\ () \ominus \mathsf{return}\ () \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{unit}}\ \{\gamma \to \beta\}}{\Delta; \Phi_a; \Gamma \vdash \mathsf{return}\ () \ominus \mathsf{return}\ () \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(|\beta \cap [i,n]|*r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}}\ \text{R-EXEC}$$

$$\dfrac{\Delta; \Phi_a; \Gamma \vdash k < n \ominus k < n \lesssim 0 : \mathrm{unit} + \mathrm{unit} \quad \Delta; \Phi_a; \Gamma \vdash\ T \ominus\ T \lesssim 0 : \{\gamma \to S\}\ \exists\_ : \overset{\mathrm{diff}(|\beta \cap [i,n]|*r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}}{\Delta; \Phi_a; \Gamma, \{x : \tau\} \vdash V \ominus V \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(|\beta \cap [i,n]|*r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}}\ \text{R-CASE}$$

Using R-Let rules,instantiate P_1 with $\{\gamma \to \beta\}$ and $P_2 = \mathsf{empty}, Q = \{\gamma \to \beta \cup [i,n]\}$, we get

$$\dfrac{\Delta; \Phi_a; \Gamma \vdash \mathsf{read}\ a\ k \ominus \mathsf{read}\ a\ k \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(0)}{\mathrm{int}}\ \{\gamma \to \beta\} \quad \Delta; \Phi_a; \Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(|\beta \cap [i,n]|*r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}}{\Delta; \Phi_a; \Gamma, \{x : \mathrm{int}\} \vdash T \ominus T \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(|\beta \cap [i,n]|*r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}}\ \text{R-LET}$$

Because we know: $(\beta \cup \{i\}) \cap [i+1, n] = \beta \cap [i+1, n]$ and $\beta \cup \{i\} \cup [i+1, n] = \beta \cup [i, n]$

.

Using R-Let rules,instantiate P_1 with $\{\gamma \to \beta\}$ and $P_2 = \mathsf{empty}, Q_1 = \{\gamma \to \beta \cup \{i\}\}$, $Q_2 = \mathsf{empty}, Q = \{\gamma \to \beta \cup \{i\} \cup [i+1, n]\}$, we get

$$\frac{\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ a\ k\ (f\ x) \ominus \mathsf{updt}\ a\ k\ (f\ x) \lesssim 0 : \{\gamma \to \beta\}\ \exists\_ : \overset{\mathrm{diff}(r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup \{i\}\} \quad \Delta;\Phi_a;\Gamma \vdash M \ominus M \lesssim 0 : \{\gamma \to \beta \cup \{i\}\}\ \exists\_ : \overset{\mathrm{diff}(|(\beta \cup \{i\}) \cap [i+1,n]| * r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup \{i\} \cup [i+1,n]\}}{\Delta;\Phi_a;\Gamma \vdash Z \ominus Z \lesssim 0 : \{\gamma \to S\}\ \exists\_ : \overset{\mathrm{diff}(|\beta \cap [i+1,n]| * r + r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}}\ \text{R-LET}$$

We want to show the typing of M. By applying $R-IApp$, instantiating β with $\beta \cup \{i\}$ and i with $i+1$. Then apply $R-App$ (instantiating k with $k+1$) several times, we can easily get $\Delta;\Phi_a;\Gamma \vdash (\mathsf{celim}(\mathsf{map}\ f)\ a\ (k+1)\ n) \ominus (\mathsf{celim}(\mathsf{map}\ f)\ a\ (k+1)\ n) \lesssim 0 :$ $\{\gamma \to \beta \cup \{i\}\}\ \exists\_ : \overset{\mathrm{diff}(|\beta \cap [i+1,n]| * r)}{\mathrm{unit}}\ \{\gamma \to \beta \cup [i,n]\}$. When apply R-Celim, We also check the constraint $i+1 \leq n$.

A.6.8 Square and Multiply (SAM)

This examples implements the square and multiply algorithm for the positive power of the a number. It bases on the observation that $x^m = (x^2)^{\frac{m}{2}}$ if m is even and $x^m = x * (x^2)^{\frac{m}{2}}$ if m is odd.

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{sam}\ ().\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda x.\lambda a.\lambda i.\lambda n. \\
\quad \mathsf{if}\ i \leq (n-1)\ \mathsf{then} \\
\quad\quad \mathsf{let}\ \{b\} = \mathsf{read}\ a\ i\ \mathsf{in} \\
\quad\quad \mathsf{let}\ \{r\} = \mathsf{celim}(\mathsf{sam}\ ()\ []\ []\ []\ []\ []\ [])\ x\ a\ (i+1)\ n\ \mathsf{in} \\
\quad\quad\quad \mathsf{return}(\mathsf{func}\ x\ b\ r) \\
\quad \mathsf{else} \\
\quad\quad \mathsf{if}\ x = 0\ \mathsf{return}\ 1 \\
\quad\quad \mathsf{return}\ x
\end{array}
$$

where func is a function which depends on the value of b (1 or 0), returns the result $x * 1$ if $b = 1$, returns 1 otherwise. The relatonal type is derived by split the cases on the assumption $I \in \beta$ and on the two cases, func will generates different costs based on the assumption whether $I \in \beta$ or not.

$$
\begin{array}{l}
\vdash \mathsf{sam} \ominus \mathsf{sam} \lesssim 0 : \\
\end{array}
\begin{array}{l}
\mathrm{unit} \to \forall \gamma_1.\forall I, N, X, \beta.(I < N) \supset \mathrm{int}[X] \to \mathsf{Array}_{\gamma_1}[N]\ U(\mathrm{bool}, \mathrm{bool}) \\
\to \mathrm{int}[I] \to \mathrm{int}[N] \to \{\gamma_1 \to \beta\}\ \overset{\mathrm{diff}(\beta \cap [I,N])}{\exists\_}.U(\mathrm{int})\ \{\gamma_1 \to \beta\}
\end{array}
$$

function func

$$\mathsf{fix}\ \mathsf{func}\,(x).\lambda r.\lambda b.$$

$$\mathsf{if}\, b\ \mathsf{then}$$

$$\mathsf{return}\, x * r * r$$

$$\mathsf{else}$$

$$\mathsf{return}\, r * r$$

$$\vdash \mathsf{func} \ominus \mathsf{func} \lesssim 0 : \; U(\mathrm{int}) \to U(\mathrm{int}) \to \mathrm{int} \xrightarrow{\mathrm{diff}\,(0)} U(\mathrm{int})$$

$$\vdash \mathsf{func} \ominus \mathsf{func} \lesssim 0 : \; U(\mathrm{int}) \to U(\mathrm{int}) \to U(\mathrm{int}) \xrightarrow{\mathrm{diff}\,(1)} U(\mathrm{int})$$

A.6.9 Constant-time Comparison

$$
\begin{array}{l}
\mathsf{fix}\ \mathsf{comp}\ ().\lambda l_1.\lambda l_2.\lambda i.\lambda n.\\
\quad \mathsf{if}\ i < n\ \mathsf{then}\\
\quad\quad \mathsf{let}\ \{a\} = \mathsf{read}\ l_1\ i\ \mathsf{in}\\
\quad\quad \mathsf{let}\ \{b\} = \mathsf{read}\ l_2\ i\ \mathsf{in}\\
\quad\quad \mathsf{return}\,\mathsf{boolAnd}\big(\mathsf{celim}(comp()[][][])l_1\ l_2\ (i+1)\ n, \mathsf{eq}(a,b)\big)\\
\quad \mathsf{else}\\
\quad\quad \mathsf{return}\ true
\end{array}
$$

where function boolAnd has the type $(U(\text{bool}) \times U(\text{bool}) \xrightarrow{\mathsf{diff}(0)} U(\text{bool})$, and eq has type $(U(\text{int}) \times U(\text{int}) \xrightarrow{\mathsf{diff}(0)} U(\text{bool})$.

$$
\begin{array}{rl}
 & \text{unit} \to \forall \gamma_1, \gamma_2. \forall I, N, \beta_1, \beta_2. (I \geq 0)\\
\vdash \mathsf{comp} \ominus \mathsf{comp} \lesssim 0 : & \supset \mathsf{Array}_{\gamma_1}[N]\ U(\text{int},\text{int}) \to \mathsf{Array}_{\gamma_2}[N]\ U(\text{int},\text{int}) \to \text{int}[I] \to\\
 & \text{int}[N] \to \{\gamma_1 \to \beta_1, \gamma_2 \to \beta_2\}\ \exists \_. \overset{\mathsf{diff}(0)}{U}(\text{bool})\ \{\gamma_1 \to \beta_1, \gamma_2 \to \beta_2\}
\end{array}
$$

When we assume the two arrays has the same length N, start comparing the two arrays from the same index i on the two runs. No relative cost is generated because the execution paths across the two runs are the same.

APPENDIX B

Appendix for BiARel

We use some abbreviations throughout in the appendix, "TS" for "To show", RTS stands for “remains to show", STS stands for “suffices to show". "IH" for induction hypothesis, "IH2" for induction hypothesis on the second goal, which is usually when simultaneously proving multiple goals in a lemma, similar for "IH3", "IH4", and so on.

B.1 RELSTLC

B.1.1 Syntax of relSTLC

$$
\begin{array}{llll}
\text{Types} & \tau & ::= & \text{bool}_r \mid \text{bool}_u \mid \tau_1 \to \tau_2 \\
\text{Terms} & t & ::= & x \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \lambda x.t \mid t_1\ t_2 \\
\text{Value} & v & := & \text{true} \mid \text{false} \mid \lambda x.t
\end{array}
$$

Figure B.1: Syntax of values and terms in relSTLC.

$$
\begin{array}{llll}
\text{Types} & \tau & ::= & \text{bool}_r \mid \text{bool}_u \mid \tau_1 \to \tau_2 \\
\text{Terms} & t & ::= & x \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \lambda x.t \mid t_1\ t_2 \\
\text{Value} & v & := & \text{true} \mid \text{false} \mid \lambda x.t
\end{array}
$$

Figure B.2: Syntax of values and terms in relSTLC-CORE.

$$
\begin{array}{llll}
\text{Types} & \tau & ::= & \text{bool}_r \mid \text{bool}_u \mid \tau_1 \to \tau_2 \\
\text{Terms} & t & ::= & x \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \lambda x.t \mid t_1\ t_2 \mid (t : \tau) \\
\text{Value} & v & := & \text{true} \mid \text{false} \mid \lambda x.t
\end{array}
$$

Figure B.3: Syntax of values and terms in BirelSTLC.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x \backsim x : \tau}\ \textbf{r-var} \qquad \frac{\mathtt{b} \in \{\text{true}, \text{false}\}}{\Gamma \vdash \mathtt{b} \backsim \mathtt{b} : \text{bool}_r}\ \textbf{r-bool} \qquad \frac{\mathtt{b}_1, \mathtt{b}_2 \in \{\text{true}, \text{false}\}}{\Gamma \vdash \mathtt{b}_1 \backsim \mathtt{b}_2 : \text{bool}_u}\ \textbf{r-u-bool}$$

$$\frac{\Gamma \vdash t \backsim t' : \text{bool}_r \qquad \Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Gamma \vdash t_2 \backsim t_2' : \tau}{\Gamma \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 \backsim \text{if } t' \text{ then } t_1' \text{ else } t_2' : \tau}\ \textbf{r-if}$$

$$\frac{\Gamma, x : \tau_1 \vdash t_1 \backsim t_2 : \tau_2}{\Gamma \vdash \lambda x.t_1 \backsim \lambda x.t_2 : \tau_1 \to \tau_2}\ \textbf{r-fix} \qquad \frac{\Gamma \vdash t_1 \backsim t_1' : \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 \backsim t_2' : \tau_1}{\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' : \tau_2}\ \textbf{r-app}$$

$$\frac{\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \models \tau \sqsubseteq \tau'}{\Gamma \vdash t_1 \backsim t_2 : \tau'}\ \textbf{r-}\sqsubseteq$$

Figure B.4: Typing rules of relSTLC.

$$\frac{}{\models \text{bool}_r \sqsubseteq \text{bool}_u}\ \textbf{bool} \qquad \frac{\models \tau_1' \sqsubseteq \tau_1 \qquad \models \tau_2 \sqsubseteq \tau_2'}{\models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'} \to \qquad \frac{}{\models \tau \sqsubseteq \tau}\ \textbf{refl}$$

$$\frac{\models \tau_1 \sqsubseteq \tau_2 \qquad \models \tau_2 \sqsubseteq \tau_3}{\models \tau_1 \sqsubseteq \tau_3}\ \textbf{trans}$$

Figure B.5: Subtyping rules of relSTLC.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x \backsim x \uparrow \tau}\ \textbf{alg-r-var} \qquad \frac{\mathtt{b} \in \{\text{true}, \text{false}\}}{\Gamma \vdash \mathtt{b} \backsim \mathtt{b} \uparrow \text{bool}_r}\ \textbf{alg-r-bool}$$

$$\frac{\mathtt{b}_1, \mathtt{b}_2 \in \{\text{true}, \text{false}\} \qquad b_1 \neq b_2}{\Gamma \vdash \mathtt{b}_1 \backsim \mathtt{b}_2 \downarrow \text{bool}_u}\ \textbf{alg-r-u-bool}$$

$$\frac{\Gamma \vdash t \backsim t' \uparrow \text{bool}_r \qquad \Gamma \vdash t_1 \backsim t_1' \downarrow \tau \qquad \Gamma \vdash t_2 \backsim t_2' \downarrow \tau}{\Gamma \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 \backsim \text{if } t' \text{ then } t_1' \text{ else } t_2' \downarrow \tau}\ \textbf{alg-r-if}$$

$$\frac{\Gamma, x : \tau_1 \vdash t_1 \backsim t_2 \downarrow \tau_2}{\Gamma \vdash \lambda x.t_1 \backsim \lambda x.t_2 \downarrow \tau_1 \to \tau_2}\ \textbf{alg-r-fix}$$

$$\frac{\Gamma \vdash t_1 \backsim t_1' \uparrow \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 \backsim t_2' \downarrow \tau_1}{\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' \uparrow \tau_2}\ \textbf{alg-r-app} \qquad \frac{\Gamma \vdash t \backsim t' \uparrow \tau' \qquad \models \tau' \leq \tau}{\Gamma \vdash t \backsim t' \downarrow \tau}\ \textbf{alg-}\uparrow\downarrow$$

$$\frac{\Gamma \vdash t \backsim t' \downarrow \tau}{\Gamma \vdash (t : \tau) \backsim (t' : \tau) \uparrow \tau}\ \textbf{alg-r-anno-}\uparrow$$

Figure B.6: Algorithmic typing rules of relSTLC.

$$\frac{}{\models \text{bool}_r \leq \text{bool}_r}\ \textbf{alg-bool-r} \qquad \frac{}{\models \text{bool}_u \leq \text{bool}_u}\ \textbf{alg-bool-u}$$

$$\frac{}{\models \text{bool}_r \leq \text{bool}_u}\ \textbf{alg-bool} \qquad \frac{\models \tau_1' \leq \tau_1 \qquad \models \tau_2 \leq \tau_2'}{\models \tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'}\ \textbf{alg-}\to$$

Figure B.7: Algorithmic subtyping rules of relSTLC.

$$\begin{array}{lcl} |.| & : & \text{Expression} \to \text{Expression} \\ \\ |x| & = & x \\ |t_1\ t_2| & = & |t_1|\,|t_2| \\ \vdots & & \\ |(t : \tau)| & = & |t| \end{array}$$

Figure B.8: Annotation erasure.

B.1.2 Metatheory of relSTLC

Lemma 20 (Reflexivity of algorithmic subtyping)**.** *The reflexivity property holds for algorithm subtyping for RelSTLC, denoted* $\models \tau \leq \tau$.

Proof. By induction on the binary type.

Case

$\mathbf{bool}_r$

It is proved by **alg-bool-r**.

Case

$\mathbf{bool}_u$

It is proved by **alg-bool-u**.

Case

$\tau_1 \to \tau_2$

By **alg-**$\to$, TS: $\models \tau_1 \leq \tau_1$ and $\models \tau_2 \leq \tau_2$, which is obtained by IH on τ_1 and τ_2 respectively.

□

Lemma 21 (Transitivity of algorithmic subtyping)**.** *The transitivity property holds for algorithm subtyping for RelSTLC.*
$\models \tau_1 \leq \tau_2$ *and* $\models \tau_2 \leq \tau_3$, *then* $\models \tau_1 \leq \tau_3$.

Proof. By simultaneous induction on the first two subtyping derivation.

Case

$$\frac{}{\models \mathbf{bool}_r \leq \mathbf{bool}_r}\ \textbf{alg-bool-r}\ ,\quad \frac{}{\models \mathbf{bool}_r \leq \mathbf{bool}_r}\ \textbf{alg-bool-r}$$

TS: $\models \mathrm{bool}_r \leq \mathrm{bool}_r$, which is proved by **alg-bool-r**.

Case

$$\frac{}{\models \mathbf{bool}_u \leq \mathbf{bool}_u}\ \textbf{alg-bool-u},\quad \frac{}{\models \mathbf{bool}_u \leq \mathbf{bool}_u}\ \textbf{alg-bool-u}$$

TS: $\models \text{bool}_u \leq \text{bool}_u$, which is proved by **alg-bool-u**.

Case

$$\frac{}{\models \mathbf{bool}_r \leq \mathbf{bool}_u}\ \textbf{alg-bool}\,,\quad \frac{}{\models \mathbf{bool}_u \leq \mathbf{bool}_u}\ \textbf{alg-bool-u}$$

TS: $\models \text{bool}_r \leq \text{bool}_u$, which is proved by **alg-bool**.

Case

$$\frac{}{\models \mathbf{bool}_r \leq \mathbf{bool}_r}\ \textbf{alg-bool-r}\,,\quad \frac{}{\models \mathbf{bool}_r \leq \mathbf{bool}_u}\ \textbf{alg-bool}$$

TS: $\models \text{bool}_r \leq \text{bool}_u$, which is proved by **alg-bool**.

Case

$$\frac{\models \tau_1' \leq \tau_1 \qquad \models \tau_2 \leq \tau_2'}{\models \tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'}\ \textbf{alg-}\to,\quad \frac{\models \tau_1'' \leq \tau_1' \qquad \models \tau_2' \leq \tau_2''}{\models \tau_1' \to \tau_2' \leq \tau_1'' \to \tau_2''}\ \textbf{alg-}\to$$

TS: $\models \tau_1 \to \tau_2 \leq \tau_1'' \to \tau_2''$.

By IH on $\models \tau_1'' \leq \tau_1'$ and $\models \tau_1' \leq \tau_1$, $\models \tau_1'' \leq \tau_1$.

By IH on $\models \tau_2' \leq \tau_1''$ and $\models \tau_2 \leq \tau_2'$, $\models \tau_2 \leq \tau_2''$.

It is proved by using the two statements and rule **alg-**$\to$.

□

Next, we can show that the algorithmic formulation of subtyping ($\tau_1 \leq \tau_2$) coincides with the declarative formulation of subtyping ($\tau_1 \sqsubseteq \tau_2$). We do this in two steps.

Lemma 22 (Soundness of algorithmic subtyping)**.** *If* $\models \tau \leq \tau'$ *then* $\models \tau \sqsubseteq \tau'$.

Proof. Proof is by straightforward induction on the given algorithmic subtyping derivation.

Case

$$\frac{\models \tau_1' \leq \tau_1 \qquad \models \tau_2 \leq \tau_2'}{\models \tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'}\ \textbf{alg-}\to$$

TS: $\models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'$

By IH on $\models \tau_1' \leq \tau_1$, $\models \tau_1' \sqsubseteq \tau_1$.

By IH on $\models \tau_2 \leq \tau_2'$, $\models \tau_2 \sqsubseteq \tau_2'$.

It is proved using these two statements and subtyping rule $\to$.

Case

$$\frac{}{\models \mathbf{bool}_r \leq \mathbf{bool}_u}\ \textbf{alg-bool}$$

TS: $\models \text{bool}_r \sqsubseteq \text{bool}_u$, which is proved by subtyping rule **bool**.

Case

$$\frac{}{\models \mathbf{bool}_u \leq \mathbf{bool}_u}\ \textbf{alg-bool-u}$$

TS: $\models \text{bool}_u \sqsubseteq \text{bool}_u$, which is proved by subtyping rule **refl**.

Case

$$\frac{}{\models \mathbf{bool}_r \leq \mathbf{bool}_r}\ \textbf{alg-bool-r}$$

TS: $\models \text{bool}_r \sqsubseteq \text{bool}_r$, which is proved by subtyping rule **refl**.

□

Lemma 23 (Completeness of algorithmic subtyping)**.** *If* $\models \tau \sqsubseteq \tau'$ *then* $\models \tau \leq \tau'$.

Proof. Proof is by induction on the given subtyping derivation.

Case

$$\frac{\models \tau_1' \sqsubseteq \tau_1 \qquad \models \tau_2 \sqsubseteq \tau_2'}{\models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'}\ \to$$

TS: $\models \tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'$

By IH on $\models \tau_1' \sqsubseteq \tau_1$, $\models \tau_1' \leq \tau_1$.

By IH on $\models \tau_2 \sqsubseteq \tau_2'$, $\models \tau_2 \leq \tau_2'$.

It is proved using these two statements and algorithmic subtyping rule **alg-→**.

Case

$$\frac{}{\models \mathbf{bool}_r \sqsubseteq \mathbf{bool}_u}\ \mathbf{bool}$$

TS: $\models \text{bool}_r \leq \text{bool}_u$, which is proved by algorithmic subtyping rule **alg-bool**.

Case

$$\frac{}{\models \tau \sqsubseteq \tau}\ \mathbf{refl}$$

TS: $\models \tau \leq \tau$, which is proved by Lemma 20.

Case

$$\frac{\models \tau_1 \sqsubseteq \tau_2 \qquad \models \tau_2 \sqsubseteq \tau_3}{\models \tau_1 \sqsubseteq \tau_3}\ \mathbf{trans}$$

TS: $\models \tau_1 \leq \tau_3$.

By IH on $\models \tau_1 \sqsubseteq \tau_2$, $\models \tau_1 \leq \tau_2$.

By IH on $\models \tau_2 \sqsubseteq \tau_3$, $\models \tau_2 \leq \tau_3$.

It is proved using these two statements and Lemma 21.

□

Lemma 24 (Soundness of algorithmic typing)**.** *The following holds,* $|.|$ *is the annotation erasure function.*

1. *If* $\Gamma \vdash t_1 \backsim t_2 \downarrow \tau$ *then* $\Gamma \vdash |t_1| \backsim |t_2| : \tau$.
2. *If* $\Gamma \vdash t_1 \backsim t_2 \uparrow \tau$ *then* $\Gamma \vdash |t_1| \backsim |t_2| : \tau$.

Proof. Proof is by simultaneous induction on the given algorithmic checking and inference derivations and Lemma 22.

Case

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x \backsim x \uparrow \tau}\ \textbf{alg-r-var}$$

TS: $\Gamma \vdash |x| \backsim |x| : \tau$

It is proved by using the premise $\Gamma(x) = \tau$ and typing rule **r-var**.

Case

$$\frac{\Gamma \vdash t \backsim t' \uparrow \textbf{bool}_r \qquad \Gamma \vdash t_1 \backsim t_1' \downarrow \tau \qquad \Gamma \vdash t_2 \backsim t_2' \downarrow \tau}{\Gamma \vdash \textbf{if } t \textbf{ then } t_1 \textbf{ else } t_2 \backsim \textbf{if } t' \textbf{ then } t_1' \textbf{ else } t_2' \downarrow \tau}\ \textbf{alg-r-if}$$

TS: $\Gamma \vdash |\text{if } t \text{ then } t_1 \text{ else } t_2| \backsim |\text{if } t' \text{ then } t_1' \text{ else } t_2'| : \tau$

By IH2 on $\Gamma \vdash t \backsim t' \uparrow \text{bool}_r$,$\Gamma$

$jstype|t||t'|\text{bool}_r$.

By IH on $\Gamma \vdash t_1 \backsim t_1' \downarrow \tau, \Gamma \vdash |t_1| \backsim |t_1'| : \tau$.

By IH on $\Gamma \vdash t_2 \backsim t_2' \downarrow \tau, \Gamma \vdash |t_2| \backsim |t_2'| : \tau$.

Because $|\text{if } t \text{ then } t_1 \text{ else } t_2| = \text{if } |e| \text{ then } |t_1| \text{ else } |t_2|$. By using these three statements and typing rule **r-if**, we get the conclusion.

Case

$$\frac{\Gamma, x : \tau_1 \vdash t_1 \backsim t_2 \downarrow \tau_2}{\Gamma \vdash \lambda x.t_1 \backsim \lambda x.t_2 \downarrow \tau_1 \to \tau_2}\ \textbf{alg-r-fix}$$

TS: $\Gamma \vdash \lambda x.|t_1| \backsim \lambda x.|t_1| : \tau_1 \to \tau_2$.

By IH on $\Gamma, x : \tau_1 \vdash t_1 \backsim t_2 \downarrow \tau_2, \Gamma, x : \tau_1 \vdash |t_1| \backsim |t_2| : \tau_2$.

It is proved by the above statment and tying rule **r-fix**.

Case

$$\frac{\Gamma \vdash t \backsim t' \uparrow \tau' \qquad \models \tau' \leq \tau}{\Gamma \vdash t \backsim t' \downarrow \tau}\ \textbf{alg-}\uparrow\downarrow$$

TS: $\Gamma \vdash |t| \backsim |t'| : \tau$.

By Lemma 22 on $\models \tau \leq \tau', \models \tau \sqsubseteq \tau'$.

By IH2 on $\Gamma \vdash t \backsim t' \uparrow \tau, \Gamma, \vdash |t| \backsim |t'| : \tau$.

It is proved by the above statment and tying rule **r-⊑**.

Case

$$\frac{\Gamma \vdash t \backsim t' \downarrow \tau}{\Gamma \vdash (t:\tau) \backsim (t':\tau) \uparrow \tau}\ \textbf{alg-r-anno-}\uparrow$$

TS: $\Gamma \vdash |(t:\tau)| \backsim |(t':\tau)| : \tau$, which is simplified as $\Gamma \vdash t \backsim t' : \tau$.

By IH on $\Gamma \vdash t \backsim t' \downarrow \tau$, $\Gamma \vdash t \backsim t' : \tau$.

It is proved by the above statement.

□

Lemma 25 (Completeness of algorithmic typing)**.** *If* $\Gamma \vdash t_1 \backsim t_2 : \tau$ *then there exists* t_1' *and* t_2' *such that* $\Gamma \vdash t_1' \backsim t_2' \downarrow \tau$ *and* $|t_i'| = t_i$ *for* $i \in \{1,2\}$.

Proof. Proof is by induction on the given typing derivation and Lemma 23.

Case

$$\frac{\Gamma, x:\tau_1 \vdash t_1 \backsim t_2 : \tau_2}{\Gamma \vdash \lambda x.t_1 \backsim \lambda x.t_2 : \tau_1 \to \tau_2}\ \textbf{r-fix}$$

By IH on $\Gamma, x:\tau_1 \vdash t_1 \backsim t_2 : \tau_2$, $\exists t_1', t_2'.\Gamma, x:\tau_1 \vdash t_1' \backsim t_2' \downarrow \tau_2$ and $|t_i'| = t_i$.

Then, using the above statement and algorithmic typing rule **alg-r-fix**, we can construct the derivation where $t_1'' = \lambda x.t_1'$ and $t_2'' = \lambda x.t_2'$.

$$\Gamma \vdash t_1'' \backsim t_2'' \downarrow \tau_1 \to \tau_2$$

And we show that $|t_i''| = |\lambda x.t_i'| = \lambda x.|t_i'| = \lambda x.t_i'$.

Case

$$\frac{\Gamma \vdash t \backsim t' : \textbf{bool}_r \qquad \Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Gamma \vdash t_2 \backsim t_2' : \tau}{\Gamma \vdash \textbf{if}\ t\ \textbf{then}\ t_1\ \textbf{else}\ t_2 \backsim \textbf{if}\ t'\ \textbf{then}\ t_1'\ \textbf{else}\ t_2' : \tau}\ \textbf{r-if}$$

By IH on $\Gamma \vdash t \backsim t' : \text{bool}_r$, $\exists t'', t'''.\Gamma, \vdash t'' \backsim t''' \downarrow \text{bool}_r$ and $|t''| = e, |t'''| = t'$.

By using algorithmic typing rule **alg-r-anno-**$\uparrow$, $\Gamma \vdash (t'' : \text{bool}_r) \backsim (t''' : \text{bool}_r) \uparrow \text{bool}_r$.

By IH on $\Gamma \vdash t_1 \backsim t_1' : \tau$, $\exists t_1'', t_1'''.\Gamma, \vdash t_1'' \backsim t_1''' \downarrow \tau$ and $|t_1''| = t_1, |t_1'''| = t_1'$.

By IH on $\Gamma \vdash t_2 \backsim t_2' : \tau, \exists t_2'', t_2'''.\Gamma, \vdash t_2'' \backsim t_2''' \downarrow \tau$ and $|t_2''| = t_2, |t_2'''| = t_2'$.

Then, using the above statement and algorithmic typing rule **alg-r-if**, we can construct the derivation

where $t_3 = \text{if } (t'' : \text{bool}_r) \text{ then } t_1'' \text{ else } t_2''$ and $t_4 = \text{if } (t''' : \text{bool}_r) \text{ then } t_1''' \text{ else } t_2'''$.

$$\Gamma \vdash t_3 \backsim t_4 \downarrow \tau$$

And we show that

$|t_3| = |\text{if } (t'' : \text{bool}_r) \text{ then } t_1'' \text{ else } t_2''|$

$= \text{if } |(t'' : \text{bool}_r)| \text{ then } |t_1''| \text{ else } |t_2''| = \text{if } e \text{ then } t_1 \text{ else } t_2$, similarly for t_4.

Case

$$\frac{\Gamma \vdash t_1 \backsim t_1' : \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 \backsim t_2' : \tau_1}{\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' : \tau_2}\ \textbf{r-app}$$

By IH on $\Gamma \vdash t_1 \backsim t_1' : \tau_1 \to \tau_2, \exists t_1'', t_1'''.\Gamma, \vdash t_1'' \backsim t_1''' \downarrow \tau_1 \to \tau_2$ and $|t_1''| = t_1, |t_1'''| = t_1'$.

By using **alg-r-anno-**$\uparrow$, $\Gamma \vdash (t_1'' : \tau_1 \to \tau_2) \backsim (t_1''' : \tau_1 \to \tau_2) \uparrow \tau_1 \to \tau_2$.

By IH on $\Gamma \vdash t_2 \backsim t_2' : \tau_1, \exists t_2'', t_2'''.\Gamma, \vdash t_2'' \backsim t_2''' \downarrow \tau_1$ and $|t_2''| = t_2, |t_2'''| = t_2'$.

Then, using the above statement and algorithmic typing rule **alg-r-app**, we can construct the derivation where $t_3 = t_1''\ t_2''$ and $t_4 = t_1'''\ t_2'''$.

$$\Gamma \vdash t_3 \backsim t_4 \downarrow \tau_2$$

And we show that $|t_3| = |t_1''\ t_2''| = |t_1''|\,|t_2''| = t_1\ t_2$, , similarly for t_4.

Case

$$\frac{\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \models \tau \sqsubseteq \tau'}{\Gamma \vdash t_1 \backsim t_2 : \tau'}\ \textbf{r-}\sqsubseteq$$

By IH on $\Gamma \vdash t_1 \backsim t_2 : \tau, \exists t_1', t_2'.\Gamma, \vdash t_1' \backsim t_2' \downarrow \tau$ and $|t_1'| = t_1, |t_2'| = t_2$.

By using algorithmic typing rule **alg-r-anno-**$\uparrow$, $\Gamma \vdash (t_1' : \tau) \backsim (t_2' : \tau) \uparrow \tau$.

By using Lemma 22 on premise $\models \tau \sqsubseteq \tau'$, $\models \tau \leq \tau'$.

Then, using the above statement and algorithmic typing rule **alg-r-**$\uparrow\downarrow$, we can construct the derivation where $t_3 = (t_1' : \tau)$ and $t_4 = (t_2' : \tau)$.

$$\Gamma \vdash t_3 \backsim t_4 \downarrow \tau$$

And we show that $|t_3| = |(t_1' : \tau)| = |t_1'| = t_1$, similarly for t_4.

□

B.2 RELREF

B.2.1 Syntax of RelRef

$$
\begin{array}{llll}
\text{Types} & \tau & ::= & \text{bool}_r \mid \text{bool}_u \mid \tau_1 \to \tau_2 \mid \text{list}[n]^\alpha\,\tau \mid \forall i{::}S.\,\tau \mid \exists i{::}S.\,\tau \\
 & & & \mid \Box\tau \mid C \,\&\, \tau \mid C \supset \tau \\
\text{Terms} & t & ::= & x \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \text{fix } f(x).t \mid t_1\ t_2 \\
 & & & \mid (\ \text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2) \mid \Lambda.t \mid t[\,] \mid \text{let } x = t_1 \text{ in } t_2 \\
 & & & \mid \text{pack } t \mid \text{unpack } t_1 \text{ as } x \text{ in } t_2 \mid \text{clet } t_1 \text{ as } x \text{ in } t_2 \mid \text{celim } t \\
 & & & \mid \text{nil } \mid \text{cons}(t_1, t_2) \\
\text{Value} & v & := & \text{true} \mid \text{false} \mid \text{fix } f(x).v \mid \text{nil } \mid \text{cons}(t_1, t_2) \mid \Lambda.t \mid \text{pack } v \\
\text{Indices} & I, n & ::= & i \mid 0 \mid I+1 \mid I_1 + I_2 \mid I_1 - I_2 \mid \frac{I_1}{I_2} \mid I_1 \cdot I_2 \mid \lceil I \rceil \mid \lfloor I \rfloor \mid \\
 & ,\alpha & & \texttt{min}(I_1, I_2) \mid \texttt{max}(I_1, I_2).
\end{array}
$$

Figure B.9: Syntax of values and terms,index terms in RelRef.

$$
\begin{array}{llll}
 & \tau & ::= & \text{bool}_r \mid \text{bool}_u \mid \tau_1 \to \tau_2 \mid \text{list}[n]^\alpha\,\tau \mid \forall i{::}S.\,\tau \mid \exists i{::}S.\,\tau \mid \Box\tau \mid C \,\&\, \tau \mid C \supset \tau \\
\text{Terms} & t & ::= & x \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \text{fix } f(x).t \mid \text{fix}_{NC}\ f(x).t \mid t_1\ t_2 \mid \text{NC } t \\
 & & & \mid \text{split } (t_1, t_2) \text{ with } C \mid \text{contra } t \mid \text{der } t \mid \Lambda i.t \mid t[I] \mid \text{pack } t \text{ with } I \\
 & & & \mid \text{unpack } t_1 \text{ as } (x, i) \text{ in } t_2 \mid \text{clet } t_1 \text{ as } x \text{ in } t_2 \mid \text{celim } t \mid \text{let } x = t_1 \text{ in } t_2 \\
 & & & \mid \text{cons}_{NC}(t_1, t_2) \mid \text{cons}_C(t_1, t_2) \mid \begin{pmatrix} \text{case } t \text{ of nil } \to t_1 \\ \mid h ::_{NC} tl \to t_2 \mid h ::_C tl \ \to t_3 \end{pmatrix} \\
\text{Value} & v & := & \text{true} \mid \text{false} \mid \text{fix } f(x).v \mid \text{fix}_{NC}\ f(x).t \mid \text{nil } \mid \text{cons}(t_1, t_2) \mid \text{pack } t \text{ with } I \\
 & & & \mid \Lambda i.t \mid \text{cons}_{NC}(v_1, v_2) \mid \text{cons}_C(v_1, v_2)
\end{array}
$$

Figure B.10: Syntax of values and terms in RelRef CORE.

$$
\begin{array}{lcl}
\text{Types} & \tau & ::= \quad \mathrm{bool}_r \mid \mathrm{bool}_u \mid \tau_1 \to \tau_2 \mid \mathrm{list}[n]^{\alpha}\,\tau \mid \forall i{::}S.\,\tau \mid \exists i{::}S.\,\tau \mid \Box\tau \mid C \,\&\, \tau \mid C \supset \tau \\
\text{Terms} & t & ::= \quad x \mid \mathrm{true} \mid \mathrm{false} \mid \mathrm{if}\ t\ \mathrm{then}\ t_1\ \mathrm{else}\ t_2 \mid \mathrm{fix}\ f(x).t \mid \mathrm{fix}_{NC}\ f(x).t \mid t_1\ t_2 \mid \mathrm{NC}\ t \\
& & \quad \mid \mathrm{split}\ (t_1, t_2)\ \mathrm{with}\ C \mid \mathrm{contra}\ t \mid \mathrm{der}\ t \mid \Lambda i.t \mid t[I] \mid \mathrm{pack}\ t\ \mathrm{with}\ I \\
& & \quad \mid \mathrm{unpack}\ t_1\ \mathrm{as}\ (x, i)\ \mathrm{in}\ t_2 \mid (t : \tau) \mid \mathrm{clet}\ t_1\ \mathrm{as}\ x\ \mathrm{in}\ t_2 \\
& & \quad \mid \mathrm{celim}\ t \mid \mathrm{let}\ x = t_1\ \mathrm{in}\ t_2 \\
& & \quad \mid \mathrm{cons}_{NC}(t_1, t_2) \mid \mathrm{cons}_C(t_1, t_2) \mid \begin{pmatrix} \mathrm{case}\ t\ \mathrm{of\ nil}\ \to t_1 \\ \mid h ::_{NC} tl \to t_2 \mid h ::_C tl\ \to t_3 \end{pmatrix} \\
\text{Value} & v & := \quad \mathrm{true} \mid \mathrm{false} \mid \mathrm{fix}\ f(x).v \mid \mathrm{fix}_{NC}\ f(x).t \mid \mathrm{nil}\ \mid \mathrm{cons}(t_1, t_2) \mid \mathrm{pack}\ t\ \mathrm{with}\ I \\
& & \quad \mid \Lambda i.t \mid \mathrm{cons}_{NC}(v_1, v_2) \mid \mathrm{cons}_C(v_1, v_2)
\end{array}
$$

Figure B.11: Syntax of values and terms in BiRelRef.

$$\frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \backsim x : \tau}\ \textbf{rr-var} \qquad \frac{\mathtt{b} \in \{\text{true}, \text{false}\}}{\Delta;\Phi_a;\Gamma \vdash \mathtt{b} \backsim \mathtt{b} : \text{bool}_r}\ \textbf{rr-bool}$$

$$\frac{\mathtt{b}_1, \mathtt{b}_2 \in \{\text{true}, \text{false}\}}{\Delta;\Phi_a;\Gamma \vdash \mathtt{b}_1 \backsim \mathtt{b}_2 : \text{bool}_u}\ \textbf{rr-u-bool}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \text{bool}_r \qquad \Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \tau}{\Delta;\Phi_a;\Gamma \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 \backsim \text{if } t' \text{ then } t_1' \text{ else } t_2' : \tau}\ \textbf{rr-if}$$

$$\frac{\Delta;\Phi_a;x:\tau_1, f:\tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t_1 \backsim \text{fix } f(x).t_2 : \tau_1 \to \tau_2}\ \textbf{rr-fix}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \tau_1 \to \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \tau_1}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' : \tau_2}\ \textbf{rr-app}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t : \tau \\ \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \backsim t : \Box\tau}\ \textbf{rr-nochange}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau'}\ \textbf{rr-}\sqsubseteq$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\Box(\tau_1 \to \tau_2), \Gamma \vdash t \backsim t : \tau_2 \\ \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash \text{fix } f(x).t \backsim \text{fix } f(x).t : \Box(\tau_1 \to \tau_2)}\ \textbf{rr-fixNC}$$

$$\frac{\begin{array}{c}i :: S, \Delta;\Phi_a;\Gamma \vdash t \backsim t' : \tau \\ i \notin \text{FIV}(\Phi_a;\Gamma)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \Lambda.t \backsim \Lambda.t' : \forall i{::}S.\tau}\ \textbf{rr-iLam} \qquad \frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \forall i{::}S.\tau \\ \Delta \vdash I :: S\end{array}}{\Delta;\Phi_a;\Gamma \vdash t[\,] \backsim t'[\,] : \tau\{I/i\}}\ \textbf{rr-iApp}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \text{pack } t \backsim \text{pack } t' : \exists i{::}S.\tau}\ \textbf{rr-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \exists i{::}S.\tau_1 \\ i :: S, \Delta;\Phi_a;x:\tau_1, \Gamma \vdash t_2 \backsim t_2' : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } x \text{ in } t_2 \backsim \text{unpack } t_1' \text{ as } x \text{ in } t_2' : \tau_2}\ \textbf{rr-unpack1}$$

Figure B.12: Typing rules of RelRef, part 1.

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' : \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : C \supset \tau}\ \textbf{rr-c-implI} \qquad \frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : C \supset \tau \qquad \Delta;\Phi_a \vDash C}{\Delta;\Phi_a;\Gamma \vdash \text{celim } t \backsim \text{celim } t' : \tau}\ \textbf{rr-c-implE}$$

$$\frac{\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{nil } \backsim \text{nil } : \text{list}[0]^\alpha\,\tau}\ \textbf{rr-nil}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \text{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}(t_1,t_2) \backsim \text{cons}(t_1',t_2') : \text{list}[n+1]^{\alpha+1}\,\tau}\ \textbf{rr-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \text{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}(t_1,t_2) \backsim \text{cons}(t_1',t_2') : \text{list}[n+1]^{\alpha}\,\tau}\ \textbf{rr-cons2}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \text{list}[n]^\alpha\,\tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau, tl : \text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h:\tau, tl : \text{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{ case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2 \backsim \text{ case } t' \text{ of nil } \to t_1' \mid h :: tl \to t_2' : \tau'}\ \textbf{rr-caseL}$$

$$\frac{\Delta;\Phi_a \vDash C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' : \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : C \,\&\, \tau}\ \textbf{rr-c-andI}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : C \,\&\, \tau_1 \qquad \Delta;\Phi_a \wedge C;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \backsim \text{clet } t_1' \text{ as } x \text{ in } t_2' : \tau_2}\ \textbf{rr-c-andE}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \Delta \vdash C\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau}\ \textbf{rr-split}$$

$$\frac{\Delta;\Phi_a \vDash \bot \qquad \Delta;\Phi_a \vdash \Gamma\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau}\ \textbf{rr-contra}$$

Figure B.13: Typing rules of RelRef, part 2.

$$\frac{}{\models \mathrm{bool}_r \sqsubseteq \mathrm{bool}_u}\ \textbf{bool} \qquad \frac{\models \tau_1' \sqsubseteq \tau_1 \qquad \models \tau_2 \sqsubseteq \tau_2'}{\models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'} \to \qquad \frac{}{\models \tau \sqsubseteq \tau}\ \textbf{refl}$$

$$\frac{}{\models \Box(\tau_1 \to \tau_2) \sqsubseteq \Box\tau_1 \to \Box\tau_2} \to \Box\textbf{diff}$$

$$\frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \leq \alpha' \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \sqsubseteq \mathrm{list}[n']^{\alpha'}\,\tau'}\ \textbf{l1}$$

$$\frac{\Delta;\Phi_a \models \alpha \doteq 0}{\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \sqsubseteq \mathrm{list}[n]^{\alpha}\,\Box\tau}\ \textbf{l2} \qquad \frac{}{\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\Box\tau \sqsubseteq \Box\,(\mathrm{list}[n]^{\alpha}\,\tau)}\ \textbf{l}\Box$$

$$\frac{}{\Delta;\Phi_a \models \mathrm{int} \sqsubseteq \Box\,\mathrm{int}}\ \textbf{int-}\Box \qquad \frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \tau}\ \textbf{T} \qquad \frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \Box\Box\tau}\ \textbf{D}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2}{\Delta;\Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2}\ \textbf{B-}\Box \qquad \frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2 \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_3}{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_3}\ \textbf{trans}$$

$$\frac{i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \forall i{::}S.\,\tau \sqsubseteq \forall i{::}S.\,\tau'}\ \forall_{\textbf{diff}} \qquad \frac{}{\Delta;\Phi_a \models \Box\,(\forall i{::}S.\,\tau) \sqsubseteq \forall i{::}S.\,\Box\tau}\ \forall\,\Box$$

$$\frac{i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \exists i{::}S.\,\tau \sqsubseteq \exists i{::}S.\,\tau'}\ \exists \qquad \frac{}{\Delta;\Phi_a \models \exists i{::}S.\,\Box\tau \sqsubseteq \Box\,(\exists i{::}S.\,\tau)}\ \exists\Box$$

$$\frac{\Delta;\Phi_a \wedge C' \models C \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models C \supset \tau \sqsubseteq C' \supset \tau'}\ \textbf{c-impl} \qquad \frac{}{\Delta;\Phi_a \models \Box\,(C \supset \tau) \sqsubseteq C \supset \Box\tau}\ \textbf{c-impl-}\Box$$

$$\frac{\Delta;\Phi_a \wedge C \models C' \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models C \,\&\, \tau \sqsubseteq C' \,\&\, \tau'}\ \textbf{c-and} \qquad \frac{}{\Delta;\Phi_a \models C \,\&\, \Box\tau \sqsubseteq \Box\,(C \,\&\, \tau)}\ \textbf{c-and-}\Box$$

Figure B.14: Subtyping rules of RelRef.

$\boxed{\Delta;\Phi_a \models \tau_1 \equiv \tau_2}$ checks whether τ_1 is equivalent to τ_2.

$$\frac{\Delta;\Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta;\Phi_a \models \tau_2 \equiv \tau_2'}{\Delta;\Phi_a \models \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'}\ \textbf{eq-fun} \qquad \frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \doteq \alpha' \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \equiv \mathrm{list}[n']^{\alpha'}\,\tau'}\ \textbf{eq-list}$$

$$\frac{\Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \Box\tau \equiv \Box\tau'}\textbf{eq-B-}\Box \qquad \frac{i,\Delta;\Phi_a \models \tau \equiv \tau' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \exists i{::}S.\,\tau \equiv \exists i{::}S.\,\tau'}\ \textbf{eq-}\exists$$

$$\frac{\Delta;C' \wedge \Phi_a \models C \qquad \Delta;C \wedge \Phi_a \models C' \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau'}\textbf{eq-c-prod}$$

$$\frac{\Delta;C \wedge \Phi_a \models C' \qquad \Delta;C' \wedge \Phi_a \models C \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models C \supset \tau \equiv C' \supset \tau'}\textbf{eq-c-impl}$$

$$\frac{i,\Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \forall i{::}S.\,\tau \equiv \forall i{::}S.\,\tau'}\textbf{eq-}\forall$$

Figure B.15: Type equivalence rules of RelRef.

$$\frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \backsim x :^c \tau}\ \textbf{c-r-var}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t_1 \backsim \text{fix } f(x).t_2 :^c \tau_1 \to \tau_2}\ \textbf{c-r-fix}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\square(\tau_1 \to \tau_2), \square\Gamma \vdash t \backsim t :^c \tau_2}{\Delta;\Phi_a;\square\Gamma, \Gamma' \vdash \text{fix}_{NC}\ f(x).t \backsim \text{fix}_{NC}\ f(x).t :^c \square(\tau_1 \to \tau_2)}\ \textbf{c-r-fixNC}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \text{list}[n]^\alpha\, \tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}_C(t_1, t_2) \backsim \text{cons}_C(t_1', t_2') :^c \text{list}[n+1]^{\alpha+1}\, \tau}\ \textbf{c-r-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \square\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \text{list}[n]^\alpha\, \tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}_{NC}(t_1, t_2) \backsim \text{cons}_{NC}(t_1', t_2') :^c \text{list}[n+1]^\alpha\, \tau}\ \textbf{c-r-cons2}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \text{list}[n]^\alpha\, \tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' :^c \tau' \\ i, \Delta;\Phi_a \wedge n = i+1;h:\square\tau, tl:\text{list}[i]^\alpha\, \tau, \Gamma \vdash t_2 \backsim t_2' :^c \tau' \\ i, \beta, \Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h':\tau, tl:\text{list}[i]^\beta\, \tau, \Gamma \vdash t_3 \backsim t_3' :^c \tau'\end{array}}{\begin{array}{rcl}\text{case } t \text{ of nil } & \to t_1 & \text{case } t' \text{ of nil } \to t_1' \\ \Delta;\Phi_a;\Gamma \vdash \mid h ::_N tl & \to t_2 & \backsim \mid h ::_N tl \to t_2' \qquad :^c \tau' \\ \mid h' ::_C tl' & \to t_3 & \mid h' ::_C tl' \to t_3'\end{array}}\ \textbf{c-r-caseL}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 :^c \square\tau}{\Delta;\Phi_a;\Gamma \vdash \text{der } t_1 \backsim \text{der } t_2 :^c \tau}\ \textbf{c-der} \qquad \frac{\Delta;\Phi_a;\square\Gamma \vdash t \backsim t :^c \tau}{\Delta;\Phi_a;\square\Gamma, \Gamma' \vdash \text{NC } t \backsim \text{NC } t :^c \square\tau}\ \textbf{c-nochange}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau'}\ \textbf{c-r-}\equiv$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \backsim t_2 :^c \tau \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1' \backsim t_2' :^c \tau}{\Delta;\Phi_a;\Gamma \vdash \text{split } (t_1, t_1') \text{ with } C \backsim \text{split } (t_2, t_2') \text{ with } C :^c \tau}\ \textbf{c-r-split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Gamma\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{contra } t_1 \backsim \text{contra } t_2 :^c \tau}\ \textbf{c-r-contra}$$

Figure B.16: Typing rules of RelRef Core, part 1.

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \forall i{::}S.\tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash t[I] \backsim t'[I] :^c \tau\{I/i\}}\ \textbf{c-r-iApp}$$

$$\frac{i :: S, \Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau \qquad i \notin \mathrm{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda i.t \backsim \Lambda i.t' :^c \forall i{::}S.\tau}\ \textbf{c-r-iLam}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' :^c \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c C \supset \tau}\ \textbf{c-r-cimpI} \qquad \frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash \mathrm{celim}\ t \backsim \mathrm{celim}\ t' :^c \tau}\ \textbf{c-r-cimplE}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' :^c \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c C \,\&\, \tau}\ \textbf{c-r-candI}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c C \,\&\, \tau_1 \qquad \Delta;\Phi_a \wedge C; x : \tau_1, \Gamma \vdash t_2 \backsim t_2' :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{clet}\ t_1\ \mathrm{as}\ x\ \mathrm{in}\ t_2 \backsim \mathrm{clet}\ t_1'\ \mathrm{as}\ x\ \mathrm{in}\ t_2' :^c \tau_2}\ \textbf{c-r-candE}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau_1 \to \tau_2 \\ \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \tau_1\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' :^c \tau_2}\ \textbf{c-r-app}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau_1 \qquad \Delta;\Phi_a; x : \tau_1, \Gamma \vdash t_2 \backsim t_2' :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{let}\ x = t_1\ \mathrm{in}\ t_2 \backsim \mathrm{let}\ x = t_1'\ \mathrm{in}\ t_2' :^c \tau_2}\ \textbf{c-r-let}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \mathrm{pack}\ t\ \mathrm{with}\ I \backsim \mathrm{pack}\ t'\ \mathrm{with}\ I :^c \exists i{::}S.\tau}\ \textbf{c-r-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \exists i{::}S.\tau_1 \\ i :: S, \Delta;\Phi_a; x : \tau_1, \Gamma \vdash t_2 \backsim t_2' :^c \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{unpack}\ t_1\ \mathrm{as}\ (x,i)\ \mathrm{in}\ t_2 \backsim \mathrm{unpack}\ t_1'\ \mathrm{as}\ (x,i)\ \mathrm{in}\ t_2' :^c \tau_2}\ \textbf{c-r-unpack1}$$

Figure B.17: Typing rules of RelRef Core, part 2.

$$\frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \backsim x \rightsquigarrow x \backsim x : \tau}\ \textbf{e-r-var}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t_1 \backsim \text{fix } f(x).t_2 \rightsquigarrow \text{fix } f(x).{t_1}^* \backsim \text{fix } f(x).{t_2}^* : \tau_1 \to \tau_2}\ \textbf{e-r-fix}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\square(\tau_1 \to \tau_2), \Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau_2 \\ \forall x_i \in dom(\Gamma),\ \ t_i = \text{coerce}_{\Gamma(x_i),\square\Gamma(x_i)} \qquad \forall x_i \in dom(\Gamma),\ \ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \square\Gamma(x) \\ t^{**} = \text{let } \overline{y_i = t_i\ x_i} \text{ in } \text{fix}_{NC}\ f(x).t^*\,[\overline{\text{der } y_i / x_i}]\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t \backsim \text{fix } f(x).t \rightsquigarrow t^{**} \backsim t^{**} : \square(\tau_1 \to \tau_2)}$$

e-r-fixNC

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : C \supset \tau}\ \textbf{e-r-c-impI}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash \text{celim } t \backsim \text{celim } t' \rightsquigarrow \text{celim } t^* \backsim \text{celim } {t'}^* : \tau}\ \textbf{e-r-c-implE}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : C\,\&\,\tau}\ \textbf{e-r-andI}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : C\,\&\,\tau_1 \\ \Delta;\Phi_a \wedge C;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau_2\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \backsim \text{clet } t_1' \text{ as } x \text{ in } t_2' \rightsquigarrow \text{clet } t_1^* \text{ as } x \text{ in } t_2^* \backsim \text{clet } t_1'^* \text{ as } x \text{ in } t_2'^* : \tau_2\end{array}}$$

e-r-c-andE

Figure B.18: Embedding typing rules of RelRef, part 1.

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \mathrm{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}(t_1,t_2) \backsim \mathrm{cons}(t_1',t_2') \rightsquigarrow \mathrm{cons}_C(t_1^*,t_2^*) \backsim \mathrm{cons}_C(t_1'^*,t_2'^*) : \mathrm{list}[n+1]^{\alpha+1}\,\tau}$$

e-r-cons1

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \mathrm{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}(t_1,t_2) \backsim \mathrm{cons}(t_1',t_2') \rightsquigarrow \mathrm{cons}_{NC}(t_1',t_2') \backsim \mathrm{cons}_{NC}(t_1^*,t_2^*) : \mathrm{list}[n+1]^{\alpha}\,\tau}$$

e-r-cons2

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau \\ \forall x_i \in dom(\Gamma),\ \ t_i = \mathrm{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \qquad \forall x_i \in dom(\Gamma),\ \ \Delta;\Phi_a \models \Gamma(x_i) \sqsubseteq \Box\Gamma(x_i)\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma,\Gamma' \\ \vdash t \backsim t \rightsquigarrow \mathrm{let}\ \overline{y_i = t_i\ x_i}\ \mathrm{in\ NC}\ t^*\,[\overline{\mathrm{der}\ y_i\,/\,x_i}] \backsim \mathrm{let}\ \overline{y_i = t_i\ x_i}\ \mathrm{in\ NC}\ t^*\,[\overline{\mathrm{der}\ y_i\,/\,x_i}] : \Box\tau\end{array}}$$

e-nochange

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : \mathrm{list}[n]^\alpha\,\tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h : \Box\tau, tl : \mathrm{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h : \tau, tl : \mathrm{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^{**} \backsim t_2'^{**} : \tau'\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\mathrm{case}\ t\ \mathrm{of\ nil}\ \to t_1 \\ \mid h :: tl \to t_2\end{array} \backsim \begin{array}{l}\mathrm{case}\ t\ \mathrm{of\ nil}\ \to t_1' \\ \mid h :: tl \to t_2'\end{array} \rightsquigarrow : \tau' \\ \begin{array}{l}\mathrm{case}\ t^*\ \mathrm{of\ nil}\ \to t_1^* \\ \mid h ::_{NC} tl \to t_2^* \\ \mid h ::_C tl \ \ \to t_2^{**}\end{array} \backsim \begin{array}{l}\mathrm{case}\ t'^*\ \mathrm{of\ nil}\ \to \\ \quad t_1'^* \\ \mid h ::_{NC} tl \to t_2'^* \\ \mid h ::_C tl \ \ \to t_2'^{**}\end{array}\end{array}}$$

e-r-caseL

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : \tau \qquad i \notin \mathrm{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda.t \backsim \Lambda.t' \rightsquigarrow \Lambda i.t^* \backsim \Lambda i.t'^* : \forall i{::}S.\tau}\ \textbf{e-r-iLam}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \forall i{::}S.\tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash t[] \backsim t'[] \rightsquigarrow t^*\,[I] \backsim t'^*\,[I] : \tau\{I\,/\,i\}}\ \textbf{e-r-iApp}$$

Figure B.19: Embedding rules of RelRef, part 2.

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \mathrm{coerce}_{\tau,\tau'}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t'\ t_1^* \backsim t'\ t_2^* : \tau'}\ \textbf{e-r-}\sqsubseteq$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \mathrm{pack}\ t \backsim \mathrm{pack}\ t' \rightsquigarrow \mathrm{pack}\ t^*\ \mathrm{with}\ I \backsim \mathrm{pack}\ t'^*\ \mathrm{with}\ I : \exists i{::}S.\,\tau}\ \textbf{e-r-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \exists i{::}S.\,\tau_1 \\ i :: S, \Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2) \\ t_1^{**} = \mathrm{unpack}\ t_1^*\ \mathrm{as}\ (x,i)\ \mathrm{in}\ t_2^* \qquad t_2^{**} = \mathrm{unpack}\ t_1'^*\ \mathrm{as}\ (x,i)\ \mathrm{in}\ t_2'^*\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{unpack}\ t_1\ \mathrm{as}\ x\ \mathrm{in}\ t_2 \backsim \mathrm{unpack}\ t_1'\ \mathrm{as}\ x\ \mathrm{in}\ t_2' \rightsquigarrow t_1^{**} \backsim t_2^{**} : \tau_2}\ \textbf{e-r-unpack}$$

$$\frac{\begin{array}{c}\Delta;C \wedge \Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \\ \Delta;\neg C \wedge \Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t_1^{**} \backsim t_2^{**} : \tau \qquad \Delta \vdash C\ \mathsf{wf}\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \mathrm{split}\ (t_1^*, t_1^{**})\ \mathrm{with}\ C \backsim \mathrm{split}\ (t_2^*, t_2^{**})\ \mathrm{with}\ C : \tau}\ \textbf{e-r-split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Gamma\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \mathrm{contra}\ t_1 \backsim \mathrm{contra}\ t_2 : \tau}\ \textbf{e-r-contra}$$

Figure B.20: Embedding rules of RelRef, part 3.

$$\frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2 \qquad \Phi = \Phi_1 \wedge \Phi_2}{\Delta;\psi_a;\Phi_a \models \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2' \Rightarrow \Phi} \textbf{alg-r-fun}$$

$$\frac{i,\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \forall i{::}S.\tau \equiv \forall i{::}S.\tau' \Rightarrow \forall i :: S.\Phi} \textbf{alg-r-forall}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi \qquad \Phi' = \Phi \wedge n \doteq n' \wedge \alpha \doteq \alpha'}{\Delta;\psi_a;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \equiv \mathrm{list}[n']^{\alpha'}\,\tau' \Rightarrow \Phi'} \textbf{alg-r-list}$$

$$\frac{i,\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi \qquad i \notin FV(\Phi_a)}{\Delta;\psi_a;\Phi_a \models \exists i{::}S.\tau \equiv \exists i{::}S.\tau' \Rightarrow \forall i :: S.\Phi} \textbf{alg-r-}\exists$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi} \textbf{alg-r-cprod}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \Box\tau_1 \equiv \Box\tau_2 \Rightarrow \Phi} \textbf{alg-r-}\Box$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models C \supset \tau \equiv C' \supset \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi} \textbf{alg-r-cimpl}$$

Figure B.21: Algorithmic type equivalence rules of RelRef.

$$\frac{\Delta;\psi_a;\Phi_a;\Box\Gamma \vdash t \ominus t \downarrow \tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma',\Box\Gamma \vdash \mathrm{NC}\ t \ominus \mathrm{NC}\ t \downarrow \Box\tau \Rightarrow \Phi}\ \textbf{alg-r-nochange-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \uparrow \Box\tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{der}\ t_1 \ominus \mathrm{der}\ t_2 \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-der-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;f:\tau_1 \to \tau_2, x:\tau_1,\Gamma \vdash t \ominus t' \downarrow \tau_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).t \ominus \mathrm{fix}\ f(x).t' \downarrow \tau_1 \to \tau_2 \Rightarrow \Phi}\ \textbf{alg-r-fix-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;f:\Box(\tau_1 \to \tau_2), x:\tau_1,\Box\Gamma \vdash t \ominus t' \downarrow \tau_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma',\Box\Gamma \vdash \mathrm{fix}_{NC}\ f(x).t \ominus \mathrm{fix}_{NC}\ f(x).t \downarrow \Box(\tau_1 \to \tau_2) \Rightarrow \Phi}\ \textbf{alg-r-fix-}\downarrow\Box$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \tau_1 \to \tau_2 \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \uparrow \tau_2 \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-app-}\uparrow$$

$$\frac{\Delta,\psi_a;\Phi \vdash \tau\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{nil}\ \ominus \mathrm{nil}\ \downarrow \mathrm{list}[n]^\alpha\,\tau \Rightarrow n \doteq 0}\ \textbf{alg-r-nil-}\downarrow$$

$$\frac{\begin{array}{c} i,\beta \in \mathrm{fresh}(\mathbb{N}) \\ \Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau \Rightarrow \Phi_1 \qquad \Delta;i,\beta,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \mathrm{list}[i]^\beta\,\tau \Rightarrow \Phi_2 \\ \Phi_2' = n \doteq (i+1) \wedge \exists\beta :: \mathbb{N}.\Phi_2 \wedge \alpha \doteq \beta + 1 \end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{cons}_C(t_1,t_2) \ominus \mathrm{cons}_C(t_1',t_2') \downarrow \mathrm{list}[n]^\alpha\,\tau \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}\ \textbf{alg-r-consC-}\downarrow$$

$$\frac{\begin{array}{c} i \in \mathrm{fresh}(\mathbb{N}) \qquad \Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \Box\tau \Rightarrow \Phi_1 \\ \Delta;i,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \mathrm{list}[i]^\alpha\,\tau \Rightarrow \Phi_2 \qquad \Phi_2' = \Phi_2 \wedge n \doteq (i+1) \end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{cons}_{NC}(t_1,t_2) \ominus \mathrm{cons}_{NC}(t_1',t_2') \downarrow \mathrm{list}[n]^\alpha\,\tau \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}\ \textbf{alg-r-consNC-}\downarrow$$

$$\frac{\begin{array}{c} \Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \mathrm{list}[n]^\alpha\,\tau \Rightarrow \Phi_e \qquad \Delta;\psi_a;n \doteq 0 \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau' \Rightarrow \Phi_1 \\ i :: \mathbb{N},\Delta;\psi_a;n \doteq i+1 \wedge \Phi_a;h:\Box\tau, tl:\mathrm{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \ominus t_2' \downarrow \tau' \Rightarrow \Phi_2 \\ i :: \mathbb{N},\beta :: \mathbb{N},\Delta;\psi_a;n \doteq i+1 \wedge \alpha \doteq \beta+1 \wedge \Phi_a;h:\tau, tl:\mathrm{list}[i]^\beta\,\tau,\Gamma \vdash t_3 \ominus t_3' \downarrow \tau' \Rightarrow \Phi_3 \\ \Phi_{body} = (n \doteq 0 \to \Phi_1) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to (\Phi_2 \wedge \forall\beta :: \mathbb{N}.(\alpha \doteq \beta+1) \to \Phi_3)) \end{array}}{\begin{array}{lll} \Delta;\psi_a;\Phi_a;\Gamma \vdash & \mathrm{case}\ t\ \mathrm{of\ nil}\ \to t_1 & \mathrm{case}\ t'\ \mathrm{of\ nil}\ \to t_1' \\ & \mid h ::_{NC} tl \to t_2 & \ominus \mid h ::_{NC} tl \to t_2' \qquad \downarrow \tau' \Rightarrow (\Phi_e \wedge \Phi_{body}) \\ & \mid h ::_C tl \to t_3 & \mid h ::_C tl \to t_3' \end{array}}$$

$$\textbf{alg-r-caseL-}\downarrow$$

Figure B.22: Algorithmic typing rules of RelRef, part 1.

$$\frac{\Gamma(x) = \tau}{\Delta;\psi_a;\Phi_a;\Gamma \vdash x \ominus x \uparrow \tau \Rightarrow \top}\ \textbf{alg-r-var-}\uparrow$$

$$\frac{i :: S, \Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \Lambda i.t \ominus \Lambda i.t' \downarrow \forall i{::}S.\tau \Rightarrow (\forall i :: S.\Phi)}\ \textbf{alg-r-iLam-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \forall i{::}S.\tau' \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Gamma \vdash e\,[I] \ominus t'\,[I] \uparrow \tau'\{I/i\} \Rightarrow \Phi}\ \textbf{alg-r-iApp-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau\{I/i\} \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{pack } t \text{ with } I \ominus \text{pack } t' \text{ with } I \downarrow \exists i{::}S.\tau \Rightarrow \Phi}\ \textbf{alg-r-pack-}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \exists i{::}S.\tau_1 \Rightarrow \Phi_1 \\ i :: S, \Delta;\psi_a;\Phi_a;x:\tau_1,\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2 \Rightarrow \Phi_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2) \qquad \Phi = (\Phi_1 \wedge \forall i :: S.\Phi_2)\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2 \ominus \text{unpack } t_1' \text{ as } (x,i) \text{ in } t_2' \downarrow \tau_2 \Rightarrow \Phi}$$

$$\textbf{alg-r-unpack-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi \wedge C;\Gamma \vdash t_1 \ominus t_1 \downarrow \tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \downarrow C \,\&\, \tau \Rightarrow C \wedge (C \to \Phi)}\ \textbf{alg-r-c-andI}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow C \,\&\, \tau_1 \Rightarrow \Phi_1 \\ \Delta;\psi_a;\Phi \wedge C;x:\tau_1,\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2 \Rightarrow \Phi_2 \qquad \Phi_2' = C \to \Phi_2\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \ominus \text{clet } t_1' \text{ as } x \text{ in } t_2' \downarrow \tau_2 \Rightarrow (\Phi_1 \wedge \Phi_2')}\ \textbf{alg-r-c-andE}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi \wedge C;\Gamma \vdash t_1 \ominus t_1 \downarrow \tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \downarrow C \supset \tau \Rightarrow (C \to \Phi)}\ \textbf{alg-r-c-implI}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow C \supset \tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{celim } t \ominus \text{celim } t' \uparrow \tau \Rightarrow (C \wedge \Phi)}\ \textbf{alg-r-c-implE}\downarrow$$

Figure B.23: Algorithmic typing rules of RelRef, part 2.

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models \tau' \equiv \tau \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-}\uparrow\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash \tau\ \mathsf{wf} \qquad \mathrm{FIV}(\tau) \in \Delta}{\Delta;\psi_a;\Phi_a;\Gamma \vdash (t:\tau) \ominus (t':\tau) \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-anno-}\uparrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;C \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau \Rightarrow \Phi_1 \\ \Delta;\psi_a;\neg C \wedge \Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau \Rightarrow \Phi_2 \qquad \Delta \vdash C\ \mathsf{wf}\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{split}\ (t_1,t_2)\ \mathrm{with}\ C \ominus \mathrm{split}\ (t_1',t_2')\ \mathrm{with}\ C \downarrow \tau \Rightarrow C \to \Phi_1 \wedge \neg C \to \Phi_2}$$

alg-r-split$\downarrow$

$$\frac{\Delta;\psi_a;\Phi_a \models \bot}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{contra}\ t \ominus \mathrm{contra}\ t' \downarrow \tau \Rightarrow \top}\ \textbf{alg-r-contra}\downarrow$$

$$\frac{\Gamma \vdash t \ominus t' \uparrow \mathrm{bool}_r \Rightarrow \Phi_1 \qquad \Gamma \vdash t_1 \ominus t_1' \downarrow \tau \Rightarrow \Phi_2 \qquad \Gamma \vdash t_2 \ominus t_2' \downarrow \tau \Rightarrow \Phi_3}{\Gamma \vdash \mathrm{if}\ t\ \mathrm{then}\ t_1\ \mathrm{else}\ t_2 \ominus \mathrm{if}\ t'\ \mathrm{then}\ t_1'\ \mathrm{else}\ t_2' \downarrow \tau \Rightarrow \Phi_1 \wedge \Phi_2 \wedge \Phi_3}\ \textbf{alg-r-if}$$

Figure B.24: Algorithmic typing rules of RelRef, part 3.

B.2.2 Metatheory of RelRef

B.2.3 RelRef Proof

Lemma 26 (Substitution of RelRef Core)**.** *Assume that*

1. $\Delta; \Psi_a; \Gamma \vdash t_1 \backsim t_2 :^c \tau$ (1).
2. $\Delta; \Psi_a; \Gamma, x : \tau \vdash t_1' \backsim t_2' :^c \tau'$ (2).

then $\Delta; \Psi_a; \Gamma \vdash t_1'[t_1/x] \backsim t_2'[t_2/x] :^c \tau'$.

Proof. By induction on the typing derivation of the second assumption (2).

Case

$$\frac{(\Gamma, x:\tau)(z) = \tau'}{\Delta; \Phi_a; \Gamma, x:\tau \vdash z \backsim z :^c \tau'}\ \textbf{c-r-var}$$

Subcase: $z = x$.

TS: $\Delta; \Phi_a; \Gamma \vdash z[t_1/x] \backsim z[t_2/x] :^c \tau'$.

Because $x = z$,$z[t_1/x] = t_1$, $z[t_2/x] = t_1$, STS: $\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_2 :^c \tau'$.

From $(\Gamma, x:\tau)(z) = \tau'$ and $x = z$, we know $\tau = \tau'$.

By the above statements, STS: $\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_2 :^c \tau$, which is proved by the assumption.

Subcase: $z \neq x$.

we know: $\Gamma(z) = \tau'$ $(\star)$.

TS: $\Delta; \Phi_a; \Gamma \vdash z[t_1/x] \backsim z[t_2/x] :^c \tau'$.

Because $x \neq z$, $z[t_1/x] = z$, $z[t_2/x] = z$, STS: $\Delta; \Phi_a; \Gamma \vdash z \backsim z :^c \tau'$.

By the core rule **c-r-vars** and $(\star)$, we construct the derivation :

$$\frac{\Gamma(z) = \tau'}{\Delta; \Phi_a; \Gamma \vdash z \backsim z :^c \tau'}\ \textbf{c-r-var}$$

Case

$$\frac{\Delta;\Phi_a;\Gamma,x:\tau\vdash t_1''\backsim t_2'':^c\Box\tau' \quad (3)}{\Delta;\Phi_a;\Gamma,x:\tau\vdash \textbf{der}\ t_1''\backsim \textbf{der}\ t_2'':^c\tau'}\ \textbf{c-der}$$

$t_1' = \text{der}\ t_1'',\ t_2' = \text{der}\ t_2''.$

TS: $\Delta;\Phi_a;\Gamma\vdash \text{der}\ (t_1''[t_1/x])\backsim \text{der}\ (t_2''[t_2/x]):^c\tau'.$

By IH on (1) and (3), $\Delta;\Psi_a;\Gamma\vdash t_1''[t_1/x]\backsim t_2''[t_2/x]:^c\Box\tau'$ (4).

By core rule **c-der** and (4), we can derive

$$\frac{\Delta;\Phi_a;\Gamma,x:\tau\vdash t_1''[t_1/x]\backsim t_2''[t_2/x]:^c\Box\tau' \quad (4)}{\Delta;\Phi_a;\Gamma,x:\tau\vdash \text{der}\ (t_1''[t_1/x])\backsim \text{der}\ (t_2''[t_2/x]):^c\tau'}\ \textbf{c-der}.$$

Case

$$\frac{\Delta;\Phi_a;z:\tau_1,f:\tau_1\to\tau_2,\Gamma,x:\tau\vdash t_1''\backsim t_2'':^c\tau_2 \quad (3)}{\Delta;\Phi_a;\Gamma,x:\tau\vdash \textbf{fix}\ f(z).t_1''\backsim \textbf{fix}\ f(z).t_2'':^c\tau_1\to\tau_2}\ \textbf{c-r-fix}$$

$t_1' = \text{fix}\ f(z).t_1'',\ t_2' = \text{fix}\ f(z).t_2'', \tau' = \tau_1\to\tau_2.$

TS: $\Delta;\Phi_a;\Gamma\vdash \text{fix}\ f(z).(t_1''[t_1/x])\backsim \text{fix}\ f(z).(t_2''[t_2/x]):^c\tau'.$

By IH on (1) and (3), $\Delta;\Psi_a;z:\tau_1,f:\tau_1\to\tau_2,\Gamma\vdash t_1''[t_1/x]\backsim t_2''[t_2/x]:^c\tau_2$ (4).

By core rule **c-r-fix** and (4), we can derive :

$\Delta;\Phi_a;\Gamma\vdash \text{fix}\ f(z).(t_1''[t_1/x])\backsim \text{fix}\ f(z).(t_2''[t_2/x]):^c\tau'.$

Case

$$\frac{\Delta;\Phi_a;z:\tau_1,f:\Box(\tau_1\to\tau_2),\Box\Gamma,x:\tau\vdash t\backsim t:^c\tau_2 \quad (3)}{\Delta;\Phi_a;\Box\Gamma,\Gamma',x:\tau\vdash \textbf{fix}_{NC}\ f(z).t\backsim \textbf{fix}_{NC}\ f(z).t:^c\Box(\tau_1\to\tau_2)}\ \textbf{c-r-fixNC}$$

$t_1' = \text{fix}\ f(z).t,\ t_2' = \text{fix}\ f(z).t, \tau' = \Box(\tau_1\to\tau_2).$

TS: $\Delta;\Phi_a;\Box\Gamma,\Gamma'\vdash \text{fix}\ f(z).(t[t_1/x])\backsim \text{fix}\ f(z).(t[t_2/x]):^c\tau'.$

By IH on (1) and (3), $\Delta;\Phi_a;z:\tau_1,f:\Box(\tau_1\to\tau_2),\Box\Gamma\vdash t[t_1/x]\backsim t[t_2/x]:^c\tau_2$ (4).

By core rule **c-r-fixNC** and (4), we can derive :

$\Delta;\Phi_a;\Box\Gamma,\Gamma'\vdash \text{fix}\ f(z).(t[t_1/x])\backsim \text{fix}\ f(z).(t[t_2/x]):^c\tau'.$

Case

$$\frac{\Delta;\Phi_a;\Gamma,x:\tau\vdash t_3\backsim t_3' :^c \tau'' \quad (3) \qquad \Delta;\Phi_a;\Gamma,x:\tau\vdash t_4\backsim t_4' :^c \mathbf{list}[n]^\alpha\,\tau'' \quad (4)}{\Delta;\Phi_a;\Gamma,x:\tau\vdash \mathbf{cons}_C(t_3,t_4)\backsim \mathbf{cons}_C(t_3',t_4') :^c \mathbf{list}[n+1]^{\alpha+1}\,\tau''}\ \textbf{c-r-cons1}$$

$t_1' = \mathrm{cons}_C(t_3,t_4),\ t_2' = \mathrm{cons}_C(t_3',t_4'), \tau' = \mathrm{list}[n+1]^{\alpha+1}\,\tau''.$

TS: $\Delta;\Phi_a;\Gamma\vdash \mathrm{cons}_C(t_3[t_1/x],t_4[t_1/x])\backsim \mathrm{cons}_C(t_3'[t_2/x],t_4'[t_2/x]) :^c \tau'.$

By IH on (1) and (3),$\Delta;\Phi_a;\Gamma\vdash t_3[t_1/x]\backsim t_3'[t_2/x] :^c \tau''$ (5).

By IH on (1) and (4),$\Delta;\Phi_a;\Gamma\vdash t_4[t_1/x]\backsim t_4'[t_2/x] :^c \mathrm{list}[n]^\alpha\,\tau''$ (6).

By core rule **c-r-cons1** and (5),(6), we can derive :

$\Delta;\Phi_a;\Gamma\vdash \mathrm{cons}_C(t_3[t_1/x],t_4[t_1/x])\backsim \mathrm{cons}_C(t_3'[t_2/x],t_4'[t_2/x]) :^c \tau'.$

Case

$$\frac{\Delta;\Phi_a;\Box\Gamma,x:\tau\vdash t\backsim t :^c \tau'' \quad (3)}{\Delta;\Phi_a;\Box\Gamma,\Gamma',x:\tau\vdash \mathbf{NC}\ t\backsim \mathbf{NC}\ t :^c \Box\tau''}\ \textbf{c-nochange}$$

$t_1' = \mathrm{NC}\ t,\ t_2' = \mathrm{NC}\ t, \tau' = \Box\tau''.$

TS: $\Delta;\Phi_a;\Gamma\vdash \mathrm{NC}\ t[t_1/x]\backsim \mathrm{NC}\ t[t_2/x] :^c \tau'.$

By IH on (1) and (3),$\Delta;\Phi_a;\Gamma\vdash t[t_1/x]\backsim t[t_2/x] :^c \tau''$ (4).

By core rule **c-nochange** and (4), we can derive :

$\Delta;\Phi_a;\Gamma\vdash \mathrm{NC}\ t[t_1/x]\backsim \mathrm{NC}\ t[t_2/x]) :^c \tau'$

□

Lemma 27 (Reflexivity of Algorithmic Binary Type Equivalence)**.** $\forall\Delta,\psi_a,\Phi_a,\tau$. *there exists* Φ *s.t* $\Delta;\psi_a;\Phi_a\models\tau\equiv\tau\Rightarrow\Phi$ *and* $\Delta;\psi_a;\Phi_a\models\Phi$.

Proof. By induction on the binary type.

Case

$\forall i::S.\tau$

Assume Δ,ψ_a,Φ_a.

TS: $\exists \Phi$ s.t $\Delta; \psi_a; \Phi_a \models \forall i{::}S.\tau \equiv \forall i{::}S.\tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$

By IH on τ where $\Delta' = i, \Delta$, $\exists \Phi'$ s.t $\Delta'; \psi_a; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi'$ and $\Delta'; \psi_a; \Phi_a \models \Phi'$.

By the above statement and algorithmic type eq rules **alg-r-forall**, we conclude the following statement where $\Phi = \forall i{::}S.\Phi'$.

$\Delta; \psi_a; \Phi_a \models \forall i{::}S.\tau \equiv \forall i{::}S.\tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$.

Case

$\tau_1 \rightarrow \tau_2$

Assume Δ, ψ_a, Φ_a.

TS: $\exists \Phi$ s.t $\Delta; \psi_a; \Phi_a \models \tau_1 \rightarrow \tau_2 \equiv \tau_1 \rightarrow \tau_2 \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$

By IH on τ_1, $\exists \Phi_1$ s.t $\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_1 \Rightarrow \Phi_1$ and $\Delta; \psi_a; \Phi_a \models \Phi_1$.

By IH on τ_2, $\exists \Phi_2$ s.t $\Delta; \psi_a; \Phi_a \models \tau_2 \equiv \tau_2 \Rightarrow \Phi_2$ and $\Delta; \psi_a; \Phi_a \models \Phi_2$.

By the above statement and algorithmic type eq rules **alg-r-fun**, we conclude the following statement where $\Phi = \Phi_1 \wedge \Phi_2$.

$\Delta; \psi_a; \Phi_a \models \tau_1 \rightarrow \tau_2 \equiv \tau_1 \rightarrow \tau_2 \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$.

Case

$\mathbf{list}[n]^{\alpha}\,\tau$

Assume Δ, ψ_a, Φ_a.

TS: $\exists \Phi$ s.t $\Delta; \psi_a; \Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \equiv \mathrm{list}[n]^{\alpha}\,\tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$

By IH on τ, $\exists \Phi_1$ s.t $\Delta; \psi_a; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi_1$ and $\Delta; \psi_a; \Phi_a \models \Phi_1$.

By the above statement and algorithmic type eq rules **alg-r-list**, we conclude the following statement where $\Phi = \Phi_1 \wedge n = n \wedge \alpha = \alpha$.

$\Delta; \psi_a; \Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \equiv \mathrm{list}[n]^{\alpha}\,\tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$.

Case

$\exists i{::}S.\tau$

Assume Δ, ψ_a, Φ_a.

TS: $\exists \Phi$ s.t $\Delta; \psi_a; \Phi_a \models \exists i{::}S.\tau \equiv \exists i{::}S.\tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$

By IH on τ, choose $i \notin FV(\Phi_a)$, $\exists \Phi_1$ s.t $i, \Delta; \psi_a; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi_1$ and $i, \Delta; \psi_a; \Phi_a \models \Phi_1$.

By the above statement and algorithmic type eq rules **alg-r-**$\exists$, we conclude the following statement where $\Phi = \forall i{::}S.\, \Phi_1$.

$\Delta; \psi_a; \Phi_a \models \exists i{::}S.\, \tau \equiv \exists i{::}S.\, \tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$.

Case

$\square \tau$

Assume Δ, ψ_a, Φ_a.

TS: $\exists \Phi$ s.t $\Delta; \psi_a; \Phi_a \models \square \tau \equiv \square \tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$

By IH on τ,$\exists \Phi_1$ s.t $\Delta; \psi_a; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi_1$ and $\Delta; \psi_a; \Phi_a \models \Phi_1$.

By the above statement and algorithmic type eq rules **B-**$\square$, we conclude the following statement where $\Phi = \Phi_1$.

$\Delta; \psi_a; \Phi_a \models \square \tau \equiv \square \tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$.

Case

$C \,\&\, \tau$

Assume Δ, ψ_a, Φ_a.

TS: $\exists \Phi$ s.t $\Delta; \psi_a; \Phi_a \models C \,\&\, \tau \equiv C \,\&\, \tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$

By IH on τ,$\exists \Phi_1$ s.t $\Delta; \psi_a; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi_1$ and $\Delta; \psi_a; \Phi_a \models \Phi_1$.

By the above statement and algorithmic type eq rules **alg-r-cprod**, we conclude the following statement where $\Phi = C \leftrightarrow C \wedge \Phi_1$.

$\Delta; \psi_a; \Phi_a \models C \,\&\, \tau \equiv C \,\&\, \tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$.

Case

$C \supset \tau$

Assume Δ, ψ_a, Φ_a.

TS: $\exists \Phi$ s.t $\Delta; \psi_a; \Phi_a \models C \supset \tau \equiv C \supset \tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$

By IH on τ,$\exists \Phi_1$ s.t $\Delta; \psi_a; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi_1$ and $\Delta; \psi_a; \Phi_a \models \Phi_1$.

By the above statement and algorithmic type eq rules **alg-r-cimpl**, we conclude the fol-

lowing statement where $\Phi = C \leftrightarrow C \wedge \Phi_1$.

$\Delta; \psi_a; \Phi_a \models C \supset \tau \equiv C \supset \tau \Rightarrow \Phi$ and $\Delta; \psi_a; \Phi_a \models \Phi$.

□

Theorem 28 (Soundness of the Algorithmic Binary Type Equality)**.** *Assume that*

1. $\Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi$
2. $FIV(\Phi_a, \tau, \tau') \subseteq \Delta, \psi_a$
3. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable and* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta; \Phi_a[\theta_a] \models \tau[\theta_a] \equiv \tau'[\theta_a]$.

Proof. By induction on the algorithmic binary type equivalence derivation.

Case

$$\frac{\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta; \psi_a; \Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2 \qquad \Phi = \Phi_1 \wedge \Phi_2}{\Delta; \psi_a; \Phi_a \models \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2' \Rightarrow \Phi}\ \textbf{alg-r-fun}$$

Assume $FIV(\Phi_a, \tau_1 \to \tau_2, \tau_1' \to \tau_2') \subseteq \Delta, \psi_a$ and $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ is provable

and $\Delta \rhd \theta_a : \psi_a$ is derivable.

TS: $\Delta; \Phi_a[\theta_a] \models (\tau_1 \to \tau_2)[\theta_a] \equiv (\tau_1' \to \tau_2')[\theta_a]$.

We know $\Phi = \Phi_1 \wedge \Phi_2$ s.t $FIV(\Phi_a, \tau_1, \tau_1') \subseteq \Delta, \psi_a$ and $\Delta; \Phi_a[\theta_a] \models \Phi_1[\theta_a]$ is provable.

Similaryly, $FIV(\Phi_a, \tau_2, \tau_2') \subseteq \Delta, \psi_a$ and $\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta_a]$ is provable.

By IH on $\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1$ with the same substitution θ_a and the above statements, $\Delta; \Phi_a[\theta_a] \models \tau_1[\theta_a] \equiv \tau_1'[\theta_a]$.

By IH on $\Delta; \psi_a; \Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_2$ with the same substitution θ_a and the above statements , $\Delta; \Phi_a[\theta_a] \models \tau_2[\theta_a] \equiv \tau_2'[\theta_a]$.

By the above statements and RelRef type equivalence rule **eq-fun**, we conclude that $\Delta; \Phi_a[\theta_a] \models (\tau_1 \to \tau_2)[\theta_a] \equiv (\tau_1' \to \tau_2')[\theta_a]$.

Case

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi \qquad \Phi' = \Phi \wedge n \doteq n' \wedge \alpha \doteq \alpha'}{\Delta;\psi_a;\Phi_a \models \textbf{list}[n]^{\alpha}\,\tau \equiv \textbf{list}[n']^{\alpha'}\,\tau' \Rightarrow \Phi'}\ \textbf{alg-r-list}$$

Assume FIV$(\Phi_a, \text{list}[n]^{\alpha}\,\tau, \text{list}[n']^{\alpha'}\,\tau') \subseteq \Delta, \psi_a$ and $\Delta;\Phi_a[\theta_a] \models \Phi'[\theta_a]$ is provable and $\Delta \rhd \theta_a : \psi_a$ is derivable.

TS: $\Delta;\Phi_a[\theta_a] \models (\text{list}[n]^{\alpha}\,\tau)[\theta_a] \equiv (\text{list}[n']^{\alpha'}\,\tau')[\theta_a]$.

We know $\Phi' = \Phi \wedge n \doteq n' \wedge \alpha \doteq \alpha'$ s.t

From assumption FIV$(\Phi_a, \text{list}[n]^{\alpha}\,\tau, \text{list}[n']^{\alpha'}\,\tau') \subseteq \Delta, \psi_a$, we know

FIV$(\Phi_a, \tau, n, \alpha, \tau', n', \alpha') \subseteq \Delta, \psi_a$.

From assumption $\Delta;\Phi_a[\theta_a] \models \Phi'[\theta_a]$, we get $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ is provable

and $\Delta;\Phi_a[\theta_a] \models (n \doteq n')[\theta_a]$ and $\Delta;\Phi_a[\theta_a] \models (\alpha \doteq \alpha')[\theta_a]$.

By IH on the first premise $\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi$ with the same substitution θ_a

and the above statements, $\Delta;\Phi_a[\theta_a] \models \tau[\theta_a] \equiv \tau'[\theta_a]$.

By the above statements and RelRef type equivalence rule **eq-list**, we conclude that

$\Delta;\Phi_a[\theta_a] \models (\text{list}[n]^{\alpha}\,\tau)[\theta_a] \equiv (\text{list}[n']^{\alpha'}\,\tau')[\theta_a]$.

Case

$$\frac{i, \Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \forall i{::}S.\,\tau \equiv \forall i{::}S.\,\tau' \Rightarrow \forall\, i :: S.\Phi}\ \textbf{alg-r-forall}$$

Assume FIV$(\Phi_a, \forall i{::}S.\,\tau, \forall i{::}S.\,\tau') \subseteq \Delta, \psi_a$ and $\Delta;\Phi_a[\theta_a] \models \Phi'[\theta_a]$ is provable and $\Delta \rhd \theta_a : \psi_a$ is derivable.

TS: $\Delta;\Phi_a[\theta_a] \models \forall i{::}S.\,\tau[\theta_a] \equiv \forall i{::}S.\,\tau'[\theta_a]$.

We know $\Phi' = \forall i{::}S.\Phi$ s.t

From assumption FIV$(\Phi_a, \forall i{::}S.\,\tau, \forall i{::}S.\,\tau') \subseteq \Delta, \psi_a$, we know FIV$(\Phi_a, \tau, \tau', i) \subseteq (i, \Delta), \psi_a$.

From assumption $\Delta;\Phi_a[\theta_a] \models \forall i{::}S.\Phi[\theta_a]$, we get $i, \Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ is provable.

By IH on the first premise $i, \Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi$ with the above statements,

$i, \Delta;\Phi_a[\theta_a] \models \tau[\theta_a] \equiv \tau'[\theta_a]$.

By the above statements and RelRef type equivalence rule **eq-**$\forall$, we conclude that $\Delta; \Phi_a[\theta_a] \models \forall i{::}S.\tau[\theta_a] \equiv \forall i{::}S.\tau'[\theta_a]$.

Case

$$\frac{\Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi}\textbf{alg-r-cprod}$$

Assume $\text{FIV}(\Phi_a, C \,\&\, \tau, C' \,\&\, \tau') \subseteq \Delta, \psi_a$ and $\Delta; \Phi_a[\theta_a] \models \Phi'[\theta_a]$ is provable and $\Delta \rhd \theta_a : \psi_a$ is derivable.

TS: $\Delta; \Phi_a[\theta_a] \models (C \,\&\, \tau)[\theta_a] \equiv (C' \,\&\, \tau')[\theta_a]$.

We know $\Phi' = C \leftrightarrow C' \wedge \Phi$ s.t

From assumption $\text{FIV}(\Phi_a, C \,\&\, \tau, C' \,\&\, \tau') \subseteq \Delta, \psi_a$, we know $\text{FIV}(\Phi_a, \tau, \tau', C, C') \subseteq \Delta, \psi_a$.

From assumption $\Delta; \Phi_a[\theta_a] \models (C \leftrightarrow C' \wedge \Phi)[\theta_a]$, we get $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ is provable and $\Delta; \Phi_a[\theta_a] \models C[\theta_a]$ and $\Delta; \Phi_a[\theta_a] \models C'[\theta_a]$.

By IH on the first premise $\Delta; \psi_a; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi$ with the above statements, $\Delta; \Phi_a[\theta_a] \models \tau[\theta_a] \equiv \tau'[\theta_a]$.

By the above statements and RelRef type equivalence rule **eq-c-prod**, we conclude that $\Delta; \Phi_a[\theta_a] \models (C \,\&\, \tau)[\theta_a] \equiv (C' \,\&\, \tau')[\theta_a]$.

□

Theorem 29 (Completeness of the Binary Algorithmic Type Equivalence)**.** *Assume that* $\Delta; \Phi_a \models \tau \equiv \tau'$. *Then* $\exists \Phi$. *such that* $\Delta; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the binary equivalence typing derivation.

Case

$$\frac{\Delta; \Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta; \Phi_a \models \tau_2 \equiv \tau_2'}{\Delta; \Phi_a \models \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'}\textbf{eq-fun}$$

By IH on the first premise $\Delta; \Phi_a \models \tau_1 \equiv \tau_1'$, there exists Φ_1 s.t $\Delta; \Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1$ and $\Delta; \Phi_a \models \Phi_1$.

By IH on the second premise $\Delta;\Phi_a \models \tau_2 \equiv \tau_2'$, there exists Φ_2 s.t $\Delta;\Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2$ and $\Delta;\Phi_a \models \Phi_2$.

By the above statements and algorithmic type equivalence rules **alg-r-fun** with an empty Ψ_a, we conclude the following statement where $\Phi = \Phi_1 \wedge \Phi_2$.

$\Delta;\Phi_a \models \tau_1 \to \tau_2 \equiv \tau_1 \to \tau_2 \Rightarrow \Phi$ and $\Delta;\Phi_a \models \Phi$.

Case

$$\frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \doteq \alpha' \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \mathbf{list}[n]^{\alpha}\,\tau \equiv \mathbf{list}[n']^{\alpha'}\,\tau'}\ \textbf{eq-list}$$

By IH on the third premise $\Delta;\Phi_a \models \tau \equiv \tau'$, there exists Φ_1 s.t $\Delta;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_1$ and $\Delta;\Phi_a \models \Phi_1$.

By the above statements, the first and second premises and algorithmic type equivalence rules **alg-r-list** with an empty Ψ_a, we conclude the following statement where $\Phi = \Phi_1 \wedge n \doteq n' \wedge \alpha \doteq \alpha'$.

$\Delta;\Phi_a \models \mathrm{list}[n]^{\alpha}\,\tau \equiv \mathrm{list}[n']^{\alpha'}\,\tau' \Rightarrow \Phi$ and $\Delta;\Phi_a \models \Phi$.

Case

$$\frac{\Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \Box\tau \equiv \Box\tau'}\ \textbf{eq-B-}\Box$$

By IH on the premise $\Delta;\Phi_a \models \tau \equiv \tau'$, there exists Φ_1 s.t $\Delta;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_1$ and $\Delta;\Phi_a \models \Phi_1$.

By the above statements and algorithmic type equivalence rules **alg-r-**$\Box$ with an empty Ψ_a, we conclude the following statement where $\Phi = \Phi_1$.

$\Delta;\Phi_a \models \Box\tau \equiv \Box\tau' \Rightarrow \Phi$ and $\Delta;\Phi_a \models \Phi$.

Case

$$\frac{i,\Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \forall i{::}S.\,\tau \equiv \forall i{::}S.\,\tau'}\ \textbf{eq-}\forall$$

By IH on the premise $i,\Delta;\Phi_a \models \tau \equiv \tau'$, there exists Φ_1 s.t $i,\Delta;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_1$ and

$i, \Delta; \Phi_a \models \Phi_1$.

By the above statements and algorithmic type equivalence rules **alg-r-forall** with an empty Ψ_a, we conclude the following statement where $\Phi = \forall i{::}S.\, \Phi_1$.

$\Delta; \Phi_a \models \forall i{::}S.\, \tau \equiv \forall i{::}S.\, \tau' \Rightarrow \Phi$ and $\Delta; \Phi_a \models \Phi$.

Case

$$\frac{\Delta; C' \wedge \Phi_a \models C \qquad \Delta; C \wedge \Phi_a \models C' \qquad \Delta; \Phi_a \models \tau \equiv \tau'}{\Delta; \Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau'}\ \textbf{eq-c-prod}$$

By IH on the third premise $\Delta; \Phi_a \models \tau \equiv \tau'$, there exists Φ_1 s.t $i, \Delta; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_1$ and $\Delta; \Phi_a \models \Phi_1$.

From the first two premises $\Delta; C' \wedge \Phi_a \models C$ and $\Delta; C \wedge \Phi_a \models C'$, we know that $\Delta; \Phi_a \models C \leftrightarrow C'$.

By the above statements and the algorithmic type equivalence rules **alg-r-cprod** with an empty Ψ_a, we conclude the following statement where $\Phi = C \leftrightarrow C' \wedge \Phi_1$.

$\Delta; \Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau' \Rightarrow \Phi$ and $\Delta; \Phi_a \models \Phi$.

□

B.2.3.1 Proof of Lemma 4

Lemma 30 (Existence of coercions for relational subtyping)**.** *If* $\Delta; \Phi_a \models \tau \sqsubseteq \tau'$ *then there exists* $e \in$ *RelRef Core s.t.* $\Delta; \Phi_a; \cdot \vdash t \backsim t :^c \tau \to \tau'$.

Proof. Proof is by induction on the subtyping derivation. We denote the witness t of type $\tau \to \tau'$ as $\text{coerce}_{\tau,\tau'}$ for clarity.

Case

$$\frac{\Delta; \Phi_a \models \tau_1' \sqsubseteq \tau_1 \ \ (\star) \qquad \Delta; \Phi_a \models \tau_2 \sqsubseteq \tau_2' \ \ (\diamond)}{\Delta; \Phi_a \models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'}\ \to$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau_1',\tau_1}\,.\ \Delta; \Phi_a; \cdot \vdash \text{coerce}_{\tau_1',\tau_1} \backsim \text{coerce}_{\tau_1',\tau_1} :^c \tau_1' \to \tau_1$

By IH on $(\diamond)$, $\exists \text{coerce}_{\tau_2,\tau_2'}\,.\ \Delta; \Phi_a; \cdot \vdash \text{coerce}_{\tau_2,\tau_2'} \backsim \text{coerce}_{\tau_2,\tau_2'} :^c \tau_2 \to \tau_2'$

Then, using these two statements,we can construct the following derivation where

$e = \text{fix } f(x).\text{fix } f(y).\text{coerce}_{\tau_2,\tau_2'}\ (x\ (\text{coerce}_{\tau_1',\tau_1}\ y))$

$\Delta;\Phi_a;\cdot \vdash t \backsim t :^c (\tau_1 \to \tau_2) \to \tau_1' \to \tau_2'$.

Case

$$\frac{}{\Delta;\Phi_a \models \textbf{int} \sqsubseteq \Box\textbf{int}}\ \textbf{int-}\Box$$

Then, we can construct the derivation using the primitive function

$\text{box}_{\text{int}} : \text{int} \to \Box\text{int}$

$$\frac{}{\Delta;\Phi_a;\cdot \vdash \text{fix } f(x).\text{box}_{\text{int}}\, x \backsim \text{fix } f(x).\text{box}_{\text{int}}\, x :^c \text{int} \to \Box\text{int}}$$

Case

$$\frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \tau}\ \textbf{T}$$

Then, we can immediately construct the derivation using the rule **c-der**.

$$\frac{}{\Delta;\Phi_a;\cdot \vdash \text{fix } f(x).\text{der } x \backsim \text{fix } f(x).\text{der } x :^c \Box\tau \to \tau}$$

Case

$$\frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \Box\Box\tau}\ \textbf{D}$$

Then, we can immediately construct the derivation using the rule **c-nochange**.

$$\frac{}{\Delta;\Phi_a;\cdot \vdash \text{fix } f(x).\text{NC } x \backsim \text{fix } f(x).\text{NC } x :^c \Box\tau \to \Box\Box\tau}$$

Case

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2 (\star)}{\Delta;\Phi_a \models \square\tau_1 \sqsubseteq \square\tau_2}\ \textbf{B-}\square$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau_1,\tau_2}\ .\ \Delta;\Phi_a;\cdot \vdash \text{coerce}_{\tau_1,\tau_2} \rightsquigarrow \text{coerce}_{\tau_1,\tau_2} :^c \tau_1 \to \tau_2$.

Then, using $(\star)$ and the rules **c-der** and **c-nochange**, we can construct the derivation

$$\frac{}{\Delta;\Phi_a;\cdot \vdash \text{fix } f(x).\text{NC } (\text{coerce}_{\tau_1,\tau_2}\ (\text{der } x)) \rightsquigarrow \text{fix } f(x).\text{NC } (\text{coerce}_{\tau_1,\tau_2}\ (\text{der } x)) :^c \square\tau_1 \to \square\tau_2}$$

Case

$$\frac{}{\Delta;\Phi_a \models \tau \sqsubseteq \tau}\ \textbf{refl}$$

Then, we can immediately construct the derivation

$$\frac{}{\Delta;\Phi_a;\cdot \vdash \text{fix } f(x).x \rightsquigarrow \text{fix } f(x).x :^c \tau \to \tau}$$

Case

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2\ (\star) \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_3\ (\diamond)}{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_3}\ \textbf{trans}$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau_1,\tau_2}\ .\ i :: S, \Delta;\Phi_a;\cdot \vdash \text{coerce}_{\tau_1,\tau_2} \rightsquigarrow \text{coerce}_{\tau_1,\tau_2} :^c \tau_1 \to \tau_2$

By IH on $(\diamond)$, $\exists \text{coerce}_{\tau_2,\tau_3}\ .\ i :: S, \Delta;\Phi_a;\cdot \vdash \text{coerce}_{\tau_2,\tau_3} \rightsquigarrow \text{coerce}_{\tau_2,\tau_3} :^c \tau_2 \to \tau_3$.

Then, using $(\star)$ and $(\diamond)$, we can construct the derivation simply by function composition

$$\frac{}{\Delta;\Phi_a;\cdot \vdash \text{fix } f(x).\text{coerce}_{\tau_2,\tau_3}\ (\text{coerce}_{\tau_1,\tau_2}\ x) \rightsquigarrow \text{fix } f(x).\text{coerce}_{\tau_2,\tau_3}\ (\text{coerce}_{\tau_1,\tau_2}\ x) :^c \tau_1 \to \tau_3}$$

Case

$$\frac{}{\Delta;\Phi_a \models \Box(\tau_1 \xrightarrow{\mathbf{diff}(D)} \tau_2) \sqsubseteq \Box\tau_1 \xrightarrow{\mathbf{diff}(0)} \Box\tau_2} \to \Box_{\mathbf{diff}}$$

Then, we can immediately construct the derivation

where $e = \text{fix } f(x).\text{fix } (y).\text{NC } (\text{der } x)\ (\text{der } y))$

$\Delta;\Phi_a;\cdot \vdash t \backsim t :^c \Box(\tau_1 \to \tau_2) \to \Box\tau_1 \to \Box\tau_2.$

Case

$$\frac{i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau' \ (\star) \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \forall i{::}S.\tau \sqsubseteq \forall i{::}S.\tau'} \forall_{\mathbf{diff}}$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau,\tau'}\ .\ \ i :: S, \Delta;\Phi_a;\cdot \vdash \text{coerce}_{\tau,\tau'} \backsim \text{coerce}_{\tau,\tau'} :^c \tau \to \tau'$,

Then, using this, and the **c-r-iLam** and **c-r-iApp** rules, we can construct the following derivation:

$\Delta;\Phi_a;\cdot$

$\vdash \text{fix } f(x).\Lambda i.\text{coerce}_{\tau,\tau'}\ (x\ [i]) \backsim \text{fix } f(x).\Lambda i.\text{coerce}_{\tau,\tau'}\ (x\ [i]) :^c (\forall i{::}S.\tau) \to \forall i{::}S.\tau'.$

Case

$$\frac{}{\Delta;\Phi_a \models \Box(\forall i{::}S.\tau) \sqsubseteq \forall i{::}S.\Box\tau} \forall\,\Box$$

Then, we can immediately construct the following derivation using **c-der**, **c-nochange**, **c-r-iLam** and **c-r-iApp** rules.

$\Delta;\Phi_a;\cdot$

$\vdash \text{fix } f(x).\Lambda i.\text{NC } ((\text{der } x)\ [i]) \backsim \text{fix } f(x).\Lambda i.\text{NC } ((\text{der } x)\ [i]) :^c \Box(\forall i{::}S.\tau) \to \forall i{::}S.\Box\tau.$

Case

$$\frac{\Delta;\Phi_a \models n \doteq n' \ (\star) \qquad \Delta;\Phi_a \models \alpha \leq \alpha' \ (\diamond) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \ (\dagger)}{\Delta;\Phi_a \models \mathbf{list}[n]^{\alpha}\,\tau \sqsubseteq \mathbf{list}[n']^{\alpha'}\,\tau'} \mathbf{l1}$$

By IH on $(\dagger)$, $\exists \text{coerce}_{\tau,\tau'}\ .\ \Delta;\Phi_a;\cdot \vdash \text{coerce}_{\tau,\tau'} \backsim \text{coerce}_{\tau,\tau'} :^c \tau \to \tau'$

We first construct the more generic term for type:

$$\text{unit} \to \forall n{::}\mathbb{N}.\,\forall n'{::}\mathbb{N}.\,\forall \alpha{::}\mathbb{N}.\,\forall \alpha'{::}\mathbb{N}.\,((n = n' \wedge \alpha \leq \alpha') \,\&\, \text{list}[n]^{\alpha}\,\tau) \to \text{list}[n']^{\alpha'}\,\tau' \qquad \text{(B.1)}$$

and then instantiate the term for eq. (B.33) later.

It can be shown that such a derivation can be constructed for expression.

$t' = \text{fix fList}(\_).\Lambda n.\Lambda n'.\Lambda\alpha.\Lambda\alpha'.\text{fix } f(x).\text{clet } x \text{ as } t \text{ in}$

$\text{case } t \text{ of}$

$\text{nil} \rightarrow \text{nil}$

$\mid h ::_N tl \rightarrow \text{let } r = \text{fList } () [n-1] [n'-1] [\alpha] [\alpha'] \; tl$

$\text{in cons}_{NC}(\text{NC } (\text{coerce}_{\tau,\tau'} \text{ der } h), r)$

$\mid h ::_C tl \rightarrow \text{let } r = \text{fList } () [n-1] [n'-1] [\alpha-1] [\alpha'-1] \; tl$

$\text{in cons}_C(\text{coerce}_{\tau,\tau'} \; h, r)$

Then, we can instantiate fList using ($\star$) and ($\diamond$) as follows where

$t'' = \text{fix } f(x).\text{fList } () \; [n][n'][\alpha][\alpha'] \; x$

$\Delta; \Phi_a; \cdot \vdash t'' \backsim t'' :^c \text{list}[n]^\alpha \tau \rightarrow \text{list}[n']^{\alpha'} \tau'.$

Case

$$\frac{}{\Delta; \Phi_a \models \mathbf{list}[n]^\alpha \Box\tau \sqsubseteq \Box(\mathbf{list}[n]^\alpha \tau)} \; \mathbf{l}\Box$$

We first construct the more generic term for type

$$\text{unit} \rightarrow \forall n::\mathbb{N}.\, \forall \alpha::\mathbb{N}.\, \text{list}[n]^\alpha \Box\tau \rightarrow \Box(\text{list}[n]^\alpha \tau) \tag{B.2}$$

and then instantiate the term for eq. (B.34) later. It can be shown that such a derivation can be constructed for expression

$t' = \text{fix fList}(\_).\Lambda n.\Lambda\alpha.\text{fix } f(x).$

$\text{case } t \text{ of}$

$\text{nil} \rightarrow \text{NC } (\text{nil })$

$\mid h ::_N tl \rightarrow \text{let } r = \text{fList } () [n-1] [\alpha] \; tl \text{ in}$

$\text{NC } (\text{cons}_{NC}(\text{der } h, \text{der } r))$

$\mid h ::_C tl \rightarrow \text{let } r = \text{fList } () [n-1] [\alpha-1] \; tl \text{ in}$

$\text{NC } (\text{cons}_C(\text{der } h, \text{der } r))$

Then, we can instantiate fList with a concrete n and α as follows

where $t'' = \text{fix } f(x).\text{fList } () \ [n][\alpha] \ x$

$\Delta; \Phi_a; \cdot \vdash t'' \ominus t'' \lesssim 0 :^c \text{list}[n]^\alpha \Box \tau \to \Box (\text{list}[n]^\alpha \tau)$

Case

$$\frac{\Delta; \Phi_a \models \alpha \doteq 0}{\Delta; \Phi_a \models \textbf{list}[n]^\alpha \tau \sqsubseteq \textbf{list}[n]^\alpha \Box \tau} \textbf{ l2}$$

We first construct the more generic term for type

$$\text{unit} \to \forall n{::}\mathbb{N}. \forall \alpha{::}\mathbb{N}. (\alpha = 0 \ \& \ \text{list}[n]^\alpha \tau) \to \text{list}[n]^\alpha \Box \tau \tag{B.3}$$

and then instantiate the term for eq. (B.35) later. It can be shown that such a derivation can be constructed for expression

$t' = \text{fix fList}(\_).\Lambda n.\Lambda \alpha.\text{fix } f(x).\text{clet } x \text{ as } t \text{ in}$

case t of

nil $\to$ nil

$| \ h ::_N tl \to \text{let } r = \text{fList } () \ [n-1][\alpha] \ tl \text{ in } \text{cons}_{NC}(\text{NC } h, r)$

$| \ h ::_C tl \to \text{contra}$

Then, we can instantiate fList with a concrete n and α (note the premise $\alpha = 0$) as follows

where $t'' = \text{fix } f(x).\text{fList } () \ [n][\alpha] \ x$

$\Delta; \Phi_a; \cdot \vdash t'' \backsim t'' :^c \text{list}[n]^0 \tau \to \text{list}[n]^0 \Box \tau.$

Case

$$\frac{i :: S, \Delta; \Phi_a \models \tau \sqsubseteq \tau' \ (\star) \qquad i \notin FV(\Phi_a)}{\Delta; \Phi_a \models \exists i{::}S. \tau \sqsubseteq \exists i{::}S. \tau'} \exists$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau,\tau'} . \ i :: S, \Delta; \Phi; \cdot \vdash \text{coerce}_{\tau,\tau'} \backsim \text{coerce}_{\tau,\tau'} :^c \tau \to \tau'$

Then, using this and the **c-r-pack** and **c-r-unpack** rules, we can construct the follow-

ing derivation where $e = \text{fix } f(x).\text{unpack } x \text{ as } (y,i) \text{ in pack } (\text{coerce}_{\tau,\tau'}\ y) \text{ with } i$

$$\Delta;\Phi_a;\cdot \vdash t \backsim t :^c (\exists i{::}S.\tau) \to \exists i{::}S.\tau'.$$

Case

$$\frac{}{\Delta;\Phi_a \models \exists i{::}S.\Box\tau \sqsubseteq \Box(\exists i{::}S.\tau)}\ \exists\Box$$

Then, we can immediately construct the following derivation using **c-der**, **c-nochange**, **c-r-pack** and **c-r-unpack** rules in in Figures B.92 and B.95

where $e = \text{fix } f(x).\text{unpack } x \text{ as } (y,i) \text{ in NC (pack der } y \text{ with } i)$.

$$\Delta;\Phi_a;\cdot \vdash t \backsim t :^c (\exists i{::}S.\Box\tau) \to \Box(\exists i{::}S.\tau).$$

Case

$$\frac{\Delta;\Phi_a \wedge C' \models C\ \ (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'\ \ (\diamond)}{\Delta;\Phi_a \models C \supset \tau \sqsubseteq C' \supset \tau'}\ \textbf{c-impl}$$

By IH on ($\diamond$), $\exists \text{coerce}_{\tau,\tau'}.\ \Delta;\Phi_a;\cdot \vdash \text{coerce}_{\tau,\tau'} \backsim \text{coerce}_{\tau,\tau'} :^c \tau \to \tau'$.

Then, using this and the premise ($\star$) along with the **c-r-c-implI** and **c-r-c-implE** rules, we can construct the following derivation where $e = \text{fix } f(x).\text{coerce}_{\tau,\tau'}\ (\text{celim } x)$

$$\Delta;\Phi_a;\cdot \vdash t \backsim t :^c (C \supset \tau) \to C' \supset \tau'.$$

Case

$$\frac{}{\Delta;\Phi_a \models \Box(C \supset \tau) \sqsubseteq C \supset \Box\tau}\ \textbf{c-impl-}\Box$$

Then, we can immediately construct the following derivation using **c-der**, **c-nochange** and **c-r-c-implE** rules where $e = \lambda x.\text{NC (celim der } x)$.

$$\Delta;\Phi_a;\cdot \vdash t \backsim t :^c \Box(C \supset \tau) \to (C \supset \Box\tau).$$

Case

$$\frac{\Delta;\Phi_a \wedge C \models C'\ \ (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'\ \ (\diamond)}{\Delta;\Phi_a \models C\ \&\ \tau \sqsubseteq C'\ \&\ \tau'}\ \textbf{c-and}$$

By IH on ($\diamond$), $\exists \text{coerce}_{\tau,\tau'}.\ \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau,\tau'} \backsim \text{coerce}_{\tau,\tau'} :^c \tau \to \tau'$

Then, using this and the premise ($\star$) along with the **c-r-c-prodI** and **c-r-c-prodE** rules, we can construct the following derivation where $e = \text{fix } f(x).\text{clet } x \text{ as } y \text{ in coerce}_{\tau,\tau'}\ y$

$\Delta;\Phi_a;\cdot \vdash t \backsim t :^c (C \,\&\, \tau) \to C' \,\&\, \tau'$.

Case

$$\frac{}{\Delta;\Phi_a \models C \,\&\, \Box\tau \sqsubseteq \Box(C \,\&\, \tau)}\ \textbf{c-and-}\Box$$

Then, we can immediately construct the following derivation using **c-der**, **c-nochange**, **c-r-c-prodI** and **c-r-c-prodE** rules where $e = \text{fix } f(x).\text{clet } x \text{ as } y \text{ in NC (der } y)$.

$\Delta;\Phi_a;\cdot \vdash t \backsim t :^c (C \,\&\, \Box\tau) \to \Box(C \,\&\, \tau)$

□

Theorem 31 (Types are preserved by embedding for RelRef)**.**

1. *If* $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t_1^* \backsim t_2^* : \tau$, *then* $\Delta;\Phi_a;\Gamma \vdash t_1^* \backsim t_2^* :^c \tau$ *and* $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau$.

Proof. Proof is by induction on the embedding derivations.

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \quad (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad (\diamond) \qquad t' = \textbf{coerce}_{\tau,\tau'} \quad (\dagger)}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t'\ t_1^* \backsim t'\ t_2^* : \tau'}\ \textbf{e-r-}\sqsubseteq$$

By IH on ($\star$), $\Delta;\Phi_a;\Gamma \vdash t_1^* \backsim t_2^* :^c \tau$ ($\star\star$) and $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau$ ($\star\star\star$).

By Lemma 30 using ($\diamond$), we know that $\Delta;\Phi_a;\cdot \vdash t' \backsim t' :^c \tau \to \tau'$ ($\diamond\diamond$).

By applying **c-r-app** rule and ($\star\star$) and ($\diamond\diamond$), we get $\Delta;\Phi_a;\Gamma \vdash t'\ t_1^* \backsim t'\ t_2^* :^c \tau'$ (♠).

By reflexivity of binary type equivalence, we know $\Delta;\Phi_a \models \tau' \equiv \tau'$ (♠♠).

Then, we conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t'\ t_1^* \backsim t'\ t_2^* :^c \tau' \quad (\spadesuit) \qquad \Delta;\Phi_a \models \tau' \equiv \tau' \quad (\spadesuit\spadesuit)}{\Delta;\Phi_a;\Gamma \vdash t'\ t_1^* \backsim t'\ t_2^* :^c \tau'}\ \textbf{c-r-}\sqsubseteq$$

By ($\star\star\star$) and ($\diamond$) and **rr-**$\sqsubseteq$, we also conclude $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau'$.

Case

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2 \text{ wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\tau_1\to\tau_2,\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \textbf{fix}\ f(x).t_1 \backsim \textbf{fix}\ f(x).t_2 \rightsquigarrow \textbf{fix}\ f(x).{t_1}^* \backsim \textbf{fix}\ f(x).{t_2}^* : \tau_1 \to \tau_2}\ \textbf{e-r-fix}$$

By IH on the premise $\Delta;\Phi_a;x:\tau_1, f:\tau_1\to\tau_2,\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau_2$, $\Delta;\Phi_a;x:\tau_1, f:\tau_1\to\tau_2,\Gamma \vdash t_1^* \backsim t_2^* :^c \tau_2$ $(\star)$.

By applying **c-r-fix** rule and $(\star)$, we get $\Delta;\Phi_a;\Gamma \vdash \text{fix}\ f(x).t_1^* \backsim \text{fix}\ f(x).t_2^* :^c \tau_1 \to \tau_2$ $(\spadesuit)$.

Similarly, using **rr-fix**, we conclude $\Delta;\Phi_a;\Gamma \vdash \text{fix}\ f(x).t_1 \backsim \text{fix}\ f(x).t_2 : \tau_1 \to \tau_2$.

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \to \tau_2 \text{ wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\square(\tau_1\to\tau_2),\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau_2 \\ \forall x_i \in dom(\Gamma),\ \ t_i = \textbf{coerce}_{\Gamma(x_i),\square\Gamma(x_i)} \qquad \forall x_i \in dom(\Gamma),\ \ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \square\Gamma(x) \\ t^{**} = \textbf{let}\ \overline{y_i = t_i\ x_i}\ \textbf{in}\ \textbf{fix}_{NC}\ f(x).t^*[\overline{\textbf{der}\ y_i/x_i}]\end{array}}{\Delta;\Phi_a;\Gamma \vdash \textbf{fix}\ f(x).t \backsim \textbf{fix}\ f(x).t \rightsquigarrow t^{**} \backsim t^{**} : \square(\tau_1\to\tau_2)}$$

e-r-fixNC

By IH on the premise $\Delta;\Phi_a;x:\tau_1, f:\square(\tau_1\to\tau_2),\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau_2$,

we have $\Delta;\Phi_a;x:\tau_1, f:\square(\tau_1\to\tau_2),\Gamma \vdash t^* \backsim t^* :^c \tau_2$ $(\star)$

and $\Delta;\Phi_a;\Gamma \vdash t \backsim t : \tau_2$ $(\star\star)$.

TS: $\Delta;\Phi_a;\Gamma \vdash t^{**} \backsim t^{**} :^c \square(\tau_1\to\tau_2)$

By applying **c-r-let** rule on t^{**}.

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_i\ x_i \backsim t_i\ x_i :^c \square\Gamma(x_i) \\ \Delta;\Phi_a;\overline{y_i:\square\Gamma(x_i)},\Gamma \vdash \text{fix}_{NC}\ f(x).t^*[\overline{\text{der}\ y_i/x_i}] \backsim \text{fix}_{NC}\ f(x).t^*[\overline{\text{der}\ y_i/x_i}] :^c \square(\tau_1\to\tau_2)\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \text{let}\ \overline{y_i = t_i\ x_i}\ \text{in}\ \text{fix}_{NC}\ f(x).t^*[\overline{\text{der}\ y_i/x_i}] \backsim \text{let}\ \overline{y_i = t_i\ x_i}\ \text{in}\ \text{fix}_{NC}\ f(x).t^*[\overline{\text{der}\ y_i/x_i}] :^c \\ \square(\tau_1\to\tau_2)\end{array}}$$

Set $\Gamma = \overline{x_i:\Gamma(x_i)}$, $\Gamma_1 = \overline{y_i:\Gamma(x_i)}$.

STS: $\Delta;\Phi_a;\square\Gamma_1,\Gamma \vdash \text{fix}_{NC}\ f(x).t^*[\overline{\text{der}\ y_i/x_i}] \backsim \text{fix}_{NC}\ f(x).t^*[\overline{\text{der}\ y_i/x_i}] :^c \square(\tau_1\to\tau_2)$.

By **c-r-fixNC**,

$$\frac{\Delta;\Phi_a;x:\tau_1,f:\Box(\tau_1\to\tau_2),\Box\Gamma_1\vdash t^*[\overline{\text{der } y_i/x_i}]\backsim t^*[\overline{\text{der } y_i/x_i}]:^c\tau_2 \quad (\diamond)}{\Delta;\Phi_a;\Box\Gamma_1,\Gamma\vdash \text{fix}_{NC}\ f(x).t^*[\overline{\text{der } y_i/x_i}]\backsim \text{fix}_{NC}\ f(x).t^*[\overline{\text{der } y_i/x_i}]:^c\Box(\tau_1\to\tau_2)}$$

STS: $\Delta;\Phi_a;x:\tau_1,f:\Box(\tau_1\to\tau_2),\Box\Gamma_1\vdash t^*[\overline{\text{der } y_i/x_i}]\backsim t^*[\overline{\text{der } y_i/x_i}]:^c\tau_2$.

From $(\star)$, we extend the context and get:

$\Delta;\Phi_a;x:\tau_1,f:\Box(\tau_1\to\tau_2),\Box\Gamma_1,\Gamma\vdash t^*\backsim t^*:^c\tau_2 \quad (\star\star)$.

By Lemma 26 on $(\star\star)$ for the number of x_i in Γ times, we conclude :

$\Delta;\Phi_a;x:\tau_1,f:\Box(\tau_1\to\tau_2),\Box\Gamma_1\vdash t^*[\overline{\text{der } y_i/x_i}]\backsim t^*[\overline{\text{der } y_i/x_i}]:^c\tau_2$.

Similarly, using $(\star\star)$ and the third premise and **rr-fixNC**, we conclude

$\Delta;\Phi_a;\Gamma\vdash \text{fix } f(x).t\backsim \text{fix } f(x).t:\Box\tau_1\to\tau_2$.

Case

$$\frac{\Delta;\Phi_a;\Gamma\vdash t_1\backsim t_1'\rightsquigarrow {t_1}^*\backsim {t_1'}^*:\tau \quad (\star) \qquad \Delta;\Phi_a;\Gamma\vdash t_2\backsim t_2'\rightsquigarrow {t_2}^*\backsim {t_2'}^*:\mathbf{list}[n]^\alpha\tau \quad (\diamond)}{\Delta;\Phi_a;\Gamma\vdash \mathbf{cons}(t_1,t_2)\backsim\mathbf{cons}(t_1',t_2')\rightsquigarrow\mathbf{cons}_C(t_1^*,t_2^*)\backsim\mathbf{cons}_C(t_1'^*,t_2'^*):\mathbf{list}[n+1]^{\alpha+1}\tau}\ \textbf{e-r-cons1}$$

By IH on the premise $\star$, $\Delta;\Phi_a;\Gamma\vdash t_1^*\backsim t_1'^*:^c\tau \quad (\star\star)$

By IH on the premise $\diamond$, $\Delta;\Phi_a;\Gamma\vdash t_2^*\backsim t_2'^*:^c\text{list}[n]^\alpha\tau \quad (\diamond\diamond)$

By applying **c-r-con1** rule using $\star\star$ and $\diamond\diamond$. we get

$\Delta;\Phi_a;\Gamma\vdash \text{cons}_C(t_1^*,t_2^*)\backsim\text{cons}_C(t_1^*,t_2^*):^c\text{list}[n+1]^{\alpha+1}\tau \quad (\spadesuit)$.

Case

$$\frac{\Delta;\Phi_a;\Gamma\vdash t\backsim t\rightsquigarrow t^*\backsim t^*:\tau \qquad \forall x_i\in dom(\Gamma),\ \ t_i=\mathbf{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \qquad \forall x_i\in dom(\Gamma),\ \ \Delta;\Phi_a\models\Gamma(x_i)\sqsubseteq\Box\Gamma(x_i)}{\Delta;\Phi_a;\Gamma,\Gamma'\vdash t\backsim t\rightsquigarrow\mathbf{let}\ \overline{y_i=t_i\ x_i}\ \mathbf{in\ NC}\ t^*[\overline{\mathbf{der}\ y_i/x_i}]\backsim\mathbf{let}\ \overline{y_i=t_i\ x_i}\ \mathbf{in\ NC}\ t^*[\overline{\mathbf{der}\ y_i/x_i}]:\Box\tau}\ \textbf{e-nochange}$$

To Show:

$\Delta;\Phi_a;\Gamma,\Gamma'\vdash \text{let } \overline{y_i=t_i\ x_i}\text{ in NC } t^*[\overline{\text{der } y_i/x_i}]\backsim\text{let } \overline{y_i=t_i\ x_i}\text{ in NC } t^*[\overline{\text{der } y_i/x_i}]:^c\Box\tau$.

By IH on the first premise $\Delta;\Phi_a;\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau$,

$\Delta;\Phi_a;\Gamma \vdash t^* \backsim t^* :^c \tau \quad (\star)$ and $\Delta;\Phi_a;\Gamma \vdash t \backsim t : \tau \quad (\star\star)$.

By applying **c-r-let** rule.

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma,\Gamma' \vdash t_i\ x_i \backsim t_i\ x_i :^c \Box\Gamma(x_i) \\ \Delta;\Phi_a;\Box\Gamma_1,\Gamma,\Gamma' \vdash \mathrm{NC}\ t^*\,[\overline{\mathrm{der}\ y_i/x_i}] \backsim \mathrm{NC}\ t^*\,[\overline{\mathrm{der}\ y_i/x_i}] :^c \Box\tau\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma,\Gamma' \\ \vdash \mathrm{let}\ \overline{y_i = t_i\ x_i}\ \mathrm{in}\ \mathrm{NC}\ t^*\,[\overline{\mathrm{der}\ y_i/x_i}] \backsim \mathrm{let}\ \overline{y_i = t_i\ x_i}\ \mathrm{in}\ \mathrm{NC}\ t^*\,[\overline{\mathrm{der}\ y_i/x_i}] :^c \Box\tau\end{array}}\ \textbf{c-r-let}$$

Set $\Gamma = \overline{x_i : \Gamma(x_i)}$, $\Gamma_1 = \overline{y_i : \Gamma(x_i)}$.

STS:$\Delta;\Phi_a;\Box\Gamma_1,\Gamma,\Gamma' \vdash \mathrm{NC}\ t^*\,[\overline{\mathrm{der}\ y_i/x_i}] \backsim \mathrm{NC}\ t^*\,[\overline{\mathrm{der}\ y_i/x_i}] :^c \Box\tau$.

By applying **c-nochange**.

$$\frac{\Delta;\Phi_a;\Box\Gamma_1 \vdash t^*\,[\overline{y_i/x_i}] \backsim t^*\,[\overline{y_i/x_i}] :^c \tau}{\Delta;\Phi_a;\Box\Gamma_1,\Gamma,\Gamma' \vdash \mathrm{NC}\ t^*\,[\overline{\mathrm{der}\ y_i/x_i}] \backsim \mathrm{NC}\ t^*\,[\overline{\mathrm{der}\ y_i/x_i}] :^c \Box\tau}\ \textbf{c-nochange}$$

STS: $\Delta;\Phi_a;\Box\Gamma_1 \vdash t^*\,[\overline{y_i/x_i}] \backsim t^*\,[\overline{y_i/x_i}] :^c \tau$.

From $(\star)$, we extend the context and get

$\Delta;\Phi_a;\Gamma,\Box\Gamma_1 \vdash t^* \backsim t^* :^c \tau \quad (\star\star)$

By Lemma 26 on $(\star\star)$ for the number of x_i of Γ times, we conclude :

$\Delta;\Phi_a;\Box\Gamma_1 \vdash t^*\,[\overline{y_i/x_i}] \backsim t^*\,[\overline{y_i/x_i}] :^c \tau$.

Similarly, using $(\star\star)$ and the third premise and **rr-nochange**,

we conclude $\Delta;\Phi_a;\Gamma \vdash t \backsim t : \Box\tau$.

□

Theorem 32 (Completeness of embedding)**.**

1. *If* $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau$*, then there exist* t_1^*, t_2^* *such that*

$$\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t_1^* \backsim t_2^* : \tau.$$

Proof. By induction on the typing derivations.

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t : \tau \qquad \forall x \in dom(\Gamma).\ (\star)\Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\ (\diamond)}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \backsim t : \Box\tau}\ \textbf{rr-nochange}$$

By IH on $(\star)$, we get $\exists\, t^*$ such that $\Delta;\Phi_a;\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau$ $(\dagger)$.

By Lemma 30 on $(\diamond)$, we get $\exists t_i = \text{coerce}_{\Gamma(x_i),\Box(\Gamma(x_i)}$ for all $x_i \in dom(\Gamma)$ $(\dagger\dagger)$.

By **e-nochange** embedding rule using $(\dagger)$ and $(\dagger\dagger)$, we can conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau\ (\dagger) \qquad \forall x_i \in dom(\Gamma),\ t_i = \text{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)}\ (\dagger\dagger) \qquad \forall x_i \in dom(\Gamma),\ \Delta;\Phi_a \models \Gamma(x_i) \sqsubseteq \Box\Gamma(x_i)}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \backsim t \rightsquigarrow \text{let } \overline{y_i = t_i\ x_i} \text{ in NC } t^*\,[\overline{\text{der } y_i / x_i}] \backsim \text{let } \overline{y_i = t_i\ x_i} \text{ in NC } t^*\,[\overline{\text{der } y_i / x_i}] : \Box\tau}$$

e-nochange.

Case

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \textbf{list}[n]^\alpha\,\tau\ (\star) \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' : \tau'\ (\diamond) \\ i,\Delta;\Phi_a \wedge n = i+1;h : \Box\tau, tl : \textbf{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau'\ (\dagger) \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h : \tau, tl : \textbf{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau'\ (\spadesuit)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \textbf{case } t \textbf{ of nil} \to t_1 \mid h :: tl \to t_2 \backsim \textbf{case } t' \textbf{ of nil} \to t_1' \mid h :: tl \to t_2' : \tau'}\ \textbf{rr-caseL}$$

By IH on $(\star)$, we get $\exists\, t^*$ and $\exists\, t'^*$ s.t. $\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \text{list}[n]^\alpha\,\tau$ $(\star\star)$.

By IH on $(\diamond)$, we get $\exists\, t_1^*$ and $\exists\, t_1'^*$ s.t. $\Delta; n \doteq 0 \wedge \Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow t_1^* \backsim t_1'^* : \tau'$ $(\diamond\diamond)$.

By IH on $(\dagger)$, we get $\exists\, t_2^*$ and $\exists\, t_2'^*$ s.t.

$i :: S,\Delta; n \doteq i+1 \wedge \Phi_a; h : \Box\tau, tl : \text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^* \backsim t_2'^* : \tau'$ $(\dagger\dagger)$.

By IH on $(\spadesuit)$, we get $\exists\, t_2^{**}$ and $\exists\, t_2'^{**}$ s.t.

$i :: S,\beta :: S,\Delta; n \doteq i+1 \wedge \alpha \doteq \beta+1 \wedge \Phi_a; h : \tau, tl : \text{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^{**} \backsim t_2'^{**} : \tau'$ $(\spadesuit\spadesuit)$.

By **e-caseL** embedding rule using $(\star\star)$, $(\diamond\diamond)$, and $(\spadesuit\spadesuit)$, we can conclude as follows

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \text{list}[n]^\alpha\,\tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \tau' \qquad i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau, tl:\text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau' \qquad i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h:\tau, tl:\text{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^{**} \backsim t_2'^{**} : \tau'}{\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\text{case } t \text{ of nil } \to t_1 \\ \mid h :: tl \to t_2\end{array} \backsim \begin{array}{l}\text{case } t \text{ of nil } \to t_1' \\ \mid h :: tl \to t_2'\end{array} \rightsquigarrow : \tau'}$$

$$\begin{array}{l}\text{case } t^* \text{ of nil } \to t_1^* \\ \mid h ::_{NC} tl \to t_2^* \\ \mid h ::_C tl \to t_2^{**}\end{array} \backsim \begin{array}{l}\text{case } t'^* \text{ of nil } \to \\ \quad t_1'^* \\ \mid h ::_{NC} tl \to t_2'^* \\ \mid h ::_C tl \to t_2'^{**}\end{array}$$

e-r-caseL

Case

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2 \text{ wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\Box(\tau_1 \to \tau_2),\Gamma \vdash t \backsim t : \tau_2 \quad (\star) \qquad \forall x \in dom(\Gamma).\Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x) \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash \textbf{fix}\ f(x).t \backsim \textbf{fix}\ f(x).t : \Box(\tau_1 \to \tau_2)}\ \textbf{rr-fixNC}$$

By IH on $(\star)$, we get:$\exists\, t^*$ s.t. $\Delta;\Phi_a;x:\tau_1, f:\Box(\tau_1 \to \tau_2),\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau_2 \quad (\star\star)$.

By Lemma 30 on $(\diamond)$, we get $\exists t_i = \text{coerce}_{\Gamma(x_i),\Box(\Gamma(x_i)}$ for all $x_i \in dom(\Gamma)$ $(\diamond\diamond)$.

By **e-fixNC** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude as follows:

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2 \text{ wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\Box(\tau_1 \to \tau_2),\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau_2 \qquad \forall x_i \in dom(\Gamma),\ t_i = \text{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \qquad \forall x_i \in dom(\Gamma),\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x) \qquad t^{**} = \text{let } \overline{y_i = t_i\ x_i} \text{ in } \text{fix}_{NC}\ f(x).t^*[\overline{\text{der } y_i / x_i}]}{\Delta;\Phi_a;\Gamma \vdash \text{fix}\ f(x).t \backsim \text{fix}\ f(x).t \rightsquigarrow t^{**} \backsim t^{**} : \Box(\tau_1 \to \tau_2)}$$

e-r-fixNC.

Case

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \tau \quad (\star) \qquad i \notin \textbf{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda t \backsim \Lambda t' : \forall i{::}S.\tau}\ \textbf{rr-iLam}$$

By IH on $(\star)$, we get $\exists\, t^*$ and $\exists\, t'^*$ such that $i :: S,\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau \quad (\star\star)$.

By **e-iLam** embedding rule using $(\star\star)$, we can conclude as follows:

$$\frac{i :: S, \Delta; \Phi_a; \Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau \qquad i \notin \mathrm{FIV}(\Phi_a; \Gamma)}{\Delta; \Phi_a; \Gamma \vdash \Lambda.t \backsim \Lambda.t' \rightsquigarrow \Lambda i.t^* \backsim \Lambda i.t'^* : \forall i{::}S.\tau}\ \textbf{e-r-iLam}.$$

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \backsim t' : \forall i{::}S.\tau \ \ (\star) \qquad \Delta \vdash I : S \ \ (\diamond)}{\Delta; \Phi_a; \Gamma \vdash t[] \backsim t'[] : \tau\{I/i\}}\ \textbf{rr-iApp}$$

By IH on $(\star)$, we get $\exists\, t^*$ such that $\Delta; \Phi_a; \Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \forall i{::}S.\tau \ \ (\star\star)$.

By **e-iApp** embedding rule using $(\star\star)$ and $(\diamond)$, we can conclude as follows:

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \forall i{::}S.\tau \qquad \Delta \vdash I : S}{\Delta; \Phi_a; \Gamma \vdash t[] \backsim t'[] \rightsquigarrow t^*[I] \backsim t'^*[I] : \tau\{I/i\}}\ \textbf{e-r-iApp}.$$

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \backsim t' : \tau\{I/i\} \ \ (\star) \qquad \Delta \vdash I :: S \ \ (\diamond)}{\Delta; \Phi_a; \Gamma \vdash \textbf{pack}\ t \backsim \textbf{pack}\ t' : \exists i{::}S.\tau}\ \textbf{rr-pack}$$

By IH on $(\star)$, we get $\exists\, t^*$ and $\exists\, t'^*$ such that $\Delta; \Phi_a; \Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau\{I/i\} \ \ (\star\star)$.

By **e-pack** embedding rule using $(\star\star)$ and $(\diamond)$, we can conclude as follows:

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta; \Phi_a; \Gamma \vdash \text{pack } t \backsim \text{pack } t' \rightsquigarrow \text{pack } t^* \text{ with } I \backsim \text{pack } t'^* \text{ with } I : \exists i{::}S.\tau}\ \textbf{e-r-pack}.$$

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_1' : \exists i{::}S.\tau_1 \ \ (\star) \qquad i :: S, \Delta; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \backsim t_2' : \tau_2 \ \ (\diamond) \qquad i \notin FV(\Phi_a; \Gamma, \tau_2, t_2)}{\Delta; \Phi_a; \Gamma \vdash \textbf{unpack}\ t_1\ \textbf{as}\ x\ \textbf{in}\ t_2 \backsim \textbf{unpack}\ t_1'\ \textbf{as}\ x\ \textbf{in}\ t_2' : \tau_2}\ \textbf{rr-unpack1}$$

By IH on $(\star)$, we get $\exists\, t_1^*$ and $\exists\, t_1'^*$ such that $\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_1' \rightsquigarrow t_1^* \backsim t_1'^* : \exists i{::}S.\tau_1 \ \ (\star\star)$.

By IH on $(\diamond)$, we get $\exists\, t_2^*$ and $\exists\, t_2'^*$

such that $i :: S, \Delta; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^* \backsim t_2'^* : \tau_2 \ \ (\diamond\diamond)$.

By **e-unpack** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude as follows:

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \exists i{::}S.\tau_1 \\ i :: S, \Delta;\Phi_a; x:\tau_1, \Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2) \\ t_1^{**} = \text{unpack } t_1^* \text{ as } (x,i) \text{ in } t_2^* \qquad t_2^{**} = \text{unpack } t_1'^* \text{ as } (x,i) \text{ in } t_2'^*\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } x \text{ in } t_2 \backsim \text{unpack } t_1' \text{ as } x \text{ in } t_2' \rightsquigarrow t_1^{**} \backsim t_2^{**} : \tau_2}\ \textbf{e-r-unpack}.$$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau \quad (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau'}\ \textbf{rr-}\sqsubseteq$$

By IH on $(\star)$, we get $\exists\, t_1^*, t_2^*$ such that $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \quad (\star\star)$.

By Lemma 30 on $(\diamond)$, we can show that $\exists t' = \text{coerce}_{\tau,\tau'} \quad (\diamond\diamond)$.

By **e-r-**$\sqsubseteq$ rule using $(\star\star)$, $(\diamond\diamond)$ and $(\diamond)$, we conclude as follows

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \text{coerce}_{\tau,\tau'}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t'\ t_1^* \backsim t'\ t_2^* : \tau'}\ \textbf{e-r-}\sqsubseteq$$

□

B.2.3.2 Proof of Theorem 5

Theorem 33 (Soundness of algorithmic typechecking)**.**

1. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Gamma,\tau) \subseteq dom(\Delta,\psi_a)$ *and* θ_a *is a valid substitution for* ψ_a *such that* $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds.*
   Then, $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \backsim |t'| :^c \tau[\theta_a]$.

2. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$ *and* θ_a *is a valid substitution for* ψ_a *such that* $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds.*
   Then, $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \backsim |t'| :^c \tau[\theta_a]$.

Proof. By simultaneous induction on the given algorithmic typing derivations.

Proof of Theorem 33.1:

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Box\Gamma \vdash t \ominus t \downarrow \tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma',\Box\Gamma \vdash \mathbf{NC}\ t \ominus \mathbf{NC}\ t \downarrow \Box\tau \Rightarrow \Phi}\ \textbf{alg-r-nochange-}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Gamma'[\theta_a],\Box\Gamma[\theta_a] \vdash \mathrm{NC}\,|t| \ominus \mathrm{NC}\,|t| : \Box\tau[\theta_a]$.

By the main assumptions, we have $\mathrm{FIV}(\Phi_a,\Gamma',\Box\Gamma,\tau) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$

Using $(\star)$, we can show that

a) $\mathrm{FIV}(\Phi_a,\Box\Gamma,\tau) \subseteq dom(\Delta,\psi_a)$.

By IH1 on the premise using a) and $(\star\star)$, we can show that

$$\Delta;\Phi_a[\theta_a];\Box\Gamma[\theta_a] \vdash |t| \ominus |t| : \tau[\theta_a] \tag{B.4}$$

By the **c-nochange** rule using eq. (B.4), we obtain

$\Delta;\Phi_a[\theta_a];\Gamma'[\theta_a],\Box\Gamma[\theta_a] \vdash \mathrm{NC}\,|t| \ominus \mathrm{NC}\,|t| : \Box\tau[\theta_a]$.

Case

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \mathbf{list}[n]^{\alpha}\,\tau \Rightarrow \Phi_e \qquad \Delta;\psi_a;n \doteq 0 \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau' \Rightarrow \Phi_1 \\ i :: \mathbb{N},\Delta;\psi_a;n \doteq i+1 \wedge \Phi_a;h : \Box\tau, tl : \mathbf{list}[i]^{\alpha}\,\tau,\Gamma \vdash t_2 \ominus t_2' \downarrow \tau' \Rightarrow \Phi_2 \\ i :: \mathbb{N},\beta :: \mathbb{N},\Delta;\psi_a;n \doteq i+1 \wedge \alpha \doteq \beta+1 \wedge \Phi_a;h : \tau, tl : \mathbf{list}[i]^{\beta}\,\tau,\Gamma \vdash t_3 \ominus t_3' \downarrow \tau' \Rightarrow \Phi_3 \\ \Phi_{body} = (n \doteq 0 \to \Phi_1) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to (\Phi_2 \wedge \forall \beta :: \mathbb{N}.(\alpha \doteq \beta+1) \to \Phi_3))\end{array}}{\begin{array}{lll}\Delta;\psi_a;\Phi_a;\Gamma \vdash | & \mathbf{case}\ t\ \mathbf{of\ nil}\ \to t_1 & \mathbf{case}\ t'\ \mathbf{of\ nil}\ \to t_1' \\ | & h ::_{NC} tl \to t_2 & \ominus\ | \ h ::_{NC} tl \to t_2' \qquad \downarrow \tau' \Rightarrow (\Phi_e \wedge \Phi_{body}) \\ | & h ::_{C} tl \ \to t_3 & |\ h ::_{C} tl \ \to t_3'\end{array}}\ \textbf{alg-r-}$$

caseL-$\downarrow$

TS:

$$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash \begin{array}{l}\mathrm{case}\ t\ \mathrm{of\ nil}\ \to |t_1| \\ h ::_N tl \to |t_2| \\ |\ h ::_C tl \to |t_3|\end{array} \quad \ominus \quad \begin{array}{l}\mathrm{case}\ t'\ \mathrm{of\ nil}\ \to |t_1'| \\ |\ h ::_N tl \to |t_2'| \\ |\ h ::_C tl \to |t_3'|\end{array} \quad : \tau'[\theta_a]$$

By the main assumptions, we have $\mathrm{FIV}(\Phi_a,\Gamma,\tau') \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models (\Phi_e \wedge \Phi_{body})[\theta_a]$ $(\star\star)$

Using $(\star)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta;\Phi_a[\theta_a] \models \Phi_e[\theta_a]$

b) $\Delta; n[\theta_a] \doteq 0 \wedge \Phi_a[\theta_a] \models \Phi_1[\theta_a]$

c) $i :: S, \Delta; n[\theta_a] \doteq i+1 \wedge \Phi_a[\theta_a] \models \Phi_2[\theta_a]$

d) $i :: S, \beta :: S, \Delta; n[\theta_a] \doteq i+1 \wedge \alpha[\theta_a] \doteq \beta+1 \wedge \Phi_a[\theta_a] \models \Phi_3[\theta_a]$

By IH2 on the first premise using a) and ($\star$), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| : \text{list}[n[\theta_a]]^{\alpha[\theta_a]}\, \tau[\theta_a] \tag{B.5}$$

By IH1 on the second premise using b) and ($\star$), we can show that

$$\Delta; n[\theta_a] \doteq 0 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_1| \ominus |t_1'| : \tau'[\theta_a] \tag{B.6}$$

By IH1 on the third premise using c) and ($\star$), we can show that

$$i :: S, \Delta; n[\theta_a] \doteq i+1 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_2| \ominus |t_2'| : \tau'[\theta_a] \tag{B.7}$$

By IH1 on the fourth premise using d) and ($\star$), we can show that

$$i :: S, \beta :: S, \Delta; n[\theta_a] \doteq i+1 \wedge \alpha[\theta_a] \doteq \beta+1 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_3| \ominus |t_3'| : \tau'[\theta_a] \tag{B.8}$$

Then by **c-r-caseL** rule using eqs. (B.5) to (B.8), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \begin{array}{lll} \text{case } t \text{ of nil } \to |t_1| & \text{case } t' \text{ of nil } \to |t_1'| & \\ \mid h ::_N tl \to |t_2| & \mid h ::_N tl \to |t_2'| & : \tau'[\theta_a] \\ \mid h ::_C tl \to |t_3| & \mid h ::_C tl \to |t_3'| & \end{array}$$

Case

$$\frac{\begin{array}{c} i \in \textbf{fresh}(\mathbb{N}) \qquad \Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \Box\tau \Rightarrow \Phi_1 \\ \Delta; i, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow \textbf{list}[i]^{\alpha}\, \tau \Rightarrow \Phi_2 \qquad \Phi_2' = \Phi_2 \wedge n \doteq (i+1) \end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \textbf{cons}_{NC}(t_1, t_2) \ominus \textbf{cons}_{NC}(t_1', t_2') \downarrow \textbf{list}[n]^{\alpha}\, \tau \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}\ \textbf{alg-r-consNC-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \text{cons}_{NC}(|t_1|, |t_2|) \ominus \text{cons}_{NC}(|t_1'|, |t_2'|) : \text{list}[n[\theta_a]]^{\alpha[\theta_a]}\, \tau[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma, \text{list}[n]^{\alpha}\, \tau) \subseteq dom(\Delta, \psi_a)$ ($\star$) and

$\Delta; \Phi_a[\theta_a] \models (\Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2')[\theta_a]$ ($\star\star$)

Using ($\star$), ($\star\star$)'s derivation must be in a form such that we have

a) $\Delta; \Phi_a[\theta_a] \models \Phi_1[\theta_a]$

b) $\Delta \vdash I :: \mathbb{N}$

c) $\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta_a, i \mapsto I]$

d) $\Delta; \Phi_a[\theta_a] \models (I+1) \doteq n[\theta_a]$

By IH1 on the second premise using ($\star$) and a), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_1| \ominus |t_1'| : \Box \tau[\theta_a] \tag{B.9}$$

By IH1 on the third premise using ($\star$) and b), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_2| \ominus |t_2'| : \text{list}[I]^{\alpha[\theta_a]}\, \tau[\theta_a] \tag{B.10}$$

By **c-r-cons2** typing rule using eqs. (B.9) and (B.10), we obtain
$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \text{cons}_{NC}(|t_1|, |t_2|) \ominus \text{cons}_{NC}(|t_1'|, |t_2'|) : \text{list}[I+1]^{\alpha[\theta_a]}\, \tau[\theta_a]$.
We conclude by applying **c-r-⊑** rule to this using d) .

Proof of Theorem 33.2:

Case

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi \qquad \Delta; \Phi_a \vdash \tau \text{ wf} \qquad \textbf{FIV}(\tau) \in \Delta}{\Delta; \psi_a; \Phi_a; \Gamma \vdash (t : \tau) \ominus (t' : \tau) \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-anno-}\uparrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |(t : \tau, t)| \ominus |(t' : \tau, t)| : \tau[\theta_a]$.
Since by definition, $\forall t.\ |(t : \_)| = |t|$, STS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| : \tau[\theta_a]$.
By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma) \subseteq dom(\Delta, \psi_a)$ ($\star$) and
$\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ ($\star\star$)
Using the third premise, we can show that

a) $\text{FIV}(\Phi_a, \Gamma, \tau) \subseteq dom(\Delta, \psi_a)$.

By IH2 on the first premise using ($\star\star$) and a), we can conclude that
$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| : \tau[\theta_a]$.

Case

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \uparrow \Box \tau \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \textbf{der}\ t_1 \ominus \textbf{der}\ t_2 \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-der-}\uparrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \text{der}\,|t| \backsim \text{der}\,|t'| :^c \tau[\theta\,\theta_a]$.

By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Gamma) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$ such that $\Delta \rhd \theta_a : \psi_a$ are derivable.
By IH2 on the first premise using $(\star)$ and $\star\star$, we obtain

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| : \Box\tau[\theta_a] \tag{B.11}$$

Then, by **c-der** rule using eq. (B.62), we can conclude that
$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \mathrm{der}\,|t| \ominus \mathrm{der}\,|t'| : \tau[\theta\,\theta_a]$.

□

B.2.3.3 Proof of Theorem 6

Theorem 34 (Completeness of algorithmic typechecking)**.**

1. *Assume that* $\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_2 :^c \tau$. *Then, there exist* t_1', t_2' *such that* $\Delta; \cdot; \Phi_a; \Gamma \vdash t_1' \ominus t_2' \downarrow \tau \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$ *and* $|t_1'| = t_1$ *and* $|t_2'| = t_2$.

Proof. By simultaneous induction on the given Core typing derivations.

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_2 :^c \Box\tau}{\Delta; \Phi_a; \Gamma \vdash \mathbf{der}\ t_1 \backsim \mathbf{der}\ t_2 :^c \tau}\ \textbf{c-der}$$

By IH on the premise, $\exists t_1', t_2'$ such that

a) $\Delta; \cdot; \Phi_a; \Gamma \vdash t_1' \ominus t_2' \downarrow \tau, t \Rightarrow \Phi$

b) $\Delta; \Phi_a \models \Phi$

c) $|t_1'| = t_1$ and $|t_2'| = t_2$

Then, we can conclude by using a), b) and c) as follows:

$$\frac{\Delta; \cdot; \Phi_a; \Gamma \vdash t_1' \ominus t_2' \downarrow \Box\tau \Rightarrow \Phi}{\Delta; \cdot; \Phi_a; \Gamma \vdash \mathrm{der}\ t_1' \ominus \mathrm{der}\ t_2' \downarrow \tau \Rightarrow \Phi}\ \textbf{alg-r-der-}\downarrow \text{ and}$$

1.

$$\dfrac{\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \Box\tau \Rightarrow \Phi}{\Delta;\cdot;\Phi_a;\Gamma \vdash \text{der}\, t_1' \ominus \text{der}\, t_2' \downarrow \tau \Rightarrow \Phi}\,\textbf{alg-r-der-}\downarrow}{\Delta;\cdot;\Phi_a;\Gamma \vdash (\text{der}\, t_1' : \tau) \ominus (\text{der}\, t_2' : \tau) \uparrow \tau \Rightarrow \Phi}\,\textbf{alg-r-anno-}\uparrow \qquad \Delta;\Phi_a \models \tau \equiv \tau \Rightarrow \Phi' \text{ by } \textit{Lemma}\ 27}{\Delta;\cdot;\Phi_a;\Gamma \vdash (\text{der}\, t_1' : \tau) \ominus (\text{der}\, t_2' : \tau) \downarrow \tau \Rightarrow \Phi \wedge \Phi'}\,\textbf{alg-r-}\uparrow\downarrow$$

2. By c), $|(\text{der}\, t_i' : \tau)| = \text{der}\, |t_i'|$.

3. By b) and Lemma 27.

Case

$$\dfrac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau'}\,\textbf{c-r-}\equiv$$

By IH on the first premise, $\exists t_1', t_2'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1'| = t$ and $|t_2'| = t'$

By IH on the second premise,

d) $\Delta;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_2$

e) $\Delta;\Phi_a \models \Phi_2$.

Then, we can conclude as follows

1. By using a) and d)

$$\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau \Rightarrow \Phi_1}{\Delta;\cdot;\Phi_a;\Gamma \vdash (t_1' : \tau) \ominus (t_2' : \tau) \uparrow \tau \Rightarrow \Phi_1}\ \textbf{alg-r-anno-}\uparrow \qquad \Delta;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_2}{\Delta;\cdot;\Phi_a;\Gamma \vdash (t_1' : \tau) \ominus (t_2' : \tau) \downarrow \tau' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-}\uparrow\downarrow$$

2. By using b), e) ,we can show that $\Delta;\Phi_a \models \Phi_1 \wedge \Phi_2$

3. By c), $|(t_1' : \tau)| = t_1'$ and $(t_2' : \tau) = t_2'$

Case

$$\dfrac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau_1 \rightarrow \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \tau_1}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' :^c \tau_2}\ \textbf{c-r-app}$$

By IH on the first premise, $\exists t_1, t_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1 \rightarrow \tau_2 \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1| = t_1$ and $|t_1'| = t_1'$

By IH on the second premise, $\exists t_2, t_2'$ such that

d) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1 \Rightarrow \Phi_2$

e) $\Delta;\Phi_a \models \Phi_2$

f) $|t_2| = t_2$ and $|t_2'| = t_2'$

Then, we can conclude as follows

1.

$$\dfrac{\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1 \to \tau_2 \Rightarrow \Phi_1}{\Delta;\cdot;\Phi_a;\Gamma \vdash (t1 : \tau_1 \to \tau_2) \ominus t1' : \tau_1 \to \tau_2) \uparrow \tau_1 \to \tau_2 \Rightarrow \Phi_1}\ \textbf{alg-r-anno-}\uparrow \quad \dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1 \Rightarrow \Phi_2}{}}{\Delta;\cdot;\Phi_a;\Gamma \vdash E_1 \ominus E_2 \uparrow \tau_2 \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{c-r-app}}{\Delta;\cdot;\Phi_a;\Gamma \vdash E_1 \ominus E_2 \downarrow \tau_2 \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-}\uparrow\downarrow$$

where $E_1 = (t_1 : \tau_1 \to \tau_2)\ t_2$ and $E_2 = (t_1' : \tau_1 \to \tau_2)\ t_2'$.

2. By using b) and e) .

3. Using c) and f), $|(t_1 : \tau_1 \to \tau_2)\ t_2| = t_1\ t_2$ and $|(t_1' : \tau_1 \to \tau_2)\ t_2'| = t_1'\ t_2'$.

Case

$$\dfrac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \exists i{::}S.\tau_1 \qquad i :: S, \Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_2 \backsim t_2' :^c \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)}{\Delta;\Phi_a;\Gamma \vdash \textbf{unpack}\ t_1\ \textbf{as}\ (x,i)\ \textbf{in}\ t_2 \backsim \textbf{unpack}\ t_1'\ \textbf{as}\ (x,i)\ \textbf{in}\ t_2' :^c \tau_2}\ \textbf{c-r-unpack1}$$

By IH on the first premise, $\exists t_1, t_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \exists i{::}S.\tau_1 \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1| = t_1$ and $|t_1'| = t_1'$

By IH on the second premise, $\exists t_2, t_2'$ such that

d) $i :: S, \Delta;\cdot;\Phi_a;x : \tau_1,\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2 \Rightarrow \Phi_2$

e) $i :: S, \Delta;\Phi_a \models \Phi_2$

f) $|t_2| = t_2$ and $|t_2'| = t_2'$

Then, we can conclude as follows

1.

$$\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \exists i{::}S.\tau_1 \Rightarrow \Phi_1 \;\; (a)}{\Delta;\cdot;\Phi_a;\Gamma \vdash E_1 \ominus E_1' \uparrow \exists i{::}S.\tau_1 \Rightarrow \Phi_1}\,\textbf{alg-r-anno-}\uparrow \qquad i :: S, \Delta;\Phi_a; x:\tau_1,\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2 \Rightarrow \Phi_2 \qquad \Phi' = \Phi_1 \wedge \Phi_2'}{\Delta;\cdot;\Phi_a;\Gamma \vdash \text{unpack } E_1 \text{ as } (x,i) \text{ in } t_2 \ominus \text{unpack } E_1' \text{ as } (x,i) \text{ in } t_2' \downarrow \tau_2 \Rightarrow \Phi'}$$

alg-r-unpack-↓

where $E_1 = (t_1 : \exists i{::}S.\tau_1)$ and $E_2 = (t_1' : \exists i{::}S.\tau_1)$

2. By using b) and t).

3. Using c) and f), $|\text{unpack } (t_1 : \exists i{::}S.\tau_1) \text{ as } (x,i) \text{ in } t_2| = \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2$ and $|\text{unpack } (t_1' : \exists i{::}S.\tau_1) \text{ as } (x,i) \text{ in } t_2'| = \text{unpack } t_1' \text{ as } (x,i) \text{ in } t_2'$.

□

B.3 RELINF

B.3.1 Syntax of RelInf

This section does not appear in the main paper, but it works as a supplemental system that helps us to understand the refinement type $U(A, A)$, which is an important component in later systems.

Unary types	A	::=	$\text{bool} \mid \text{int} \mid A_1 \to A_2$
Relational types	τ	::=	$\text{bool}_r \mid \text{int} \mid \tau_1 \to \tau_2 \mid U(A_1, A_2)$
Terms	t	::=	$x \mid n \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \text{fix } f(x).t \mid t_1\ t_2$
Value	v	:=	$n \mid \text{true} \mid \text{false} \mid \text{fix } f(x).t$

Figure B.25: Syntax of values and terms in Rel Inf.

Unary types	A	::=	$\text{bool} \mid \text{int} \mid A_1 \to A_2$
Relational types	τ	::=	$\text{bool}_r \mid \text{int} \mid \tau_1 \to \tau_2 \mid U(A_1, A_2)$
Terms	t	::=	$x \mid n \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2$
			$\mid \text{fix } f(x).t \mid t_1\ t_2 \mid \text{switch } t$
Value	v	:=	$n \mid \text{true} \mid \text{false} \mid \text{fix } f(x).t$

Figure B.26: Syntax of values and terms in Rel Inf-CORE.

Unary types	A	::=	$\text{bool} \mid \text{int} \mid A_1 \to A_2$
Relational types	τ	::=	$\text{bool}_r \mid \text{int} \mid \tau_1 \to \tau_2 \mid U(A_1, A_2)$
Terms	t	::=	$\ldots \mid \text{switch } t \mid (t : \tau) \mid (t : A)$
Value	v	:=	$n \mid \text{true} \mid \text{false} \mid \text{fix } f(x).t$

Figure B.27: Syntax of values and terms in BiRelInf.

$$\frac{\Omega(x) = A}{\Omega \vdash x : A}\ \textbf{var} \qquad \frac{\mathtt{b} \in \{\text{true}, \text{false}\}}{\Omega \vdash \mathtt{b} : \text{bool}}\ \textbf{bool} \qquad \frac{}{\Omega \vdash \mathtt{n} : \text{int}}\ \textbf{const}$$

$$\frac{\vdash^{\mathsf{A}} A_1 \to A_2\ \mathsf{wf} \qquad x : A_1, f : A_1 \to A_2, \Omega \vdash t : A_2}{\Omega \vdash \text{fix}\ f(x).t : A_1 \to A_2}\ \textbf{fix}$$

$$\frac{\Omega \vdash t_1 : A_1 \to A_2 \qquad \Omega \vdash t_2 : A_1}{\Omega \vdash t_1\ t_2 : A_2}\ \textbf{app} \qquad \frac{\Omega \vdash t : \text{bool} \qquad \Omega \vdash t_1 : \tau \qquad \Omega \vdash t_2 : \tau}{\Omega \vdash \text{if}\ t\ \text{then}\ t_1\ \text{else}\ t_2 : \tau}\ \textbf{if}$$

$$\frac{\Omega \vdash t : A \qquad \models^{\mathsf{A}} A \sqsubseteq A'}{\Omega \vdash t : A'}\ \sqsubseteq_{\text{exec}}$$

Figure B.28: Unary typing rules of Rel Inf.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x \backsim x : \tau}\ \textbf{r-var} \qquad \frac{\mathtt{b} \in \{\text{true}, \text{false}\}}{\Gamma \vdash \mathtt{b} \backsim \mathtt{b} : \text{bool}_r}\ \textbf{r-bool} \qquad \frac{|\Gamma|_1 \vdash t_1 : A_1 \qquad |\Gamma|_2 \vdash t_2 : A_2}{\Gamma \vdash t_1 \backsim t_2 : U(A_1, A_2)}\ \textbf{r-switch}$$

$$\frac{\Gamma \vdash t \backsim t' : \text{bool}_r \qquad \Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Gamma \vdash t_2 \backsim t_2' : \tau}{\Gamma \vdash \text{if}\ t\ \text{then}\ t_1\ \text{else}\ t_2 \backsim \text{if}\ t'\ \text{then}\ t_1'\ \text{else}\ t_2' : \tau}\ \textbf{r-if}$$

$$\frac{\Delta; \Phi_a; x : \tau_1, f : \tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 : \tau_2}{\Delta; \Phi_a; \Gamma \vdash \text{fix}\ f(x).t_1 \backsim \text{fix}\ f(x).t_2 : \tau_1 \to \tau_2}\ \textbf{r-fix}$$

$$\frac{\Gamma \vdash t_1 \backsim t_1' : \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 \backsim t_2' : \tau_1}{\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' : \tau_2}\ \textbf{r-app} \qquad \frac{\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \models \tau \sqsubseteq \tau'}{\Gamma \vdash t_1 \backsim t_2 : \tau'}\ \textbf{r-}\sqsubseteq$$

Figure B.29: Relational typing rules of Rel Inf.

$$\frac{\models^{\mathsf{A}} A_1' \sqsubseteq A_1 \qquad \models^{\mathsf{A}} A_2 \sqsubseteq A_2'}{\models^{\mathsf{A}} A_1 \to A_2 \sqsubseteq A_1' \to A_2'}\ \to_{\textbf{exec}} \qquad \frac{}{\models^{\mathsf{A}} A \sqsubseteq A}\ \textbf{u-refl}$$

$$\frac{\models^{\mathsf{A}} A_1 \sqsubseteq A_2 \qquad \models^{\mathsf{A}} A_2 \sqsubseteq A_3}{\models^{\mathsf{A}} A_1 \sqsubseteq A_3}\ \textbf{u-tran}$$

Figure B.30: Unary subtyping rules of Rel Inf.

$$\frac{}{\models U(A_1 \to A_2, A_1' \to A_2') \sqsubseteq U(A_1, A_1') \to U(A_2, A_2')} \to \textbf{execdiff}$$

$$\frac{\models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \models^{\mathsf{A}} A_2 \sqsubseteq A_2'}{\models U(A_1, A_2) \sqsubseteq U(A_1', A_2')}\ \mathbf{U} \qquad \frac{}{\models \tau \sqsubseteq U(|\tau|1, |\tau|2)}\ \mathbf{W} \qquad \frac{\models \tau_1' \sqsubseteq \tau_1 \qquad \models \tau_2 \sqsubseteq \tau_2'}{\models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'} \to$$

$$\frac{}{\models \tau \sqsubseteq \tau}\ \textbf{refl} \qquad \frac{\models \tau_1 \sqsubseteq \tau_2 \qquad \models \tau_2 \sqsubseteq \tau_3}{\models \tau_1 \sqsubseteq \tau_3}\ \textbf{trans}$$

Figure B.31: Relational subtyping rules of Rel Inf.

$$\begin{array}{rcl} |.|_{i \in \{1,2\}} & : & \text{Binary type} \to \text{Unary type} \\ |\text{int}|_i & = & \text{int} \\ |\text{bool}_r|_i & = & \text{bool} \\ |\tau_1 \to \tau_2|_i & = & |\tau_1|_i \to |\tau_2|_i \\ |U(A_1, A_2)|_i & = & A_i \\ |\Gamma, x : \tau|_i & = & |\Gamma|_i, x : |\tau|_i \\ |\emptyset|_i & = & \emptyset \end{array}$$

Figure B.32: RelInf refinement removal operation.

$$\frac{}{\models \text{bool}_r \equiv \text{bool}_r}\ \textbf{eq-boolr} \qquad \frac{}{\models \text{int} \equiv \text{int}}\ \textbf{eq-intr} \qquad \frac{\models \tau_1 \equiv \tau_1' \qquad \models \tau_2 \equiv \tau_2'}{\models \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'}\ \textbf{eq-fun}$$

$$\frac{\models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \qquad \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \qquad \models^{\mathsf{A}} A_2' \sqsubseteq A_2}{\models U(A_1, A_2) \equiv U(A_1', A_2')}\ \textbf{eq-U}$$

Figure B.33: Relational type equivalence rules of Rel Inf.

$$\frac{x : A_1, f : A_1 \to A_2, \Omega \vdash t :^c A_2}{\Omega \vdash \text{fix } f(x).t :^c A_1 \to A_2}\ \textbf{c-u-fix} \qquad \frac{\Omega(x) = A}{\Omega \vdash x :^c A}\ \textbf{c-u-var} \qquad \frac{}{\Omega \vdash \mathtt{n} :^c \text{int}}\ \textbf{c-u-const}$$

$$\frac{\Omega \vdash t :^c A \qquad \models A \sqsubseteq A'}{\Omega \vdash^{t'}_{k'} t :^c A'}\ \textbf{c-u-}\sqsubseteq$$

Figure B.34: Unary typing rules of Rel Inf Core.

$$\frac{|\Gamma|1 \vdash t_1 :^c A_1 \qquad |\Gamma|2 \vdash t_2 :^c A_2}{\Gamma \vdash \text{switch } t_1 \backsim \text{switch } t_2 :^c U(A_1, A_2)}\ \textbf{c-switch}$$

$$\frac{\vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad x : \tau_1, f : \tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 :^c \tau_2}{\Gamma \vdash \text{fix } f(x).t_1 \backsim \text{fix } f(x).t_2 :^c \tau_1 \to \tau_2}\ \textbf{c-r-fix} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x \backsim x :^c \tau}\ \textbf{c-r-var}$$

$$\frac{}{\Gamma \vdash \mathtt{n} \backsim \mathtt{n} :^c \text{int}}\ \textbf{c-r-const} \qquad \frac{\Gamma \vdash t \backsim t' :^c \tau \qquad \models \tau \equiv \tau'}{\Gamma \vdash t \backsim t' :^c \tau'}\ \textbf{c-r-}\sqsubseteq$$

Figure B.35: Relational typing rules of Rel Inf Core.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x \backsim x \rightsquigarrow x \backsim x : \tau}\ \textbf{e-r-var} \qquad \frac{}{\Gamma \vdash \mathtt{n} \backsim \mathtt{n} \rightsquigarrow \mathtt{n} \backsim \mathtt{n} : \text{int}}\ \textbf{e-r-const}$$

$$\frac{\vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad x : \tau_1, f : \tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau_2}{\Gamma \vdash \text{fix } f(x).t_1 \backsim \text{fix } f(x).t_2 \rightsquigarrow \text{fix } f(x).{t_1}^* \backsim \text{fix } f(x).{t_2}^* : \tau_1 \to \tau_2}\ \textbf{e-r-fix}$$

$$\frac{\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \qquad \models \tau \sqsubseteq \tau' \qquad t' = \text{coerce}_{\tau,\tau'}}{\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t'\ t_1^* \backsim t'\ t_2^* : \tau'}\ \textbf{e-r-}\sqsubseteq$$

$$\frac{|\Gamma|1 \vdash t_1 \rightsquigarrow t_1^* : A_1 \qquad |\Gamma|2 \vdash t_2 \rightsquigarrow t_2^* : A_2}{\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \text{switch } t_1^* \backsim \text{switch } t_2^* : U(A_1, A_2)}\ \textbf{e-switch}$$

Figure B.36: Relational embedding rules of Rel Inf.

$$\frac{\Omega(x) = A}{\Omega \vdash x \rightsquigarrow x : A}\ \textbf{e-u-var} \qquad \frac{}{\Omega \vdash \mathtt{n} \rightsquigarrow \mathtt{n} : \text{int}}\ \textbf{e-u-const}$$

$$\frac{\vdash^{\mathsf{A}} A_1 \to A_2\ \mathsf{wf} \qquad ; x : A_1, f : A_1 \to A_2, \Omega \vdash t \rightsquigarrow t^* : A_2}{\Omega \vdash \text{fix } f(x).t \rightsquigarrow \text{fix } f(x).t^* : A_1 \to A_2}\ \textbf{e-u-fix}$$

$$\frac{\Omega \vdash t \rightsquigarrow t^* : A \qquad \models^{\mathsf{A}} A \sqsubseteq A'}{\Omega \vdash t \rightsquigarrow t^* : A'}\ \textbf{e-u-}\sqsubseteq$$

Figure B.37: Unary embedding rules of Rel Inf.

$$\frac{\Omega(x) = \tau}{\Omega \vdash x \uparrow A}\ \textbf{alg-u-var} \qquad \frac{\mathtt{b} \in \{\text{true}, \text{false}\}}{\Omega \vdash \mathtt{b} \uparrow \text{bool}}\ \textbf{alg-u-bool} \qquad \frac{\Omega \vdash t \uparrow A' \qquad \models A' \sqsubseteq A}{\Omega \vdash t \downarrow A}\ \textbf{alg-u-}\uparrow\downarrow$$

$$\frac{\Omega \vdash t \downarrow A}{\Omega \vdash (t : A) \uparrow A}\ \textbf{alg-u-anno-}\uparrow \qquad \frac{\Omega \vdash t \uparrow \text{bool} \qquad \Omega \vdash t_1 \downarrow A \qquad \Omega \vdash t_2 \downarrow A}{\Omega \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 \downarrow A}\ \textbf{alg-u-if}$$

$$\frac{\Omega, x : A_1, f : A_1 \to A_2 \vdash t_1 \downarrow A_2}{\Omega \vdash \text{fix } f(x).t_1 \downarrow A_1 \to A_2}\ \textbf{alg-u-fix}$$

Figure B.38: Unary algorithmic rules of Rel Inf.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x \ominus x \uparrow \tau}\ \textbf{alg-r-var} \qquad \frac{\mathtt{b} \in \{\text{true}, \text{false}\}}{\Gamma \vdash \mathtt{b} \ominus \mathtt{b} \uparrow \text{bool}_r}\ \textbf{alg-r-bool}$$

$$\frac{|\Gamma|1 \vdash t_1 \uparrow A_1 \qquad |\Gamma|2 \vdash t_2 \uparrow A_2}{\Gamma \vdash \text{switch } t_1 \ominus \text{switch } t_2 \uparrow U(A_1, A_2)}\ \textbf{alg-r-switch}\uparrow$$

$$\frac{|\Gamma|1 \vdash t_1 \downarrow A_1 \qquad |\Gamma|2 \vdash t_2 \downarrow A_2}{\Gamma \vdash \text{switch } t_1 \ominus \text{switch } t_2 \downarrow U(A_1, A_2)}\ \textbf{alg-r-switch}\downarrow$$

$$\frac{\Gamma \vdash t \ominus t' \uparrow \text{bool}_r \qquad \Gamma \vdash t_1 \ominus t_1' \downarrow \tau \qquad \Gamma \vdash t_2 \ominus t_2' \downarrow \tau}{\Gamma \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 \ominus \text{if } t' \text{ then } t_1' \text{ else } t_2' \downarrow \tau}\ \textbf{alg-r-if}$$

$$\frac{\Gamma, x : \tau_1, f : \tau_1 \to \tau_2 \vdash t_1 \ominus t_2 \downarrow \tau_2}{\Gamma \vdash \text{fix } f(x).t_1 \ominus \text{fix } f(x).t_2 \downarrow \tau_1 \to \tau_2}\ \textbf{alg-r-fix}$$

$$\frac{\Gamma \vdash t_1 \ominus t_1' \uparrow \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1}{\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \uparrow \tau_2}\ \textbf{alg-r-app} \qquad \frac{\Gamma \vdash t \ominus t' \uparrow \tau' \qquad \models \tau' \equiv \tau}{\Gamma \vdash t \ominus t' \downarrow \tau}\ \textbf{alg-}\uparrow\downarrow$$

$$\frac{\Gamma \vdash t \ominus t' \downarrow \tau}{\Gamma \vdash (t : \tau) \ominus (t : \tau) \uparrow \tau}\ \textbf{alg-r-anno-}\uparrow$$

Figure B.39: Algorithmic typing rules of Rel Inf.

$$\frac{}{\models \mathrm{bool}_r \equiv \mathrm{bool}_r}\ \textbf{alg-boolr} \qquad \frac{}{\models \mathrm{int} \equiv \mathrm{int}}\ \textbf{alg-intr} \qquad \frac{\models^{\mathsf{A}} A \sqsubseteq A' \qquad \models^{\mathsf{A}} A' \sqsubseteq A}{\models U\,A \equiv U\,A'}\ \textbf{alg-u}$$

$$\frac{\models \tau_1' \equiv \tau_1 \qquad \models \tau_2 \equiv \tau_2'}{\models \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'}\ \textbf{alg-}\to$$

Figure B.40: Algorithmic subtyping rules of Rel Inf.

B.3.2 Metatheory of RelInf

B.3.3 RefInf Proof

Lemma 35 (Reflexivity of Algorithmic Binary Type Equivalence)**.** $\models \tau \equiv \tau$.

Proof. By induction on the binary type.

Case

bool$_r$

It is proved by algorithmic binary type equivalence rule **alg-boolr**.

Case

$\tau_1 \to \tau_2$

By IH on τ_1,$\models \tau_1 \equiv \tau_1$.

By IH on τ_2, $\models \tau_2 \equiv \tau_2$

By the above statements and rule **alg-**$\to$, we conclude

$\models \tau_1 \to \tau_2 \equiv \tau_1 \to \tau_2$.

Case

$U\,(A_1, A_2)$

TS: $\models U\,(A_1, A_2) \equiv U\,(A_1, A_2)$.

By unary subtyping rule **u-refl**, $\models^{\mathsf{A}} A_1 \sqsubseteq A_1$ and $\models^{\mathsf{A}} A_2 \sqsubseteq A_2$.

By the above statements and rule **alg-u**, we conclude

$\models U\,(A_1, A_2) \equiv U\,(A_1, A_2)$.

□

Lemma 36 (Existence of coercions for relational subtyping)**.** *If* $\models \tau \sqsubseteq \tau'$ *then there exists* $\mathit{coerce}_{\tau,\tau'} \in \mathit{Core\ s.t.}\ \cdot \vdash \mathit{coerce}_{\tau,\tau'} \rightharpoonup \mathit{coerce}_{\tau,\tau'} :^c \tau \to \tau'$.

Proof. Proof is by induction on the subtyping derivation. We denote the witness e of type $\tau \to \tau'$ as $\mathrm{coerce}_{\tau,\tau'}$ for clarity.

Case

$$\frac{\models \tau_1' \sqsubseteq \tau_1 \ (\star) \qquad \models \tau_2 \sqsubseteq \tau_2' \ (\diamond)}{\models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'} \to$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau_1',\tau_1} \,.\, \cdot \vdash \text{coerce}_{\tau_1',\tau_1} \backsim \text{coerce}_{\tau_1',\tau_1} :^c \tau_1' \to \tau_1$

By IH on $(\diamond)$, $\exists \text{coerce}_{\tau_2,\tau_2'} \,.\, \cdot \vdash \text{coerce}_{\tau_2,\tau_2'} \backsim \text{coerce}_{\tau_2,\tau_2'} :^c \tau_2 \to \tau_2'$

Then, using these two statements,we can construct the following derivation where

$e = \text{fix } f(x).\text{fix } f(y).\text{coerce}_{\tau_2,\tau_2'} \ (x \ (\text{coerce}_{\tau_1',\tau_1} \ y))$

$\cdot \vdash t \backsim t :^c (\tau_1 \to \tau_2) \to \tau_1' \to \tau_2'$

Case

$$\frac{}{\cdot \models \tau \sqsubseteq U\,(|\tau|1, |\tau|2)} \ \mathbf{W}$$

Then, we can immediately construct the derivation using the rule **c-switch**.

$$\frac{}{\cdot \vdash \text{fix } f(x).\text{switch } x \backsim \text{fix } f(x).\text{switch } x :^c \tau \to U\,(|\tau|1, |\tau|2)}$$

Case

$$\frac{\models \tau_1 \sqsubseteq \tau_2 \ (\star) \qquad \models \tau_2 \sqsubseteq \tau_3 \ (\diamond)}{\models \tau_1 \sqsubseteq \tau_3} \ \mathbf{tran}$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau_1,\tau_2} \,.\, i :: S, \cdot \vdash \text{coerce}_{\tau_1,\tau_2} \backsim \text{coerce}_{\tau_1,\tau_2} :^c \tau_1 \to \tau_2$

By IH on $(\diamond)$, $\exists \text{coerce}_{\tau_2,\tau_3} \,.\, i :: S, \cdot \vdash \text{coerce}_{\tau_2,\tau_3} \backsim \text{coerce}_{\tau_2,\tau_3} :^c \tau_2 \to \tau_3$.

Then, using $(\star)$ and $(\diamond)$, we can construct the derivation simply by function composition

$$\frac{}{\cdot \vdash \text{fix } f(x).\text{coerce}_{\tau_2,\tau_3} \ (\text{coerce}_{\tau_1,\tau_2} \ x) \backsim \text{fix } f(x).\text{coerce}_{\tau_2,\tau_3} \ (\text{coerce}_{\tau_1,\tau_2} \ x) :^c \tau_1 \to \tau_3}$$

Case

$$\frac{}{\models U\,(A_1 \to A_2, A_1' \to A_2') \sqsubseteq U\,(A_1, A_1') \to U\,(A_2, A_2')}\ \to \textbf{execdiff}$$

Then, we can immediately construct the following derivation

where $e = \text{fix}\ f(x).\text{fix}\ f(y).\text{switch}\ (x\ y)$ using the **c-switch** and **c-app** rules.

$\cdot \vdash t \backsim t :^c (U\,(A_1 \to A_2, A_1' \to A_2')) \to U\,(A_1, A_1') \to U\,(A_2, A_2')$.

Case

$$\frac{\models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \models^{\mathsf{A}} A_2 \sqsubseteq A_2'}{\models U\,(A_1, A_2) \sqsubseteq U\,(A_1', A_2')}\ \mathbf{U}$$

Then, we can immediately construct the following derivation

where $e = \text{fix}\ f(x).\text{switch}\ x$ using the **c-switch** and **c-u-$\sqsubseteq$** rules.

$\cdot \vdash t \backsim t :^c (U\,(A_1, A_2)) \to U\,(A_1', A_2')$.

□

Theorem 37 (Types are preserved by embedding)**.**

1. *If* $\Omega \vdash t \rightsquigarrow t^* : A$, *then* $\Omega \vdash t^* :^c A$ *and* $\Omega \vdash t : A$.

2. *If* $\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t_1^* \backsim t_2^* : \tau$, *then* $\Gamma \vdash t_1^* \backsim t_2^* :^c \tau$ *and* $\Gamma \vdash t_1 \backsim t_2 : \tau$.

Proof. By simultaneous induction on the given derivations.

Proof of Theorem 37.1:

Case

$$\frac{\vdash^{\mathsf{A}} A_1 \to A_2\ \mathsf{wf} \qquad ; x : A_1, f : A_1 \to A_2, \Omega \vdash t \rightsquigarrow t^* : A_2}{\Omega \vdash \textbf{fix}\ f(x).t \rightsquigarrow \textbf{fix}\ f(x).t^* : A_1 \to A_2}\ \textbf{e-u-fix}$$

By Theorem 37.1 on the premise, we get $x : A_1, f : A_1 \to A_2, \Omega \vdash t :^c A_2$. Then, by unary core rule **c-fix**,we concludt: $\dfrac{x : A_1, f : A_1 \to A_2, \Omega \vdash t :^c A_2}{\Omega \vdash \text{fix}\ f(x).t :^c A_1 \to A_2}$ **c-u-fix**.

Proof of Theorem 37.2:

Case

$$\frac{|\Gamma|1 \vdash^{t_1}_{k_1} t_1 \rightsquigarrow t_1^* : A_1 \quad (\star) \qquad |\Gamma|2 \vdash^{t_2}_{k_2} t_2 \rightsquigarrow t_2^* : A_2 \quad (\diamond)}{\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \textbf{switch}\ t_1^* \backsim \textbf{switch}\ t_2^* : U\,(A_1, A_2)}\ \textbf{e-switch}$$

By Theorem 37.1 on $(\star)$, we get $\Delta;\Phi;\Omega \vdash t_1^* :^c A_1 \quad (\star\star)$.

By Theorem 37.1 on $(\diamond)$, we get $\Delta;\Phi;\Omega \vdash t_2^* :^c A_2 \quad (\diamond\diamond)$.

Then, we conclude as follows: $\dfrac{|\Gamma|1 \vdash t_1 :^c A_1 \qquad |\Gamma|2 \vdash t_2 :^c A_2}{\Gamma \vdash \text{switch}\ t_1 \backsim \text{switch}\ t_2 :^c U\,(A_1, A_2)}\ \textbf{c-switch}$.

□

Theorem 38 (Completeness of embedding)**.**

1. *If* $\Omega \vdash t : A$*, then there exists an* t^* *such that* $\Omega \vdash t \rightsquigarrow t^* : A$.

2. *If* $\Gamma \vdash t_1 \backsim t_2 : \tau$*, then there exist* t_1^*, t_2^* *such that* $\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t_1^* \backsim t_2^* : \tau$.

Proof. By simultaneous induction on the given typing derivations.

Proof of Theorem 38.1:

Case

$$\frac{\Omega \vdash^{t}_{k} t : A \quad (\star) \qquad \models^{\mathsf{A}} A \sqsubseteq A' \quad (\diamond)}{\Omega \vdash t : A'}\ \sqsubseteq$$

By Theorem 38.1 on $(\star)$, we get $\exists\, t^*$ such that $\Omega \vdash t \rightsquigarrow t^* : A \quad (\star\star)$.

By **e-u-**$\sqsubseteq$ rule using $(\star\star)$, $(\diamond)$, we conclude as follows

$$\frac{\Omega \vdash t \rightsquigarrow t^* : A \qquad \models^{\mathsf{A}} A \sqsubseteq A'}{\Omega \vdash t \rightsquigarrow t^* : A'}\ \textbf{e-u-}\sqsubseteq$$

Proof of Theorem 38.2:

Case

$$\frac{|\Gamma|_1 \vdash t_1 : A_1 \quad (\star) \qquad |\Gamma|_2 \vdash t_2 : A_2 \quad (\diamond)}{\Gamma \vdash t_1 \backsim t_2 : U\,(A_1, A_2)}\ \textbf{r-switch}$$

By Theorem 38.1 on $(\star)$, we get $\exists\, t_1^*$ such that $|\Gamma|_1 \vdash t_1 \rightsquigarrow {t_1}^* : A_1 \quad (\star\star)$.

By Theorem 38.1 on $(\diamond)$, we get $\exists\, t_2^*$ such that $|\Gamma|_2 \vdash t_2 \rightsquigarrow {t_2}^* : A_2 \quad (\diamond\diamond)$.

By **e-switch** embedding rule using ($\star\star$) and ($\diamond\diamond$), we can conclude as follows:

$$\frac{|\Gamma|1 \vdash^{t_1}_{k_1} t_1 \rightsquigarrow t_1^* : A_1 \quad (\star\star) \qquad |\Gamma|2 \vdash^{t_2}_{k_2} t_2 \rightsquigarrow t_2^* : A_2 \quad (\diamond\diamond)}{\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \text{switch } t_1^* \backsim \text{switch } t_2^* : U\,(A_1, A_2)}\ \textbf{e-switch}.$$

□

Theorem 39 (Soundness of algorithmic typechecking)**.**

1. *Assume that* $\Omega \vdash t \downarrow A$*, then,* $\Omega \vdash |t| :^c A$.
2. *Assume that* $\Omega \vdash t \uparrow A$*,then,* $\Omega \vdash |t| :^c A$.
3. *Assume that* $\Gamma \vdash t \ominus t' \downarrow \tau$*,then,* $\Gamma \vdash |t| \backsim |t'| :^c \tau$.
4. *Assume that* $\Gamma \vdash t \ominus t' \uparrow \tau$ *,then,* $\Gamma \vdash |t| \backsim |t'| :^c \tau$.

Proof. By simultaneous induction on the given algorithmic typing derivations.

Proof of Theorem 39.1:

Case

$$\frac{\Omega, x : A_1, f : A_1 \to A_2 \vdash t_1 \downarrow A_2}{\Omega \vdash \textbf{fix}\ f(x).t_1 \downarrow A_1 \to A_2}\ \textbf{alg-u-fix}$$

By IH1 on the premise, we get $\Omega, x : A_1, f : A_1 \to A_2 \vdash |t_1| :^c A_2 \quad (\star)$.

By the unary core rule **c-u-fix** and ($\star$), We conclude $\Omega \vdash \text{fix}\ f(x).|t_1| :^c A_1 \to A_2$.

Proof of Theorem 39.2:

Case

$$\frac{\Omega \vdash t \downarrow A}{\Omega \vdash (t : A) \uparrow A}\ \textbf{alg-u-anno-}\uparrow$$

By IH1 on the premise, we get $\Omega, \vdash |t| :^c A \quad (\star)$.

Because $|(t : A)| = |t|$, We conclude $\Omega \vdash |(t : A)| :^c A$.

Proof of Theorem 39.3:

Case

$$\frac{|\Gamma|1 \vdash t_1 \downarrow A_1 \qquad |\Gamma|2 \vdash t_2 \downarrow A_2}{\Gamma \vdash \textbf{switch}\ t_1 \backsim \textbf{switch}\ t_2 \downarrow U\,(A_1, A_2)}\ \textbf{alg-r-switch}{\downarrow}$$

By IH1 on the first premise, we get $|\Gamma|_1 \vdash |t_1| :^c A_1 \quad (\star)$.

By IH1 on the second premise, we get $|\Gamma|_2 \vdash |t_2| :^c A_2 \quad (\diamond)$.

By relational core rule **c-switch** and $(\star)$ and $(\diamond)$,

We conclude: $\Gamma \vdash \text{switch}\,|t_1| \backsim \text{switch}\ t_1|t_2| :^c U\,(A_1, A_2)$.

Proof of Theorem 39.4:

Case

$$\frac{|\Gamma|1 \vdash t_1 \uparrow A_1 \qquad |\Gamma|2 \vdash t_2 \uparrow A_2}{\Gamma \vdash \textbf{switch}\ t_1 \backsim \textbf{switch}\ t_2 \uparrow U\,(A_1, A_2)}\ \textbf{alg-r-switch}{\uparrow}$$

By IH2 on the first premise, we get $|\Gamma|_1 \vdash |t_1| :^c A_1 \quad (\star)$.

By IH2 on the second premise, we get $|\Gamma|_2 \vdash |t_2| :^c A_2 \quad (\diamond)$.

By relational core rule **c-switch** and $(\star)$ and $(\diamond)$, We conclude that

$\Gamma \vdash \text{switch}\,|t_1| \backsim \text{switch}\ t_1|t_2| :^c U\,(A_1, A_2)$.

□

Theorem 40 (Completeness of algorithmic typechecking)**.**

1. *Assume that* $\Omega \vdash t :^c A$*. Then, there exists* t' *such that* $\Omega \vdash t' \downarrow A$ *and* $|t'| = t$*.*

2. *Assume that* $\Gamma \vdash t_1 \backsim t_2 :^c \tau$*. Then, there exist* t_1', t_2' *such that* $\Gamma \vdash t_1' \ominus t_2' \downarrow \tau$ *and* $|t_1'| = t_1$ *and* $|t_2'| = t_2$*.*

Proof. By simultaneous induction on the given Core typing derivations.

Proof of Theorem 40.1:

Case

$$\frac{\Omega(x) = A}{\Omega \vdash x :^c A}\ \textbf{c-u-var}$$

We can conclude as follows

$$\dfrac{\dfrac{\Omega(x) = \tau}{\Omega \vdash x \uparrow A}\ \textbf{alg-u-var} \qquad \models A \sqsubseteq A}{\Omega \vdash x \downarrow A}\ \textbf{alg-u-}\uparrow\downarrow$$

Case

$$\dfrac{x : A_1, f : A_1 \to A_2, \Omega \vdash t :^c A_2}{\Omega \vdash \textbf{fix}\ f(x).t :^c A_1 \to A_2}\ \textbf{c-u-fix}$$

By IH1 on the premise, we get exists t'' s.t $\Omega \vdash t'' \downarrow A_2 \ \ (\star)$.

By algorithmic unary rule **alg-u-fix** and $(\star)$, we conclude where $t' = \text{fix}\ f(x).t''$ and $|t''| = t''$

$\Omega \vdash t' \downarrow A_1 \to A_2$. and $|t'| = \text{fix}\ f(x).|t''|$.

Proof of Theorem 40.2:

Case

$$\dfrac{\Gamma \vdash t_1 \backsim t_1' :^c \tau_1 \to \tau_2 \qquad \Gamma \vdash t_2 \backsim t_2' :^c \tau_1}{\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' :^c \tau_2}\ \textbf{c-r-app}$$

By IH on the first premise, $\exists t_1, t_1'$ such that

a) $\Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1 \to \tau_2$

b) $|t_1| = t_1$ and $|t_1'| = t_1'$

By IH on the second premise, $\exists t_2, t_2'$ such that

c) $\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1$

d) $|t_2| = t_2$ and $|t_2'| = t_2'$

Then, we can conclude as follows

1.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1 \to \tau_2}{\Gamma \vdash (t_1 : \tau_1 \to \tau_2) \ominus t_1' : \tau_1 \to \tau_2) \uparrow \tau_1 \to \tau_2}\ \textbf{alg-r-anno-}\uparrow \quad \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1 \Rightarrow \Phi_2}{\Gamma \vdash E_1 \ominus E_2 \uparrow \tau_2}\ \textbf{c-r-app}}{\Gamma \vdash E_1 \ominus E_2 \downarrow \tau_2}\ \textbf{alg-r-}\uparrow\downarrow$$

where $E_1 = (t_1 : \tau_1 \to \tau_2)\ t_2$ and $E_2 = (t_1' : \tau_1 \to \tau_2)\ t_2'$.

2. By using b) and e) .
3. Using c) and f), $|(t_1 : \tau_1 \to \tau_2)\ t_2| = t_1\ t_2$ and $|(t_1' : \tau_1 \to \tau_2)\ t_2'| = t_1'\ t_2'$.

□

B.4 RELREFU

B.4.1 Syntax of RelRefU

Unary types	A	::=	$\text{bool} \mid \text{int} \mid A_1 \to A_2 \mid \text{list}[n]\, A \mid \forall i{::}S.\, A \mid \exists i{::}S.\, A$ $\mid C \,\&\, A \mid C \supset A$
Relational types	τ	::=	$\text{bool}_r \mid \text{int} \mid \tau_1 \to \tau_2 \mid U(A_1, A_2) \mid \text{list}[n]^{\alpha}\, \tau \mid \forall i{::}S.\, \tau \mid$ $\exists i{::}S.\, \tau \mid \Box\tau \mid C \,\&\, \tau \mid C \supset \tau$
Terms	t	::=	$x \mid n \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \text{fix } f(x).t \mid t_1\ t_2 \mid \text{nil}$ $\mid \text{cons}(t_1, t_2) \mid \text{ case } t \text{ of nil} \to t_1 \mid h :: tl \to t_2 \mid \Lambda.t \mid t[] \mid$ $\text{pack } t \mid \text{unpack } t_1 \text{ as } x \text{ in } t_2 \mid \text{let } x = t_1 \text{ in } t_2 \mid$ $\text{clet } t_1 \text{ as } x \text{ in } t_2 \mid \text{celim } t$
Value	v	:=	$\mathtt{n} \mid \text{true} \mid \text{false} \mid \text{fix } f(x).t \mid \text{nil} \mid \text{cons}(v_1, v_2) \mid \Lambda t \mid \text{pack } v$

Figure B.41: Syntax of values and terms in RelInfRef.

Unary types	A	::=	$\text{bool} \mid \text{int} \mid A_1 \to A_2 \mid \text{list}[n]\, A \mid \forall i{::}S.\, A \mid \exists i{::}S.\, A$ $\mid C \,\&\, A \mid C \supset A$
Relational types	τ	::=	$\text{bool}_r \mid \text{int} \mid \tau_1 \to \tau_2 \mid U(A_1, A_2) \mid \text{list}[n]^{\alpha}\, \tau \mid \forall i{::}S.\, \tau \mid$ $\exists i{::}S.\, \tau \mid \Box\tau \mid C \,\&\, \tau \mid C \supset \tau$
Terms	t	::=	$x \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \text{fix } f(x).t \mid t_1\ t_2$ $\mid \text{switch } t \mid \text{NC } t \mid \text{split } (t_1, t_2) \text{ with } C \mid \text{contra } t \mid \text{der } t$ $\mid \Lambda i.t \mid t[I] \mid \text{pack } t \text{ with } I \mid \text{fix}_{NC}\ f(x).t$ $\mid eunpack t_1 \text{ as } (x, i) \text{ in } t_2 \mid \text{cons}_{NC}(t_1, t_2) \mid \text{cons}_C(t_1, t_2)$ $\mid \text{let } x = t_1 \text{ in } t_2 \mid \begin{pmatrix} \text{case } t \text{ of nil} \to t_1 \\ \mid h ::_{NC} tl \to t_2 \mid h ::_C tl \to t_3 \end{pmatrix}$
Value	v	:=	$\mathtt{n} \mid \text{fix } f(x).t \mid \mid \text{nil}$ $\mid \text{cons}_{NC}(v_1, v_2) \mid \text{cons}_C(v_1, v_2) \mid \Lambda i.t \mid \text{pack } v \text{ with } I$

Figure B.42: Syntax of values and terms in RelInfRef-CORE.

$$
\begin{array}{lrll}
\text{Unary types} & A & ::= & \text{bool} \mid \text{int} \mid A_1 \to A_2 \mid \text{list}[n]\, A \mid \forall i{::}S.\, A \mid \exists i{::}S.\, A \\
 & & & \mid C \,\&\, A \mid C \supset A \\
\text{Relational types} & \tau & ::= & \text{bool}_r \mid \text{int} \mid \tau_1 \to \tau_2 \mid U\,(A_1, A_2) \mid \text{list}[n]^{\alpha}\, \tau \mid \forall i{::}S.\, \tau \mid \\
 & & & \exists i{::}S.\, \tau \mid \Box \tau \mid C \,\&\, \tau \mid C \supset \tau \\
\text{Terms} & t & ::= & x \mid \text{true} \mid \text{false} \mid \text{if } t \text{ then } t_1 \text{ else } t_2 \mid \text{fix } f(x).t \mid t_1\ t_2 \\
 & & & \mid \text{switch } t \mid \text{NC } t \mid \text{split } (t_1, t_2) \text{ with } C \mid \text{contra } t \mid \text{der } t \\
 & & & \mid \Lambda i.t \mid t[I] \mid \text{pack } t \text{ with } I \mid \text{unpack } t_1 \text{ as } (x, i) \text{ in } t_2 \\
 & & & \mid \text{cons}_{NC}(t_1, t_2) \mid \text{cons}_C(t_1, t_2) \mid \text{fix}_{NC}\ f(x).t \mid (t : \tau) \mid (t : A) \\
 & & & \mid \text{let } x = t_1 \text{ in } t_2 \mid \begin{pmatrix} \text{case } t \text{ of nil } \to t_1 \\ \mid h ::_{NC} tl \to t_2 \mid h ::_C tl \to t_3 \end{pmatrix} \\
\text{Value} & v & := & \mathtt{n} \mid \text{fix } f(x).t \mid \text{fix}_{NC}\ f(x).t \mid \ \mid \text{nil} \\
 & & & \mid \text{cons}_{NC}(v_1, v_2) \mid \text{cons}_C(v_1, v_2) \mid \Lambda i.t \mid \text{pack } v \text{ with } I
\end{array}
$$

Figure B.43: Syntax of values and terms in BiRelInfRef.

$$\frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \backsim x : \tau}\ \textbf{r-var} \qquad \frac{\mathsf{b} \in \{\text{true}, \text{false}\}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{b} \backsim \mathsf{b} : \text{bool}_r}\ \textbf{r-bool}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \text{bool}_r \qquad \Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \tau}{\Delta;\Phi_a;\Gamma \vdash \text{if } t \text{ then } t_1 \text{ else } t_2 \backsim \text{if } t' \text{ then } t_1' \text{ else } t_2' : \tau}\ \textbf{r-if}$$

$$\frac{\Delta;\Phi_a;x : \tau_1, f : \tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t_1 \backsim \text{fix } f(x).t_2 : \tau_1 \to \tau_2}\ \textbf{r-fix}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \tau_1 \to \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \tau_1}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' : \tau_2}\ \textbf{r-app}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t : \tau \\ \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \backsim t : \Box\tau}\ \textbf{r-nochange}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau'}\ \textbf{r-}\sqsubseteq$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x : \tau_1, f : \Box(\tau_1 \to \tau_2), \Gamma \vdash t \backsim t : \tau_2 \\ \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t \backsim \text{fix } f(x).t : \Box(\tau_1 \to \tau_2)}\ \textbf{r-fixNC}$$

$$\frac{\begin{array}{c}i :: S, \Delta;\Phi_a;\Gamma \vdash t \backsim t' : \tau \\ i \notin \text{FIV}(\Phi_a;\Gamma)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \Lambda t \backsim \Lambda t' : \forall i{::}S.\tau}\ \textbf{r-iLam} \qquad \frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \forall i{::}S.\tau \\ \Delta \vdash I : S\end{array}}{\Delta;\Phi_a;\Gamma \vdash t[\,] \backsim t'[\,] : \tau\{I/i\}}\ \textbf{r-iApp}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \text{pack } t \backsim \text{pack } t' : \exists i{::}S.\tau}\ \textbf{r-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \exists i{::}S.\tau_1 \\ i :: S, \Delta;\Phi_a;x : \tau_1, \Gamma \vdash t_2 \backsim t_2' : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } x \text{ in } t_2 \backsim \text{unpack } t_1' \text{ as } x \text{ in } t_2' : \tau_2}\ \textbf{r-unpack1}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' : \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : C \supset \tau}\ \textbf{r-c-impII}$$

Figure B.44: Typing rules of RelInfRef, part 1.

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : C \supset \tau \qquad \Delta;\Phi_a \vDash C}{\Delta;\Phi_a;\Gamma \vdash \text{celim } t \backsim \text{celim } t' : \tau}\ \textbf{r-c-implE}$$

$$\frac{|\Gamma|_1 \vdash t_1 : A_1 \qquad |\Gamma|_2 \vdash t_2 : A_2}{\Gamma \vdash t_1 \backsim t_2 : U\,(A_1, A_2)}\ \textbf{r-switch}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \tau_1 \qquad \Delta;\Phi_a; x:\tau_1,\Gamma \vdash t_2 \backsim t_2' : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \backsim \text{let } x = t_1' \text{ in } t_2' : \tau_2}\ \textbf{r-let}$$

$$\frac{\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{nil } \backsim \text{nil } : \text{list}[0]^\alpha\,\tau}\ \textbf{r-nil}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \text{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}(t_1,t_2) \backsim \text{cons}(t_1',t_2') : \text{list}[n+1]^{\alpha+1}\,\tau}\ \textbf{r-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' : \text{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}(t_1,t_2) \backsim \text{cons}(t_1',t_2') : \text{list}[n+1]^\alpha\,\tau}\ \textbf{r-cons2}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \text{list}[n]^\alpha\,\tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau, tl : \text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h:\tau, tl : \text{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{ case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2 \backsim \text{ case } t' \text{ of nil } \to t_1' \mid h :: tl \to t_2' : \tau'}\ \textbf{r-caseL}$$

$$\frac{\Delta;\Phi_a \vDash C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' : \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : C\,\&\,\tau}\ \textbf{r-c-andI}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : C\,\&\,\tau_1 \qquad \Delta;\Phi_a \wedge C; x:\tau_1,\Gamma \vdash t_2 \backsim t_2' : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \backsim \text{clet } t_1' \text{ as } x \text{ in } t_2' : \tau_2}\ \textbf{r-c-andE}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1 \backsim t_2 : \tau \qquad \Delta \vdash C\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau}\ \textbf{r-split}$$

$$\frac{\Delta;\Phi_a \vDash \bot \qquad \Delta;\Phi_a \vdash \Gamma\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau}\ \textbf{r-contra}$$

Figure B.45: Typing rules of RelInfRef, part 2.

$$\frac{}{\Delta;\Phi_a \models \text{int} \sqsubseteq \Box\text{int}}\ \textbf{int-}\Box \qquad \frac{}{\Delta;\Phi_a \models \Box U(\text{int},\text{int}) \sqsubseteq \text{int}}\ \Box\textbf{U-int}$$

$$\frac{}{\Delta;\Phi_a \models \text{bool}_r \sqsubseteq \Box\text{bool}_r}\ \textbf{bool-}\Box \qquad \frac{\Delta;\Phi_a \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2'}{\Delta;\Phi_a \models \tau_1 \to \tau_2 \sqsubseteq \tau_1' \to \tau_2'}\ \to$$

$$\frac{}{\Delta;\Phi_a \models \tau \sqsubseteq \tau}\ \textbf{refl} \qquad \frac{}{\Delta;\Phi_a \models \Box(\tau_1 \to \tau_2) \sqsubseteq \Box\tau_1 \to \Box\tau_2}\ \Box\textbf{diff}$$

$$\frac{i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad i \notin \text{FIV}(\Phi_a)}{\Delta;\Phi_a \models \forall i{::}S.\,\tau \sqsubseteq \forall i{::}S.\,\tau'}\ \forall\textbf{diff} \qquad \frac{}{\Delta;\Phi_a \models \Box\forall i{::}S.\,\tau \sqsubseteq \forall i{::}S.\,\Box\tau}\ \forall\Box$$

$$\frac{}{\Delta;\Phi_a \models U(\forall i{::}S.\,A, \forall i{::}S.\,A') \sqsubseteq \forall i{::}S.\,U(A,A')}\ \forall\,\textbf{U}$$

$$\frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \leq \alpha' \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models \text{list}[n]^{\alpha}\,\tau \sqsubseteq \text{list}[n']^{\alpha'}\,\tau'}\ \textbf{l1}$$

$$\frac{\Delta;\Phi_a \models \alpha \doteq 0}{\Delta;\Phi_a \models \text{list}[n]^{\alpha}\,\tau \sqsubseteq \text{list}[n]^{\alpha}\,\Box\tau}\ \textbf{l2} \qquad \frac{}{\Delta;\Phi_a \models \text{list}[n]^{\alpha}\,\Box\tau \sqsubseteq \Box(\text{list}[n]^{\alpha}\,\tau)}\ \textbf{l}\Box$$

$$\frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \tau}\ \textbf{T} \qquad \frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \Box\Box\tau}\ \textbf{D} \qquad \frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2}{\Delta;\Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2}\ \textbf{B-}\Box$$

$$\frac{}{\models \tau \sqsubseteq U(|\tau|1, |\tau|2)}\ \textbf{W} \qquad \frac{\models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \models^{\mathsf{A}} A_2 \sqsubseteq A_2'}{\models U(A_1,A_2) \sqsubseteq U(A_1',A_2')}\ \textbf{U}$$

$$\frac{i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \exists i{::}S.\,\tau \sqsubseteq \exists i{::}S.\,\tau'}\ \exists \qquad \frac{}{\Delta;\Phi_a \models \exists i{::}S.\,\Box\tau \sqsubseteq \Box(\exists i{::}S.\,\tau)}\ \exists\Box$$

$$\frac{\Delta;\Phi_a \wedge C \models C' \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models C \,\&\, \tau \sqsubseteq C' \,\&\, \tau'}\ \textbf{c-and} \qquad \frac{}{\Delta;\Phi_a \models C \,\&\, \Box\tau \sqsubseteq \Box(C \,\&\, \tau)}\ \textbf{c-and-}\Box$$

$$\frac{\Delta;\Phi_a \wedge C' \models C \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models C \supset \tau \sqsubseteq C' \supset \tau'}\ \textbf{c-impl} \qquad \frac{}{\Delta;\Phi_a \models \Box(C \supset \tau) \sqsubseteq C \supset \Box\tau}\ \textbf{c-impl-}\Box$$

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2 \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_3}{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_3}\ \textbf{trans}$$

Figure B.46: Relational subtyping rules of RelInfRef.

$$\frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2'}{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \to A_2 \sqsubseteq A_1' \to A_2'}\,\textbf{u-}\to_{\textbf{exec}}$$

$$\frac{i :: S, \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models^{\mathsf{A}} \forall i{::}S.\, A \sqsubseteq \forall i{::}S.\, A'}\,\textbf{u-}\forall_{\textbf{exec}}$$

$$\frac{i :: S, \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models^{\mathsf{A}} \exists i{::}S.\, A \sqsubseteq \exists i{::}S.\, A'}\,\textbf{u-}\exists \qquad \frac{\Delta;\Phi_a \wedge C \models C' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'}{\Delta;\Phi_a \models^{\mathsf{A}} C \,\&\, A \sqsubseteq C' \,\&\, A'}\,\textbf{u-c-and}$$

$$\frac{\Delta;\Phi_a \wedge C' \models C \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'}{\Delta;\Phi_a \models^{\mathsf{A}} C \supset A \sqsubseteq C' \supset A'}\,\textbf{u-c-impl} \qquad \frac{}{\Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A}\,\textbf{u-refl}$$

$$\frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2 \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_3}{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_3}\,\textbf{u-tran}$$

$$\frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'}{\Delta;\Phi_a \models^{\mathsf{A}} \mathrm{list}[n]\, A \sqsubseteq \mathrm{list}[n']\, A'}\,\textbf{u-l}$$

Figure B.47: Unary subtyping rules of RelInfRef.

$\boxed{\Delta;\Phi_a \models \tau_1 \equiv \tau_2}$ checks whether τ_1 is equivalent to τ_2

$$\frac{}{\Delta;\Phi_a \models \text{bool}_r \equiv \text{bool}_r}\ \textbf{eq-bool} \qquad \frac{}{\Delta;\Phi_a \models \text{int} \equiv \text{int}}\ \textbf{eq-int}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta;\Phi_a \models \tau_2 \equiv \tau_2'}{\Delta;\Phi_a \models \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'}\ \textbf{eq-}\forall \qquad \frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \doteq \alpha' \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \text{list}[n]^{\alpha}\,\tau \equiv \text{list}[n']^{\alpha'}\,\tau'}\ \textbf{eq-list}$$

$$\frac{i,\Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \forall i{::}S.\,\tau \equiv \forall i{::}S.\,\tau'}\ \textbf{eq-}\forall \qquad \frac{i,\Delta;\Phi_a \models \tau \equiv \tau' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \exists i{::}S.\,\tau \equiv \exists i{::}S.\,\tau'}\ \textbf{eq-}\exists$$

$$\frac{\Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \Box\tau \equiv \Box\tau'}\ \textbf{eq-B-}\Box \qquad \frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2' \sqsubseteq A_2}{\Delta;\Phi_a \models U(A_1,A_2) \equiv U(A_1',A_2')}\ \textbf{eq-U}$$

$$\frac{\Delta;C \wedge \Phi_a \models C' \qquad \Delta;C' \wedge \Phi_a \models C \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models C \supset \tau \equiv C' \supset \tau'}\ \textbf{eq-c-impl}$$

$$\frac{\Delta;C' \wedge \Phi_a \models C \qquad \Delta;C \wedge \Phi_a \models C' \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau'}\ \textbf{eq-c-prod}$$

Figure B.48: Relational type equivalence rules of RelInfRef.

$$\frac{}{\Delta;\Phi_a;\Omega \vdash \mathtt{n} :^c \text{int}}\ \textbf{c-const} \qquad \frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash x :^c A}\ \textbf{c-var}$$

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A_1 \to A_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x : A_1, f : A_1 \to A_2, \Omega \vdash t :^c A_2}{\Delta;\Phi_a;\Omega \vdash \text{fix } f(x).t :^c A_1 \to A_2}\ \textbf{c-fix}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t_1 :^c A_1 \to A_2 \qquad \Delta;\Phi_a;\Omega \vdash t_2 :^c A_1}{\Delta;\Phi_a;\Omega \vdash t_1\ t_2 :^c A_2}\ \textbf{c-app}$$

$$\frac{i :: S, \Delta;\Phi_a;\Omega \vdash t :^c A \qquad i \notin \text{FIV}(\Phi_a;\Omega)}{\Delta;\Phi_a;\Omega \vdash \Lambda i.t :^c \forall i{::}S.\ A}\ \textbf{c-iLam}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t :^c \forall i{::}S.\ A \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Omega \vdash t[I] :^c A\{I/i\}}\ \textbf{c-iApp} \qquad \frac{\Delta;\Phi_a;\Omega \vdash t :^c A\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Omega \vdash \text{pack } t \text{ with } I :^c \exists i{::}S.\ A}\ \textbf{c-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash t_1 :^c \exists i{::}S.\ A_1 \\ i :: S, \Delta;\Phi_a;x : A_1, \Omega \vdash t_2 :^c A_2 \qquad i \notin FV(\Phi_a;\Gamma, A_2, L_2, t_2)\end{array}}{\Delta;\Phi_a;\Omega \vdash \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2 :^c A_2}\ \textbf{c-unpack}$$

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash \text{nil} :^c \text{list}[0]\ A}\ \textbf{c-nil} \qquad \frac{\Delta;\Phi_a;\Omega \vdash t_1 :^c A \qquad \Delta;\Phi_a;\Omega \vdash t_2 :^c \text{list}[n]\ A}{\Delta;\Phi_a;\Omega \vdash \text{cons}_C(t_1,t_2) :^c \text{list}[n+1]\ A}\ \textbf{c-cons}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t_1 :^c A_1 \qquad \Delta;\Phi_a;x : A_1, \Omega \vdash t_2 :^c A_2}{\Delta;\Phi_a;\Omega \vdash \text{let } x = t_1 \text{ in } t_2 :^c A_2}\ \textbf{c-let}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash t :^c \text{list}[n]\ A \\ \Delta;\Phi_a \wedge n = 0;\Omega \vdash t_1 :^c A' \qquad i, \Delta;\Phi_a \wedge n = i+1;h : A, tl : \text{list}[i]\ A, \Omega \vdash t_2 :^c A'\end{array}}{\Delta;\Phi_a;\Omega \vdash \text{ case } t \text{ of nil } \to t_1 \mid h ::_{NC} tl \to t_2 \mid h ::_C tl \to t_3 :^c A'}\ \textbf{c-caseL}$$

Figure B.49: Unary typing rules of RelInfRef Core, part 1.

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Omega \vdash t :^c A}{\Delta;\Phi_a;\Omega \vdash t :^c C \,\&\, A}\ \textbf{c-candI}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t_1 :^c C \,\&\, A_1 \qquad \Delta;\Phi_a \wedge C;x : A_1,\Omega \vdash t_2 :^c A_2}{\Delta;\Phi_a;\Omega \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 :^c A_2}\ \textbf{c-candE}$$

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash t :^c A}{\Delta;\Phi_a;\Omega \vdash t :^c C \supset A}\ \textbf{c-cimpI} \qquad \frac{\Delta;\Phi_a;\Omega \vdash t :^c C \supset A \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Omega \vdash \text{celim } t :^c A}\ \textbf{c-cimplE}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t :^c A \qquad \Delta;\Phi_a \models A \sqsubseteq A'}{\Delta;\Phi_a;\Omega \vdash t :^c A'}\ \textbf{c-}\sqsubseteq_{\text{exec}}$$

Figure B.50: Unary typing rules of RelInfRef Core, part 2.

$$\frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \backsim x :^c \tau}\ \textbf{c-r-var}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t_1 \backsim \text{fix } f(x).t_2 :^c \tau_1 \to \tau_2}\ \textbf{c-r-fix}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\square(\tau_1 \to \tau_2), \square\Gamma \vdash t \backsim t :^c \tau_2}{\Delta;\Phi_a;\square\Gamma \vdash \text{fix}_{NC}\ f(x).t \backsim \text{fix}_{NC}\ f(x).t :^c \square(\tau_1 \to \tau_2)}\ \textbf{c-r-fixNC}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \text{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}_C(t_1,t_2) \backsim \text{cons}_C(t_1',t_2') :^c \text{list}[n+1]^{\alpha+1}\,\tau}\ \textbf{c-r-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \square\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \text{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}_{NC}(t_1,t_2) \backsim \text{cons}_{NC}(t_1',t_2') :^c \text{list}[n+1]^\alpha\,\tau}\ \textbf{c-r-cons2}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \text{list}[n]^\alpha\,\tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' :^c \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h:\square\tau, tl:\text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' :^c \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h':\tau, tl:\text{list}[i]^\beta\,\tau,\Gamma \vdash t_3 \backsim t_3' :^c \tau'\end{array}}{\begin{array}{rcl}\text{case } t \text{ of nil } & \to t_1 & \text{case } t \text{ of nil } \to t_1' \\ \Delta;\Phi_a;\Gamma \vdash \mid h ::_N tl & \to t_2 & \backsim \mid h ::_N tl \to t_2' \qquad :^c \tau' \\ \mid h' ::_C tl' & \to t_3 & \mid h' ::_C tl' \to t_3'\end{array}}\ \textbf{c-r-caseL}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 :^c \square\tau}{\Delta;\Phi_a;\Gamma \vdash \text{der } t_1 \backsim \text{der } t_2 :^c \tau}\ \textbf{c-der} \qquad \frac{\Delta;\Phi_a;\square\Gamma \vdash t \backsim t :^c \tau}{\Delta;\Phi_a;\square\Gamma,\Gamma' \vdash \text{NC } t \backsim \text{NC } t :^c \square\tau}\ \textbf{c-nochange}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \tau'}\ \textbf{c-r-}\sqsubseteq$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \backsim t_2 :^c \tau \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1' \backsim t_2' :^c \tau}{\Delta;\Phi_a;\Gamma \vdash \text{split } (t_1,t_1') \text{ with } C \backsim \text{split } (t_2,t_2') \text{ with } C :^c \tau}\ \textbf{c-r-split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Gamma\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{contra } t_1 \backsim \text{contra } t_2 :^c \tau}\ \textbf{c-r-contra}$$

Figure B.51: Relational typing rules of RelInfRef Core, part 1.

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' :^c \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c C \supset \tau}\ \textbf{c-r-cimpI} \qquad \frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash \text{celim } t \backsim \text{celim } t' :^c \tau}\ \textbf{c-r-cimplE}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' :^c \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c C \,\&\, \tau}\ \textbf{c-r-candI}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c C \,\&\, \tau_1 \qquad \Delta;\Phi_a \wedge C;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \backsim \text{clet } t_1' \text{ as } x \text{ in } t_2' :^c \tau_2}\ \textbf{c-r-candE}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau_1 \to \tau_2 \\ \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \tau_1\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' :^c \tau_2}\ \textbf{c-r-app}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau_1 \qquad \Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \backsim \text{let } x = t_1' \text{ in } t_2' :^c \tau_2}\ \textbf{c-r-let}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \text{pack } t \text{ with } I \backsim \text{pack } t' \text{ with } I :^c \exists i{::}S.\,\tau}\ \textbf{c-r-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \exists i{::}S.\,\tau_1 \\ i :: S,\Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' :^c \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2 \backsim \text{unpack } t_1' \text{ as } (x,i) \text{ in } t_2' :^c \tau_2}\ \textbf{c-r-unpack1}$$

$$\frac{|\Gamma|1 \vdash t_1 :^c A_1 \qquad |\Gamma|2 \vdash t_2 :^c A_2}{\Gamma \vdash \text{switch } t_1 \backsim \text{switch } t_2 :^c U\,(A_1,A_2)}\ \textbf{c-switch}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \forall i{::}S.\,\tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash t[I] \backsim t'[I] :^c \tau\{I/i\}}\ \textbf{c-r-iApp}$$

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau \qquad i \notin \text{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda i.t \backsim \Lambda i.t' :^c \forall i{::}S.\,\tau}\ \textbf{c-r-iLam}$$

Figure B.52: Relational typing rules of RelInfRef Core, part 2

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A_1 \to A_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:A_1,f:A_1\to A_2,\Omega \vdash t \rightsquigarrow t^* : A_2}{\Delta;\Phi_a;\Omega \vdash \text{fix } f(x).t \rightsquigarrow \text{fix } f(x).t^* : A_1 \to A_2}\ \textbf{e-u-fix}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t_1 \rightsquigarrow {t_1}^* : A_1 \to A_2 \qquad \Delta;\Phi_a;\Omega \vdash t_2 \rightsquigarrow {t_2}^* : A_1}{\Delta;\Phi_a;\Omega \vdash t_1\ t_2 \rightsquigarrow t_1^*\ t_2^* : A_2}\ \textbf{e-u-app}$$

$$\frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash x \rightsquigarrow x : A}\ \textbf{e-u-var} \qquad \frac{\Delta;\Phi_a \vdash A\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash \text{nil} \rightsquigarrow \text{nil} : \text{list}[0]\ A}\ \textbf{e-u-nil}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t_1 \rightsquigarrow {t_1}^* : A \qquad \Delta;\Phi_a;\Omega \vdash t_2 \rightsquigarrow {t_2}^* : \text{list}[n]\ A}{\Delta;\Phi_a;\Omega \vdash \text{cons}(t_1,t_2) \rightsquigarrow \text{cons}_C(t_1^*,t_2^*) : \text{list}[n+1]\ A}\ \textbf{e-u-cons}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : \text{list}[n]\ A \qquad \Delta;\Phi_a \wedge n = 0;\Omega \vdash t_1 \rightsquigarrow {t_1}^* : A' \\ i,\Delta;\Phi_a \wedge n = i+1;h:A,tl:\text{list}[i]\ A,\Omega \vdash t_2 \rightsquigarrow {t_2}^* : A'\end{array}}{\Delta;\Phi_a;\Omega \vdash \begin{array}{l}\text{case } t \text{ of nil } \to t_1 \\ \mid h :: tl \to t_2\end{array} \rightsquigarrow \begin{array}{l}\text{case } t^* \text{ of nil } \to t_1^* \\ \mid h ::_{NC} tl \to t_2^* \qquad : A' \\ \mid h ::_C tl \to t_2^*\end{array}}\ \textbf{e-u-caseL}$$

$$\frac{i :: S,\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : A \qquad i \notin \text{FIV}(\Phi_a;\Omega)}{\Delta;\Phi_a;\Omega \vdash \Lambda.e \rightsquigarrow \Lambda i.t^* : \forall i{::}S.\ A}\ \textbf{e-u-iLam}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : \forall i{::}S.\ A \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Omega \vdash e[] \rightsquigarrow t^*[I] : A\{I/i\}}\ \textbf{e-u-iApp}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : A\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Omega \vdash \text{pack } e \rightsquigarrow \text{pack } t^* \text{ with } I : \exists i{::}S.\ A}\ \textbf{e-u-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash t_1 \rightsquigarrow {t_1}^* : \exists i{::}S.\ A_1 \\ i :: S,\Delta;\Phi_a;x:A_1,\Omega \vdash t_2 \rightsquigarrow {t_2}^* : A_2 \qquad i \notin FV(\Phi_a;\Omega,A_2,k_2,t_2)\end{array}}{\Delta;\Phi_a;\Omega \vdash \text{unpack } t_1 \text{ as } x \text{ in } t_2 \rightsquigarrow \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2 : \text{unpack } t_1^* \text{ as } (x,i) \text{ in } t_2^*\ A_2}$$

e-u-unpack

$$\frac{\Delta;\Phi_a;\Omega \vdash t_1 \rightsquigarrow {t_1}^* : A_1 \qquad \Delta;\Phi_a;x:A_1,\Omega \vdash t_2 \rightsquigarrow {t_2}^* : A_2}{\Delta;\Phi_a;\Omega \vdash \text{let } x = t_1 \text{ in } t_2 \rightsquigarrow \text{let } x = t_1^* \text{ in } t_2^* : A_2}\ \textbf{e-u-let}$$

Figure B.53: Embedding uanry typing rules of RelInfRef, part 1.

$$\frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \backsim x \rightsquigarrow x \backsim x : \tau}\ \textbf{e-r-var}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\tau_1 \to \tau_2, \Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).t_1 \backsim \mathrm{fix}\ f(x).t_2 \rightsquigarrow \mathrm{fix}\ f(x).{t_1}^* \backsim \mathrm{fix}\ f(x).{t_2}^* : \tau_1 \to \tau_2}\ \textbf{e-r-fix}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \to \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1, f:\square(\tau_1 \to \tau_2), \Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau_2 \\ \forall x_i \in dom(\Gamma),\ \ t_i = \mathrm{coerce}_{\Gamma(x_i),\square\Gamma(x_i)} \qquad \forall x_i \in dom(\Gamma),\ \ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \square\Gamma(x) \\ t^{**} = \mathrm{let}\ \overline{y_i = t_i\ x_i}\ \mathrm{in}\ \mathrm{fix}_{NC}\ f(x).t^*\,[\overline{\mathrm{der}\ y_i / x_i}]\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).t \backsim \mathrm{fix}\ f(x).t \rightsquigarrow t^{**} \backsim t^{**} : \square(\tau_1 \to \tau_2)}\ \textbf{e-r-fixNC}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : C \supset \tau}\ \textbf{e-r-c-impI}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash \mathrm{celim}\ t \backsim \mathrm{celim}\ t' \rightsquigarrow \mathrm{celim}\ t^* \backsim \mathrm{celim}\ {t'}^* : \tau}\ \textbf{e-r-c-implE}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : \tau}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim {t'}^* : C \,\&\, \tau}\ \textbf{e-r-andI}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : C \,\&\, \tau_1 \\ \Delta;\Phi_a \wedge C;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau_2\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \mathrm{clet}\ t_1\ \mathrm{as}\ x\ \mathrm{in}\ t_2 \backsim \mathrm{clet}\ t_1'\ \mathrm{as}\ x\ \mathrm{in}\ t_2' \rightsquigarrow \mathrm{clet}\ t_1^*\ \mathrm{as}\ x\ \mathrm{in}\ t_2^* \backsim \mathrm{clet}\ {t_1'}^*\ \mathrm{as}\ x\ \mathrm{in}\ {t_2'}^* : \tau_2\end{array}}\ \textbf{e-r-c-andE}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Omega \vdash t \rightsquigarrow t^* : A}{\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : C \,\&\, A}\ \textbf{e-u-andI}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t_1 \rightsquigarrow {t_1}^* : C \,\&\, A_1 \qquad \Delta;\Phi_a \wedge C;x:A_1,\Omega \vdash t_2 \rightsquigarrow {t_2}^* : A_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{clet}\ t_1\ \mathrm{as}\ x\ \mathrm{in}\ t_2 \rightsquigarrow \mathrm{clet}\ t_1^*\ \mathrm{as}\ x\ \mathrm{in}\ t_2^* : A_2}\ \textbf{e-u-c-andE}$$

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash t \rightsquigarrow t^* : A}{\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : C \supset A}\ \textbf{e-u-c-impI}$$

$$\frac{\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : C \supset A \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Omega \vdash \mathrm{celim}\ t \rightsquigarrow \mathrm{celim}\ t^* : A}\ \textbf{e-u-c-implE}$$

Figure B.54: Embedding typing rules of RelInfRef, part 2.

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \mathrm{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}(t_1,t_2) \backsim \mathrm{cons}(t_1',t_2') \rightsquigarrow \mathrm{cons}_C(t_1^*,t_2^*) \backsim \mathrm{cons}_C(t_1'^*,t_2'^*) : \mathrm{list}[n+1]^{\alpha+1}\,\tau}$$

e-r-cons1

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \mathrm{list}[n]^\alpha\,\tau}{\Delta;\Phi_a;\Gamma \vdash \mathrm{cons}(t_1,t_2) \backsim \mathrm{cons}(t_1',t_2') \rightsquigarrow \mathrm{cons}_{NC}(t_1',t_2') \backsim \mathrm{cons}_{NC}(t_1^*,t_2^*) : \mathrm{list}[n+1]^{\alpha}\,\tau}$$

e-r-cons2

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau \qquad \forall x_i \in dom(\Gamma),\ \ t_i = \mathrm{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)}}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \backsim t \rightsquigarrow \mathrm{let}\ \overline{y_i = t_i\ x_i}\ \mathrm{in\ NC}\ t^*[\overline{y_i/x_i}] \backsim \mathrm{let}\ \overline{y_i = t_i\ x_i}\ \mathrm{in\ NC}\ t^*[\overline{y_i/x_i}] : \Box\tau}$$

e-nochange

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \mathrm{list}[n]^\alpha\,\tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau, tl:\mathrm{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h:\tau, tl:\mathrm{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^{**} \backsim t_2'^{**} : \tau'\end{array}}{\begin{array}{l}\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\mathrm{case}\ t\ \mathrm{of\ nil}\ \to t_1 \\ \mid h :: tl \to t_2\end{array} \backsim \begin{array}{l}\mathrm{case}\ t\ \mathrm{of\ nil}\ \to t_1' \\ \mid h :: tl \to t_2'\end{array} \rightsquigarrow : \tau' \\ \qquad \begin{array}{l}\mathrm{case}\ t^*\ \mathrm{of\ nil}\ \to t_1^* \\ \mid h ::_{NC} tl \to t_2^* \\ \mid h ::_C tl\ \ \to t_2^{**}\end{array} \backsim \begin{array}{l}\mathrm{case}\ t'^*\ \mathrm{of\ nil}\ \to \\ \quad t_1'^* \\ \mid h ::_{NC} tl \to t_2'^* \\ \mid h ::_C tl\ \ \to t_2'^{**}\end{array}\end{array}}$$

e-r-caseL

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau \qquad i \notin \mathrm{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda.t \backsim \Lambda.t' \rightsquigarrow \Lambda i.t^* \backsim \Lambda i.t'^* : \forall i{::}S.\tau}\ \textbf{e-r-iLam}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \forall i{::}S.\tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash t[] \backsim t'[] \rightsquigarrow t^*[I] \backsim t'^*[I] : \tau\{I/i\}}\ \textbf{e-r-iApp}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \mathrm{coerce}_{\tau,\tau'}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t'\ t_1^* \backsim t'\ t_2^* : \tau'}\ \textbf{e-r-}\sqsubseteq$$

Figure B.55: Embedding rules of RelInfRef, part 3.

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \text{pack } t \backsim \text{pack } t' \rightsquigarrow \text{pack } t^* \text{ with } I \backsim \text{pack } t'^* \text{ with } I : \exists i{::}S.\,\tau} \textbf{ e-r-pack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \exists i{::}S.\,\tau_1 \\ i :: S, \Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2) \\ t_1^{**} = \text{unpack } t_1^* \text{ as } (x,i) \text{ in } t_2^* \qquad t_2^{**} = \text{unpack } t_1'^* \text{ as } (x,i) \text{ in } t_2'^*\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } x \text{ in } t_2 \backsim \text{unpack } t_1' \text{ as } x \text{ in } t_2' \rightsquigarrow t_1^{**} \backsim t_2^{**} : \tau_2} \textbf{ e-r-unpack}$$

$$\frac{\begin{array}{c}\Delta;C \wedge \Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \\ \Delta;\neg C \wedge \Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t_1^{**} \backsim t_2^{**} : \tau \qquad \Delta \vdash C \text{ wf}\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \text{split } (t_1^*, t_1^{**}) \text{ with } C \backsim \text{split } (t_2^*, t_2^{**}) \text{ with } C : \tau} \textbf{ e-r-split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Gamma \text{ wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \text{contra } t_1 \backsim \text{contra } t_2 : \tau} \textbf{ e-r-contra}$$

$$\frac{|\Gamma|1 \vdash t_1 \rightsquigarrow t_1^* : A_1 \qquad |\Gamma|2 \vdash t_2 \rightsquigarrow t_2^* : A_2}{\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \text{switch } t_1^* \backsim \text{switch } t_2^* : U\,(A_1, A_2)} \textbf{ e-switch}$$

Figure B.56: Embedding rules of RelInfRef, part 4.

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \to A_2 \sqsubseteq A_1' \to A_2' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-u-fun}$$

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \Rightarrow \Phi_1 \qquad \Phi = n \doteq n' \wedge \Phi}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} \mathrm{list}[n]\ A \sqsubseteq \mathrm{list}[n']\ A' \Rightarrow \Phi}\ \textbf{alg-u-list}$$

$$\frac{i,\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \Rightarrow \Phi_1}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} \forall i{::}S.\, A \sqsubseteq \forall i{::}S.\, A' \Rightarrow \forall i :: S.\Phi_1 \wedge \Phi_2}\ \textbf{alg-u-}\forall$$

$$\frac{i,\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \Rightarrow \Phi \qquad i \notin FV(\Phi_a)}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} \exists i{::}S.\, A \sqsubseteq \exists i{::}S.\, A' \Rightarrow \forall i :: S.\Phi}\ \textbf{alg-u-}\exists$$

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} C \supset A \sqsubseteq C' \supset A' \Rightarrow (C' \to C) \wedge \Phi}\ \textbf{alg-u-c-impl}$$

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} C\ \&\ A \sqsubseteq C'\ \&\ A' \Rightarrow (C \to C') \wedge \Phi}\ \textbf{alg-u-c-prod}$$

Figure B.57: Algortihmic unary subtyping rules of RelInfRef.

$$\frac{}{\Delta;\psi_a;\Phi_a \models \text{int} \equiv \text{int} \Rightarrow \top}\ \textbf{alg-r-int}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a \models \tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-fun}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \text{list}[n]^{\alpha}\,\tau \equiv \text{list}[n']^{\alpha'}\,\tau' \Rightarrow \Phi \wedge n \doteq n' \wedge \alpha \doteq \alpha'}\ \textbf{alg-r-list}$$

$$\frac{i,\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \forall i{::}S.\tau \equiv \forall i{::}S.\tau' \Rightarrow \forall i :: S.\Phi}\ \forall$$

$$\frac{i,\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi \qquad i \notin FV(\Phi_a)}{\Delta;\psi_a;\Phi_a \models \exists i{::}S.\tau \equiv \exists i{::}S.\tau' \Rightarrow \forall i :: S.\Phi}\ \textbf{alg-r-}\exists \qquad \frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \Box\tau_1 \equiv \Box\tau_2 \Rightarrow \Phi}\ \textbf{B-}\Box$$

$$\frac{\begin{array}{ll}\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \Rightarrow \Phi_1 & \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1' \\ \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2 & \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2' \sqsubseteq A_2 \Rightarrow \Phi_2'\end{array}}{\Delta;\psi_a;\Phi_a \models U\,(A_1,A_2) \equiv U\,(A_1',A_2') \Rightarrow \Phi_1 \wedge \Phi_1' \wedge \Phi_2 \wedge \Phi_2'}\ \textbf{U}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models C \supset \tau \equiv C' \supset \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi}\ \textbf{c-impl}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi}\ \textbf{c-prod}$$

Figure B.58: Algorithmic equivalence binary rules of RelInfRef.

$$\frac{\Omega(x) = A}{\Delta;\psi;\Phi_a;\Omega \vdash x \uparrow A \Rightarrow \top}\ \textbf{alg-u-var-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;f : A_1 \to A_2, x : A_1, \Omega \vdash t \downarrow A_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{fix } f(x).t \downarrow A_1 \to A_2 \Rightarrow \Phi}\ \textbf{alg-u-fix-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow A_1 \to A_2 \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a;\Omega \vdash t_2 \downarrow A_1 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1\ t_2 \uparrow A_2 \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-u-app-}\uparrow$$

$$\frac{i :: S, \Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \Lambda i.t \downarrow \forall i{::}S.\, A \Rightarrow \forall i :: S.\Phi}\ \textbf{alg-u-iLam-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow \forall i{::}S.\, A' \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Omega \vdash t\,[I] \uparrow A'\{I/i\} \Rightarrow \Phi}\ \textbf{alg-u-iApp-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A\{I/i\} \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{pack } t \text{ with } I \downarrow \exists i{::}S.\, A \Rightarrow \Phi}\ \textbf{alg-u-pack-}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow \exists i{::}S.\, A_1 \Rightarrow \Phi_1 \qquad i \notin FV(\Phi_a;\Omega, A_2) \\ i :: S, \Delta;\psi_a;\Phi_a;x : A_1, \Omega \vdash t_2 \downarrow A_2 \Rightarrow \Phi_2 \qquad \Phi = \Phi_1 \wedge \forall i :: S.\Phi_2\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{unpack } t_1 \text{ as } (x, i) \text{ in } t_2 \downarrow A_2 \Rightarrow \Phi}\ \textbf{alg-u-unpack-}\downarrow$$

$$\frac{\begin{array}{c}i \in \text{fresh}(\mathbb{N}) \qquad \Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \downarrow A \Rightarrow \Phi_1 \\ \Delta; i, \psi_a;\Phi_a;\Omega \vdash t_2 \downarrow \text{list}[i]\, A \Rightarrow \Phi_2 \qquad \Phi_2' = \Phi_2 \wedge n \doteq (i+1)\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{cons}_C(t_1, t_2) \downarrow \text{list}[n]\, A \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}\ \textbf{alg-u-cons-}\downarrow$$

$$\frac{\Delta, \psi_a;\Phi \vdash^{\mathsf{A}} A\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{nil} \downarrow \text{list}[n]\, A \Rightarrow n \doteq 0}\ \textbf{alg-u-nil-}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow \text{list}[n]\, A \Rightarrow \Phi_1 \qquad \Delta;\psi_a; n \doteq 0 \wedge \Phi_a;\Omega \vdash t_1 \downarrow A' \Rightarrow \Phi_2 \\ i \in \text{fresh}(\mathbb{N}) \qquad i :: \mathbb{N}, \Delta;\psi_a; n \doteq i+1 \wedge \Phi_a; h : A, tl : \text{list}[i]\, A, \Omega \vdash t_2 \downarrow A' \Rightarrow \Phi_3 \\ \Phi_{body} = (n \doteq 0 \to \Phi_2) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to \Phi_3)\end{array}}{\begin{array}{l}\text{case } t \text{ of nil } \to t_1 \\ \Delta;\psi_a;\Phi_a;\Omega \vdash \mid h ::_{NC} tl \to t_2 \qquad \downarrow A' \Rightarrow (\Phi_1 \wedge \Phi_{body}) \\ \qquad\quad \mid h ::_C tl \ \to t_3\end{array}}$$

$$\textbf{alg-u-caseL-}\downarrow$$

Figure B.59: Algorithmic unary rules of RelInfRef, part 1.

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow A_1 \Rightarrow \Phi_1 \qquad \Delta;\psi_a;x:A_1,\Omega \vdash t_2 \downarrow A_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{let } x = t_1 \text{ in } t_2 \downarrow A_2 \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi_2}\ \textbf{alg-u-let-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash^{\mathsf{A}} A\ \mathsf{wf} \qquad \mathrm{FIV}(A,k,t) \in \Delta}{\Delta;\psi_a;\Phi_a;\Omega \vdash (t:A,k,t) \uparrow A \Rightarrow \Phi}\ \textbf{alg-u-anno-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A' \sqsubseteq A \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-}\uparrow\downarrow$$

$$\frac{\Delta;\psi_a;\Phi \wedge C;\Omega \vdash t \downarrow A \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow k \Rightarrow C \wedge (C \to \Phi)}\ \textbf{alg-u-c-andI}\downarrow$$

$$\frac{\Delta;\Phi \wedge C;\Omega \vdash t \downarrow A \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow C \supset A \Rightarrow C \to \Phi}\ \textbf{alg-u-c-implI}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow C \supset A \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{celim } t \uparrow A \Rightarrow C \wedge \Phi}\ \textbf{alg-u-c-implE}\uparrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow C\ \&\ A_1 \Rightarrow \Phi_1 \\ k_2,t_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta;k_2,t_2,\psi,\psi_a;\Phi \wedge C;x:A_1,\Omega \vdash t_2 \downarrow A_2 \Rightarrow \Phi_2 \\ \Phi_2' = C \to \Phi_2 \wedge k \doteq (k_1+k_2) \wedge (t_1+t_2) \doteq t\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \downarrow A_2 \Rightarrow \exists(\psi).(\Phi_1 \wedge \exists k_2,t_2 :: \mathbb{R}.\Phi_2')}\ \textbf{alg-u-c-andE}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;C \wedge \Phi_a;\Omega \vdash t_1 \downarrow A \Rightarrow \Phi_1 \\ \Delta;\psi_a;\neg C \wedge \Phi_a;\Omega \vdash t_2 \downarrow A \Rightarrow \Phi_2 \qquad \Delta \vdash C\ \mathsf{wf}\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{split } (t_1,t_2) \text{ with } C \downarrow A \Rightarrow C \to \Phi_1 \wedge \neg C \to \Phi_2}\ \textbf{alg-u-split}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a \models \bot}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{contra } t \downarrow A \Rightarrow \top}\ \textbf{alg-u-contra}\downarrow$$

Figure B.60: Algorithmic unary rules of RelInfRef, part 2.

$$\frac{\Gamma(x) = \tau}{\Delta;\psi_a;\Phi_a;\Gamma \vdash x \ominus x \uparrow \tau \Rightarrow \top}\ \textbf{alg-r-var-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;f:\tau_1 \to \tau_2, x:\tau_1,\Gamma \vdash t \ominus t' \downarrow \tau_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{fix } f(x).t \ominus \text{fix } f(x).t' \downarrow \tau_1 \to \tau_2 \Rightarrow \Phi}\ \textbf{alg-r-fix-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;f:\Box(\tau_1 \to \tau_2), x:\tau_1,\Box\Gamma \vdash t \ominus t' \downarrow \tau_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma',\Box\Gamma \vdash \text{fix}_{NC}\ f(x).t \ominus \text{fix}_{NC}\ f(x).t \downarrow \Box(\tau_1 \to \tau_2) \Rightarrow \Phi}\ \textbf{alg-r-fix-}\downarrow\Box$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \tau_1 \to \tau_2 \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \uparrow \tau_2 \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-app-}\uparrow$$

$$\frac{\Delta,\psi_a;\Phi \vdash \tau\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{nil} \ominus \text{nil} \downarrow \text{list}[n]^\alpha\,\tau \Rightarrow n \doteq 0}\ \textbf{alg-r-nil-}\downarrow$$

$$\frac{\begin{array}{c} i,\beta \in \text{fresh}(\mathbb{N}) \qquad \Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau \Rightarrow \Phi_1 \\ \Delta;i,\beta,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \text{list}[i]^\beta\,\tau \Rightarrow \Phi_2 \qquad \Phi_2' = n \doteq (i+1) \wedge \exists\beta :: \mathbb{N}.\Phi_2 \wedge \alpha \doteq \beta + 1 \end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{cons}_C(t_1,t_2) \ominus \text{cons}_C(t_1',t_2') \downarrow \text{list}[n]^\alpha\,\tau \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}$$

alg-r-consC-↓

$$\frac{\begin{array}{c} i \in \text{fresh}(\mathbb{N}) \qquad \Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \Box\tau \Rightarrow \Phi_1 \\ \Delta;i,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \text{list}[i]^\alpha\,\tau \Rightarrow \Phi_2 \qquad \Phi_2' = \Phi_2 \wedge n \doteq (i+1) \end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{cons}_{NC}(t_1,t_2) \ominus \text{cons}_{NC}(t_1',t_2') \downarrow \text{list}[n]^\alpha\,\tau \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}$$

alg-r-consNC-↓

$$\frac{\begin{array}{c} \Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \text{list}[n]^\alpha\,\tau \Rightarrow \Phi_e \qquad \Delta;\psi_a;n \doteq 0 \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau' \Rightarrow \Phi_1 \\ i :: \mathbb{N},\Delta;\psi_a;n \doteq i+1 \wedge \Phi_a;h:\Box\tau, tl:\text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \ominus t_2' \downarrow \tau' \Rightarrow \Phi_2 \\ i :: \mathbb{N},\beta :: \mathbb{N},\Delta;\psi_a;n \doteq i+1 \wedge \alpha \doteq \beta+1 \wedge \Phi_a;h:\tau, tl:\text{list}[i]^\beta\,\tau,\Gamma \vdash t_3 \ominus t_3' \downarrow \tau' \Rightarrow \Phi_3 \\ \Phi_{body} = (n \doteq 0 \to \Phi_1) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to (\Phi_2 \wedge \forall\beta :: \mathbb{N}.(\alpha \doteq \beta+1) \to \Phi_3)) \end{array}}{\begin{array}{l} \Delta;\psi_a;\Phi_a;\Gamma \vdash \left|\begin{array}{l} \text{case } t \text{ of nil } \to t_1 \\ \quad\left|\begin{array}{l} h ::_{NC} tl \to t_2 \\ h ::_C tl \to t_3 \end{array}\right. \end{array}\right. \ominus \left|\begin{array}{l} \text{case } t' \text{ of nil } \to t_1' \\ \quad\left|\begin{array}{l} h ::_{NC} tl \to t_2' \\ h ::_C tl \to t_3' \end{array}\right. \end{array}\right. \downarrow \tau' \Rightarrow \Phi_e \wedge \Phi_{body} \end{array}}$$

alg-r-caseL-↓

Figure B.61: Algorithmic relational typing rules of RelInfRef, part 1.

$$\frac{i :: S, \Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \Lambda i.t \ominus \Lambda i.t' \downarrow \forall i{::}S.\tau \Rightarrow (\forall i :: S.\Phi)}\ \textbf{alg-r-iLam-}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \uparrow \forall i{::}S.\tau' \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Gamma \vdash t\,[I] \ominus t'\,[I] \uparrow \tau'\{I/i\} \Rightarrow \Phi}\ \textbf{alg-r-iApp-}\uparrow$$

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \downarrow \tau\{I/i\} \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \text{pack } t \text{ with } I \ominus \text{pack } t' \text{ with } I \downarrow \exists i{::}S.\tau \Rightarrow \Phi}\ \textbf{alg-r-pack-}\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \uparrow \exists i{::}S.\tau_1 \Rightarrow \Phi_1 \\ i :: S, \Delta; \psi_a; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2 \Rightarrow \Phi_2 \qquad i \notin FV(\Phi_a; \Gamma, \tau_2) \qquad \Phi = (\Phi_1 \wedge \forall i :: S.\Phi_2)\end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \text{unpack } t_1 \text{ as } (x, i) \text{ in } t_2 \ominus \text{unpack } t_1' \text{ as } (x, i) \text{ in } t_2' \downarrow \tau_2 \Rightarrow \Phi}$$

alg-r-unpack-↓

$$\frac{\Delta; \psi_a; \Phi \wedge C; \Gamma \vdash t_1 \ominus t_1 \downarrow \tau \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \downarrow C \,\&\, \tau \Rightarrow C \wedge (C \rightarrow \Phi)}\ \textbf{alg-r-c-andI}\downarrow$$

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \uparrow C \,\&\, \tau_1 \Rightarrow \Phi_1 \\ \Delta; \psi_a; \Phi \wedge C; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2 \Rightarrow \Phi_2 \qquad \Phi_2' = C \rightarrow \Phi_2\end{array}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \ominus \text{clet } t_1' \text{ as } x \text{ in } t_2' \downarrow \tau_2 \Rightarrow \Phi_1 \wedge \Phi_2'}\ \textbf{alg-r-c-andE}\downarrow$$

Figure B.62: Algorithmic relational typing rules of RelInfRef, part 2.

$$\frac{\Delta;\psi_a;\Phi_a;\Box\Gamma \vdash t \ominus t \downarrow \tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma',\Box\Gamma \vdash \mathrm{NC}\ t \ominus \mathrm{NC}\ t \downarrow \Box\tau \Rightarrow \Phi}\ \textbf{alg-r-nochange-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \uparrow \Box\tau \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{der}\ t_1 \ominus \mathrm{der}\ t_2 \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-der-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models \tau' \equiv \tau \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-}\uparrow\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash \tau\ \mathsf{wf} \qquad \mathrm{FIV}(\tau) \in \Delta}{\Delta;\psi_a;\Phi_a;\Gamma \vdash (t:\tau) \ominus (t':\tau) \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-anno-}\uparrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;C \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau \Rightarrow \Phi_1 \\ \Delta;\psi_a;\neg C \wedge \Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau \Rightarrow \Phi_2 \qquad \Delta \vdash C\ \mathsf{wf}\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{split}\ (t_1,t_2)\ \mathrm{with}\ C \ominus \mathrm{split}\ (t_1',t_2')\ \mathrm{with}\ C \downarrow \tau \Rightarrow C \to \Phi_1 \wedge \neg C \to \Phi_2}\ \textbf{alg-r-split}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a \models \bot}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{contra}\ t \ominus \mathrm{contra}\ t' \downarrow \tau \Rightarrow \top}\ \textbf{alg-r-contra}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi;|\Gamma|1 \vdash t_1 \uparrow A_1 \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi;|\Gamma|2 \vdash t_2 \uparrow A_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{switch}\ t_1 \ominus \mathrm{switch}\ t_2 \uparrow U\,(A_1,A_2) \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-switch}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi;|\Gamma|1 \vdash t_1 \downarrow A_1 \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi;|\Gamma|2 \vdash t_2 \downarrow A_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{switch}\ t_1 \ominus \mathrm{switch}\ t_2 \downarrow U\,(A_1,A_2) \Rightarrow (\Phi_1 \wedge \Phi_2)}\ \textbf{alg-r-switch}\downarrow$$

Figure B.63: Algorithmic relational typing rules of RelInfRef, part 3.

B.4.2 Metatheory of RelRefU

We also use the name RelInfRef for RelRefU.

Lemma 41 (Substitution of RelInfRef unary Core)**.** *Assume that*

1. $\Delta; \Psi_a; \Omega \vdash t_1 :^c A$ (1).

2. $\Delta; \Psi_a; \Omega, x : A \vdash t_1' :^c A'$ (2).

then $\Delta; \Psi_a; \Omega \vdash t_1'[t_1/x] :^c A'$.

Proof. By induction on the typing derivation of the second assumption (2).

Case

$$\frac{\Omega(x) = A}{\Delta; \Phi_a; \Omega \vdash x :^c A}\ \textbf{c-var}$$

Subcase: $z = x$.

TS: $\Delta; \Psi_a; \Omega \vdash z[t_1/x] :^c A'$.

Because $x = z, z[t_1/x] = t_1$, STS: $\Delta; \Phi_a; \Gamma \vdash t_1 :^c A'$.

From $(\Omega, x : A)(z) = A'$ and $x = z$, we know $A = A'$.

By the above statements, STS: $\Delta; \Psi_a; \Omega \vdash t_1 :^c A$, which is proved by the assumption.

Subcase: $z \neq x$.

we know: $\Omega(z) = A'$ $(\star)$.

TS: $\Delta; \Psi_a; \Omega \vdash z[t_1/x] :^c A'$.

Because $x \neq z$, $z[t_1/x] = z$, STS: $\Delta; \Psi_a; \Omega \vdash z :^c A'$.

By the core rule **c-r-vars** and $(\star)$, we construct the derivation :

$$\frac{\Omega(z) = A'}{\Delta; \Psi_a; \Omega \vdash z :^c A'}\ \textbf{c-r-var}$$

Case

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A_1 \to A_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:A_1,f:A_1\to A_2,\Omega \vdash t :^c A_2}{\Delta;\Phi_a;\Omega \vdash \mathbf{fix}\ f(x).t :^c A_1 \to A_2}\ \textbf{c-fix}$$

$t_1' = \text{fix}\ f(z).t_1$, $A' = A_1 \to A_2$.

TS: $\Delta;\Phi_a;\Omega \vdash \text{fix}\ f(z).(t_1''[t_1/x]) :^c A'$.

By IH on (1) and (3), $\Delta;\Psi_a;\Omega \vdash t_1''[t_1/x] :^c A_2$ (4).

By core rule **c-fix** and (4), we can derive :

$\Delta;\Phi_a;\Omega \vdash \text{fix}\ f(z).(t_1''[t_1/x]) :^c A'$. □

Lemma 42 (Substitution of RelInfRef Core)**.** *Assume that*

1. $\Delta;\Psi_a;\Gamma \vdash t_1 \backsim t_2 :^c \tau$ (1).
2. $\Delta;\Psi_a;\Gamma,x:\tau \vdash t_1' \backsim t_2' :^c \tau'$ (2).

then $\Delta;\Psi_a;\Gamma \vdash t_1'[t_1/x] \backsim t_2'[t_2/x] :^c \tau'$.

Proof. By induction on the typing derivation of the second assumption (2).

Case

$$\frac{(\Gamma,x:\tau)(z) = \tau'}{\Delta;\Phi_a;\Gamma,x:\tau \vdash z \backsim z :^c \tau'}\ \textbf{c-r-var}$$

Subcase: $z = x$.

TS: $\Delta;\Phi_a;\Gamma \vdash z[t_1/x] \backsim z[t_2/x] :^c \tau'$.

Because $x = z$,$z[t_1/x] = t_1$, $z[t_2/x] = t_1$, STS: $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 :^c \tau'$.

From $(\Gamma,x:\tau)(z) = \tau'$ and $x = z$, we know $\tau = tau'$.

By the above statements, STS: $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 :^c \tau$, which is proved by the assumption.

Subcase: $z \neq x$.

we know: $\Gamma(z) = \tau'$ ($\star$).

TS: $\Delta;\Phi_a;\Gamma \vdash z[t_1/x] \backsim z[t_2/x] :^c \tau'$.

Because $x \neq z$, $z[t_1/x] = z$, $z[t_2/x] = z$, STS: $\Delta;\Phi_a;\Gamma \vdash z \backsim z :^c \tau'$.

By the core rule **c-r-vars** and ($\star$), we construct the derivation :

$$\frac{\Gamma(z) = \tau'}{\Delta;\Phi_a;\Gamma \vdash z \backsim z :^c \tau'}\ \textbf{c-r-var}$$

Case

$$\frac{\Delta;\Phi_a;\Gamma, x:\tau \vdash t_1'' \backsim t_2'' :^c \Box\tau' \quad (3)}{\Delta;\Phi_a;\Gamma, x:\tau \vdash \textbf{der}\ t_1'' \backsim \textbf{der}\ t_2'' :^c \tau'}\ \textbf{c-der}$$

$t_1' = \text{der}\ t_1''$, $t_2' = \text{der}\ t_2''$.

TS: $\Delta;\Phi_a;\Gamma \vdash \text{der}\ (t_1''[t_1/x]) \backsim \text{der}\ (t_2''[t_2/x]) :^c \tau'$.

By IH on (1) and (3), $\Delta;\Psi_a;\Gamma \vdash t_1''[t_1/x] \backsim t_2''[t_2/x] :^c \Box\tau'$ (4).

By core rule **c-der** and (4), we can derive

$$\frac{\Delta;\Phi_a;\Gamma, x:\tau \vdash t_1''[t_1/x] \backsim t_2''[t_2/x] :^c \Box\tau' \quad (4)}{\Delta;\Phi_a;\Gamma, x:\tau \vdash \text{der}\ (t_1''[t_1/x]) \backsim \text{der}\ (t_2''[t_2/x]) :^c \tau'}\ \textbf{c-der}.$$

Case

$$\frac{\Delta;\Phi_a;z:\tau_1, f:\tau_1 \to \tau_2, \Gamma, x:\tau \vdash t_1'' \backsim t_2'' :^c \tau_2 \quad (3)}{\Delta;\Phi_a;\Gamma, x:\tau \vdash \textbf{fix}\ f(z).t_1'' \backsim \textbf{fix}\ f(z).t_2'' :^c \tau_1 \to \tau_2}\ \textbf{c-r-fix}$$

$t_1' = \text{fix}\ f(z).t_1$, $t_2' = \text{fix}\ f(z).t_2''$, $\tau' = \tau_1 \to \tau_2$.

TS: $\Delta;\Phi_a;\Gamma \vdash \text{fix}\ f(z).(t_1''[t_1/x]) \backsim \text{fix}\ f(z).(t_2''[t_2/x]) :^c \tau'$.

By IH on (1) and (3), $\Delta;\Psi_a;\Gamma \vdash t_1''[t_1/x] \backsim t_2''[t_2/x] :^c \tau_2$ (4).

By core rule **c-r-fix** and (4), we can derive :

$\Delta;\Phi_a;\Gamma \vdash \text{fix}\ f(z).(t_1''[t_1/x]) \backsim \text{fix}\ f(z).(t_2''[t_2/x]) :^c \tau'$.

Case

$$\frac{\Delta;\Phi_a;z:\tau_1, f:\Box(\tau_1 \to \tau_2), \Box\Gamma, x:\tau \vdash t \backsim t :^c \tau_2 \quad (3)}{\Delta;\Phi_a;\Box\Gamma, \Gamma', x:\tau \vdash \textbf{fix}_{NC}\ f(z).t \backsim \textbf{fix}_{NC}\ f(z).t :^c \Box(\tau_1 \to \tau_2)}\ \textbf{c-r-fixNC}$$

$t_1' = \text{fix}\ f(z).t$, $t_2' = \text{fix}\ f(z).t$, $\tau' = \Box(\tau_1 \to \tau_2)$.

TS: $\Delta;\Phi_a;\Box\Gamma, \Gamma' \vdash \text{fix}\ f(z).(t[t_1/x]) \backsim \text{fix}\ f(z).(t[t_2/x]) :^c \tau'$.

By IH on (1) and (3), $\Delta;\Phi_a;z:\tau_1, f:\Box(\tau_1 \to \tau_2), \Box\Gamma \vdash t_1''[t_1/x] \backsim t_2''[t_2/x] :^c \tau_2$ (4).

By core rule **c-r-fixNC** and (4), we can derive :

$\Delta; \Phi_a; \square\Gamma, \Gamma' \vdash \text{fix } f(z).(t[t_1/x]) \backsim \text{fix } f(z).(t[t_2/x]) :^c \tau'$.

Case

$$\frac{\Delta; \Phi_a; \Gamma, x:\tau \vdash t_3 \backsim t_3' :^c \tau'' \quad (3) \qquad \Delta; \Phi_a; \Gamma, x:\tau \vdash t_4 \backsim t_4' :^c \mathbf{list}[n]^\alpha \tau'' \quad (4)}{\Delta; \Phi_a; \Gamma, x:\tau \vdash \mathbf{cons}_C(t_3, t_4) \backsim \mathbf{cons}_C(t_3', t_4') :^c \mathbf{list}[n+1]^{\alpha+1} \tau''}\ \textbf{c-r-cons1}$$

$t_1' = \text{cons}_C(t_3, t_4),\ t_2' = \text{cons}_C(t_3', t_4'), \tau' = \text{list}[n+1]^{\alpha+1} \tau''$.

TS: $\Delta; \Phi_a; \Gamma \vdash \text{cons}_C(t_3[t_1/x], t_4[t_1/x]) \backsim \text{cons}_C(t_3'[t_2/x], t_4'[t_2/x]) :^c \tau'$.

By IH on (1) and (3),$\Delta; \Phi_a; \Gamma \vdash t_3[t_1/x] \backsim t_3'[t_2/x] :^c \tau''$ (5).

By IH on (1) and (4),$\Delta; \Phi_a; \Gamma \vdash t_4[t_1/x] \backsim t_4'[t_2/x] :^c \tau''$ (6).

By core rule **c-r-fixNC** and (5),(6), we can derive :

$\Delta; \Phi_a; \Gamma \vdash \text{cons}_C(t_3[t_1/x], t_4[t_1/x]) \backsim \text{cons}_C(t_3'[t_2/x], t_4'[t_2/x]) :^c \tau'$.

Case

$$\frac{\Delta; \Phi_a; \square\Gamma, x:\tau \vdash t \backsim t :^c \tau'' \quad (3)}{\Delta; \Phi_a; \square\Gamma, \Gamma', x:\tau \vdash \mathbf{NC}\ t \backsim \mathbf{NC}\ t :^c \square\tau''}\ \textbf{c-nochange}$$

$t_1' = \text{NC } t,\ t_2' = \text{NC } t, \tau' = \square\tau''$.

TS: $\Delta; \Phi_a; \Gamma \vdash \text{NC } t[t_1/x] \backsim \text{NC } t[t_2/x]) :^c \tau'$.

By IH on (1) and (3),$\Delta; \Phi_a; \Gamma \vdash t[t_1/x] \backsim t[t_2/x] :^c \tau''$ (4).

By core rule **c-nochange** and (4), we can derive :

$\Delta; \Phi_a; \Gamma \vdash \text{NC } t[t_1/x] \backsim \text{NC } t[t_2/x]) :^c \tau'$

Case

$$\frac{\Delta; \Phi_a; |\Gamma|1 \vdash t_1'' :^c A_1 \quad (3) \qquad \Delta; \Phi_a; |\Gamma|2 \vdash t_2'' :^c A_2 \quad (4)}{\Delta; \Phi_a; \Gamma \vdash \mathbf{switch}\ t_1'' \backsim \mathbf{switch}\ t_2'' :^c U(A_1, A_2)}\ \textbf{c-switch}$$

$t_1' = \text{switch } t_1'',\ t_2' = \text{switch } t_2'', \tau' = U(A_1, A_2)$.

TS: $\Delta; \Phi_a; \Gamma \vdash t_1' \backsim t_2' :^c \tau'$.

By Lemma 41 on (1) and (3),$\Delta; \Phi_a; |\Gamma|1 \vdash t_1''[t_1/x] :^c A_1$ (5).

By Lemma 41 on (1) and (4),$\Delta; \Phi_a; |\Gamma|2 \vdash t_2''[t_2/x] :^c A_2$ (6).

By **c-switch** and (5),(6), we conclude that $\Delta; \Phi_a; \Gamma \vdash t_1' \backsim t_2' :^c \tau'$.

□

Lemma 43 (Reflexivity of Unary Algorithmic Subtyping)**.** $\Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A \Rightarrow \Phi$ *and* $\Delta;\Phi_a \models \Phi$.

Proof. By induction on the unary type. □

Lemma 44 (Reflexivity of Algorithmic Binary Type Equivalence)**.** $\forall \Delta, \psi_a, \Phi_a$, *there exists* Φ *s.t* $\Delta;\psi_a;\Phi_a \models \tau \equiv \tau \Rightarrow \Phi$ *and* $\Delta;\psi_a;\Phi_a \models \Phi$.

Proof. By induction on the binary type. Most cases are the same as **RelRef**'s corresponding proof.

Case

$U(A_1, A_2)$

By algorithmic equivalence rule **U**.

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1 \Rightarrow \Phi_1 \ (\star) \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1 \Rightarrow \Phi_1 \quad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2 \Rightarrow \Phi_2 \ (\diamond) \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a \models U(A_1, A_2) \equiv U(A_1, A_2) \Rightarrow \Phi_1 \wedge \Phi_1 \wedge \Phi_2 \wedge \Phi_2}\ \mathbf{U}$$

By Lemma 43 on ($\star$) and ($\diamond$), we conclude

$\Delta;\psi_a;\Phi_a \models U(A_1, A_2) \equiv U(A_1, A_2) \Rightarrow \Phi_1 \wedge \Phi_1 \wedge \Phi_2 \wedge \Phi_2$ and $\Delta;\psi_a;\Phi_a \models \Phi_1 \wedge \Phi_1 \wedge \Phi_2 \wedge \Phi_2$.

□

Lemma 45 (Transitivity of Unary Algorithmic Subtyping)**.** *If* $\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2 \Rightarrow \Phi_1$ *and* $\Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_3 \Rightarrow \Phi_2$ *and* $\Delta;\Phi_a \models \Phi_1 \wedge \Phi_2$, *then* $\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_3 \Rightarrow \Phi_3$ *for some* Φ_3 *such that* $\Delta;\Phi_a \models \Phi_3$.

Proof. By induction on the first two algorithmic subtyping derivation.

Case

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1 \ (1) \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2 \ (2)}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \to A_2 \sqsubseteq A_1' \to A_2' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-u-fun}$$

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1'' \sqsubseteq A_1' \Rightarrow \Phi_1' \ (3) \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2' \sqsubseteq A_2'' \Rightarrow \Phi_2 \ (4)}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1' \to A_2' \sqsubseteq A_1'' \to A_2'' \Rightarrow \Phi_1' \wedge \Phi_2'}\ \textbf{alg-u-fun}$$

By IH on (1) and (3), we get

$\models^{\mathsf{A}} A_1'' \sqsubseteq A_1 \Rightarrow \Phi_1 \wedge \Phi_1'$ and $\Delta;\Phi_a \models \Phi_1 \wedge \Phi_1'$.

By IH on (2) and (4), we get

$\models^{\mathsf{A}} A_2 \sqsubseteq A_2'' \Rightarrow \Phi_2 \wedge \Phi_2'$ and $\Delta;\Phi_a \models \Phi_2 \wedge \Phi_2'$.

By rule **alg-u-fun** and above statements, we conclude

$\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \to A_2 \sqsubseteq A_1'' \to A_2'' \Rightarrow \Phi_1' \wedge \Phi_2' \wedge \Phi_1 \wedge \Phi_2$.

□

Theorem 46 (Soundness of the Algorithmic Unary Subtyping)**.** *Assume that*

1. $\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \Rightarrow \Phi$
2. $FIV(\Phi_a, A, A') \subseteq \Delta, \psi_a$
3. $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta;\Phi_a[\theta_a] \models^{\mathsf{A}} A[\theta_a] \sqsubseteq A'[\theta_a]$.

Proof. By induction on the algorithmic unary subtyping derivation.

Case

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1 \ (1) \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2 \ (2)}{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \to A_2 \sqsubseteq A_1' \to A_2' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-u-fun}$$

By IH on (1) and assumption 2,3, we get $\Delta;\Phi_a[\theta_a] \models^{\mathsf{A}} A_1'[\theta_a] \sqsubseteq A_1[\theta_a]$.

By IH on (2) and assumption 2,3, we get $\Delta;\Phi_a[\theta_a] \models^{\mathsf{A}} A_2[\theta_a] \sqsubseteq A_2'[\theta_a]$.

By subtyping rule $\to_{\mathbf{exec}}$, then we conclude

$\Delta;\Phi_a[\theta_a] \models^{\mathsf{A}} A_1 \to A_2[\theta_a] \sqsubseteq A_1' \to A_2'[\theta_a]$.

□

Theorem 47 (Completeness of the Unary Algorithmic Subtyping)**.** *Assume that* $\Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'$. *Then* $\exists\Phi$. *such that* $\Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \Rightarrow \Phi$ *and* $\Delta;\Phi_a \models \Phi$.

Proof. By induction on the unary subtyping derivation.

Case

$$\frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \ (1) \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \ (2)}{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \to A_2 \sqsubseteq A_1' \to A_2'} \to \textbf{exec}$$

By IH on (1), there exists Φ_1 s.t $\Delta;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1$ and $\Delta;\Phi_a \models \Phi_1$.

By IH on (2), there exists Φ_2 s.t $\Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2$ and $\Delta;\Phi_a \models \Phi_2$.

By **alg-u-fun** and the above statements, we conclude there eixsts $\Phi = \Phi_1 \wedge \Phi_2$,

s.t $\Delta;\Phi_a \models^{\mathsf{A}} A_1 \to A_2 \sqsubseteq A_1' \to A_2' \Rightarrow \Phi$ and $\Delta;\Phi_a \models \Phi$.

□

Theorem 48 (Soundness of the Algorithmic Binary Type Equality)**.** *Assume that*

1. $\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi$
2. $FIV(\Phi_a,\tau,\tau') \subseteq \Delta,\psi_a$
3. $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta;\Phi_a[\theta_a] \models \tau[\theta_a] \equiv \tau'[\theta_a]$.

Proof. By induction on the algorithmic binary type equivalence derivation. Same as RelRef.

Case

$$\frac{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1' \quad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2 \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2' \sqsubseteq A_2 \Rightarrow \Phi_2'}{\Delta;\psi_a;\Phi_a \models U(A_1,A_2) \equiv U(A_1',A_2') \Rightarrow \Phi_1 \wedge \Phi_1' \wedge \Phi_2 \wedge \Phi_2'} \mathbf{U}$$

From assumption 3, $\Delta;\Phi_a[\theta_a] \models (\Phi_1 \wedge \Phi_1' \wedge \Phi_2 \wedge \Phi_2')[\theta_a]$

we know $\Delta; \Phi_a[\theta_a] \models \Phi_1[\theta_a]$ (a)

$\Delta; \Phi_a[\theta_a] \models \Phi_1'[\theta_a]$ (b)

$\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta_a]$ (c)

$\Delta; \Phi_a[\theta_a] \models \Phi_2'[\theta_a]$ (d)

By IH on the first premise and (a), the second and (b), the third and (c) and the fourth and (d) and the rule **eq-U**, we can conclude that $\Delta; \Phi_a[\theta_a] \models U(A_1, A_2)[\theta_a] \equiv U(A_1', A_2')[\theta_a]$.

□

Theorem 49 (Completeness of the Binary Algorithmic Type Equivalence)**.** *Assume that* $\Delta; \Phi_a \models \tau \equiv \tau'$. *Then* $\exists \Phi$. *such that* $\Delta; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the binary subtyping derivation. Same as RelRef.

Case

$$\frac{\Delta; \Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \Delta; \Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \quad \Delta; \Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \qquad \Delta; \Phi_a \models^{\mathsf{A}} A_2' \sqsubseteq A_2}{\Delta; \Phi_a \models U(A_1, A_2) \equiv U(A_1', A_2')}\ \textbf{eq-U}$$

By IH on the first premise, we know $\Delta; \Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \Rightarrow \Phi_1$ and $\Delta; \Phi_a \models \Phi_1$.

Similarly by IH on the other premises respectively, we get

$\Delta; \Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1'$ and $\Delta; \Phi_a \models \Phi_1'$

$\Delta; \Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2$ and $\Delta; \Phi_a \models \Phi_2$

$\Delta; \Phi_a \models^{\mathsf{A}} A_2' \sqsubseteq A_2 \Rightarrow \Phi_2'$ and $\Delta; \Phi_a \models \Phi_2'$

By the above statements and the relational algorithmic type equivalence rule **U**, we conclude where $\Phi = \Phi_1 \wedge \Phi_1' \wedge \Phi_2 \wedge \Phi_2'$:

$\Delta; \Phi_a \models U(A_1, A_2) \equiv U(A_1', A_2') \Rightarrow \Phi$ and $\Delta; \Phi_a \models \Phi$

□

Lemma 50 (Existence of coercions for relational subtyping)**.** *If* $\Delta; \Phi_a \models \tau \sqsubseteq \tau'$ *then there exists* $coerce_{\tau,\tau'} \in Core$ *s.t.* $\Delta; \Phi; \cdot \vdash coerce_{\tau,\tau'} \rightharpoonup coerce_{\tau,\tau'} :^c \tau \to \tau'$.

Proof. Proof is by induction on the subtyping derivation. We denote the witness e of type $\tau \to \tau'$ as $\text{coerce}_{\tau,\tau'}$ for clarity.

Case

$$\frac{}{\Delta;\Phi_a \models \Box U\,(\mathbf{int},\mathbf{int}) \sqsubseteq \mathbf{int}}\ \Box\mathbf{U\text{-}int}$$

Then, we can construct the derivation using the primitive function

$\text{box}_U : \Box U\,(\text{int},\text{int}) \to \text{int}$

$$\frac{}{\Delta;\Phi;\cdot \vdash \text{fix}\ f(x).\text{box}_U\, x \rightharpoonup \text{fix}\ f(x).\text{box}_U\, x :^c \Box U\,(\text{int},\text{int}) \to \text{int}}$$

Case

$$\frac{}{\Delta;\Phi_a \models U\,(\forall i{::}S.\,A, \forall i{::}S.\,A') \sqsubseteq \forall i{::}S.\,U\,(A,A')}\ \forall\mathbf{U}$$

Then, we can immediately construct the following derivation

where $e = \text{fix}\ f(x).\Lambda i.\text{switch}\,(x\,[i])$ using the **c-switch** and **c-iApp** rules.

$\Delta;\Phi;\cdot \vdash t \rightharpoonup t :^c (U\,(\forall i{::}S.\,A, \forall i{::}S.\,A')) \to \forall i{::}S.\,U\,(A,A')$

□

Theorem 51 (Types are preserved by embedding)**.**

1. *If* $\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : A$*, then* $\Delta;\Phi_a;\Omega \vdash t^* :^c A$ *and* $\Delta;\Phi_a;\Omega \vdash t : A$.

2. *If* $\Delta;\Phi_a;\Gamma \vdash t_1 \rightharpoonup t_2 \rightsquigarrow t_1^* \rightharpoonup t_2^* : \tau$*, then* $\Delta;\Phi_a;\Gamma \vdash t_1^* \rightharpoonup t_2^* :^c \tau$ *and* $\Delta;\Phi_a;\Gamma \vdash t_1 \rightharpoonup t_2 : \tau$.

Proof. By simultaneous induction on the given derivations.

Proof of Theorem 51.1:

Case

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A_1 \to A_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:A_1,f:A_1\to A_2,\Omega \vdash t \rightsquigarrow t^* : A_2}{\Delta;\Phi_a;\Omega \vdash \mathbf{fix}\ f(x).t \rightsquigarrow \mathbf{fix}\ f(x).t^* : A_1 \to A_2}\ \textbf{e-u-fix}$$

By Theorem 37.1 on the premise, we get $x:A_1, f:A_1\to A_2, \Omega \vdash t :^c A_2$. Then, by unary core rule **c-fix**,we conclude:

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A_1 \to A_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:A_1,f:A_1\to A_2,\Omega \vdash t :^c A_2}{\Delta;\Phi_a;\Omega \vdash \mathrm{fix}\ f(x).t :^c A_1 \to A_2}\ \textbf{c-fix}.$$

Proof of Theorem 51.2:

Case

$$\frac{\Delta;\Phi_a;|\Gamma|1 \vdash t_1 \rightsquigarrow t_1^* : A_1 \quad (\star) \qquad \Delta;\Phi_a;|\Gamma|2 \vdash t_2 \rightsquigarrow t_2^* : A_2 \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \mathbf{switch}\ t_1^* \backsim \mathbf{switch}\ t_2^* : U(A_1,A_2)}\ \textbf{e-switch}$$

By Theorem 51.1 on $(\star)$, we get $\Delta;\Phi;\Omega \vdash t_1^* :^c A_1 \quad (\star\star)$.

By Theorem 51.1 on $(\diamond)$, we get $\Delta;\Phi;\Omega \vdash t_2^* :^c A_2 \quad (\diamond\diamond)$.

Then, we conclude as follows: $\dfrac{\Delta;\Phi_a;|\Gamma|1 \vdash t_1 :^c A_1 \qquad \Delta;\Phi_a;|\Gamma|2 \vdash t_2 :^c A_2}{\Delta;\Phi_a;\Gamma \vdash \mathrm{switch}\ t_1 \backsim \mathrm{switch}\ t_2 :^c U(A_1,A_2)}\ \textbf{c-switch}$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \quad (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad (\diamond) \qquad t' = \mathbf{coerce}_{\tau,\tau'}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t'\ t_1^* \backsim t'\ t_2^* : \tau'}\ \textbf{e-r-}\sqsubseteq$$

By Theorem 51.2 on $(\star)$, $\Delta;\Phi;\Gamma \vdash t_1^* \backsim t_2^* :^c \tau \quad (\star\star)$.

By Theorem 50 using $(\diamond)$, we know that $\Delta;\Phi;\cdot \vdash t' \backsim t' :^c \tau \to \tau' \quad (\diamond\diamond)$.

By applying **c-r-app** rule on $(\star\star)$ and $(\diamond\diamond)$, we get $\Delta;\Phi;\Gamma \vdash t'\ t_1^* \backsim t'\ t_2^* :^c \tau' \quad (\spadesuit)$.

By reflexivity of binary type equivalence, we know $\Delta;\Phi_a \models \tau' \equiv \tau' \quad (\spadesuit\spadesuit)$.

Then, we conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t'\ t_1^* \backsim t'\ t_2^* :^c \tau' \quad (\spadesuit) \qquad \Delta;\Phi_a \models \tau' \equiv \tau' \quad (\spadesuit\spadesuit)}{\Delta;\Phi_a;\Gamma \vdash t'\ t_1^* \backsim t'\ t_2^* :^c \tau'}\ \textbf{c-r-}\sqsubseteq$$

□

Theorem 52 (Completeness of embedding)**.**

1. *If* $\Delta;\Phi_a;\Omega \vdash t : A$*, then there exists an* t^* *such that* $\Delta;\Phi;\Omega \vdash t \rightsquigarrow t^* : A$.

2. *If* $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau$, *then there exist* t_1^*, t_2^* *such that* $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t_1^* \backsim t_2^* : \tau$.

Proof. By simultaneous induction on the given derivations.

Proof of Theorem 52.1:

Case

$$\frac{\Delta;\Phi_a;\Omega \vdash^{t}_{k} t : A \quad (\star) \qquad \Delta;\Phi_a;\models^{\mathsf{A}} A \sqsubseteq A' \quad (\diamond)}{\Delta;\Phi_a;\Omega \vdash t : A'} \sqsubseteq$$

By Theorem 52.1 on $(\star)$, we get $\exists\, t^*$ such that $\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : A \quad (\star\star)$.

By **e-u-**$\sqsubseteq$ rule using $(\star\star)$, $(\diamond)$, we conclude as follows

$$\frac{\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : A \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'}{\Delta;\Phi_a;\Omega \vdash t \rightsquigarrow t^* : A'}\ \textbf{e-u-}\sqsubseteq$$

Proof of Theorem 52.2:

Case

$$\frac{\Delta;\Phi;|\Gamma|_1 \vdash t_1 : A_1 \quad (\star) \qquad \Delta;\Phi;|\Gamma|_2 \vdash t_2 : A_2 \quad (\diamond)}{\Delta;\Phi;\Gamma \vdash t_1 \backsim t_2 : U\,(A_1, A_2)}\ \textbf{r-switch}$$

By Theorem 52.1 on $(\star)$, we get $\exists\, t_1^*$ such that $\Delta;\Phi;\Omega \vdash^{t_1}_{k_1} t_1 \rightsquigarrow {t_1}^* : A_1 \quad (\star\star)$.

By Theorem 52.1 on $(\diamond)$, we get $\exists\, t_2^*$ such that $\Delta;\Phi;\Omega \vdash^{t_2}_{k_2} t_2 \rightsquigarrow {t_2}^* : A_2 \quad (\diamond\diamond)$.

By **e-switch** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude as follows:

$$\frac{\Delta;\Phi_a;|\Gamma|1 \vdash t_1 \rightsquigarrow t_1^* : A_1 \quad (\star\star) \qquad \Delta;\Phi_a;|\Gamma|2 \vdash t_2 \rightsquigarrow t_2^* : A_2 \quad (\diamond\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow \text{switch } t_1^* \backsim \text{switch } t_2^* : U\,(A_1, A_2)}\ \textbf{e-switch}.$$

Case

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t : \tau \\ \forall x \in dom(\Gamma).\ (\star)\Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\ (\diamond)\end{array}}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \backsim t : \Box\tau}\ \textbf{rr-nochange}$$

By IH2 on $(\star)$, we get $\exists\, t^*$ such that $\Delta;\Phi_a;\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau \quad (\dagger)$.

By Lemma 30 on $(\diamond)$, we get $\exists t_i = \text{coerce}_{\Gamma(x_i),\Box(\Gamma(x_i)}$ for all $x_i \in dom(\Gamma)$ $(\dagger\dagger)$.

By **e-nochange** embedding rule using $(\dagger)$ and $(\dagger\dagger)$, we can conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau \quad (\dagger) \qquad \forall x_i \in dom(\Gamma),\ t_i = \text{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \quad (\dagger\dagger) \qquad \forall x_i \in dom(\Gamma),\ \Delta;\Phi_a \models \Gamma(x_i) \sqsubseteq \Box\Gamma(x_i)}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \backsim t \rightsquigarrow \text{let } \overline{y_i = t_i\ x_i} \text{ in NC } t^*\,\overline{[\text{der } y_i / x_i]} \backsim \text{let } \overline{y_i = t_i\ x_i} \text{ in NC } t^*\,\overline{[\text{der } y_i / x_i]} : \Box\tau}\ \textbf{e-nochange}$$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \mathbf{list}[n]^\alpha\,\tau \quad (\star) \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' : \tau' \quad (\diamond)}{\begin{array}{c} i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau,tl:\mathbf{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau' \quad (\dagger) \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h:\tau,tl:\mathbf{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' : \tau' \quad (\spadesuit) \end{array}}\ \textbf{rr-caseL}$$

$\Delta;\Phi_a;\Gamma \vdash$ **case** t **of nil** $\rightarrow t_1 \mid h :: tl \rightarrow t_2 \backsim$ **case** t' **of nil** $\rightarrow t_1' \mid h :: tl \rightarrow t_2' : \tau'$

By IH2 on $(\star)$, we get $\exists\, t^*$ and $\exists\, t'^*$ s.t. $\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \text{list}[n]^\alpha\,\tau \quad (\star\star)$.

By IH2 on $(\diamond)$, we get $\exists\, t_1^*$ and $\exists\, t_1'^*$ s.t. $\Delta; n \doteq 0 \wedge \Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow t_1^* \backsim t_1'^* : \tau' \quad (\diamond\diamond)$.

By IH2 on $(\dagger)$, we get $\exists\, t_2^*$ and $\exists\, t_2'^*$ s.t.

$i :: S,\Delta; n \doteq i+1 \wedge \Phi_a;h:\Box\tau,tl:\text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^* \backsim t_2'^* : \tau' \quad (\dagger\dagger)$.

By IH2 on $(\spadesuit)$, we get $\exists\, t_2^{**}$ and $\exists\, t_2'^{**}$ s.t.

$i :: S,\beta :: S,\Delta; n \doteq i+1 \wedge \alpha \doteq \beta+1 \wedge \Phi_a;h:\tau,tl:\text{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^{**} \backsim t_2'^{**} : \tau' \quad (\spadesuit\spadesuit)$.

By **e-caseL** embedding rule using $(\star\star)$, $(\diamond\diamond)$, and $(\spadesuit\spadesuit)$, we can conclude as follows

$$\frac{\begin{array}{c} \Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \text{list}[n]^\alpha\,\tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau,tl:\text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h:\tau,tl:\text{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^{**} \backsim t_2'^{**} : \tau' \end{array}}{\begin{array}{l} \Delta;\Phi_a;\Gamma \vdash \begin{array}{l} \text{case } t \text{ of nil } \rightarrow t_1 \\ \mid h :: tl \rightarrow t_2 \end{array} \backsim \begin{array}{l} \text{case } t \text{ of nil } \rightarrow t_1' \\ \mid h :: tl \rightarrow t_2' \end{array} \rightsquigarrow : \tau' \\ \qquad \begin{array}{l} \text{case } t^* \text{ of nil } \rightarrow t_1^* \\ \mid h ::_{NC} tl \rightarrow t_2^* \\ \mid h ::_C tl \ \rightarrow t_2^{**} \end{array} \backsim \begin{array}{l} \text{case } t'^* \text{ of nil } \rightarrow \\ \quad t_1'^* \\ \mid h ::_{NC} tl \rightarrow t_2'^* \\ \mid h ::_C tl \ \rightarrow t_2'^{**} \end{array} \end{array}}\ \textbf{e-r-caseL}$$

Case

$$\frac{\Delta;\Phi_a \vdash \tau_1 \rightarrow \tau_2\ \text{wf} \qquad \Delta;\Phi_a;x:\tau_1,f:\Box(\tau_1 \rightarrow \tau_2),\Gamma \vdash t \backsim t : \tau_2 \quad (\star) \qquad \forall x \in dom(\Gamma).\Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x) \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash \textbf{fix}\ f(x).t \backsim \textbf{fix}\ f(x).t : \Box(\tau_1 \rightarrow \tau_2)}\ \textbf{rr-fixNC}$$

By IH2 on $(\star)$, we get:

$\exists\, t^*$ such that $\Delta;\Phi_a;x:\tau_1,f:\Box(\tau_1 \rightarrow \tau_2),\Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau_2 \quad (\star\star)$.

By Theorem 50 on ($\diamond$), we get $\exists t_i = \text{coerce}_{\Gamma(x_i),\Box(\Gamma(x_i)}$ for all $x_i \in dom(\Gamma)$ ($\diamond\diamond$).

By **e-fixNC** embedding rule using ($\star\star$) and ($\diamond\diamond$), we can conclude as follows:

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \to \tau_2 \text{ wf} \qquad \Delta;\Phi_a; x:\tau_1, f:\Box(\tau_1 \to \tau_2), \Gamma \vdash t \backsim t \rightsquigarrow t^* \backsim t^* : \tau_2 \\ \forall x_i \in dom(\Gamma),\ \ t_i = \text{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \qquad \forall x_i \in dom(\Gamma),\ \ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x) \\ t^{**} = \text{let } \overline{y_i = t_i\ x_i} \text{ in } \text{fix}_{NC}\ f(x).t^*[\overline{\text{der } y_i / x_i}]\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t \backsim \text{fix } f(x).t \rightsquigarrow t^{**} \backsim t^{**} : \Box(\tau_1 \to \tau_2)}$$

e-r-fixNC.

Case

$$\frac{\begin{array}{c}i :: S, \Delta;\Phi_a;\Gamma \vdash t \backsim t' : \tau \quad (\star) \\ i \notin \textbf{FIV}(\Phi_a;\Gamma)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \Lambda t \backsim \Lambda t' : \forall i{::}S.\tau}\ \textbf{rr-iLam}$$

By IH2 on ($\star$), we get $\exists\, t^*$ and $\exists\, t'^*$ such that $i :: S, \Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau$ ($\star\star$).

By **e-iLam** embedding rule using ($\star\star$) , we can conclude as follows:

$$\frac{i :: S, \Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau \qquad i \notin \text{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda.t \backsim \Lambda.t' \rightsquigarrow \Lambda i.t^* \backsim \Lambda i.t'^* : \forall i{::}S.\tau}\ \textbf{e-r-iLam}.$$

Case

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \forall i{::}S.\tau \quad (\star) \\ \Delta \vdash I : S\ (\diamond)\end{array}}{\Delta;\Phi_a;\Gamma \vdash t[] \backsim t'[] : \tau\{I/i\}}\ \textbf{rr-iApp}$$

By IH2 on ($\star$), we get $\exists\, t^*$ such that $\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \forall i{::}S.\tau$ ($\star\star$).

By **e-iApp** embedding rule using ($\star\star$) and ($\diamond$), we can conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \forall i{::}S.\tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash t[] \backsim t'[] \rightsquigarrow t^*[I] \backsim t'^*[I] : \tau\{I/i\}}\ \textbf{e-r-iApp}.$$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' : \tau\{I/i\} \quad (\star) \qquad \Delta \vdash I :: S\ (\diamond)}{\Delta;\Phi_a;\Gamma \vdash \textbf{pack } t \backsim \textbf{pack } t' : \exists i{::}S.\tau}\ \textbf{rr-pack}$$

By IH2 on ($\star$), we get $\exists\, t^*$ and $\exists\, t'^*$ such that $\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau\{I/i\}$ ($\star\star$).

By **e-pack** embedding rule using ($\star\star$) and ($\diamond$) , we can conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' \rightsquigarrow t^* \backsim t'^* : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \text{pack } t \backsim \text{pack } t' \rightsquigarrow \text{pack } t^* \text{ with } I \backsim \text{pack } t'^* \text{ with } I : \exists i{::}S.\tau}\ \textbf{e-r-pack}.$$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' : \exists i{::}S.\tau_1 \quad (\star) \qquad i :: S,\Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' : \tau_2 \quad (\diamond) \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)}{\Delta;\Phi_a;\Gamma \vdash \textbf{unpack } t_1 \textbf{ as } x \textbf{ in } t_2 \backsim \textbf{unpack } t_1' \textbf{ as } x \textbf{ in } t_2' : \tau_2}\ \textbf{rr-unpack1}$$

By IH2 on $(\star)$, we get $\exists\, t_1^*$ and $\exists\, t_1'^*$ such that $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow t_1^* \backsim t_1'^* : \exists i{::}S.\tau_1 \quad (\star\star)$.

By IH2 on $(\diamond)$, we get $\exists\, t_2^*$ and $\exists\, t_2'^*$

such that $i :: S,\Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow t_2^* \backsim t_2'^* : \tau_2 \quad (\diamond\diamond)$.

By **e-unpack** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' \rightsquigarrow {t_1}^* \backsim {t_1'}^* : \exists i{::}S.\tau_1 \qquad i :: S,\Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_2 \backsim t_2' \rightsquigarrow {t_2}^* \backsim {t_2'}^* : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2) \qquad t_1^{**} = \text{unpack } t_1^* \text{ as } (x,i) \text{ in } t_2^* \qquad t_2^{**} = \text{unpack } t_1'^* \text{ as } (x,i) \text{ in } t_2'^*}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } x \text{ in } t_2 \backsim \text{unpack } t_1' \text{ as } x \text{ in } t_2' \rightsquigarrow t_1^{**} \backsim t_2^{**} : \tau_2}\ \textbf{e-r-unpack}.$$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau \quad (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 : \tau'}\ \textbf{rr-}\sqsubseteq$$

By IH2 on $(\star)$, we get $\exists\, t_1^*, t_2^*$ such that $\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \quad (\star\star)$.

By Lemma 50 on $(\diamond)$, we can show that $\exists t' = \text{coerce}_{\tau,\tau'} \quad (\diamond\diamond)$.

By **e-r-**$\sqsubseteq$ rule using $(\star\star)$, $(\diamond\diamond)$ and $(\diamond)$, we conclude as follows

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow {t_1}^* \backsim {t_2}^* : \tau \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \text{coerce}_{\tau,\tau'}}{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 \rightsquigarrow t'\ t_1^* \backsim t'\ t_2^* : \tau'}\ \textbf{e-r-}\sqsubseteq$$

□

Theorem 53 (Soundness of algorithmic typechecking)**.**

1. *Assume that* $\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A \rightarrow \Phi$ *and* $FIV(\Phi_a,\Omega,A) \subseteq dom(\Delta,\psi_a)$ *and* θ_a *is a valid substitution for* ψ_a *s.t.* $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds.*

   Then, $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a]$.

2. *Assume that* $\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$ *and* θ_a *is the valid substitutions for* ψ_a *such that* $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds.*

   Then, $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a]$.

3. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Gamma,\tau) \subseteq dom(\Delta,\psi_a)$ *and* θ_a *is a valid substitution for* ψ_a *such that* $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds.*

Then, $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \backsim |t'| :^c \tau[\theta_a]$.

4. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$ *and* θ_a *is the valid substitution for* ψ_a *such that* $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *holds.*

   Then, $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \backsim |t'| :^c \tau[\theta_a]$.

Proof. By simultaneous induction on the given algorithmic typing derivations.

Proof of Theorem 53.1:

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A' \sqsubseteq A \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-}\uparrow\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a]$.

By the main assumptions, we have $FIV(\Phi_a,\Omega,A) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and $\Delta;\Phi_a[\theta_a] \models \Phi_1 \wedge \Phi_2[\theta_a]$ $(\star\star)$

Using $(\star)$ and $(\diamond)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \rhd \theta_a : \psi_a$

b) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta_a]$

c) $\Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a]$

By IH2 on the first premise using $(\star)$, a) and b), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A'[\theta_a] \tag{B.12}$$

By Theorem 46 using the second premise and c), we obtain

$$\Delta;\Phi_a[\theta_a] \models^{\mathsf{A}} A'[\theta_a] \sqsubseteq A[\theta_a] \tag{B.13}$$

Note that due to $(\star)$, we can conclude by the **c-⊑** exec rule using equation B.38,B.39 that $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a]$.

Case

$$\frac{\Delta;\psi_a;\Phi_a;f : A_1 \to A_2, x : A_1, \Omega \vdash t \downarrow A_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \textbf{fix}\ f(x).t \downarrow A_1 \to A_2 \Rightarrow \Phi}\ \textbf{alg-u-fix-}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash \text{fix } f(x).|t| :^c A_1[\theta_a] \to A_2[\theta_a]$.
By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega, A_1 \to A_2) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$
Using $(\star)$, we can show that

a) $\text{FIV}(\Phi_a, \Omega, A_1, A_1 \to A_2) \subseteq dom(\Delta, \psi_a)$.

We also can show that $(\star\star)$'s derivation must be in a form such that we have

b) $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$

By IH1 on the first premise using a) and b), we can show that

$$\Delta;\Phi_a[\theta_a];x : A_1[\theta_a], f : A_1[\theta_a] \to A_2[\theta_a],\Omega[\theta_a] \vdash |t| :^c A_2[\theta_a] \tag{B.14}$$

By the **c-fix** rule using equation B.40, we obtain
$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash \text{fix } f(x).|t| :^c A_1[\theta_a] \to A_2[\theta_a]$.

Case

$$\frac{i :: S, \Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \Lambda i.t \downarrow \forall i{::}S.\, A \Rightarrow \forall\, i :: S.\Phi}\ \textbf{alg-u-iLam-}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash \Lambda i.|t| :^c \forall i{::}S.\, A[\theta_a]$.
By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega, \forall i{::}S.\, A) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta;\Phi_a[\theta_a] \models \forall\, i :: S.\Phi[\theta_a]$ $(\star\star)$
Using $(\star)$, we can show that

a) $\text{FIV}(\Phi_a, \Omega, A) \subseteq i, dom(\Delta, \psi_a)$.

We can also show that $(\star\star)$'s derivation must be in a form such that we have

b) $i :: S, \Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$

By IH1 on the premise using a) and b), we can show that

$$i :: S, \Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a] \tag{B.15}$$

By the **c-iLam** rule using eq. (B.41), we obtain $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash \Lambda i.|t| :^c \forall i{::}S.\, A[\theta_a]$.

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A\{I/i\} \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Omega \vdash \textbf{pack } t \textbf{ with } I \downarrow \exists i{::}S.\, A \Rightarrow \Phi}\ \textbf{alg-u-pack-}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash \text{pack } |t| \text{ with } I :^c \exists i{::}S.\, A[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega,\exists i{::}S.\, A) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$

Using $(\star)$ and the second premise, we can show that

a) $\text{FIV}(\Phi_a,\Omega,A\{I/i\}) \subseteq dom(\Delta,\psi_a)$.

By IH1 on the premise using a) and $(\star\star)$, we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a]\{I/i\} \tag{B.16}$$

By the **c-pack** rule using eq. (B.42) and the second premise, we obtain

$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash \text{pack } |t| \text{ with } I :^c \exists i{::}S.\, A[\theta_a]$.

Case

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow \exists i{::}S.\, A_1 \Rightarrow \Phi_1 \qquad i \notin FV(\Phi_a;\Omega,A_2) \\ i :: S,\Delta;\psi_a;\Phi_a;x : A_1,\Omega \vdash t_2 \downarrow A_2 \Rightarrow \Phi_2 \qquad \Phi = \Phi_1 \wedge \forall i :: S.\Phi_2\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \textbf{unpack } t_1 \textbf{ as } (x,i) \textbf{ in } t_2 \downarrow A_2 \Rightarrow \Phi}\ \textbf{alg-u-unpack-}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash \text{unpack } |t_1| \text{ with } (x,i) \text{ in } |t_2| :^c A_2[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega,A_2) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models \Phi_1 \wedge \forall i :: S.\Phi_2[\theta_a]$ $(\star\star)$

Using $(\star)$, $(\diamond)$ and the 3rd premise, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta_a]$

b) $i :: S,\Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a]$

By IH2 on the first premise using $(\star)$, a) , we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t_1| :^c \exists i{::}S.\, A_1[\theta_a] \tag{B.17}$$

From $(\star)$, we can show that

c) $\text{FIV}(\Phi_a,A_1,\Omega,A_2) \subseteq i, dom(\Delta,\psi,\psi_a)$

By IH1 on the second premise using b), c), ($\star$) , we obtain

$$i :: S, \Delta; \Phi_a[\theta_a]; x : A_1[\theta\,\theta_a], \Omega[\theta_a] \vdash |t_2| :^c A_2[\theta_a] \tag{B.18}$$

Then by the **c-unpack** rule using eqs. (B.43) and (B.44),
we can show that $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash \text{unpack } |t_1| \text{ with } (x, i) \text{ in } |t_2| :^c A_2[\theta_a]$.

Case

$$\frac{\Delta; \psi_a; C \wedge \Phi_a; \Omega \vdash t_1 \downarrow A \Rightarrow \Phi_1 \qquad \Delta; \psi_a; \neg C \wedge \Phi_a; \Omega \vdash t_2 \downarrow A \Rightarrow \Phi_2 \qquad \Delta \vdash C \text{ wf}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \textbf{split}\ (t_1, t_2)\ \textbf{with}\ C \downarrow A \Rightarrow C \rightarrow \Phi_1 \wedge \neg C \rightarrow \Phi_2}\ \textbf{alg-u-split}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash \text{split } (|t_1|, |t_2|) \text{ with } C :^c A[\theta_a]$.
By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega, A) \subseteq dom(\Delta, \psi_a)$ ($\star$) and
$\Delta; \Phi_a[\theta_a] \models (C \rightarrow \Phi_1 \wedge \neg C \rightarrow \Phi_2)[\theta_a]$ ($\star\star$)
Using ($\star$) and the third premise, we can show that

a) $\text{FIV}(C \wedge \Phi_a, \Omega, A) \subseteq dom(\Delta, \psi_a)$.

b) $\text{FIV}(\neg C \wedge \Phi_a, \Omega, A) \subseteq dom(\Delta, \psi_a)$.

Using ($\star\star$) and the third premise, we can show that

c) $\Delta; C \wedge \Phi_a[\theta_a] \models \Phi_1[\theta_a]$

d) $\Delta; \neg C \wedge \Phi_a[\theta_a] \models \Phi_2[\theta_a]$

By IH1 on the first premise using ($\star$) and c), we can show that

$$\Delta; C \wedge \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |t_1| :^c A[\theta_a]\{I[\theta_a]/i\} \tag{B.19}$$

By IH1 on the second premise using ($\star$) and d), we can show that

$$\Delta; \neg C \wedge \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |t_2| :^c A[\theta_a]\{I[\theta_a]/i\} \tag{B.20}$$

By the **c-split** rule using eqs. (B.45) and (B.46) and the third premise, we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash \text{split } (|t_1|, |t_2|) \text{ with } C :^c A[\theta_a]$.
Case

$$\frac{\Delta;\Phi \wedge C;\Omega \vdash t \downarrow A \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow C \supset A \Rightarrow C \to \Phi}\ \textbf{alg-u-c-impI}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c C[\theta_a] \supset A[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega,C \supset A,k,t) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models (C \to \Phi)[\theta_a]$ $(\star\star)$

Using $(\star)$, we can show that

a) $\text{FIV}(C \wedge \Phi_a,\Omega,A) \subseteq dom(\Delta,\psi_a)$.

By IH1 on the premise using $(\star)$ and a), we can show that

$$\Delta;C[\theta_a] \wedge \Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a] \tag{B.21}$$

By the **c-cimpI** rule using eq. (B.47), we obtain $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c C[\theta_a] \supset A[\theta_a]$.

Proof of Theorem 53.2:

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash^{\mathsf{A}} A\ \mathsf{wf} \qquad \textbf{FIV}(A,k,t) \in \Delta}{\Delta;\psi_a;\Phi_a;\Omega \vdash (t : A,k,t) \uparrow A \Rightarrow \Phi}\ \textbf{alg-u-anno-}\uparrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |(t : A,k,t)| :^c A[\theta_a]$.

Since by definition, $\forall t.\ |(t : \_)| = |t|$, STS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$

Using the third premise, we can show that

a) $\text{FIV}(\Phi_a,\Omega,A) \subseteq dom(\Delta,\psi_a)$.

By IH1 on the first premise using $(\star\star)$ and a), we can conclude that

$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| :^c A[\theta_a]$.

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow A_1 \to A_2 \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a;\Omega \vdash t_2 \downarrow A_1 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1\ t_2 \uparrow A_2 \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-u-app-}\uparrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t_1|\,|t_2| :^c A_2[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models (\Phi_1 \wedge \Phi_2)[\theta_a]$ $(\star\star)$ such that $\Delta \rhd \theta_a : \psi_a$ are derivable.

From $(\star\star)$, we know

a) $\Delta; \Phi_a[\theta_a] \models \Phi_1[\theta_a]$

b) $\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta_a]$

By IH on the first premise using $(\star)$ and (a), we obtain

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |t_1| :^c A_1[\theta_a] \to A_2[\theta_a] \tag{B.22}$$

By IH2 on the third premise using $(\star)$ and (b), we obtain

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |t_2| :^c A_1[\theta\,\theta_a] \tag{B.23}$$

Then, by using **c-app** rule using eqs. (B.48) and (B.49), we can show that $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |t_1|\,|t_2| :^c A_2[\theta_a]$.

Case

$$\frac{\Delta; \psi_a; \Phi_a; \Omega \vdash t \uparrow \forall i{::}S.\, A' \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Omega \vdash t\,[I] \uparrow A'\{I/i\} \Rightarrow \Phi}\ \textbf{alg-u-iApp-}\uparrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |t|\,[I] :^c (A'\{I/i\})[\theta\,\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and $\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta_a]$ $(\star\star)$ such that $\Delta \rhd \theta_a : \psi_a$ are derivable.

By IH2 on the first premise using $(\star)$, we obtain

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |t| :^c \forall i{::}S.\, A'[\theta_a] \tag{B.24}$$

Then, by **c-iApp** rule using eq. (B.50) and the second premise, we can conclude that $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash |t|\,[I] :^c A'[\theta_a]\{I/i\}$.

Proof of Theorem 53.3:

Case

$$\frac{\Delta; \psi_a; \Phi_a; \Box\Gamma \vdash t \ominus t \downarrow \tau \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma', \Box\Gamma \vdash \textbf{NC}\ t \ominus \textbf{NC}\ t \downarrow \Box\tau \Rightarrow \Phi}\ \textbf{alg-r-nochange-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma'[\theta_a], \Box\Gamma[\theta_a] \vdash \text{NC}\,|e| \ominus \text{NC}\,|e| : \Box\tau[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma', \Box\Gamma, \tau) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$
Using $(\star)$, we can show that

a) $\text{FIV}(\Phi_a, \Box\Gamma, \tau) \subseteq dom(\Delta, \psi_a)$.

By IH3 on the premise using a) and $(\star\star)$, we can show that

$$\Delta; \Phi_a[\theta_a]; \Box\Gamma[\theta_a] \vdash |t| \ominus |t| : \tau[\theta_a] \tag{B.25}$$

By the **c-nochange** rule using eq. (B.25),
we obtain $\Delta; \Phi_a[\theta_a]; \Gamma'[\theta_a], \Box\Gamma[\theta_a] \vdash \text{NC}\,|t| \ominus \text{NC}\,|t| : \Box\tau[\theta_a]$.

Case

$$\frac{\begin{array}{c}\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \uparrow \mathbf{list}[n]^{\alpha}\,\tau \Rightarrow \Phi_e \qquad \Delta; \psi_a; n \doteq 0 \wedge \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \tau' \Rightarrow \Phi_1 \\ i :: \mathbb{N}, \Delta; \psi_a; n \doteq i+1 \wedge \Phi_a; h : \Box\tau, tl : \mathbf{list}[i]^{\alpha}\,\tau, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau' \Rightarrow \Phi_2 \\ i :: \mathbb{N}, \beta :: \mathbb{N}, \Delta; \psi_a; n \doteq i+1 \wedge \alpha \doteq \beta + 1 \wedge \Phi_a; h : \tau, tl : \mathbf{list}[i]^{\beta}\,\tau, \Gamma \vdash t_3 \ominus t_3' \downarrow \tau' \Rightarrow \Phi_3 \\ \Phi_{body} = (n \doteq 0 \to \Phi_1) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to (\Phi_2 \wedge \forall \beta :: \mathbb{N}.(\alpha \doteq \beta + 1) \to \Phi_3))\end{array}}{\begin{array}{l}\Delta; \psi_a; \Phi_a; \Gamma \vdash \begin{array}{l}\mathbf{case}\ t\ \mathbf{of\ nil}\ \to t_1 \\ \mid h ::_{NC} tl \to t_2 \\ \mid h ::_{C} tl \to t_3\end{array} \ominus \begin{array}{l}\mathbf{case}\ t'\ \mathbf{of\ nil}\ \to t_1' \\ \mid h ::_{NC} tl \to t_2' \\ \mid h ::_{C} tl \to t_3'\end{array} \downarrow \tau' \Rightarrow (\Phi_e \wedge \Phi_{body})\end{array}}\ \textbf{alg-r-caseL-}\downarrow$$

TS:

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \begin{array}{l}\text{case } t \text{ of nil } \to |t_1| \\ \mid h ::_N tl \to |t_2| \\ \mid h ::_C tl \to |t_3|\end{array} \ominus \begin{array}{l}\text{case } t' \text{ of nil } \to |t_1'| \\ \mid h ::_N tl \to |t_2'| \\ \mid h ::_C tl \to |t_3'|\end{array} : \tau'[\theta_a]$$

By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma, \tau') \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models (\Phi_e \wedge \Phi_{body})[\theta_a]$ $(\star\star)$
Using $(\star)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta; \Phi_a[\theta_a] \models \Phi_e[\theta_a]$

b) $\Delta; n[\theta_a] \doteq 0 \wedge \Phi_a[\theta_a] \models \Phi_1[\theta_a]$

c) $i :: S, \Delta; n[\theta_a] \doteq i+1 \wedge \Phi_a[\theta_a] \models \Phi_2[\theta_a]$

d) $i :: S, \beta :: S, \Delta; n[\theta_a] \doteq i+1 \wedge \alpha[\theta_a] \doteq \beta+1 \wedge \Phi_a[\theta_a] \models \Phi_3[\theta_a]$

By IH4 on the first premise using a) and ($\star$), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| : \text{list}[n[\theta_a]]^{\alpha[\theta_a]}\, \tau[\theta_a] \tag{B.26}$$

By IH3 on the second premise using b) and ($\star$), we can show that

$$\Delta; n[\theta_a] \doteq 0 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_1| \ominus |t_1'| : \tau'[\theta_a] \tag{B.27}$$

By IH3 on the third premise using c) and ($\star$), we can show that

$$i :: S, \Delta; n[\theta_a] \doteq i+1 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_2| \ominus |t_2'| : \tau'[\theta_a] \tag{B.28}$$

By IH3 on the fourth premise using d) and ($\star$), we can show that

$$i :: S, \beta :: S, \Delta; n[\theta_a] \doteq i+1 \wedge \alpha[\theta_a] \doteq \beta+1 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_3| \ominus |t_3'| : \tau'[\theta_a] \tag{B.29}$$

Then by **c-r-caseL** rule using eqs. (B.26) to (B.29), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \begin{array}{l} \text{case } t \text{ of nil } \rightarrow |t_1| \\ \mid h ::_N tl \rightarrow |t_2| \\ \mid h ::_C tl \rightarrow |t_3| \end{array} \ominus \begin{array}{l} \text{case } t' \text{ of nil } \rightarrow |t_1'| \\ \mid h ::_N tl \rightarrow |t_2'| \\ \mid h ::_C tl \rightarrow |t_3'| \end{array} : \tau'[\theta_a]$$

Case

$$\frac{i \in \textbf{fresh}(\mathbb{N}) \qquad \Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \square\tau \Rightarrow \Phi_1 \qquad \Delta; i, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow \textbf{list}[i]^{\alpha}\, \tau \Rightarrow \Phi_2 \qquad \Phi_2' = \Phi_2 \wedge n \doteq (i+1)}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \textbf{cons}_{NC}(t_1, t_2) \ominus \textbf{cons}_{NC}(t_1', t_2') \downarrow \textbf{list}[n]^{\alpha}\, \tau \Rightarrow \Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2'}\ \textbf{alg-r-consNC-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \text{cons}_{NC}(|t_1|, |t_2|) \ominus \text{cons}_{NC}(|t_1'|, |t_2'|) : \text{list}[n[\theta_a]]^{\alpha[\theta_a]}\, \tau[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma, \text{list}[n]^{\alpha}\, \tau) \subseteq dom(\Delta, \psi_a)$ ($\star$) and

$\Delta; \Phi_a[\theta_a] \models (\Phi_1 \wedge \exists i :: \mathbb{N}.\Phi_2')[\theta_a]$ ($\star\star$)

Using ($\star$), ($\star\star$)'s derivation must be in a form such that we have

a) $\Delta; \Phi_a[\theta_a] \models \Phi_1[\theta_a]$

b) $\Delta \vdash I :: \mathbb{N}$

c) $\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta_a, i \mapsto I]$

d) $\Delta; \Phi_a[\theta_a] \models (I+1) \doteq n[\theta_a]$

By IH3 on the second premise using ($\star$) and a), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_1| \ominus |t_1'| : \Box \tau[\theta_a] \tag{B.30}$$

By IH3 on the third premise using ($\star$) and b), we can show that

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_2| \ominus |t_2'| : \mathrm{list}[I]^{\alpha[\theta_a]} \tau[\theta_a] \tag{B.31}$$

By **c-r-cons2** typing rule using eqs. (B.30) and (B.31), we obtain
$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \mathrm{cons}_{NC}(|t_1|, |t_2|) \ominus \mathrm{cons}_{NC}(|t_1'|, |t_2'|) : \mathrm{list}[I+1]^{\alpha[\theta_a]} \tau[\theta_a]$.
We conclude by applying **c-r-⊑** rule to this using d) .

Proof of Theorem 53.4:

Case

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \downarrow \tau \Rightarrow \Phi \qquad \Delta; \Phi_a \vdash \tau \ \mathsf{wf} \qquad \mathbf{FIV}(\tau) \in \Delta}{\Delta; \psi_a; \Phi_a; \Gamma \vdash (t : \tau) \ominus (t' : \tau) \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-anno-}\uparrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |(t : \tau, t)| \ominus |(t' : \tau, t)| : \tau[\theta_a]$.
Since by definition, $\forall t.\ |(t : \_, \_)| = |t|$, STS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| : \tau[\theta_a]$.
By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Gamma) \subseteq dom(\Delta, \psi_a)$ ($\star$) and
$\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ ($\star\star$)
Using the third premise, we can show that

a) $\mathrm{FIV}(\Phi_a, \Gamma, \tau) \subseteq dom(\Delta, \psi_a)$.

By IH4 on the first premise using ($\star\star$) and a), we can conclude that
$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| : \tau[\theta_a]$.

Case

$$\frac{\Delta; \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \uparrow \Box \tau \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathbf{der}\ t_1 \ominus \mathbf{der}\ t_2 \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-der-}\uparrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \mathrm{der}\,|t| \backsim \mathrm{der}\,|t'| :^c \tau[\theta\,\theta_a]$.

By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Gamma) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$ such that $\Delta \rhd \theta_a : \psi_a$ are derivable.
By IH4 on the first premise using $(\star)$ and $\star\star$, we obtain

$$\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t| \ominus |t'| : \Box \tau[\theta_a] \tag{B.32}$$

Then, by **c-der** rule using eq. (B.32), we can conclude that $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \mathrm{der}\,|t| \ominus \mathrm{der}\,|t'| : \tau[\theta\,\theta_a]$.

□

Theorem 54 (Completeness of algorithmic typechecking)**.**

1. *Assume that* $\Delta; \Phi_a; \Omega \vdash t :^c A$. *Then, there exists* t' *such that* $\Delta; \cdot; \Phi_a; \Omega \vdash t' \downarrow A$ *and* $\Delta; \Phi_a \models \Phi$ *and* $|t'| = t$.

2. *Assume that* $\Delta; \Phi_a; \Gamma \vdash t_1 \backsim t_2 :^c \tau$. *Then, there exist* t_1', t_2' *such that* $\Delta; \cdot; \Phi_a; \Gamma \vdash t_1' \ominus t_2' \downarrow \tau \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$ *and* $|t_1'| = t_1$ *and* $|t_2'| = t_2$.

Proof. By simultaneous induction on the given Core typing derivations.

Proof of Theorem 54.1:

Case

$$\frac{\Omega(x) = A}{\Delta; \Phi_a; \Omega \vdash x :^c A}\ \textbf{c-var}$$

We can conclude as follows

$$\frac{\dfrac{\Omega(x) = A}{\Delta; \cdot; \Phi_a; \Omega \vdash x \uparrow A \Rightarrow \top}\ \textbf{alg-u-var-}\uparrow \qquad \Delta; \Phi_a \models^{\mathsf{A}} A \sqsubseteq A \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash x \downarrow A \Rightarrow \top}\ \textbf{alg-r-}\uparrow\downarrow$$

Case

$$\frac{\Delta; \Phi_a; \Omega \vdash t_1 :^c A \qquad \Delta; \Phi_a; \Omega \vdash t_2 :^c \mathbf{list}[n]\, A}{\Delta; \Phi_a; \Omega \vdash \mathbf{cons}_C(t_1, t_2) :^c \mathbf{list}[n+1]\, A}\ \textbf{c-cons}$$

By IH1 on the first premise, $\exists t_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Omega \vdash t_1' \downarrow A \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1'| = t_1$

By IH1 on the second premise, $\exists t_2'$ such that

d) $\Delta;\cdot;\Phi_a;\Omega \vdash t_2' \downarrow \text{list}[n]\, A \Rightarrow \Phi_2$

e) $\Delta;\Phi_a \models \Phi_2$

f) $|t_2'| = t_2$

Then, we can conclude as follows

1.

$$\frac{\begin{array}{c} i \in \text{fresh}(\mathbb{N}) \qquad \Delta;\psi_a;\Phi_a;\Omega \vdash t_1' \downarrow A \Rightarrow \Phi_1' \\ \Delta; i, k_2', t_2', \psi_a;\Phi_a;\Omega \vdash t_2' \downarrow \text{list}[i]\, A \Rightarrow \Phi_2' \qquad \Phi_2'' = (\Phi_2 \wedge n+1 \doteq (i+1)\wedge) \end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{cons}_C(t_1', t_2') \downarrow \text{list}[n+1]\, A \Rightarrow \Phi_1' \wedge \exists i :: \mathbb{N}.\Phi_2''}\ \textbf{alg-u-cons-}\downarrow$$

2. Using c) and f), $|\text{cons}_C(t_1', t_2')| = \text{cons}_C(t_1, t_2)$.

Proof of Theorem 54.2:

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_2 :^c \Box\tau}{\Delta;\Phi_a;\Gamma \vdash \textbf{der}\ t_1 \backsim \textbf{der}\ t_2 :^c \tau}\ \textbf{c-der}$$

By IH2 on the premise, $\exists t_1', t_2'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau, t \Rightarrow \Phi$

b) $\Delta;\Phi_a \models \Phi$

c) $|t_1'| = t_1$ and $|t_2'| = t_2$

Then, we can conclude by using a), b) and c) as follows:

$$\frac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \Box\tau \Rightarrow \Phi}{\Delta;\cdot;\Phi_a;\Gamma \vdash \text{der}\ t_1' \ominus \text{der}\ t_2' \downarrow \tau \Rightarrow \Phi}\ \textbf{alg-r-der-}\downarrow \text{ and}$$

1.

$$\dfrac{\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \Box\tau \Rightarrow \Phi}{\Delta;\cdot;\Phi_a;\Gamma \vdash \text{der } t_1' \ominus \text{der } t_2' \downarrow \tau \Rightarrow \Phi}\ \textbf{alg-r-der-}\downarrow}{\Delta;\cdot;\Phi_a;\Gamma \vdash (\text{der } t_1' : \tau) \ominus (\text{der } t_2' : \tau) \uparrow \tau \Rightarrow \Phi}\ \textbf{alg-r-anno-}\uparrow \qquad \Delta;\Phi_a \models \tau \equiv \tau \Rightarrow \Phi' \text{ by } \textit{Lemma } 27}{\Delta;\cdot;\Phi_a;\Gamma \vdash (\text{der } t_1' : \tau) \ominus (\text{der } t_2' : \tau) \downarrow \tau \Rightarrow \Phi \wedge \Phi'}\ \textbf{alg-r-}\uparrow\downarrow$$

2. By c), $|(\text{der } t_i' : \tau)| = \text{der } |t_i'|$.

3. By b) and Lemma 44.

Case

$$\dfrac{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a;\Gamma \vdash t \backsim t' :^c \tau'}\ \textbf{c-r-}\equiv$$

By IH2 on the first premise, $\exists t_1', t_2'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1'| = t$ and $|t_2'| = t'$

By IH2 on the second premise,

d) $\Delta;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_2$

e) $\Delta;\Phi_a \models \Phi_2$.

Then, we can conclude as follows

1. By using a) and d)

$$\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau \Rightarrow \Phi_1}{\Delta;\cdot;\Phi_a;\Gamma \vdash (t_1' : \tau) \ominus (t_2' : \tau) \uparrow \tau \Rightarrow \Phi_1}\ \textbf{alg-r-anno-}\uparrow \qquad \Delta;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_2}{\Delta;\cdot;\Phi_a;\Gamma \vdash (t_1' : \tau) \ominus (t_2' : \tau) \downarrow \tau' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-}\uparrow\downarrow$$

2. By using b), e) ,we can show that $\Delta;\Phi_a \models \Phi_1 \wedge \Phi_2$

3. By c), $|(t_1' : \tau)| = t_1'$ and $(t_2' : \tau) = t_2'$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \tau_1 \to \tau_2 \quad \Delta;\Phi_a;\Gamma \vdash t_2 \backsim t_2' :^c \tau_1}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \backsim t_1'\ t_2' :^c \tau_2}\ \textbf{c-r-app}$$

By IH2 on the first premise, $\exists t_1, t_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1 \to \tau_2 \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1| = t_1$ and $|t_1'| = t_1'$

By IH2 on the second premise, $\exists t_2, t_2'$ such that

d) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1 \Rightarrow \Phi_2$

e) $\Delta;\Phi_a \models \Phi_2$

f) $|t_2| = t_2$ and $|t_2'| = t_2'$

Then, we can conclude as follows

1.

$$\frac{\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1 \to \tau_2 \Rightarrow \Phi_1}{\Delta;\cdot;\Phi_a;\Gamma \vdash (t_1 : \tau_1 \to \tau_2) \ominus t_1' : \tau_1 \to \tau_2) \uparrow \tau_1 \to \tau_2 \Rightarrow \Phi_1}\ \textbf{alg-r-anno-}\uparrow \quad \dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1 \Rightarrow \Phi_2}{}\ \textbf{c-r-app}}{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \uparrow \tau_2 \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-}\uparrow\downarrow}{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \downarrow \tau_2 \Rightarrow \Phi_1 \wedge \Phi_2}$$

where $t_1 = (t_1 : \tau_1 \to \tau_2)\ t_2$ and $t_2 = (t_1' : \tau_1 \to \tau_2)\ t_2'$.

2. By using b) and e) .

3. Using c) and f), $|(t_1 : \tau_1 \to \tau_2)\ t_2| = t_1\ t_2$ and $|(t_1' : \tau_1 \to \tau_2)\ t_2'| = t_1'\ t_2'$.

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \backsim t_1' :^c \exists i{::}S.\tau_1 \quad i :: S, \Delta;\Phi_a;x : \tau_1, \Gamma \vdash t_2 \backsim t_2' :^c \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)}{\Delta;\Phi_a;\Gamma \vdash \textbf{unpack}\ t_1\ \textbf{as}\ (x,i)\ \textbf{in}\ t_2 \backsim \textbf{unpack}\ t_1'\ \textbf{as}\ (x,i)\ \textbf{in}\ t_2' :^c \tau_2}\ \textbf{c-r-unpack1}$$

By IH2 on the first premise, $\exists t_1, t_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \exists i{::}S.\tau_1 \Rightarrow \Phi_1$

b) $\Delta; \Phi_a \models \Phi_1$

c) $|t_1| = t_1$ and $|t_1'| = t_1'$

By IH2 on the second premise, $\exists t_2, t_2'$ such that

d) $i :: S, \Delta; \cdot; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2 \Rightarrow \Phi_2$

e) $i :: S, \Delta; \Phi_a \models \Phi_2$

f) $|t_2| = t_2$ and $|t_2'| = t_2'$

Then, we can conclude as follows

1.

$$\dfrac{\dfrac{\Delta; \cdot; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \exists i{::}S.\tau_1 \Rightarrow \Phi_1 \;\; (a)}{\Delta; \cdot; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \uparrow \exists i{::}S.\tau_1 \Rightarrow \Phi_1}\,\textbf{alg-r-anno-}\uparrow \qquad i :: S, \Delta; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2 \Rightarrow \Phi_2 \qquad \Phi' = \Phi_1 \wedge \Phi_2'}{\Delta; \cdot; \Phi_a; \Gamma \vdash \text{unpack } t_1 \text{ as } (x, i) \text{ in } t_2 \ominus \text{unpack } t_1' \text{ as } (x, i) \text{ in } t_2' \downarrow \tau_2 \Rightarrow \Phi'}\,\textbf{alg-r-unpack-}\downarrow$$

where $t_1 = (t_1 : \exists i{::}S.\tau_1)$ and $t_2 = (t_1' : \exists i{::}S.\tau_1)$

2. By using b) and e).

3. Using c) and f), $|\text{unpack } (t_1 : \exists i{::}S.\tau_1) \text{ as } (x, i) \text{ in } t_2| = \text{unpack } t_1 \text{ as } (x, i) \text{ in } t_2$ and $|\text{unpack } (t_1' : \exists i{::}S.\tau_1) \text{ as } (x, i) \text{ in } t_2'| = \text{unpack } t_1' \text{ as } (x, i) \text{ in } t_2'$.

□

B.5 RELCOST

The content of this section is from the Ph.D. thesis of Ezgi Cicek Çiçek (2018).

It first presents RelCost's syntax, typing and subtyping rules. Then, it introduces RelCostCore and the embedding of RelCost into RelCostCore. Finally, the bidirectional system BiRelCost is introduced.

Terms	t	::=	$x \mid \mathtt{n} \mid \text{fix}\, f(x).t \mid t_1\ t_2 \mid \zeta\ t \mid \langle t_1, t_2\rangle \mid \pi_1(t) \mid \pi_2(t)$ $\mid \text{inl}\ t \mid \text{inr}\ t \mid\ \text{case}\ (t, x.t_1, y.t_2) \mid \text{cons}(t_1, t_2)$ $\mid\ \text{case}\ t\ \text{of nil} \to t_1 \mid h :: tl \to t_2 \mid \Lambda.t \mid t[]$ $\mid \text{pack}\ t \mid \text{unpack}\ t_1\ \text{as}\ x\ \text{in}\ t_2 \mid \text{let}\ x = t_1\ \text{in}\ t_2$ $\mid \text{clet}\ t_1\ \text{as}\ x\ \text{in}\ t_2 \mid \text{celim}\ t \mid () \mid \text{nil}$
Values	v	::=	$\mathtt{n} \mid \text{fix}\, f(x).v \mid \langle v_1, v_2\rangle \mid \text{inl}\ v \mid \text{inr}\ v \mid \text{nil}$ $\mid \text{cons}(v_1, v_2) \mid \Lambda t \mid \text{pack}\ v \mid ()$
Relational types	τ	::=	$\text{unit} \mid \text{int} \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \xrightarrow{\text{diff}(D)} \tau_2$ $\mid \forall i \overset{\text{diff}(t)}{::} S.\tau \mid \exists i{::}S.\tau \mid U(A_1, A_2) \mid \square\tau$ $\mid C\ \&\ \tau \mid C \supset \tau \mid \text{list}[n]^{\alpha}\ \tau$
Unary types	A	::=	$\text{unit} \mid \text{int} \mid A_1 \times A_2 \mid A_1 + A_2 \mid A_1 \xrightarrow{\text{exec}(k,t)} A_2$ $\mid \text{list}[n]\ A \mid \forall i \overset{\text{exec}(k,t)}{::} S.A \mid \exists i{::}S.A$ $\mid C\ \&\ A \mid C \supset A$
Sorts	S	::=	$\mathbb{N} \mid \mathbb{R}$
Index terms	I, k, t, α	::=	$i \mid 0 \mid \infty \mid I + 1 \mid I_1 + I_2 \mid I_1 - I_2 \mid \frac{I_1}{I_2} \mid I_1 \cdot I_2 \mid \lceil I \rceil$ $\mid \lfloor I \rfloor \mid \mathtt{log}_2(I) \mid I_1^{I_2} \mid \mathtt{min}(I_1, I_2)$ $\mid \mathtt{max}(I_1, I_2) \mid \sum_{i=I_1}^{I_n} I$
Constraints	C	::=	$I_1 \doteq I_2 \mid I_1 < I_2 \mid \neg C \mid$
Constraint env.	Φ	::=	$\top \mid C \wedge \Phi$
Sort env.	Δ	::=	$\emptyset \mid \Delta, i :: S$
Unary type env.	Ω	::=	$\emptyset \mid \Omega, x : A$
Relational type env.	Γ	::=	$\emptyset \mid \Gamma, x : \tau$
Primitive env.	Υ	::=	$\emptyset \mid \Upsilon, \zeta : \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \mid \Upsilon, \zeta : A_1 \xrightarrow{\text{exec}(L,U)} A_2$
RelCost typing judg.			$\Omega \vdash_L^U t : A$ $\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$
core typing judg.			$\Omega \vdash_L^U t :^c A$ $\Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \tau$

Figure B.64: Syntax of types and contexts of RelCost.

$$
\begin{array}{lrl}
\text{Terms} & t & ::= \; x \mid \mathtt{n} \mid \text{fix}\, f(x).t \mid t_1\ t_2 \mid \zeta\ t \mid \langle t_1, t_2\rangle \mid \pi_1(t) \mid \pi_2(t) \\
& & \mid \text{inl}\ t \mid \text{inr}\ t \mid \ \text{case}\ (t, x.t_1, y.t_2) \mid \text{nil}\ \mid \text{cons}(t_1, t_2) \\
& & \mid \ \text{case}\ t\ \text{of nil} \to t_1 \mid h :: tl \to t_2 \mid \Lambda.t \mid t[] \\
& & \mid \text{pack}\ t \mid \text{unpack}\ t_1\ \text{as}\ x\ \text{in}\ t_2 \mid \text{let}\ x = t_1\ \text{in}\ t_2 \mid () \\
& & \mid \text{clet}\ t_1\ \text{as}\ x\ \text{in}\ t_2 \mid \text{celim}\ t \\
\text{Values} & v & ::= \; \mathtt{n} \mid \text{fix}\, f(x).v \mid \langle v_1, v_2\rangle \mid \text{inl}\ v \mid \text{inr}\ v \mid \text{nil}\ \mid \text{cons}(v_1, v_2) \mid \Lambda t \mid \text{pack}\ v \mid ()
\end{array}
$$

Figure B.65: Syntax of values and terms of RelCost.

$$
\begin{array}{lrl}
\text{Terms} & t & ::= \; x \mid \mathtt{n} \mid \text{fix}\, f(x).t \mid \text{fix}_{NC}\, f(x).t \mid t_1\ t_2 \mid \zeta\ t \mid \langle t_1, t_2\rangle \mid \pi_1(t) \mid \pi_2(t) \\
& & \mid \text{inl}\ t \mid \text{inr}\ t \mid \ \text{case}\ (t, x.t_1, y.t_2) \mid \text{nil}\ \mid \text{cons}_{NC}(t_1, t_2) \mid \text{cons}_C(t_1, t_2) \\
& & \mid \ \text{case}\ t\ \text{of nil} \to t_1 \mid h ::_{NC} tl \to t_2 \mid h ::_C tl \to t_3 \mid \Lambda i.t \mid t\,[I] \\
& & \mid \text{pack}\ t\ \text{with}\ I \mid \text{unpack}\ t_1\ \text{as}\ (x, i)\ \text{in}\ t_2 \mid \text{let}\ x = t_1\ \text{in}\ t_2 \mid () \mid \text{contra}\ t \\
& & \mid \text{clet}\ t_1\ \text{as}\ x\ \text{in}\ t_2 \mid \text{celim}\ t \mid \text{der}\ t \mid \text{switch}\ t \mid \text{NC}\ t \mid \text{split}\ (t_1, t_2)\ \text{with}\ C \\
\text{Values} & v & ::= \; \mathtt{n} \mid \text{fix}\, f(x).t \mid \text{fix}_{NC}\, f(x).t \mid \langle v_1, v_2\rangle \mid \text{inl}\ v \mid \text{inr}\ v \mid \text{nil} \\
& & \mid \text{cons}_{NC}(v_1, v_2) \mid \text{cons}_C(v_1, v_2) \mid \Lambda i.t \mid \text{pack}\ v\ \text{with}\ I \mid ()
\end{array}
$$

Figure B.66: Syntax of values and terms of RelCostCore.

$$
\begin{array}{lrl}
\text{Terms} & t & ::= \; x \mid \mathtt{n} \mid \text{fix}\, f(x).t \mid \text{fix}_{NC}\, f(x).t \mid t_1\ t_2 \mid \zeta\ t \mid \langle t_1, t_2\rangle \mid \pi_1(t) \mid \pi_2(t) \\
& & \mid \text{inl}\ t \mid \text{inr}\ t \mid \ \text{case}\ (t, x.t_1, y.t_2) \mid \text{nil}\ \mid \text{cons}_{NC}(t_1, t_2) \mid \text{cons}_C(t_1, t_2) \\
& & \mid \ \text{case}\ t\ \text{of nil} \to t_1 \mid h ::_{NC} tl \to t_2 \mid h ::_C tl \to t_3 \mid \Lambda i.t \mid t\,[I] \\
& & \mid \text{pack}\ t\ \text{with}\ I \mid \text{unpack}\ t_1\ \text{as}\ (x, i)\ \text{in}\ t_2 \mid \text{let}\ x = t_1\ \text{in}\ t_2 \mid () \\
& & \mid \text{clet}\ t_1\ \text{as}\ x\ \text{in}\ t_2 \mid \text{celim}\ t \mid \text{der}\ \mid \text{switch}\ t \mid \text{NC}\ t \mid \text{split}\ (t_1, t_2)\ \text{with}\ C \\
& & \mid \text{contra}\ t \mid (t : \tau, t) \mid (t : A, k, t) \\
\text{Values} & v & ::= \; \mathtt{n} \mid \text{fix}\, f(x).t \mid \text{fix}_{NC}\, f(x).t \mid \langle v_1, v_2\rangle \mid \text{inl}\ v \mid \text{inr}\ v \mid \text{nil} \\
& & \mid \text{cons}_{NC}(v_1, v_2) \mid \text{cons}_C(v_1, v_2) \mid \Lambda i.t \mid \text{pack}\ v\ \text{with}\ I \mid ()
\end{array}
$$

Figure B.67: Syntax of values and terms of BiRelCost.

$\boxed{\Delta;\Phi_a \vdash^A A\ \texttt{wf}}$ checks well-formedness of the unary type A
$\boxed{\Delta;\Phi_a \vdash \tau\ \texttt{wf}}$ checks well-formedness of the binary type τ

$$\frac{}{\Delta;\Phi_a \vdash \text{unit}\ \texttt{wf}}\ \textbf{wf-unit} \qquad \frac{}{\Delta;\psi;\Phi_a \vdash \text{int}\ \texttt{wf}}\ \textbf{wf-int}$$

$$\frac{\Delta;\psi;\Phi_a \vdash \tau_1\ \texttt{wf} \qquad \Delta;\psi;\Phi_a \vdash \tau_2\ \texttt{wf}}{\Delta;\psi;\Phi_a \vdash \tau_1 \times \tau_2\ \texttt{wf}}\ \textbf{wf-prod} \qquad \frac{\Delta;\psi;\Phi_a \vdash \tau_1\ \texttt{wf} \qquad \Delta;\psi;\Phi_a \vdash \tau_2\ \texttt{wf}}{\Delta;\psi;\Phi_a \vdash \tau_1 + \tau_2\ \texttt{wf}}\ \textbf{wf-sum}$$

$$\frac{\Delta;\psi;\Phi_a \vdash \tau_1\ \texttt{wf} \qquad \Delta;\psi;\Phi_a \vdash \tau_2\ \texttt{wf} \qquad \Delta;\Phi_a \vdash t :: \mathbb{R}}{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\text{diff}(D)} \tau_2\ \texttt{wf}}\ \textbf{wf-fun}$$

$$\frac{\Delta,\Phi_a \vdash n :: \mathbb{N} \qquad \Delta,\Phi_a \vdash \alpha :: \mathbb{N} \qquad \Delta;\Phi_a \vdash \tau\ \texttt{wf}}{\Delta;\Phi_a \vdash \text{list}[n]^{\alpha}\ \tau\ \texttt{wf}}\ \textbf{wf-list}$$

$$\frac{i :: S,\Delta;\Phi_a \vdash \tau\ \texttt{wf} \qquad i :: S,\Delta;\Phi_a \vdash t :: \mathbb{R}}{\Delta;\Phi_a \vdash \forall i \overset{\text{diff}(D)}{::} S.\tau\ \texttt{wf}}\ \textbf{wf-}\forall \qquad \frac{i :: S,\Delta;\Phi_a \vdash \tau\ \texttt{wf}\Phi}{\Delta;\Phi_a \vdash \exists i{::}S.\tau\ \texttt{wf}\Phi}\ \textbf{wf-}\exists$$

$$\frac{\Delta;\psi;\Phi_a \vdash^A A_1\ \texttt{wf} \qquad \Delta;\psi;\Phi_a \vdash^A A_2\ \texttt{wf}}{\Delta;\psi;\Phi_a \vdash U(A_1,A_2)\ \texttt{wf}}\ \textbf{wf-}U \qquad \frac{\Delta;\psi;\Phi_a \vdash \tau\ \texttt{wf}}{\Delta;\psi;\Phi_a \vdash \Box\tau\ \texttt{wf}}\ \textbf{wf-box}$$

$$\frac{\Delta;\Phi_a \vdash C\ \text{wf} \qquad \Delta;C \wedge \Phi_a \vdash \tau\ \texttt{wf}}{\Delta;\Phi_a \vdash C \supset \tau\ \texttt{wf}}\ \textbf{wf-C}\supset \qquad \frac{\Delta;\Phi_a \vdash C\ \text{wf} \qquad \Delta;\Phi_a \vdash \tau\ \texttt{wf}}{\Delta;\Phi_a \vdash C\ \&\ \tau\ \texttt{wf}}\ \textbf{wf-C}\&$$

Figure B.68: Well-formedness of relational types of RelCost.

$\boxed{\Delta \vdash^A A\ \texttt{wf}}$ checks well-formedness of the unary type $A\ \Phi$

$$\frac{}{\Delta;\Phi_a \vdash^A \text{unit}\ \texttt{wf}}\ \textbf{wf-u-unit} \qquad \frac{}{\Delta;\psi;\Phi_a \vdash^A \text{int}\ \texttt{wf}}\ \textbf{wf-u-int}$$

$$\frac{\Delta;\psi;\Phi_a \vdash^A A_1\ \texttt{wf} \qquad \Delta;\psi;\Phi_a \vdash^A A_2\ \texttt{wf}}{\Delta;\psi;\Phi_a \vdash^A A_1 \times A_2\ \texttt{wf}}\ \textbf{wf-u-prod}$$

$$\frac{\Delta;\psi;\Phi_a \vdash^A A_1\ \texttt{wf} \qquad \Delta;\psi;\Phi_a \vdash^A A_2\ \texttt{wf}}{\Delta;\psi;\Phi_a \vdash^A A_1 + A_2\ \texttt{wf}}\ \textbf{wf-u-sum}$$

$$\frac{\Delta;\psi;\Phi_a \vdash^A A_1\ \texttt{wf} \qquad \Delta;\psi;\Phi_a \vdash^A A_2\ \texttt{wf} \qquad \Delta;\Phi_a \vdash k :: \mathbb{R} \qquad \Delta;\Phi_a \vdash t :: \mathbb{R}}{\Delta;\Phi_a \vdash^A A_1 \xrightarrow{\text{exec}(L,U)} A_2\ \texttt{wf}}\ \textbf{wf-u-fun}$$

$$\frac{\Delta,\Phi_a \vdash n :: \mathbb{N} \qquad \Delta;\Phi_a \vdash^A A\ \texttt{wf}}{\Delta;\Phi_a \vdash^A \text{list}[n]\ A\ \texttt{wf}}\ \textbf{wf-u-list}$$

$$\frac{i :: S,\Delta;\Phi_a \vdash^A A\ \texttt{wf} \qquad i :: S,\Delta;\Phi_a \vdash L :: \mathbb{R} \qquad i :: S,\Delta;\Phi_a \vdash U :: \mathbb{R}}{\Delta;\Phi_a \vdash^A \forall i \overset{\text{exec}(L,U)}{::} S.\ A\ \texttt{wf}}\ \textbf{wf-u-}\forall$$

$$\frac{i :: S,\Delta;\Phi_a \vdash^A A\ \texttt{wf}}{\Delta;\Phi_a \vdash^A \exists i{::}S.\ A\ \texttt{wf}}\ \textbf{wf-u-}\exists \qquad \frac{\Delta;\Phi_a \vdash C\ \text{wf} \qquad \Delta;C \wedge \Phi_a \vdash^A A\ \texttt{wf}}{\Delta;\Phi_a \vdash^A C \supset A\ \texttt{wf}}\ \textbf{wf-u-C}\rangle$$

$$\frac{\Delta;\Phi_a \vdash C\ \text{wf} \qquad \Delta;\Phi_a \vdash^A A\ \texttt{wf}}{\Delta;\Phi_a \vdash^A C\ \&\ A\ \texttt{wf}}\ \textbf{wf-u-C}\wedge$$

Figure B.69: Well-formedness of unary types of RelCost.

$$\frac{}{\Delta \rhd [] : \cdot} \qquad \frac{\Delta \vdash I :: S \qquad \Delta \rhd \theta : \psi}{\Delta \rhd \theta[M \mapsto I] : M :: S,\psi}$$

Figure B.70: Sorting of Substitutions of RelCost.

$\boxed{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2}$ Relational type τ_1 is a subtype of relational type τ_2

$\boxed{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2}$ Type A_1 is a subtype of type A_2

$$\frac{}{\Delta;\Phi_a \models \text{int} \sqsubseteq \square\text{int}}\ \textbf{int-}\square \qquad \frac{}{\Delta;\Phi_a \models \square U\,(\text{int},\text{int}) \sqsubseteq \text{int}}\ \square\textbf{U-int}$$

$$\frac{}{\Delta;\Phi_a \models \text{unit} \sqsubseteq \square\text{unit}}\ \textbf{unit} \qquad \frac{\Delta;\Phi_a \models \tau_1' \sqsubseteq \tau_1 \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2' \qquad \Delta;\Phi_a \models D \leq D'}{\Delta;\Phi_a \models \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\text{diff}(D')} \tau_2'}\ \to\textbf{diff}$$

$$\frac{}{\Delta;\Phi_a \models \square(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2) \sqsubseteq \square\tau_1 \xrightarrow{\text{diff}(0)} \square\tau_2}\ \to\square\textbf{diff}$$

$$\frac{}{\Delta;\Phi_a \models U\,(A_1 \xrightarrow{\text{exec}(L,U)} A_2) \sqsubseteq U\,A_1 \xrightarrow{\text{diff}(U-L)} U\,A_2}\ \to\textbf{execdiff}$$

$$\frac{i :: S,\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad i :: S,\Delta;\Phi_a \models D \leq D' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \forall i \overset{\text{diff}(D)}{::} S.\tau \sqsubseteq \forall i \overset{\text{diff}(D')}{::} S.\tau'}\ \forall\textbf{diff}$$

$$\frac{}{\Delta;\Phi_a \models \square(\forall i \overset{\text{diff}(D)}{::} S.\tau) \sqsubseteq \forall i \overset{\text{diff}(0)}{::} S.\square\tau}\ \forall\,\square$$

$$\frac{}{\Delta;\Phi_a \models U\,(\forall i \overset{\text{exec}(L,U)}{::} S.A, \forall i \overset{\text{exec}(L',U')}{::} S.A') \sqsubseteq \forall i \overset{\text{diff}(U-L')}{::} S.U\,(A,A')}\ \forall\textbf{U}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_1' \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2'}{\Delta;\Phi_a \models \tau_1 \times \tau_2 \sqsubseteq \tau_1' \times \tau_2'}\ \times \qquad \frac{}{\Delta;\Phi_a \models \square\tau_1 \times \square\tau_2 \equiv \square(\tau_1 \times \tau_2)}\ \times\square$$

$$\frac{}{\Delta;\Phi_a \models U\,(A_1 \times A_2, A_1' \times A_2') \sqsubseteq U\,(A_1,A_1') \times U\,(A_2,A_2')}\ \times\textbf{U}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_1' \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2'}{\Delta;\Phi_a \models \tau_1 + \tau_2 \sqsubseteq \tau_1' + \tau_2'}\ + \qquad \frac{}{\Delta;\Phi_a \models \square\tau_1 + \square\tau_2 \sqsubseteq \square(\tau_1 + \tau_2)}\ +\square$$

$$\frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \leq \alpha' \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models \text{list}[n]^{\alpha}\,\tau \sqsubseteq \text{list}[n']^{\alpha'}\,\tau'}\ \textbf{l1}$$

Figure B.71: Subtyping rules of RelCost, part 1.

$\boxed{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2}$ Binary type τ_1 is a subtype of type τ_2

$$\frac{\Delta;\Phi_a \models \alpha \doteq 0}{\Delta;\Phi_a \models \text{list}[n]^\alpha\,\tau \sqsubseteq \text{list}[n]^\alpha\,\Box\tau}\ \textbf{l2} \qquad \frac{}{\Delta;\Phi_a \models \text{list}[n]^\alpha\,\Box\tau \sqsubseteq \Box(\text{list}[n]^\alpha\,\tau)}\ \textbf{l}\Box$$

$$\frac{i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \exists i{::}S.\,\tau \sqsubseteq \exists i{::}S.\,\tau'}\ \exists \qquad \frac{}{\Delta;\Phi_a \models \exists i{::}S.\,\Box\tau \sqsubseteq \Box(\exists i{::}S.\,\tau)}\ \exists\Box$$

$$\frac{\Delta;\Phi_a \wedge C \models C' \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models C \,\&\, \tau \sqsubseteq C' \,\&\, \tau'}\ \textbf{c-and} \qquad \frac{}{\Delta;\Phi_a \models C \,\&\, \Box\tau \sqsubseteq \Box(C \,\&\, \tau)}\ \textbf{c-and-}\Box$$

$$\frac{\Delta;\Phi_a \wedge C' \models C \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'}{\Delta;\Phi_a \models C \supset \tau \sqsubseteq C' \supset \tau'}\ \textbf{c-impl} \qquad \frac{}{\Delta;\Phi_a \models \Box(C \supset \tau) \sqsubseteq C \supset \Box\tau}\ \textbf{c-impl-}\Box$$

$$\frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \tau}\ \textbf{T} \qquad \frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \Box\Box\tau}\ \textbf{D} \qquad \frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2}{\Delta;\Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2}\ \textbf{B-}\Box$$

$$\frac{}{\Delta;\Phi_a \models \tau \sqsubseteq U\,|\tau|}\ \textbf{W} \qquad \frac{\Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'}{\Delta;\Phi_a \models U\,A \sqsubseteq U\,A'}\ \textbf{U} \qquad \frac{}{\Delta;\Phi_a \models \tau \sqsubseteq \tau}\ \textbf{refl}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2 \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_3}{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_3}\ \textbf{trans}$$

Figure B.72: Subtyping rules of RelCost, part 2.

$\boxed{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2}$ Type A_1 is a subtype of type A_2

$$\frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \qquad \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'}{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \xrightarrow{\mathrm{exec}(L,U)} A_2 \sqsubseteq A_1' \xrightarrow{\mathrm{exec}(L',U')} A_2'} \to\textbf{exec}$$

$$\frac{\begin{array}{c} i :: S, \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \qquad i :: S, \Delta;\Phi_a \models L' \leq L \\ i :: S, \Delta;\Phi_a \models U \leq U' \qquad i \notin FV(\Phi_a) \end{array}}{\Delta;\Phi_a \models^{\mathsf{A}} \forall i \overset{\mathrm{exec}(L,U)}{::} S.\, A \sqsubseteq \forall i \overset{\mathrm{exec}(L',U')}{::} S.\, A}\ \textbf{u-}\forall\textbf{exec}$$

$$\frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2'}{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \times A_2 \sqsubseteq A_1' \times A_2'}\ \textbf{u-}\times$$

$$\frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2'}{\Delta;\Phi_a \models^{\mathsf{A}} A_1 + A_2 \sqsubseteq A_1' + A_2'}\ \textbf{u-}+ \qquad \frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'}{\Delta;\Phi_a \models^{\mathsf{A}} \mathrm{list}[n]\, A \sqsubseteq \mathrm{list}[n']\, A'}\ \textbf{u-l}$$

$$\frac{i :: S, \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models^{\mathsf{A}} \exists i{::}S.\, A \sqsubseteq \exists i{::}S.\, A'}\ \textbf{u-}\exists \qquad \frac{\Delta;\Phi_a \wedge C \models C' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'}{\Delta;\Phi_a \models^{\mathsf{A}} C\ \&\ A \sqsubseteq C'\ \&\ A'}\ \textbf{u-c-and}$$

$$\frac{\Delta;\Phi_a \wedge C' \models C \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A'}{\Delta;\Phi_a \models^{\mathsf{A}} C \supset A \sqsubseteq C' \supset A'}\ \textbf{u-c-impl} \qquad \frac{}{\Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A}\ \textbf{u-refl}$$

$$\frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2 \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_3}{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_3}\ \textbf{u-tran}$$

Figure B.73: Unary subtyping rules of RelCost.

General rules

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 : A \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 : A}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim U_1 - L_2 : U\,A}\ \textbf{switch}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t \lesssim D : \tau \qquad \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)}{\Delta;\Phi_a;\Gamma,\Gamma';\Omega \vdash t \ominus t \lesssim 0 : \Box\tau}\ \textbf{nochange}$$

Constant integers and unit

$$\frac{}{\Delta;\Phi_a;\Omega \vdash^0_0 \mathtt{n} : \mathrm{int}}\ \textbf{const} \qquad \frac{}{\Delta;\Phi_a;\Gamma \vdash \mathtt{n} \ominus \mathtt{n} \lesssim 0 : \mathrm{int}}\ \textbf{r-const} \qquad \frac{}{\Delta;\Phi_a;\Omega \vdash^0_0 () : \mathrm{unit}}\ \textbf{unit}$$

$$\frac{}{\Delta;\Phi_a;\Gamma \vdash () \ominus () \lesssim 0 : \mathrm{unit}}\ \textbf{r-unit}$$

Variables x

$$\frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash^0_0 x : A}\ \textbf{var} \qquad \frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \ominus x \lesssim 0 : \tau}\ \textbf{r-var}$$

inl t

$$\frac{\Delta;\Phi_a;\Omega \vdash^U_L t : A_1 \qquad \Delta;\Phi_a \vdash^{\mathsf{A}} A_2\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^U_L \mathrm{inl}\ t : A_1 + A_2}\ \textbf{inl}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_1 \qquad \Delta;\Phi_a \vdash \tau_2\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{inl}\ t \ominus \mathrm{inl}\ t' \lesssim D : \tau_1 + \tau_2}\ \textbf{r-inl}$$

inr t

$$\frac{\Delta;\Phi_a;\Omega \vdash^U_L t : A_2 \qquad \Delta;\Phi_a \vdash^{\mathsf{A}} A_1\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^U_L \mathrm{inr}\ t : A_1 + A_2}\ \textbf{inr}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_2 \qquad \Delta;\Phi_a \vdash \tau_1\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathrm{inr}\ t \ominus \mathrm{inr}\ t' \lesssim D : \tau_1 + \tau_2}\ \textbf{r-inr}$$

Figure B.74: Typing rules of RelCost, part 1.

$\boxed{\text{case } (t, x.t_1, y.t_2)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t : A_1 + A_2 \qquad \Delta;\Phi_a;x:A_1,\Omega \vdash_{L'}^{U'} t_1 : A \qquad \Delta;\Phi_a;y:A_2,\Omega \vdash_{L'}^{U'} t_2 : A}{\Delta;\Phi_a;\Omega \vdash_{L+L'+c_{case}}^{U+U'+c_{case}} \text{case } (t, x.t_1, y.t_2) : A}\ \textbf{case}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_1 + \tau_2 \\ \Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_1 \ominus t_1' \lesssim D' : \tau \qquad \Delta;\Phi_a;y:\tau_2,\Gamma \vdash t_2 \ominus t_2' \lesssim D' : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{case } (t, x.t_1, y.t_2) \ominus \text{case } (t', x.t_1', y.t_2') \lesssim D + D' : \tau}\ \textbf{r-case}$$

$\boxed{\text{fix } f(x).t}$

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A_1 \xrightarrow{\text{exec}(L,U)} A_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:A_1,f:A_1 \xrightarrow{\text{exec}(L,U)} A_2,\Omega \vdash_L^U t : A_2}{\Delta;\Phi_a;\Omega \vdash_0^0 \text{fix } f(x).t : A_1 \xrightarrow{\text{exec}(L,U)} A_2}\ \textbf{fix}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\text{diff}(D)} \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1,f:\tau_1 \xrightarrow{\text{diff}(D)} \tau_2,\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t_1 \ominus \text{fix } f(x).t_2 \lesssim 0 : \tau_1 \xrightarrow{\text{diff}(D)} \tau_2}\ \textbf{r-fix}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\text{diff}(D)} \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1,f:\Box(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2),\Gamma \vdash t \ominus t \lesssim D : \tau_2 \\ \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t \ominus \text{fix } f(x).t \lesssim 0 : \Box(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2)}\ \textbf{r-fixNC}$$

$\boxed{t_1\ t_2}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 : A_1}{\Delta;\Phi_a;\Omega \vdash_{L_1+L_2+L+c_{app}}^{t_1+t_2+U+c_{app}} t_1\ t_2 : A_2}\ \textbf{app}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \\ \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_1\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \lesssim D_1 + D_2 + D : \tau_2}\ \textbf{r-app}$$

Figure B.75: Typing rules of RelCost, part 2.

$\boxed{\langle t_1, t_2\rangle}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : A_1 \qquad \Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : A_2}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \langle t_1, t_2\rangle : A_1 \times A_2}\ \textbf{prod}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau_1 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \langle t_1, t_2\rangle \ominus \langle t_1', t_2'\rangle \lesssim D_1 + D_2 : \tau_1 \times \tau_2}\ \textbf{r-prod}$$

$\boxed{\pi_1(t)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A_1 \times A_2}{\Delta;\Phi_a;\Omega \vdash^{U+c_{proj}}_{L+c_{proj}} \pi_1(e) : A_1}\ \textbf{proj1} \qquad \frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau_1 \times \tau_2}{\Delta;\Phi_a;\Gamma \vdash \pi_1(e) \ominus \pi_1(t') \lesssim D : \tau_1}\ \textbf{r-proj1}$$

$\boxed{\pi_2(t)}$

Symmetric rules.

$\boxed{\text{nil}}$

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \text{nil} : \text{list}[0]\, A}\ \textbf{nil} \qquad \frac{\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{nil} \ominus \text{nil} \lesssim 0 : \text{list}[0]^{\alpha}\, \tau}\ \textbf{r-nil}$$

$\boxed{\text{cons}(t_1, t_2)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : A \qquad \Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 : \text{list}[n]\, A}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \text{cons}(t_1, t_2) : \text{list}[n+1]\, A}\ \textbf{cons}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \text{list}[n]^{\alpha}\, \tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}(t_1, t_2) \ominus \text{cons}(t_1', t_2') \lesssim D_1 + D_2 : \text{list}[n+1]^{\alpha+1}\, \tau}\ \textbf{r-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \text{list}[n]^{\alpha}\, \tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}(t_1, t_2) \ominus \text{cons}(t_1', t_2') \lesssim D_1 + D_2 : \text{list}[n+1]^{\alpha}\, \tau}\ \textbf{r-cons2}$$

Figure B.76: Typing rules of RelCost, part 3.

$\boxed{\text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2}$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash_L^U t : \text{list}[n]\ A \\ \Delta;\Phi_a \wedge n = 0;\Omega \vdash_{L'}^{U'} t_1 : A' \qquad i,\Delta;\Phi_a \wedge n = i+1;h : A, tl : \text{list}[i]\ A,\Omega \vdash_{L'}^{U'} t_2 : A'\end{array}}{\Delta;\Phi_a;\Omega \vdash_{L+L'+c_{caseL}}^{U+U'+c_{caseL}} \text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2 : A'}\ \textbf{caseL}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \text{list}[n]^\alpha\ \tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \ominus t_1' \lesssim D' : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h : \square\tau, tl : \text{list}[i]^\alpha\ \tau,\Gamma \vdash t_2 \ominus t_2' \lesssim D' : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h : \tau, tl : \text{list}[i]^\beta\ \tau,\Gamma \vdash t_2 \ominus t_2' \lesssim D' : \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2 \ominus \text{case } t' \text{ of nil } \to t_1' \mid h :: tl \to t_2' \lesssim D + D' : \tau'}$$

r-caseL

$\boxed{\text{leaf}}$

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash_0^0 \text{leaf} : \text{tree}[0]\ A}\ \textbf{leaf} \qquad \frac{\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{leaf} \ominus \text{leaf} \lesssim 0 : \text{tree}[0]^\alpha\ \tau}\ \textbf{r-leaf}$$

$\boxed{\text{node}(t_l, t, t_r)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t : A \qquad \Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_l : \text{tree}[i]\ A \qquad \Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_r : \text{tree}[j]\ A}{\Delta;\Phi_a;\Omega \vdash_{L+L_1+L_2}^{U+U_1+U_2} \text{node}(t_l, t, t_r) : \text{tree}[i+j+1]\ A}\ \textbf{node}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau \\ \Delta;\Phi_a;\Gamma \vdash t_l \ominus t_l' \lesssim D_1 : \text{tree}[i]^\alpha\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_r \ominus t_r' \lesssim D_2 : \text{tree}[j]^\beta\ \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{node}(t_l, t, t_r) \ominus \text{node}(t_l', t', t_r') \lesssim D + D_1 + D_2 : \text{tree}[i+j+1]^{\alpha+\beta+1}\ \tau}\ \textbf{r-node1}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \square\tau \\ \Delta;\Phi_a;\Gamma \vdash t_l \ominus t_l' \lesssim D_1 : \text{tree}[i]^\alpha\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_r \ominus t_r' \lesssim D_2 : \text{tree}[j]^\beta\ \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{node}(t_l, t, t_r) \ominus \text{node}(t_l', t', t_r') \lesssim D + D_1 + D_2 : \text{tree}[i+j+1]^{\alpha+\beta}\ \tau}\ \textbf{r-node2}$$

Figure B.77: Typing rules of RelCost, part 4.

$\boxed{\text{case } t \text{ of leaf} \to t_1 \mid \text{node}(l,x,r) \to t_2}$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash_L^U t : \text{tree}[n]\ A \qquad \Delta;\Phi_a \wedge n = 0;\Omega \vdash_{L'}^{U'} t_1 : A' \\ i,j,\Delta;\Phi_a \wedge n = i+j+1;x:A,l:\text{tree}[i]\ A,r:\text{tree}[j]\ A,\Omega \vdash_{L'}^{U'} t_2 : A'\end{array}}{\Delta;\Phi_a;\Omega \vdash_{L+L'+c_{caseT}}^{U+U'+c_{caseT}} \text{case } t \text{ of leaf} \to t_1 \mid \text{node}(l,x,r) \to t_2 : A'}\ \textbf{caseT}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \text{tree}[n]^{\alpha}\ \tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \ominus t_1' \lesssim D' : \tau' \\ i,j,\beta,\theta,\Delta;\Phi_a \wedge n = i+j+1 \wedge \alpha = \beta+\theta;x:\square\tau,l:\text{tree}[i]^{\beta}\ \tau,r:\text{tree}[j]^{\theta}\ \tau,\Gamma \\ \vdash t_2 \ominus t_2' \lesssim D' : \tau' \\ i,j,\beta,\theta,\Delta;\Phi_a \wedge n = i+j+1 \wedge \alpha = \beta+\theta+1;x:\tau,l:\text{tree}[i]^{\beta}\ \tau,r:\text{tree}[j]^{\theta}\ \tau,\Gamma \\ \vdash t_2 \ominus t_2' \lesssim D' : \tau'\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \text{case } t \text{ of leaf} \to t_1 \mid \text{node}(l,x,r) \to t_2 \ominus \text{case } t' \text{ of leaf} \to t_1' \mid \text{node}(l,x,r) \to t_2' \\ \lesssim D + D' : \tau'\end{array}}$$

r-caseT

$\boxed{\Lambda t}$

$$\frac{i :: S,\Delta;\Phi_a;\Omega \vdash_L^U t : A \qquad i \notin \text{FIV}(\Phi_a;\Omega)}{\Delta;\Phi_a;\Omega \vdash_0^0 \Lambda.t : \forall i \overset{\text{exec}(L,U)}{::} S.\ A}\ \textbf{iLam}$$

$$\frac{\begin{array}{c}i :: S,\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau \\ i \notin \text{FIV}(\Phi_a;\Gamma)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \Lambda.t \ominus \Lambda.t' \lesssim 0 : \forall i \overset{\text{diff}(D)}{::} S.\ \tau}\ \textbf{r-iLam}$$

$\boxed{t[\,]}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t : \forall i \overset{\text{exec}(L',U')}{::} S.\ A \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Omega \vdash_{L+L'[I/i]}^{U+U'[I/i]} t[\,] : A\{I/i\}}\ \textbf{iApp}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \forall i \overset{\text{diff}(D')}{::} S.\ \tau \\ \Delta \vdash I : S\end{array}}{\Delta;\Phi_a;\Gamma \vdash t[\,] \ominus t'[\,] \lesssim D + D'[I/i] : \tau\{I/i\}}\ \textbf{r-iApp}$$

Figure B.78: Typing rules of RelCost, part 5.

pack t

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t : A\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Omega \vdash_L^U \text{pack } t : \exists i{::}S.\, A}\ \textbf{pack}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \text{pack } t \ominus \text{pack } t' \lesssim D : \exists i{::}S.\, \tau}\ \textbf{r-pack}$$

unpack t as x in t'

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : \exists i{::}S.\, A_1 \\ i :: S, \Delta;\Phi_a;x : A_1,\Omega \vdash_{L_2}^{U_2} t_2 : A_2 \qquad i \notin FV(\Phi_a;\Gamma,A_2,L_2,t_2)\end{array}}{\Delta;\Phi_a;\Omega \vdash_{L_1+L_2}^{U_1+U_2} \text{unpack } t_1 \text{ as } x \text{ in } t_2 : A_2}\ \textbf{unpack}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \exists i{::}S.\, \tau_1 \\ i :: S, \Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,t_2)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } x \text{ in } t_2 \ominus \text{unpack } t_1' \text{ as } x \text{ in } t_2' \lesssim D_1 + D_2 : \tau_2}\ \textbf{r-unpack1}$$

Primitive application

$$\frac{\Upsilon(\zeta) = A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash_{L'}^{U'} t : A_1}{\Delta;\Phi_a;\Omega \vdash_{L+L'+c_{app}}^{U+U'+c_{app}} \zeta\ t : A_2}\ \textbf{primapp}$$

$$\frac{\Upsilon(\zeta) = \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D' : \tau_1}{\Delta;\Phi_a;\Gamma \vdash \zeta\ t \ominus \zeta\ t' \lesssim D + D' : \tau_2}\ \textbf{r-primapp}$$

let $x = t_1$ in t_2

$$\frac{\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 : A_1 \qquad \Delta;\Phi_a;x : A_1,\Omega \vdash_{L_2}^{U_2} t_2 : A_2}{\Delta;\Phi_a;\Omega \vdash_{L_1+L_2+c_{let}}^{U_1+U_2+c_{let}} \text{let } x = t_1 \text{ in } t_2 : A_2}\ \textbf{let}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \tau_1 \qquad \Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \ominus \text{let } x = t_1' \text{ in } t_2' \lesssim D_1 + D_2 : \tau_2}\ \textbf{r-let1}$$

Figure B.79: Typing rules of RelCost, part 6.

$C \,\&\, \tau$ intro. rules

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Omega \vdash^{U}_{L} t : A}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : C \,\&\, A}\ \textbf{c-andI}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \ominus t' \lesssim D : \tau}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : C \,\&\, \tau}\ \textbf{c-andI}$$

$C \,\&\, \tau$ elim. rules

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 : C \,\&\, A_1 \qquad \Delta;\Phi_a \wedge C;x : A_1,\Omega \vdash^{U_2}_{L_2} t_2 : A_2}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \text{clet } t_1 \text{ as } x \text{ in } t_2 : A_2}\ \textbf{c-andE}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : C \,\&\, \tau_1 \qquad \Delta;\Phi_a \wedge C;x : \tau_1,\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \ominus \text{clet } t_1' \text{ as } x \text{ in } t_2' \lesssim D_1 + D_2 : \tau_2}\ \textbf{r-c-andE}$$

$C \supset \tau$ intro. rules

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash^{U}_{L} t : A}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : C \supset A}\ \textbf{c-impI} \qquad \frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \ominus t' \lesssim D : \tau}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : C \supset \tau}\ \textbf{r-c-impI}$$

$C \supset \tau$ elim. rules

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : C \supset A \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} \text{celim } t : A}\ \textbf{c-implE}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash \text{celim } t \ominus \text{celim } t' \lesssim D : \tau}\ \textbf{r-c-implE}$$

Subtyping

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \\ \Delta;\Phi_a \models k' \leq k \qquad \Delta;\Phi_a \models t \leq t'\end{array}}{\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t : A'}\ \sqsubseteq_{\text{exec}}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad \Delta;\Phi_a \models t \leq t'}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D' : \tau'}\ \textbf{r-}\sqsubseteq$$

Figure B.80: Typing rules of RelCost, part 7.

Constraint dependent typing

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash^{U}_{L} t : A \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash^{U}_{L} t : A \qquad \Delta \vdash C\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash^{U}_{L} t : A}\ \textbf{split}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \qquad \Delta \vdash C\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau}\ \textbf{r-split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Omega\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash^{U}_{L} t : A}\ \textbf{contra} \qquad \frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Gamma\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau}\ \textbf{r-contra}$$

Asynchronous typing

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 : A_1 \\ \Delta;\Phi_a;x : U\,A_1,\Gamma \vdash t_2 \ominus t \lesssim D_2 : \tau_2\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \ominus t \lesssim t_1 + t_2 + c_{let} : \tau_2}\ \textbf{r-let-t}$$

$$\frac{\Delta;\Phi_a;|\Gamma|2 \vdash^{U_1}_{L_1} t_1 : A_1 \qquad \Delta;\Phi_a;x : U\,(A_1,A_1),\Gamma \vdash t \ominus t_2 \lesssim D_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash t \ominus \text{let } x = t_1 \text{ in } t_2 \lesssim U_2 - L_1 - c_{let} : \tau_2}\ \textbf{r-t-let}$$

$$\frac{\Delta;\Phi_a;|\Gamma|1 \vdash^{U_1}_{L_1} t_1 : A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : U\,(A_1,A_2')}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_2' \lesssim t_1 + t_2 + t + c_{app} : U\,(A_2,A_2')}\ \textbf{r-app-t}$$

$$\frac{\Delta;\Phi_a;|\Gamma|2 \vdash^{U_1}_{L_1} t_1' : A_1' \xrightarrow{\text{exec}(L,U)} A_2' \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : U\,(A_2,A_1')}{\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_1'\ t_2' \lesssim U_2 - L_1 - k - c_{app} : U\,(A_2,A_2')}\ \textbf{r-t-app}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|1 \vdash^{t}_{\_} t : A_1 + A_2 \\ \Delta;\Phi_a;x : U\,(A_1,A_1),\Gamma \vdash t_1 \ominus t' \lesssim D' : \tau \qquad \Delta;\Phi_a;y : U\,(A_2,A_2),\Gamma \vdash t_2 \ominus t' \lesssim D' : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{ case } (t,x.t_1,y.t_2) \ominus t' \lesssim D' + D + c_{case} : \tau}\ \textbf{r-case-t}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|2 \vdash^{-}_{k'} t' : A_1 + A_2 \\ \Delta;\Phi_a;x : U\,(A_1,A_1),\Gamma \vdash t \ominus t_1' \lesssim D : \tau \qquad \Delta;\Phi_a;y : U\,(A_2,A_2),\Gamma \vdash t \ominus t_2' \lesssim D : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash t \ominus \text{ case } (t',x.t_1',y.t_2') \lesssim t - k' - c_{case} : \tau}\ \textbf{r-t-case}$$

Figure B.81: Typing rules of RelCost, part 8.

$\boxed{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim D : \tau}$ Expressions $t_1 \ominus t_2$ are embedded into $t_1^* \ominus t_1^*$ with the relational type τ and the relational cost D.

$\boxed{\Delta;\Phi_a;\Omega \vdash_L^U t \rightsquigarrow t^* : A}$ Expression t is embedded into $t*$ with the unary type A and the minimum and maximum execution costs L and U, respectively.

General rules

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash_{L_1}^{U_1} t_1 \rightsquigarrow t_1^* : A \qquad \Delta;\Phi_a;|\Gamma| \vdash_{L_2}^{U_2} t_2 \rightsquigarrow t_2^* : A}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow \text{switch } t_1^* \ominus \text{switch } t_2^* \lesssim U_1 - L_2 : U\ A}\ \textbf{e-switch}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t \rightsquigarrow t^* \ominus t^* \lesssim D : \tau \qquad \forall x_i \in dom(\Gamma),\ \ t_i = \text{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)}}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \ominus t \rightsquigarrow \text{let } \overline{y_i = t_i\ x_i} \text{ in NC } t^*[\overline{y_i/x_i}] \ominus \text{let } \overline{y_i = t_i\ x_i} \text{ in NC } t^*[\overline{y_i/x_i}] \lesssim 0 : \Box\tau}$$

e-nochange

Constant integers and unit

$$\frac{}{\Delta;\Phi_a;\Omega \vdash_0^0 \mathtt{n} \rightsquigarrow \mathtt{n} : \text{int}}\ \textbf{e-u-const} \qquad \frac{}{\Delta;\Phi_a;\Gamma \vdash \mathtt{n} \ominus \mathtt{n} \rightsquigarrow \mathtt{n} \ominus \mathtt{n} \lesssim 0 : \text{int}}\ \textbf{e-r-const}$$

$$\frac{}{\Delta;\Phi_a;\Omega \vdash_0^0 () \rightsquigarrow () : \text{unit}}\ \textbf{e-u-unit} \qquad \frac{}{\Delta;\Phi_a;\Gamma \vdash () \ominus () \rightsquigarrow () \ominus () \lesssim 0 : \text{unit}}\ \textbf{e-r-unit}$$

Variables x

$$\frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash_0^0 x \rightsquigarrow x : A}\ \textbf{e-u-var} \qquad \frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \ominus x \rightsquigarrow x \ominus x \lesssim 0 : \tau}\ \textbf{e-r-var}$$

Figure B.82: Embedding rules of RelCost, part 1.

$\boxed{\text{inl } t}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A_1 \qquad \Delta;\Phi_a \vdash A_2 \text{ wf}}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} \text{inl } t \rightsquigarrow \text{inl } t^* : A_1 + A_2}\ \textbf{e-u-inl}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau_1 \qquad \Delta;\Phi_a \vdash \tau_2 \text{ wf}}{\Delta;\Phi_a;\Gamma \vdash \text{inl } t_1 \ominus \text{inl } t_2 \rightsquigarrow \text{inl } {t_1}^* \ominus \text{inl } {t_2}^* \lesssim D : \tau_1 + \tau_2}\ \textbf{e-r-inl}$$

$\boxed{\text{inr } t}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A_2 \qquad \Delta;\Phi_a \vdash A_1 \text{ wf}}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} \text{inr } t \rightsquigarrow \text{inr } t^* : A_1 + A_2}\ \textbf{e-u-inr}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau_2 \qquad \Delta;\Phi_a \vdash \tau_1 \text{ wf}}{\Delta;\Phi_a;\Gamma \vdash \text{inr } t_1 \ominus \text{inl } t_2 \rightsquigarrow \text{inr } {t_1}^* \ominus \text{inl } {t_2}^* \lesssim D : \tau_1 + \tau_2}\ \textbf{e-r-inr}$$

$\boxed{\text{case } (e, x.t_1, y.t_2)}$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A_1 + A_2 \\ \Delta;\Phi_a;x : A_1,\Omega \vdash^{U'}_{L'} t_1 \rightsquigarrow {t_1}^* : A \qquad \Delta;\Phi_a;y : A_2,\Omega \vdash^{U'}_{L'} t_2 \rightsquigarrow {t_2}^* : A\end{array}}{\Delta;\Phi_a;\Omega \vdash^{U+U'+c_{case}}_{L+L'+c_{case}} \text{case } (e, x.t_1, y.t_2) \rightsquigarrow \text{case } (t^*, x.t_1^*, y.t_2^*) : A}\ \textbf{e-u-case}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus {t'}^* \lesssim D : \tau_1 + \tau_2 \\ \Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D' : \tau \\ \Delta;\Phi_a;y : \tau_2,\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D' : \tau \\ t_1^{**} = \text{case } (t^*, x.t_1^*, y.t_2^*) \qquad t_2^{**} = \text{case } (t', x.t_1'^*, y.t_2'^*)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{case } (e, x.t_1, y.t_2) \ominus \text{case } (t', x.t_1', y.t_2') \rightsquigarrow t_1^{**} \ominus t_2^{**} \lesssim D + D' : \tau}\ \textbf{e-r-case}$$

Figure B.83: Embedding rules of RelCost, part 2.

$\boxed{\text{fix } f(x).t}$

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A_1 \xrightarrow{\text{exec}(L,U)} A_2 \ \mathsf{wf} \qquad \Delta;\Phi_a;x:A_1,f:A_1 \xrightarrow{\text{exec}(L,U)} A_2,\Omega \vdash_L^U t \rightsquigarrow t^* : A_2}{\Delta;\Phi_a;\Omega \vdash_0^0 \text{fix } f(x).t \rightsquigarrow \text{fix } f(x).t^* : A_1 \xrightarrow{\text{exec}(L,U)} A_2}\ \textbf{e-u-fix}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \ \mathsf{wf} \\ \Delta;\Phi_a;x:\tau_1,f:\tau_1 \xrightarrow{\text{diff}(D)} \tau_2,\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau_2\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t_1 \ominus \text{fix } f(x).t_2 \rightsquigarrow \text{fix } f(x).{t_1}^* \ominus \text{fix } f(x).{t_2}^* \lesssim 0 : \tau_1 \xrightarrow{\text{diff}(D)} \tau_2}\ \textbf{e-r-fix}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1,f:\square(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2),\Gamma \vdash t \ominus t \rightsquigarrow t^* \ominus t^* \lesssim D : \tau_2 \\ \forall x_i \in dom(\Gamma),\ \ t_i = \text{coerce}_{\Gamma(x_i),\square\Gamma(x_i)} \qquad t^{**} = \text{let } \overline{y_i = t_i\ x_i} \text{ in } \text{fix}_{NC}\ f(x).t^*[\overline{y_i/x_i}]\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t \ominus \text{fix } f(x).t \rightsquigarrow t^{**} \ominus t^{**} \lesssim 0 : \square(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2)}$$

e-r-fixNC

$\boxed{t_1\ t_2}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 \rightsquigarrow {t_1}^* : A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 \rightsquigarrow {t_2}^* : A_1}{\Delta;\Phi_a;\Omega \vdash_{L_1+L_2+k+c_{app}}^{U_1+t_2+t+c_{app}} t_1\ t_2 \rightsquigarrow t_1^*\ t_2^* : A_2}\ \textbf{e-u-app}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \\ \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \tau_1\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \rightsquigarrow t_1^*\ t_2^* \ominus t_1'^*\ t_2'^* \lesssim D_1 + D_2 + D : \tau_2}\ \textbf{e-r-app}$$

$\boxed{\langle t_1, t_2\rangle}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 \rightsquigarrow {t_1}^* : A_1 \qquad \Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 \rightsquigarrow {t_2}^* : A_2}{\Delta;\Phi_a;\Omega \vdash_{L_1+L_2}^{U_1+t_2} \langle t_1, t_2\rangle \rightsquigarrow \langle t_1^*, t_2^*\rangle : A_1 \times A_2}\ \textbf{e-u-prod}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \tau_1 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \langle t_1, t_2\rangle \ominus \langle t_1', t_2'\rangle \rightsquigarrow \langle t_1^*, t_2^*\rangle \ominus \langle t_1'^*, t_2'^*\rangle \lesssim D_1 + D_2 : \tau_1 \times \tau_2}\ \textbf{e-r-prod}$$

Figure B.84: Embedding rules of RelCost, part 3.

$\boxed{\pi_1(t)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash U \ominus t \rightsquigarrow U^* \ominus t^* \lesssim L : A_1 \times A_2}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} \pi_1(e) \rightsquigarrow \pi_1(t^*) : A_1}\ \textbf{e-u-proj1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \tau_1 \times \tau_2}{\Delta;\Phi_a;\Gamma \vdash \pi_1(e) \ominus \pi_1(t') \rightsquigarrow \pi_1(t^*) \ominus \pi_1(t'^*) \lesssim D : \tau_1}\ \textbf{e-r-proj1}$$

$\boxed{\text{nil}}$

$$\frac{\Delta;\Phi_a \vdash A\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \text{nil} \rightsquigarrow \text{nil} : \text{list}[0]\ A}\ \textbf{e-u-nil}$$

$$\frac{\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{nil} \ominus \text{nil} \rightsquigarrow \text{nil} \ominus \text{nil} \lesssim 0 : \text{list}[0]^{\alpha}\ \tau}\ \textbf{e-r-nil}$$

$\boxed{\text{cons}(t_1, t_2)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{L_1}_{D_1} t_1 \rightsquigarrow {t_1}^* : A \qquad \Delta;\Phi_a;\Omega \vdash^{L_2}_{D_2} t_2 \rightsquigarrow {t_2}^* : \text{list}[n]\ A}{\Delta;\Phi_a;\Omega \vdash^{U_1+t_2}_{L_1+L_2} \text{cons}(t_1, t_2) \rightsquigarrow \text{cons}_C(t_1^*, t_2^*) : \text{list}[n+1]\ A}\ \textbf{e-u-cons}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \text{list}[n]^{\alpha}\ \tau}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \text{cons}(t_1, t_2) \ominus \text{cons}(t_1', t_2') \rightsquigarrow \text{cons}_C(t_1^*, t_2^*) \ominus \text{cons}_C(t_1'^*, t_2'^*) \lesssim D_1 + D_2 : \text{list}[n+1]^{\alpha+1}\ \tau\end{array}}$$

e-r-cons1

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \text{list}[n]^{\alpha}\ \tau}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \text{cons}(t_1, t_2) \ominus \text{cons}(t_1', t_2') \rightsquigarrow \text{cons}_{NC}(t_1', t_2') \ominus \text{cons}_{NC}(t_1^*, t_2^*) \lesssim D_1 + D_2 : \text{list}[n+1]^{\alpha}\ \tau\end{array}}$$

e-r-cons2

Figure B.85: Embedding rules of RelCost, part 4.

$\boxed{\text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t \rightsquigarrow t^* : \text{list}[n]\, A \qquad \Delta;\Phi_a \wedge n = 0;\Omega \vdash_{L'}^{U'} t_1 \rightsquigarrow {t_1}^* : A' \qquad i,\Delta;\Phi_a \wedge n = i+1;h : A, tl : \text{list}[i]\, A,\Omega \vdash_{L'}^{U'} t_2 \rightsquigarrow {t_2}^* : A'}{\Delta;\Phi_a;\Omega \vdash_{L+L'+c_{caseL}}^{U+U'+c_{caseL}} \begin{array}{l}\text{case } t \text{ of nil } \to t_1 \\ \mid h :: tl \to t_2\end{array} \rightsquigarrow \begin{array}{l}\text{case } t^* \text{ of nil } \to t_1^* \\ \mid h ::_{NC} tl \to t_2^* \\ \mid h ::_C tl \to t_2^*\end{array} : A'} \textbf{ e-u-caseL}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus {t'}^* \lesssim D : \text{list}[n]^\alpha\, \tau \\ \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D' : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h : \Box\tau, tl : \text{list}[i]^\alpha\, \tau,\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D' : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h : \tau, tl : \text{list}[i]^\beta\, \tau,\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow t_3^* \ominus t_3'^* \lesssim D' : \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\text{case } t \text{ of nil } \to t_1 \\ \mid h :: tl \to t_2\end{array} \ominus \begin{array}{l}\text{case } t \text{ of nil } \to t_1' \\ \mid h :: tl \to t_2'\end{array} \rightsquigarrow\lesssim D + D' : \tau'} \textbf{ e-r-caseL}$$

$$\begin{array}{l}\text{case } t^* \text{ of nil } \to \\ t_1^* \\ \mid h ::_{NC} tl \to t_2^* \\ \mid h ::_C tl \to t_3^*\end{array} \ominus \begin{array}{l}\text{case } t'^* \text{ of nil } \to \\ t_1'^* \\ \mid h ::_{NC} tl \to t_2'^* \\ \mid h ::_C tl \to t_3'^*\end{array}$$

$\boxed{\Lambda.t}$

$$\frac{i :: S,\Delta;\Phi_a;\Omega \vdash_L^U t \rightsquigarrow t^* : A \qquad i \notin \text{FIV}(\Phi_a;\Omega)}{\Delta;\Phi_a;\Omega \vdash_0^0 \Lambda.t \rightsquigarrow \Lambda i.t^* : \forall i \overset{\text{exec}(L,U)}{::} S.\, A} \textbf{ e-u-iLam}$$

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus {t'}^* \lesssim D : \tau \qquad i \notin \text{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda.t \ominus \Lambda.t' \rightsquigarrow \Lambda i.t^* \ominus \Lambda i.{t'}^* \lesssim 0 : \forall i \overset{\text{diff}(D)}{::} S.\, \tau} \textbf{ e-r-iLam}$$

$\boxed{t[\,]}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t \rightsquigarrow t^* : \forall i \overset{\text{exec}(L',U')}{::} S.\, A \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Omega \vdash_{L+L'[I/i]+c_{iApp}}^{U+U'[I/i]+c_{iApp}} t[] \rightsquigarrow t^*[I] : A\{I/i\}} \textbf{ e-u-iApp}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus {t'}^* \lesssim D : \forall i \overset{\text{diff}(U')}{::} S.\, \tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash t[] \ominus t'[] \rightsquigarrow t^*[I] \ominus {t'}^*[I] \lesssim D + D'[I/i] : \tau\{I/i\}} \textbf{ e-r-iApp}$$

Figure B.86: Embedding rules of RelCost, part 5.

pack t

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} \text{pack } t \rightsquigarrow \text{pack } t^* \text{ with } I : \exists i{::}S.\, A}\ \textbf{e-u-pack}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \text{pack } t \ominus \text{pack } t' \rightsquigarrow \text{pack } t^* \text{ with } I \ominus \text{pack } t'^* \text{ with } I \lesssim D : \exists i{::}S.\, \tau}\ \textbf{e-r-pack}$$

unpack t as x in t'

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 \rightsquigarrow {t_1}^* : \exists i{::}S.\, A_1 \\ i :: S, \Delta;\Phi_a;x : A_1,\Omega \vdash^{U_2}_{L_2} t_2 \rightsquigarrow {t_2}^* : A_2 \qquad i \notin FV(\Phi_a;\Omega,A_2,L_2,U_2)\end{array}}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+c_{unpack}}_{L_1+L_2+c_{unpack}} \text{unpack } t_1 \text{ as } x \text{ in } t_2 \rightsquigarrow \text{unpack } t_1^* \text{ as } (x,i) \text{ in } t_2^* : A_2}\ \textbf{e-u-unpk}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \exists i{::}S.\, \tau_1 \\ i :: S, \Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,D_2) \\ t_1^{**} = \text{unpack } t_1^* \text{ as } (x,i) \text{ in } t_2^* \qquad t_2^{**} = \text{unpack } t_1'^* \text{ as } (x,i) \text{ in } t_2'^*\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } x \text{ in } t_2 \ominus \text{unpack } t_1' \text{ as } x \text{ in } t_2' \rightsquigarrow t_1^{**} \ominus t_2^{**} \lesssim D_1 + D_2 : \tau_2}\ \textbf{e-r-unpk}$$

let $x = t_1$ in t_2

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 \rightsquigarrow {t_1}^* : A_1 \qquad \Delta;\Phi_a;x : A_1,\Omega \vdash^{U_2}_{L_2} t_2 \rightsquigarrow {t_2}^* : A_2}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+c_{let}}_{L_1+L_2+c_{let}} \text{let } x = t_1 \text{ in } t_2 \rightsquigarrow \text{let } x = t_1^* \text{ in } t_2^* : A_2}\ \textbf{e-u-let}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \tau_1 \qquad \Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \tau_2}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \text{let } x = t_1 \text{ in } t_2 \ominus \text{let } x = t_1' \text{ in } t_2' \rightsquigarrow \text{let } x = t_1^* \text{ in } t_2^* \ominus \text{let } x = t_1'^* \text{ in } t_2'^* \lesssim D_1 + D_2 : \tau_2\end{array}}$$

e-r-let

$C \,\&\, \tau$ intro. rules

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : C \,\&\, A}\ \textbf{e-u-andI}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \tau}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : C \,\&\, \tau}\ \textbf{e-r-andI}$$

Figure B.87: Embedding rules of RelCost, part 6.

$C \,\&\, \tau$ elim. rules

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 \rightsquigarrow {t_1}^* : C \,\&\, A_1 \qquad \Delta;\Phi_a \wedge C; x : A_1, \Omega \vdash^{U_2}_{L_2} t_2 \rightsquigarrow {t_2}^* : A_2}{\Delta;\Phi_a;\Gamma \vdash^{U_1+U_2}_{L_1+L_2} \text{clet } t_1 \text{ as } x \text{ in } t_2 \rightsquigarrow \text{clet } t_1^* \text{ as } x \text{ in } t_2^* : A_2}\ \textbf{e-u-c-andE}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : C \,\&\, \tau_1 \\ \Delta;\Phi_a \wedge C; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \tau_2 \qquad D = D_1 + D_2\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \ominus \text{clet } t_1' \text{ as } x \text{ in } t_2' \rightsquigarrow \text{clet } t_1^* \text{ as } x \text{ in } t_2^* \ominus \text{clet } t_1'^* \text{ as } x \text{ in } t_2'^* \lesssim D : \tau_2\end{array}}$$

e-r-c-andE

$C \supset \tau$ intro. rules

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : C \supset A}\ \textbf{e-u-c-impI}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus {t'}^* \lesssim D : \tau}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus {t'}^* \lesssim D : C \supset \tau}\ \textbf{e-r-c-impI}$$

$C \supset \tau$ elim. rules

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : C \supset A \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} \text{celim } t \rightsquigarrow \text{celim } t^* : A}\ \textbf{e-u-c-implE}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus {t'}^* \lesssim D : C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash \text{celim } t \ominus \text{celim } t' \rightsquigarrow \text{celim } t^* \ominus \text{celim } {t'}^* \lesssim D : \tau}\ \textbf{e-r-c-implE}$$

Figure B.88: Embedding rules of RelCost, part 7.

Primitive application

$$\frac{\Upsilon(\zeta) = A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t \rightsquigarrow t^* : A_1}{\Delta;\Phi_a;\Omega \vdash^{U+U'}_{L+L'} \zeta\ t \rightsquigarrow \zeta\ t^* : A_2}\ \textbf{e-u-primapp}$$

$$\frac{\Upsilon(\zeta) = \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D' : \tau_1}{\Delta;\Phi_a;\Gamma \vdash \zeta\ t \ominus \zeta\ t' \rightsquigarrow \zeta\ t^* \ominus \zeta\ t'^* \lesssim D + D' : \tau_2}\ \textbf{e-r-primapp}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Omega\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow \text{contra}\ t : \tau}\ \textbf{e-u-contra}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Gamma\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow \text{contra}\ t_1 \ominus \text{contra}\ t_2 \lesssim D : \tau}\ \textbf{e-r-contra}$$

Subsumption

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \qquad \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'}{\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t \rightsquigarrow t^* : A'}\ \textbf{e-u-}\sqsubseteq$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau \\ \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \text{coerce}_{\tau,\tau'} \qquad \Delta;\Phi_a \models U \leq U'\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t'\ t_1^* \ominus t'\ t_2^* \lesssim D' : \tau'}\ \textbf{e-r-}\sqsubseteq$$

Constraint-based rules

$$\frac{\Delta;C \wedge \Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A \qquad \Delta;\neg C \wedge \Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^{**} : A \qquad \Delta \vdash C\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow \text{split}\ (t^*, t^{**})\ \text{with}\ C : A}\ \textbf{e-u-split}$$

$$\frac{\begin{array}{c}\Delta;C \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau \\ \Delta;\neg C \wedge \Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^{**} \ominus t_2^{**} \lesssim D : \tau \qquad \Delta \vdash C\ \mathsf{wf}\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow \text{split}\ (t_1^*, t_1^{**})\ \text{with}\ C \ominus \text{split}\ (t_2^*, t_2^{**})\ \text{with}\ C \lesssim D : \tau}\ \textbf{e-r-split}$$

Figure B.89: Embedding rules of RelCost, part 8.

Asynchronous typing rules

$$\frac{\Delta;\Phi_a;|\Gamma|1 \vdash^{U_1}_{L_1} t_1 \rightsquigarrow {t_1}^* : A_1 \qquad \Delta;\Phi_a;x:U(A_1,A_1),\Gamma \vdash t_2 \ominus t \rightsquigarrow {t_2}^* \ominus t^* \lesssim D_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \ominus t \rightsquigarrow \text{let } x = t_1^* \text{ in } t_2^* \ominus t^* \lesssim U_1 + D_2 + c_{let} : \tau_2}$$

e-r-let-e

$$\frac{\Delta;\Phi_a;|\Gamma|2 \vdash^{U_1}_{L_1} t_1 \rightsquigarrow {t_1}^* : A_1 \qquad \Delta;\Phi_a;x:U(A_1,A_1),\Gamma \vdash t \ominus t_2 \rightsquigarrow t^* \ominus {t_2}^* \lesssim D_2 : \tau_2}{\Delta;\Phi_a;\Gamma \vdash t \ominus \text{let } x = t_1 \text{ in } t_2 \rightsquigarrow t^* \ominus \text{let } x = t_1^* \text{ in } t_2^* \lesssim U_2 - L_1 - c_{let} : \tau_2}$$

e-r-e-r-let

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|1 \vdash^{U_1}_{L_1} t_1 \rightsquigarrow {t_1}^* : A_1 \xrightarrow{\text{exec}(L,U)} A_2 \\ \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : U(A_1, A_2')\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_2' \rightsquigarrow t_1^*\ t_2^* \ominus t_2'^* \lesssim U_1 + D_2 + U + c_{app} : U(A_2, A_2')}\ \textbf{e-r-app-e}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|2 \vdash^{U_1}_{L_1} t_1' \rightsquigarrow {t_1'}^* : A_1' \xrightarrow{\text{exec}(L,U)} A_2' \\ \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : U(A_2, A_1')\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_1'\ t_2' \rightsquigarrow t_2^* \ominus t_1'^*\ t_2'^* \lesssim U_2 - L_1 - L - c_{app} : U(A_2, A_2')}\ \textbf{e-r-e-r-app}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|1 \vdash^{U}_{\_} t \rightsquigarrow t^* : A_1 + A_2 \qquad \Delta;\Phi_a;x:U(A_1,A_1),\Gamma \vdash t_1 \ominus t' \rightsquigarrow {t_1}^* \ominus {t'}^* \lesssim D' : \tau \\ \Delta;\Phi_a;y:U(A_2,A_2),\Gamma \vdash t_2 \ominus t' \rightsquigarrow {t_2}^* \ominus {t'}^* \lesssim D' : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{ case } (e, x.t_1, y.t_2) \ominus t' \rightsquigarrow \text{ case } (t^*, x.t_1^*, y.t_2^*) \ominus t'^* \lesssim D' + U + c_{case} : \tau}$$

e-r-case-r-e

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|2 \vdash^{-}_{L'} t' \rightsquigarrow {t'}^* : A_1 + A_2 \qquad \Delta;\Phi_a;x:U(A_1,A_1),\Gamma \vdash t \ominus t_1' \rightsquigarrow t^* \ominus {t_1'}^* \lesssim D : \tau \\ \Delta;\Phi_a;y:U(A_2,A_2),\Gamma \vdash t \ominus t_2' \rightsquigarrow t^* \ominus {t_2'}^* \lesssim D : \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash t \ominus \text{ case } (t', x.t_1', y.t_2') \rightsquigarrow t^* \ominus \text{ case } (t'^*, x.t_1'^*, y.t_2'^*) \lesssim U - L' - c_{case} : \tau}$$

e-r-e-r-case

Figure B.90: Embedding rules of RelCost, part 9.

$\boxed{\Delta;\Phi_a \models \tau_1 \equiv \tau_2}$ checks whether τ_1 is equivalent to τ_2

$$\frac{}{\Delta;\Phi_a \models \text{int} \equiv \text{int}}\ \textbf{eq-int} \qquad \frac{}{\Delta;\Phi_a \models \text{unit} \equiv \text{unit}}\ \textbf{eq-unit}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta;\Phi_a \models \tau_2 \equiv \tau_2' \qquad \Delta;\Phi_a \models t \doteq t'}{\Delta;\Phi_a \models \tau_1 \xrightarrow{\text{diff}(t)} \tau_2 \equiv \tau_1' \xrightarrow{\text{diff}(t')} \tau_2'}\ \textbf{eq-fun}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta;\Phi_a \models \tau_2 \equiv \tau_2'}{\Delta;\Phi_a \models \tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2'}\ \textbf{eq-prod}$$

$$\frac{\Delta;\Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta;\Phi_a \models \tau_2 \equiv \tau_2'}{\Delta;\Phi_a \models \tau_1 + \tau_2 \equiv \tau_1' + \tau_2'}\ \textbf{eq-sum} \qquad \frac{\Delta;\Phi_a \models n \doteq n' \qquad \Delta;\Phi_a \models \alpha \doteq \alpha' \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \text{list}[n]^{\alpha}\,\tau \equiv \text{list}[n']^{\alpha'}\,\tau'}\ \textbf{eq-list}$$

$$\frac{i,\Delta;\Phi_a \models \tau \equiv \tau' \qquad i,\Delta;\Phi_a \models t \doteq t'}{\Delta;\Phi_a \models \forall i \overset{\text{diff}(t)}{::} S.\tau \equiv \forall i \overset{\text{diff}(t')}{::} S.\tau'}\ \textbf{eq-}\forall \qquad \frac{i,\Delta;\Phi_a \models \tau \equiv \tau' \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \exists i{::}S.\tau \equiv \exists i{::}S.\tau'}\ \textbf{eq-}\exists$$

$$\frac{\Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models \Box\tau \equiv \Box\tau'}\ \textbf{eq-B-}\Box \qquad \frac{\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \qquad \Delta;\Phi_a \models^{\mathsf{A}} A_2' \sqsubseteq A_2}{\Delta;\Phi_a \models U(A_1,A_2) \equiv U(A_1',A_2')}\ \textbf{eq-U}$$

$$\frac{\Delta;C \wedge \Phi_a \models C' \qquad \Delta;C' \wedge \Phi_a \models C \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models C \supset \tau \equiv C' \supset \tau'}\ \textbf{eq-c-impl}$$

$$\frac{\Delta;C' \wedge \Phi_a \models C \qquad \Delta;C \wedge \Phi_a \models C' \qquad \Delta;\Phi_a \models \tau \equiv \tau'}{\Delta;\Phi_a \models C\ \&\ \tau \equiv C'\ \&\ \tau'}\ \textbf{eq-c-prod}$$

Figure B.91: Relational type equivalence rules of RelCostCore.

General rules

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 :^c A \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 :^c A}{\Delta;\Phi_a;\Gamma \vdash \text{switch } t_1 \ominus \text{switch } t_2 \lesssim U_1 - L_2 :^c U\ A}\ \textbf{c-switch}$$

$$\frac{\Delta;\Phi_a;\Box\Gamma \vdash t \ominus t \lesssim D :^c \tau}{\Delta;\Phi_a;\Box\Gamma,\Gamma' \vdash \text{NC } t \ominus \text{NC } t \lesssim 0 :^c \Box\tau}\ \textbf{c-nochange}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \Box\tau}{\Delta;\Phi_a;\Gamma \vdash \text{der } t_1 \ominus \text{der } t_2 \lesssim D :^c \tau}\ \textbf{c-der}$$

Constant integers and unit

$$\frac{}{\Delta;\Phi_a;\Omega \vdash^0_0 \mathtt{n} :^c \text{int}}\ \textbf{c-const} \qquad \frac{}{\Delta;\Phi_a;\Gamma \vdash \mathtt{n} \ominus \mathtt{n} \lesssim 0 :^c \text{int}}\ \textbf{c-r-const}$$

$$\frac{}{\Delta;\Phi_a;\Omega \vdash^0_0 () :^c \text{unit}}\ \textbf{c-unit} \qquad \frac{}{\Delta;\Phi_a;\Gamma \vdash () \ominus () \lesssim 0 :^c \text{unit}}\ \textbf{c-r-unit}$$

Variables x

$$\frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash^0_0 x :^c A}\ \textbf{c-var} \qquad \frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \ominus x \lesssim 0 :^c \tau}\ \textbf{c-r-var}$$

inl t

$$\frac{\Delta;\Phi_a;\Omega \vdash^U_L t :^c A_1 \qquad \Delta;\Phi_a \vdash^{\mathsf{A}} A_2\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^U_L \text{inl } t :^c A_1 + A_2}\ \textbf{c-inl}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \tau_1 \qquad \Delta;\Phi_a \vdash \tau_2\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{inl } t \ominus \text{inl } t' \lesssim D :^c \tau_1 + \tau_2}\ \textbf{c-r-inl}$$

inr t

$$\frac{\Delta;\Phi_a;\Omega \vdash^U_L t :^c A_2 \qquad \Delta;\Phi_a \vdash^{\mathsf{A}} A_1\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^U_L \text{inr } t :^c A_1 + A_2}\ \textbf{c-inr}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \tau_2 \qquad \Delta;\Phi_a \vdash \tau_1\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{inr } t \ominus \text{inr } t' \lesssim D :^c \tau_1 + \tau_2}\ \textbf{c-r-inr}$$

Figure B.92: Typing rules of RelCostCore, part 1.

$\boxed{\text{case } (t, x.t_1, y.t_2)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^c A_1 + A_2 \qquad \Delta;\Phi_a;x:A_1,\Omega \vdash^{U'}_{L'} t_1 :^c A \qquad \Delta;\Phi_a;y:A_2,\Omega \vdash^{U'}_{L'} t_2 :^c A}{\Delta;\Phi_a;\Omega \vdash^{U+U'+c_{case}}_{L+L'+c_{case}} \text{case } (t, x.t_1, y.t_2) :^c A}\ \textbf{c-case}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \tau_1 + \tau_2 \qquad \Delta;\Phi_a;x:\tau_1,\Gamma \vdash t_1 \ominus t_1' \lesssim D' :^c \tau \qquad \Delta;\Phi_a;y:\tau_2,\Gamma \vdash t_2 \ominus t_2' \lesssim D' :^c \tau}{\Delta;\Phi_a;\Gamma \vdash \text{case } (t, x.t_1, y.t_2) \ominus \text{case } (t', x.t_1', y.t_2') \lesssim D + D' :^c \tau}\ \textbf{c-r-case}$$

$\boxed{\text{fix } f(x).t}$

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A_1 \xrightarrow{\text{exec}(L,U)} A_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:A_1,f:A_1 \xrightarrow{\text{exec}(L,U)} A_2,\Omega \vdash^{U}_{L} t :^c A_2}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \text{fix } f(x).t :^c A_1 \xrightarrow{\text{exec}(L,U)} A_2}\ \textbf{c-fix}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\text{diff}(D)} \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1,f:\tau_1 \xrightarrow{\text{diff}(D)} \tau_2,\Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{fix } f(x).t_1 \ominus \text{fix } f(x).t_2 \lesssim 0 :^c \tau_1 \xrightarrow{\text{diff}(D)} \tau_2}\ \textbf{c-r-fix}$$

$$\frac{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\text{diff}(D)} \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x:\tau_1,f:\Box(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2),\Box\Gamma \vdash t \ominus t \lesssim D :^c \tau_2}{\Delta;\Phi_a;\Box\Gamma \vdash \text{fix}_{NC}\ f(x).t \ominus \text{fix}_{NC}\ f(x).t \lesssim 0 :^c \Box(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2)}\ \textbf{c-r-fixNC}$$

$\boxed{t_1\ t_2}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 :^c A_1}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+U+c_{app}}_{L_1+L_2+L+c_{app}} t_1\ t_2 :^c A_2}\ \textbf{c-app}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau_1}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \lesssim t_1 + t_2 + t :^c \tau_2}\ \textbf{c-r-app}$$

Figure B.93: Typing rules of RelCostCore, part 2.

$\boxed{\langle t_1, t_2\rangle}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c A_1 \qquad \Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 :^c A_2}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \langle t_1, t_2\rangle :^c A_1 \times A_2}\ \textbf{c-prod}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \tau_1 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \langle t_1, t_2\rangle \ominus \langle t_1', t_2'\rangle \lesssim D_1 + D_2 :^c \tau_1 \times \tau_2}\ \textbf{c-r-prod}$$

$\boxed{\pi_1(t)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^c A_1 \times A_2}{\Delta;\Phi_a;\Omega \vdash^{U+c_{proj}}_{L+c_{proj}} \pi_1(e) :^c A_1}\ \textbf{c-proj1} \qquad \frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \tau_1 \times \tau_2}{\Delta;\Phi_a;\Gamma \vdash \pi_1(e) \ominus \pi_1(t') \lesssim D :^c \tau_1}\ \textbf{c-r-proj1}$$

$\boxed{\pi_2(t)}$

Symmetric rules.

$\boxed{\text{nil}}$

$$\frac{\Delta;\Phi_a \vdash^{\mathsf{A}} A\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \text{nil} :^c \text{list}[0]\, A}\ \textbf{c-nil} \qquad \frac{\Delta;\Phi_a \vdash \tau\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \text{nil} \ominus \text{nil} \lesssim 0 :^c \text{list}[0]^{\alpha}\, \tau}\ \textbf{c-r-nil}$$

$\boxed{\text{cons}(t_1, t_2)}$

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c A \qquad \Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 :^c \text{list}[n]\, A}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \text{cons}_C(t_1, t_2) :^c \text{list}[n+1]\, A}\ \textbf{c-cons}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \text{list}[n]^{\alpha}\, \tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}_C(t_1, t_2) \ominus \text{cons}_C(t_1', t_2') \lesssim D_1 + D_2 :^c \text{list}[n+1]^{\alpha+1}\, \tau}\ \textbf{c-r-cons1}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \text{list}[n]^{\alpha}\, \tau}{\Delta;\Phi_a;\Gamma \vdash \text{cons}_{NC}(t_1, t_2) \ominus \text{cons}_{NC}(t_1', t_2') \lesssim D_1 + D_2 :^c \text{list}[n+1]^{\alpha}\, \tau}\ \textbf{c-r-cons2}$$

Figure B.94: Typing rules of RelCostCore, part 3.

$\boxed{\text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2}$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash_L^U t :^c \text{list}[n]\, A \\ \Delta;\Phi_a \wedge n = 0;\Omega \vdash_{L'}^{U'} t_1 :^c A' \qquad i,\Delta;\Phi_a \wedge n = i+1;h : A, tl : \text{list}[i]\, A,\Omega \vdash_{L'}^{U'} t_2 :^c A'\end{array}}{\Delta;\Phi_a;\Omega \vdash_{L+L'+c_{caseL}}^{U+U'+c_{caseL}} \text{case } t \text{ of nil } \to t_1 \mid h ::_{NC} tl \to t_2 \mid h ::_C tl \to t_3 :^c A'}\ \textbf{c-caseL}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \text{list}[n]^\alpha\, \tau \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \ominus t_1' \lesssim D' :^c \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h : \square\tau, tl : \text{list}[i]^\alpha\, \tau,\Gamma \vdash t_2 \ominus t_2' \lesssim D' :^c \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h' : \tau, tl : \text{list}[i]^\beta\, \tau,\Gamma \vdash t_3 \ominus t_3' \lesssim D' :^c \tau'\end{array}}{\begin{array}{lll}\text{case } t \text{ of nil } \to t_1 & & \text{case } t \text{ of nil } \to t_1' \\ \Delta;\Phi_a;\Gamma \vdash \mid h ::_N tl \to t_2 & \ominus & \mid h ::_N tl \to t_2' \qquad \lesssim D + D' :^c \tau' \\ \quad \mid h' ::_C tl' \to t_3 & & \quad \mid h' ::_C tl' \to t_3'\end{array}}\ \textbf{c-r-caseL}$$

$\boxed{\Lambda t}$

$$\frac{i :: S,\Delta;\Phi_a;\Omega \vdash_L^U t :^c A \qquad i \notin \text{FIV}(\Phi_a;\Omega)}{\Delta;\Phi_a;\Omega \vdash_0^0 \Lambda i.t :^c \forall i \overset{\text{exec}(L,U)}{::} S.\, A}\ \textbf{c-iLam}$$

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \tau \qquad i \notin \text{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda i.t \ominus \Lambda i.t' \lesssim 0 :^c \forall i \overset{\text{diff}(D)}{::} S.\, \tau}\ \textbf{c-r-iLam}$$

$\boxed{t[\,]}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t :^c \forall i \overset{\text{exec}(L',U')}{::} S.\, A \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Omega \vdash_{L+L'[I/i]}^{U+U'[I/i]} t[I] :^c A\{I/i\}}\ \textbf{c-iApp}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \forall i \overset{\text{diff}(D')}{::} S.\, \tau \qquad \Delta \vdash I : S}{\Delta;\Phi_a;\Gamma \vdash t[I] \ominus t'[I] \lesssim D + D'[I/i] :^c \tau\{I/i\}}\ \textbf{c-r-iApp}$$

$\boxed{\text{pack } t}$

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t :^c A\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Omega \vdash_L^U \text{pack } t \text{ with } I :^c \exists i{::}S.\, A}\ \textbf{c-pack}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta;\Phi_a;\Gamma \vdash \text{pack } t \text{ with } I \ominus \text{pack } t' \text{ with } I \lesssim D :^c \exists i{::}S.\, \tau}\ \textbf{c-r-pack}$$

Figure B.95: Typing rules of RelCostCore, part 4.

unpack t as x in t'

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c \exists i{::}S.\, A_1 \qquad i :: S, \Delta;\Phi_a; x : A_1, \Omega \vdash^{U_2}_{L_2} t_2 :^c A_2 \qquad i \notin FV(\Phi_a;\Gamma, A_2, L_2, U_2)}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2 :^c A_2}\ \textbf{c-unpack}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \exists i{::}S.\, \tau_1 \qquad i :: S, \Delta;\Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau_2 \qquad i \notin FV(\Phi_a;\Gamma, \tau_2, D_2)}{\Delta;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2 \ominus \text{unpack } t_1' \text{ as } (x,i) \text{ in } t_2' \lesssim D_1 + D_2 :^c \tau_2}\ \textbf{c-r-unpack1}$$

Primitive application

$$\frac{\Upsilon(\zeta) = A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t :^c A_1}{\Delta;\Phi_a;\Omega \vdash^{U+U'+c_{app}}_{L+L'+c_{app}} \zeta\ t :^c A_2}\ \textbf{c-primapp}$$

$$\frac{\Upsilon(\zeta) = \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D' :^c \tau_1}{\Delta;\Phi_a;\Gamma \vdash \zeta\ t \ominus \zeta\ t' \lesssim D + D' :^c \tau_2}\ \textbf{c-r-primapp}$$

$C \,\&\, \tau$ intro. rules

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Omega \vdash^{U}_{L} t :^c A}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^c C \,\&\, A}\ \textbf{c-c-andI}$$

$$\frac{\Delta;\Phi_a \models C \qquad \Delta;\Phi_a \wedge C;\Gamma \vdash t \ominus t' \lesssim D :^c \tau}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c C \,\&\, \tau}\ \textbf{c-c-andI}$$

$C \,\&\, \tau$ elim. rules

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c C \,\&\, A_1 \qquad \Delta;\Phi_a \wedge C; x : A_1, \Omega \vdash^{U_2}_{L_2} t_2 :^c A_2}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \text{clet } t_1 \text{ as } x \text{ in } t_2 :^c A_2}\ \textbf{c-c-andE}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c C \,\&\, \tau_1 \qquad \Delta;\Phi_a \wedge C; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \ominus \text{clet } t_1' \text{ as } x \text{ in } t_2' \lesssim D_1 + D_2 :^c \tau_2}\ \textbf{c-r-c-andE}$$

Figure B.96: Typing rules of RelCostCore, part 5.

$C \supset \tau$ intro. rules

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash^{U}_{L} t :^{c} A}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^{c} C \supset A}\ \textbf{c-c-implI} \qquad \frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t \ominus t' \lesssim D :^{c} \tau}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^{c} C \supset \tau}\ \textbf{c-r-c-implI}$$

$C \supset \tau$ elim. rules

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^{c} C \supset A \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Omega \vdash^{U}_{L} \text{celim } t :^{c} A}\ \textbf{c-c-implE}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^{c} C \supset \tau \qquad \Delta;\Phi_a \models C}{\Delta;\Phi_a;\Gamma \vdash \text{celim } t \ominus \text{celim } t' \lesssim D :^{c} \tau}\ \textbf{c-r-c-implE}$$

let $x = t_1$ in t_2

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^{c} A_1 \qquad \Delta;\Phi_a;x : A_1,\Omega \vdash^{U_2}_{L_2} t_2 :^{c} A_2}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2+c_{let}}_{L_1+L_2+c_{let}} \text{let } x = t_1 \text{ in } t_2 :^{c} A_2}\ \textbf{c-let}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^{c} \tau_1 \qquad \Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^{c} \tau_2}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \ominus \text{let } x = t_1' \text{ in } t_2' \lesssim D_1 + D_2 :^{c} \tau_2}\ \textbf{c-r-let1}$$

Unary Subtyping

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^{c} A \qquad \Delta;\Phi_a \models A \sqsubseteq A' \\ \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models t \leq U'\end{array}}{\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t :^{c} A'}\ \textbf{c-}\sqsubseteq$$

Binary Subeffecting

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^{c} \tau \qquad \Delta;\Phi_a \models \tau \equiv \tau' \\ \Delta;\Phi_a \models t \leq U'\end{array}}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D' :^{c} \tau'}\ \textbf{c-r-}\equiv$$

Figure B.97: Typing rules of RelCostCore, part 6.

Constraint dependent typing

$$\frac{\Delta;\Phi_a \wedge C;\Omega \vdash^U_L t_1 :^c A \qquad \Delta;\Phi_a \wedge \neg C;\Omega \vdash^U_L t_2 :^c A}{\Delta;\Phi_a;\Omega \vdash^U_L \text{split } (t_1,t_2) \text{ with } C :^c A} \textbf{ c-split}$$

$$\frac{\Delta;\Phi_a \wedge C;\Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \tau \qquad \Delta;\Phi_a \wedge \neg C;\Gamma \vdash t_1' \ominus t_2' \lesssim D :^c \tau}{\Delta;\Phi_a;\Gamma \vdash \text{split } (t_1,t_1') \text{ with } C \ominus \text{split } (t_2,t_2') \text{ with } C \lesssim D :^c \tau} \textbf{ c-r-split}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Omega \text{ wf}}{\Delta;\Phi_a;\Omega \vdash^U_L \text{contra } t :^c A} \textbf{ c-contra}$$

$$\frac{\Delta;\Phi_a \models \bot \qquad \Delta;\Phi_a \vdash \Gamma \text{ wf}}{\Delta;\Phi_a;\Gamma \vdash \text{contra } t_1 \ominus \text{contra } t_2 \lesssim D :^c \tau} \textbf{ c-r-contra}$$

Asynchronous typing

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 :^c A_1 \\ \Delta;\Phi_a;x : U(A_1,A_1),\Gamma \vdash t_2 \ominus t \lesssim D_2 :^c \tau_2\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \ominus t \lesssim U_1 + D_2 + c_{let} :^c \tau_2} \textbf{ c-r-let-t}$$

$$\frac{\Delta;\Phi_a;|\Gamma|2 \vdash^{U_1}_{L_1} t_1 :^c A_1 \qquad \Delta;\Phi_a;x : U(A_1,A_1),\Gamma \vdash t \ominus t_2 \lesssim D_2 :^c \tau_2}{\Delta;\Phi_a;\Gamma \vdash t \ominus \text{let } x = t_1 \text{ in } t_2 \lesssim D_2 - L_1 - c_{let} :^c \tau_2} \textbf{ c-r-t-let}$$

$$\frac{\Delta;\Phi_a;|\Gamma|1 \vdash^{U_1}_{L_1} t_1 :^c A_1 \xrightarrow{\text{exec}(L,U)} A_2 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c U(A_1,A_2')}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_2' \lesssim U_1 + D_2 + U + c_{app} :^c U(A_2,A_2')} \textbf{ c-r-app-t}$$

$$\frac{\Delta;\Phi_a;|\Gamma|2 \vdash^{U_1}_{L_1} t_1' :^c A_1' \xrightarrow{\text{exec}(L,U)} A_2' \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c U(A_2,A_1')}{\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_1'\ t_2' \lesssim D_2 - L_1 - L - c_{app} :^c U(A_2,A_2')} \textbf{ c-r-t-app}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|1 \vdash^U_- t :^c A_1 + A_2 \qquad \Delta;\Phi_a;x : U(A_1,A_1),\Gamma \vdash t_1 \ominus t' \lesssim D' :^c \tau \\ \Delta;\Phi_a;y : U(A_2,A_2),\Gamma \vdash t_2 \ominus t' \lesssim D' :^c \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash \text{ case } (e,x.t_1,y.t_2) \ominus t' \lesssim D' + t + c_{case} :^c \tau} \textbf{ c-r-case-t}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;|\Gamma|2 \vdash^-_{L'} t' :^c A_1 + A_2 \qquad \Delta;\Phi_a;x : U(A_1,A_1),\Gamma \vdash t \ominus t_1' \lesssim D :^c \tau \\ \Delta;\Phi_a;y : U(A_2,A_2),\Gamma \vdash t \ominus t_2' \lesssim D :^c \tau\end{array}}{\Delta;\Phi_a;\Gamma \vdash t \ominus \text{ case } (t',x.t_1',y.t_2') \lesssim D - L' - c_{case} :^c \tau} \textbf{ c-r-t-case}$$

Figure B.98: Typing rules of RelCostCore, part 7.

$\boxed{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \downarrow \tau, D \Rightarrow \Phi}$ Under the existential variable context ψ_a and the assumption Φ_a, $t_1 \ominus t_2$ checks against the input type τ and the difference cost D. Finally, it generates the constraint Φ.

$\boxed{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \uparrow \tau \Rightarrow [\psi], D, \Phi}$ Under the existential variable context ψ_a and the assumption Φ_a, $t_1 \ominus t_2$ synthesizes the output type τ and the relative cost D where all the newly generated existential variables are defined in ψ. Finally, it generates the constraint Φ.

$\boxed{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, L, U \Rightarrow \Phi}$ Under the existential variable context ψ_a and the assumption Φ_a, t checks against the unary input type A and the minimum execution cost L and maximum execution cost U. Finally, it generates the constraint Φ.

$\boxed{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A \Rightarrow [\psi], L, U, \Phi}$ Under the existential variable context ψ_a and the assumption Φ_a, t synthesizes the unary output type A, the minimum execution cost L and maximum execution cost U, where all the newly generated existential variables are defined in ψ. Finally, it generates the constraint Φ.

switch t

$$\frac{\Delta;\psi_a;\Phi;|\Gamma| \vdash t_1 \uparrow A \Rightarrow [\psi_1], \_, U_1, \Phi_1 \qquad \Delta;\psi_a;\Phi;|\Gamma| \vdash t_2 \uparrow A \Rightarrow [\psi_2], L_2, \_, \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{switch } t_1 \ominus \text{switch } t_2 \uparrow U\ A \Rightarrow [\psi_1, \psi_2], U_1 - L_2, \Phi_1 \wedge \Phi_2}$$

alg-r-switch↑

$$\frac{\begin{array}{c} L_1, U_1, L_2, U_2 \in \text{fresh}(\mathbb{R}) \\ \Delta; L_1, U_1, \psi_a; \Phi; |\Gamma| \vdash t_1 \downarrow A, L_1, U_1 \Rightarrow \Phi_1 \qquad \Delta; L_2, U_2, \psi_a; \Phi; |\Gamma| \vdash t_2 \downarrow A, L_2, U_2 \Rightarrow \Phi_2 \end{array}}{\begin{array}{c} \Delta;\psi_a;\Phi_a;\Gamma \\ \vdash \text{switch } t_1 \ominus \text{switch } t_2 \downarrow D, U\ A \Rightarrow \exists L_1, U_1 :: \mathbb{R}.(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi_2 \wedge U_1 - L_2 \doteq D) \end{array}}$$

alg-r-switch↓

NC t

$$\frac{D' \in \text{fresh}(\mathbb{R}) \qquad \Delta; D', \psi_a; \Phi_a; \square\Gamma \vdash t \ominus t \downarrow \tau, D' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma', \square\Gamma \vdash \text{NC } t \ominus \text{NC } t \downarrow \square\tau, D \Rightarrow 0 \doteq D \wedge (\exists D' :: \mathbb{R}.\Phi)}\ \textbf{alg-r-nochange-}\downarrow$$

der t

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \uparrow \square\tau \Rightarrow [\psi], D, \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{der } t_1 \ominus \text{der } t_2 \uparrow \tau \Rightarrow [\psi], D, \Phi}\ \textbf{alg-r-der-}\uparrow$$

Figure B.99: Algorithmic typing rules of RelCost, part 1.

Constant Integers $\mathtt{n}$ and unit

$$\frac{}{\Delta;\psi_a;\Phi_a;\Omega \vdash \mathtt{n} \uparrow \mathsf{int} \Rightarrow [.], 0, 0, \top}\ \textbf{alg-u-n-}\uparrow$$

$$\frac{}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathtt{n} \ominus \mathtt{n} \uparrow \mathsf{int} \Rightarrow [.], 0, \top}\ \textbf{alg-r-n-}\uparrow$$

$$\frac{}{\Delta;\psi_a;\Phi_a;\Omega \vdash () \uparrow \mathsf{unit} \Rightarrow [.], 0, 0, \top}\ \textbf{alg-u-unit-}\uparrow$$

$$\frac{}{\Delta;\psi_a;\Phi_a;\Gamma \vdash () \ominus \mathtt{n} \uparrow \mathsf{unit} \Rightarrow [.], 0, \top}\ \textbf{alg-r-unit-}\uparrow$$

Variables x

$$\frac{\Omega(x) = A}{\Delta;\psi;\Phi_a;\Omega \vdash x \uparrow A \Rightarrow [.], 0, 0, \top}\ \textbf{alg-u-var-}\uparrow$$

$$\frac{\Gamma(x) = \tau}{\Delta;\psi_a;\Phi_a;\Gamma \vdash x \ominus x \uparrow \tau \Rightarrow [.], 0, \top}\ \textbf{alg-r-var-}\uparrow$$

inl t

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A_1, L, U \Rightarrow \Phi \qquad \Delta, \psi_a;\Phi_a \vdash^{\mathsf{A}} A_2\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \mathrm{inl}\ t \downarrow A_1 + A_2, L, U \Rightarrow \Phi}\ \textbf{alg-u-inl-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \downarrow \tau_1, D \Rightarrow \Phi \qquad \Delta, \psi_a;\Phi_a \vdash \tau_2\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{inl}\ t_1 \ominus \mathrm{inl}\ t_2 \downarrow \tau_1 + \tau_2, D \Rightarrow \Phi}\ \textbf{alg-r-inl-}\downarrow$$

inr t

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A_2, L, U \Rightarrow \Phi \qquad \Delta, \psi_a;\Phi_a \vdash^{\mathsf{A}} A_1\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \mathrm{inr}\ t \downarrow A_1 + A_2, L, U \Rightarrow \Phi}\ \textbf{alg-u-inr-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \downarrow \tau_2, D \Rightarrow \Phi \qquad \Delta, \psi_a;\Phi_a \vdash \tau_1\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{inr}\ t_1 \ominus \mathrm{inr}\ t_2 \downarrow \tau_1 + \tau_2, D \Rightarrow \Phi}\ \textbf{alg-r-inr-}\downarrow$$

Figure B.100: Algorithmic typing rules of RelCost, part 2.

$\boxed{\text{case } (t, x.t_1, y.t_2)}$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A_1 + A_2 \Rightarrow [\psi], L_e, U_e, \Phi_1 \\ L', U' \in \text{fresh}(\mathbb{R}) \qquad \Delta; L', U', \psi, \psi_a; \Phi_a; \Omega, x : A_1 \vdash t_1 \downarrow A, L', U' \Rightarrow \Phi_2 \\ \Delta; L', U', \psi, \psi_a; \Phi_a; \Omega, y : A_2 \vdash t_2 \downarrow A, L', U' \Rightarrow \Phi_3 \\ \Phi = \exists(\psi).\Phi_1 \wedge (\exists L', U' :: \mathbb{R}.\Phi_2 \wedge \Phi_3 \wedge L \doteq L' + L_e + c_{case} \wedge U' + U_e + c_{case} \doteq U)\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{ case } (e, x.t_1, y.t_2) \downarrow A, L, U \Rightarrow \Phi}$$

alg-u-case-↓

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau_1 + \tau_2 \Rightarrow [\psi], t_e, \Phi_1 \\ D' \in \text{fresh}(\mathbb{R}) \qquad \Delta; D', \psi, \psi_a; \Phi_a; \Gamma, x : \tau_1 \vdash t_1 \ominus t_1' \downarrow \tau, D' \Rightarrow \Phi_2 \\ \Delta; D', \psi, \psi_a; \Phi_a; \Gamma, y : \tau_2 \vdash t_2 \ominus t_2' \downarrow \tau, D' \Rightarrow \Phi_3 \\ \Phi = \exists(\psi).\Phi_1 \wedge (\exists D' :: \mathbb{R}.\Phi_2 \wedge \Phi_3 \wedge (D' + D_e \doteq D))\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{ case } (e, x.t_1, y.t_2) \ominus \text{ case } (e', x.t_1', y.t_2') \downarrow \tau, D \Rightarrow \Phi}\ \textbf{alg-r-case-↓}$$

$\boxed{\text{fix } f(x).t}$

$$\frac{\Delta;\psi_a;\Phi_a; f : A_1 \xrightarrow{\text{exec}(L',U')} A_2, x : A_1, \Omega \vdash t \downarrow A_2, L', U' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{fix } f(x).t \downarrow A_1 \xrightarrow{\text{exec}(L',U')} A_2, L, U \Rightarrow \Phi \wedge L \doteq 0 \wedge 0 \doteq U}\ \textbf{alg-u-fix-↓}$$

$$\frac{\Delta;\psi_a;\Phi_a; f : \tau_1 \xrightarrow{\text{diff}(D')} \tau_2, x : \tau_1, \Gamma \vdash t \ominus t' \downarrow \tau_2, D' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{fix } f(x).t \ominus \text{fix } f(x).t' \downarrow \tau_1 \xrightarrow{\text{diff}(D')} \tau_2, D \Rightarrow \Phi \wedge 0 \doteq D}\ \textbf{alg-r-fix-↓}$$

$$\frac{\Delta;\psi_a;\Phi_a; f : \Box(\tau_1 \xrightarrow{\text{diff}(D')} \tau_2), x : \tau_1, \Box\Gamma \vdash t \ominus t' \downarrow \tau_2, D' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Gamma', \Box\Gamma \vdash \text{fix}_{NC}\ f(x).t \ominus \text{fix}_{NC}\ f(x).t \downarrow \Box(\tau_1 \xrightarrow{\text{diff}(D')} \tau_2), D \Rightarrow \Phi \wedge 0 \doteq D}$$

alg-r-fix-↓□

Figure B.101: Algorithmic typing rules of RelCost, part 3.

$\boxed{t_1\ t_2}$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow A_1 \xrightarrow{\text{exec}(L_e,U_e)} A_2 \Rightarrow [\psi], L_1, U_1, \Phi_1 \qquad L_2, U_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; L_2, U_2, \psi, \psi_a; \Phi_a; \Omega \vdash t_2 \downarrow A_1, L_2, U_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1\ t_2 \uparrow A_2 \Rightarrow [L_2, L_2, \psi], L_1 + L_2 + L_e + c_{app}, U_1 + U_2 + U_e + c_{app}, \Phi_1 \wedge \Phi_2}$$

alg-u-app-↑

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \tau_1 \xrightarrow{\text{diff}(D_e)} \tau_2 \Rightarrow [\psi], D_1, \Phi_1 \qquad D_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; D_2, \psi, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1, D_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \uparrow \tau_2 \Rightarrow [D_2, \psi], D_1 + D_2 + D_e, \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-app-}\uparrow$$

$\boxed{(t_1, t_2)}$

$$\frac{L_1, U_1, L_2, U_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; L_1, U_1, \psi_a; \Phi_a; \Omega \vdash t_1 \downarrow A_1, L_1, U_1 \Rightarrow \Phi_1 \qquad \Delta; L_2, U_2, \psi_a; \Phi_a; \Omega \vdash t_2 \downarrow A_1, L_2, U_2 \Rightarrow \Phi_2}{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \\ \vdash \langle t_1, t_2\rangle \downarrow A_1 \times A_2, L, U \Rightarrow \exists L_1, U_1 :: \mathbb{R}.\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi_2 \wedge U_1 + U_2 \doteq U \wedge L \doteq L_1 + L_2\end{array}}$$

alg-u-prod-↓

$$\frac{D_1, D_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; D_1, \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1, D_1 \Rightarrow \Phi_1 \qquad \Delta; D_1, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1, D_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash \langle t_1, t_2\rangle \ominus \langle t_1', t_2'\rangle \downarrow \tau_1 \times \tau_2, D \Rightarrow \exists D_1 :: \mathbb{R}.\Phi_1 \wedge \exists D_1 :: \mathbb{R}.\Phi_2 \wedge D_1 + D_2 \doteq D}$$

alg-r-prod-↓

$\boxed{\pi_1(t)}$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A_1 \times A_2 \Rightarrow [\psi], L, U, \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \pi_1(t) \uparrow A_1 \Rightarrow [\psi], L, U, \Phi}\ \textbf{alg-u-proj1-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau_1 \times \tau_2 \Rightarrow [\psi], D, \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \pi_1(t) \ominus \pi_1(t') \uparrow \tau_1 \Rightarrow [\psi], D, \Phi}\ \textbf{alg-r-proj1-}\uparrow$$

Figure B.102: Algorithmic typing rules of RelCost, part 4.

$\boxed{\text{cons}(t_1, t_2)}$

$$\frac{\begin{array}{c} L_1, U_1, L_2, U_2 \in \text{fresh}(\mathbb{R}) \qquad i \in \text{fresh}(\mathbb{N}) \qquad \Delta; L_1, U_1, \psi_a; \Phi_a; \Omega \vdash t_1 \downarrow A, L_1, U_1 \Rightarrow \Phi_1 \\ \Delta; i, L_2, U_2, \psi_a; \Phi_a; \Omega \vdash t_2 \downarrow \text{list}[i]\, A, L_2, U_2 \Rightarrow \Phi_2 \\ \Phi_2' = (\Phi_2 \wedge n \doteq (i+1) \wedge L \doteq L_1 + L_2 \wedge U_1 + U_2 \doteq U) \end{array}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \text{cons}_C(t_1, t_2) \downarrow \text{list}[n]\, A, L, U \Rightarrow \exists L_1, U_1 :: \mathbb{R}.(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2')}$$

alg-u-cons-↓

$$\frac{\begin{array}{c} D_1, D_2 \in \text{fresh}(\mathbb{R}) \qquad i, \beta \in \text{fresh}(\mathbb{N}) \qquad \Delta; D_1, \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \tau, D_1 \Rightarrow \Phi_1 \\ \Delta; i, \beta, D_2, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow \text{list}[i]^{\beta}\, \tau, D_2 \Rightarrow \Phi_2 \\ \Phi_2' = n \doteq (i+1) \wedge \exists \beta :: \mathbb{N}.\Phi_2 \wedge \alpha \doteq \beta + 1 \wedge D_1 + D_2 \doteq D \end{array}}{\begin{array}{c} \Delta; \psi_a; \Phi_a; \Gamma \\ \vdash \text{cons}_C(t_1, t_2) \ominus \text{cons}_C(t_1', t_2') \downarrow \text{list}[n]^{\alpha}\, \tau, D \Rightarrow \exists D_1 :: \mathbb{R}.(\Phi_1 \wedge \exists D_1, D_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2') \end{array}}$$

alg-r-consC-↓

$$\frac{\begin{array}{c} D_1, D_2 \in \text{fresh}(\mathbb{R}) \qquad i \in \text{fresh}(\mathbb{N}) \qquad \Delta; D_1, \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \square\tau, D_1 \Rightarrow \Phi_1 \\ \Delta; i, D_2, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow \text{list}[i]^{\alpha}\, \tau, D_2 \Rightarrow \Phi_2 \qquad \Phi_2' = \Phi_2 \wedge n \doteq (i+1) \wedge D_1 + D_2 \doteq D \end{array}}{\begin{array}{c} \Delta; \psi_a; \Phi_a; \Gamma \\ \vdash \text{cons}_{NC}(t_1, t_2) \ominus \text{cons}_{NC}(t_1', t_2') \downarrow \text{list}[n]^{\alpha}\, \tau, D \Rightarrow \exists D_1 :: \mathbb{R}.(\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2') \end{array}}$$

alg-r-consNC-↓

$\boxed{\Lambda i.t}$

$$\frac{i :: S, \Delta; \psi_a; \Phi_a; \Omega \vdash t \downarrow A, L_e, U_e \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash \Lambda i.t \downarrow \forall i \overset{\text{exec}(L_e, U_e)}{::} S.\, A, L, U \Rightarrow (\forall i :: S.\Phi) \wedge L \doteq 0 \wedge 0 \doteq U}\ \textbf{alg-u-iLam-}\downarrow$$

$$\frac{i :: S, \Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \downarrow \tau, t_e \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \Lambda i.t \ominus \Lambda i.t' \downarrow \forall i \overset{\text{diff}(D_e)}{::} S.\, \tau, D \Rightarrow (\forall i :: S.\Phi) \wedge 0 \doteq D}\ \textbf{alg-r-iLam-}\downarrow$$

Figure B.103: Algorithmic typing rules of RelCost, part 5.

$\boxed{\text{case } t \text{ of nil } \to t_1 \mid h :: tl \to t_2}$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow \text{list}[n]\, A \Rightarrow [\psi], L_1, U_1, \Phi_1 \\ L_2, U_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; L_2, U_2, \psi, \psi_a; n \doteq 0 \wedge \Phi_a; \Omega \vdash t_1 \downarrow A', L_2, U_2 \Rightarrow \Phi_2 \qquad i \in \text{fresh}(\mathbb{N}) \\ i :: \mathbb{N}, \Delta; L_2, U_2, \psi, \psi_a; n \doteq i+1 \wedge \Phi_a; h : A, tl : \text{list}[i]\, A, \Omega \vdash t_2 \downarrow A', L_2, U_2 \Rightarrow \Phi_3 \\ \Phi_s = L \doteq (L_1 + L_2 + c_{caseL}) \wedge (U_1 + U_2 + c_{caseL}) \doteq U. \\ \Phi_{body} = (n \doteq 0 \to \Phi_2) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to \Phi_3) \wedge \Phi_s\end{array}}{\begin{array}{c}\text{case } t \text{ of nil } \to t_1 \\ \Delta;\psi_a;\Phi_a;\Omega \vdash \mid h ::_{NC} tl \to t_2 \qquad \downarrow A', L, U \Rightarrow \exists(\psi).(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi_{body}) \\ \mid h ::_{C} tl \ \to t_3\end{array}}$$

alg-u-caseL-↓

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \text{list}[n]^{\alpha}\, \tau \Rightarrow [\psi], D_1, \Phi_e \\ D_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; D_2, \psi, \psi_a; n \doteq 0 \wedge \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \tau', D_2 \Rightarrow \Phi_1 \\ i :: \mathbb{N}, \Delta; D_2, \psi, \psi_a; n \doteq i+1 \wedge \Phi_a; h : \Box\tau, tl : \text{list}[i]^{\alpha}\, \tau, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau', D_2 \Rightarrow \Phi_2 \\ i :: \mathbb{N}, \beta :: \mathbb{N}, \Delta; D_2, \psi, \psi_a; n \doteq i+1 \wedge \alpha \doteq \beta + 1 \wedge \Phi_a; h : \tau, tl : \text{list}[i]^{\beta}\, \tau, \Gamma \\ \vdash t_3 \ominus t_3' \downarrow \tau', D_2 \Rightarrow \Phi_3 \qquad \Phi_{body} = \\ (n \doteq 0 \to \Phi_1) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to (\Phi_2 \wedge \forall \beta :: \mathbb{N}.(\alpha \doteq \beta + 1) \to \Phi_3)) \wedge D_1 + D_2 \doteq D\end{array}}{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \\ \text{case } t \text{ of nil } \to t_1 \qquad \text{case } t' \text{ of nil } \to t_1' \\ \vdash \mid h ::_{NC} tl \to t_2 \quad \ominus \quad \mid h ::_{NC} tl \to t_2' \qquad \downarrow \tau', D \Rightarrow \exists(\psi).(\Phi_e \wedge \exists D_2 :: \mathbb{R}.\Phi_{body}) \\ \mid h ::_{C} tl \ \to t_3 \qquad\quad \mid h ::_{C} tl \ \to t_3'\end{array}}$$

alg-r-caseL-↓

$\boxed{t\,[I]}$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow \forall i \overset{\text{exec}(L_e,U_e)}{::} S.\, A' \Rightarrow [\psi], L, U, \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Omega \vdash t\,[I] \uparrow A'\{I/i\} \Rightarrow [\psi], L + L_e[I/i], U + U_e[I/i], \Phi}\ \textbf{alg-u-iApp-}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \forall i \overset{\text{diff}(D_e)}{::} S.\, \tau' \Rightarrow [\psi], D, \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t\,[I] \ominus t'\,[I] \uparrow \tau'\{I/i\} \Rightarrow [\psi], D + D_e[I/i], \Phi}\ \textbf{alg-r-iApp-}\uparrow$$

Figure B.104: Algorithmic typing rules of RelCost, part 6.

pack t with I

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A\{I/i\}, L, U \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{pack } t \text{ with } I \downarrow \exists i{::}S.\, A, L, U \Rightarrow \Phi}\ \textbf{alg-u-pack-}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau\{I/i\}, D \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{pack } t \text{ with } I \ominus \text{pack } t' \text{ with } I \downarrow \exists i{::}S.\, \tau, D \Rightarrow \Phi}\ \textbf{alg-r-pack-}\downarrow$$

unpack t as (x,i) in t'

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow \exists i{::}S.\, A_1 \Rightarrow [\psi], L_1, U_1, \Phi_1 \qquad L_2, U_2 \in \text{fresh}(\mathbb{R}) \\ i :: S, \Delta; L_2, U_2, \psi, \psi_a; \Phi_a; x : A_1, \Omega \vdash t_2 \downarrow A_2, L_2, U_2 \Rightarrow \Phi_2 \qquad i \notin FV(\Phi_a; \Omega, A_2, L_2, U_2) \\ \Phi = \exists(\psi).(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\forall i :: S.\Phi_2 \wedge L \doteq L_1 + L_2 + c_{unpack} \wedge U_1 + U_2 + c_{unpack} \doteq U)\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2 \downarrow A_2, L, U \Rightarrow \Phi}$$

alg-u-unpack-$\downarrow$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \exists i{::}S.\, \tau_1 \Rightarrow [\psi], D_1, \Phi_1 \\ D_2 \in \text{fresh}(\mathbb{R}) \qquad i :: S, \Delta; D_2, \psi, \psi_a; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2, D_2 \Rightarrow \Phi_2 \\ i \notin FV(\Phi_a; \Gamma, \tau_2, D_2) \qquad \Phi = \exists(\psi).(\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\forall i :: S.\Phi_2 \wedge D_1 + D_2 \doteq D)\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2 \ominus \text{unpack } t_1' \text{ as } (x,i) \text{ in } t_2' \downarrow \tau_2, D \Rightarrow \Phi}$$

alg-r-unpack-$\downarrow$

Figure B.105: Algorithmic typing rules of RelCost, part 7.

clet t_1 as x in t_2

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow C \,\&\, A_1 \Rightarrow [\psi], L_1, U_1, \Phi_1 \\ L_2, U_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; L_2, U_2, \psi, \psi_a; \Phi \wedge C; x : A_1, \Omega \vdash t_2 \downarrow A_2, L_2, U_2 \Rightarrow \Phi_2 \\ \Phi_2' = C \to \Phi_2 \wedge k \doteq (L_1 + L_2) \wedge (U_1 + U_2) \doteq U\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \downarrow A_2, L, U \Rightarrow \exists(\psi).(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi_2')}\ \textbf{alg-u-c-andE}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \ominus t_1' \uparrow C \,\&\, \tau_1 \Rightarrow [\psi], D_1, \Phi_1 \qquad D_2 \in \text{fresh}(\mathbb{R}) \\ \Delta; D_2, \psi, \psi_a; \Phi \wedge C; x : \tau_1, \Omega \vdash t_2 \ominus t_2' \downarrow \tau_2, D_2 \Rightarrow \Phi_2 \qquad \Phi_2' = C \to \Phi_2 \wedge (D_1 + D_2) \doteq D\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{clet } t_1 \text{ as } x \text{ in } t_2 \ominus \text{clet } t_1' \text{ as } x \text{ in } t_2' \downarrow \tau_2, D \Rightarrow \exists(\psi).(\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\Phi_2')}$$

alg-r-c-andE↓

celim t

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow C \supset A \Rightarrow [\psi], L, U, \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{celim } t \uparrow A \Rightarrow [\psi], L, U, C \wedge \Phi}\ \textbf{alg-u-c-implE}\uparrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow C \supset \tau \Rightarrow [\psi], D, \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{celim } t \ominus \text{celim } t' \uparrow \tau \Rightarrow [\psi], D, C \wedge \Phi}\ \textbf{alg-r-c-implE}\uparrow$$

let $x = t_1$ in t_2

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow A_1 \Rightarrow [\psi], L_1, U_1, \Phi_1 \\ L_2, U_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; L_2, U_2, \psi, \psi_a; x : A_1, \Omega \vdash t_2 \downarrow A_2, L_2, U_2 \Rightarrow \Phi_2 \\ \Phi_2' = \Phi_2 \wedge k \doteq (L_1 + L_2 + c_{let}) \wedge (U_1 + U_2 + c_{let}) \doteq U\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \text{let } x = t_1 \text{ in } t_2 \downarrow A_2, L, U \Rightarrow \exists(\psi).\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi_2'}\ \textbf{alg-u-let-}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \tau_1 \Rightarrow [\psi], D_1, \Phi_1 \\ D_2 \in \text{fresh}(\mathbb{R}) \qquad \Delta; D_2, \psi, \psi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2, D_2 \Rightarrow \Phi_2 \qquad \Phi_3 = D_1 + D_2 \doteq D\end{array}}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \ominus \text{let } x = t_1' \text{ in } t_2' \downarrow \tau_2, D \Rightarrow \exists(\psi).\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\Phi_2 \wedge \Phi_3}$$

alg-r-let-↓

Figure B.106: Algorithmic typing rules of RelCost, part 8.

nil

$$\frac{\Delta, \psi_a; \Phi \vdash^{\mathsf{A}} A \ \mathsf{wf}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathrm{nil} \downarrow \mathrm{list}[n]\, A, L, U \Rightarrow n \doteq 0 \wedge L \doteq 0 \wedge 0 \doteq U} \textbf{alg-u-nil-}\downarrow$$

$$\frac{\Delta, \psi_a; \Phi \vdash \tau \ \mathsf{wf}}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \mathrm{nil} \ominus \mathrm{nil} \downarrow \mathrm{list}[n]^{\alpha} \tau, D \Rightarrow n \doteq 0 \wedge 0 \doteq D} \textbf{alg-r-nil-}\downarrow$$

$C \,\&\, \cdot$ intro

$$\frac{\Delta; \psi_a; \Phi \wedge C; \Omega \vdash t \downarrow A, L, U \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash t \downarrow L, U, C \,\&\, A \Rightarrow C \wedge (C \to \Phi)} \textbf{alg-u-c-andI}\downarrow$$

$$\frac{\Delta; \psi_a; \Phi \wedge C; \Gamma \vdash t_1 \ominus t_1 \downarrow \tau, D \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash t_1 \ominus t_2 \downarrow C \,\&\, \tau, D \Rightarrow C \wedge (C \to \Phi)} \textbf{alg-r-c-andI}\downarrow$$

$C \supset \cdot$ intro

$$\frac{\Delta; \Phi \wedge C; \Omega \vdash t \downarrow A, L, U \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash t \downarrow C \supset A, L, U \Rightarrow C \to \Phi} \textbf{alg-u-c-impI} \downarrow$$

$$\frac{\Delta; \Phi \wedge C; \Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma \vdash t \ominus t' \downarrow C \supset \tau, D \Rightarrow C \to \Phi} \textbf{alg-r-c-impI}\downarrow$$

Figure B.107: Algorithmic typing rules of RelCost, part 9.

split t

$$\frac{\Delta;\psi_a;C\wedge\Phi_a;\Omega\vdash t_1\downarrow A,L,U\Rightarrow\Phi_1 \quad \Delta;\psi_a;\neg C\wedge\Phi_a;\Omega\vdash t_2\downarrow A,L,U\Rightarrow\Phi_2 \quad \Delta\vdash C\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Omega\vdash \text{split}\ (t_1,t_2)\ \text{with}\ C\downarrow A,L,U\Rightarrow C\to\Phi_1\wedge\neg C\to\Phi_2}\ \textbf{alg-u-split}\downarrow$$

$$\frac{\Delta;\psi_a;C\wedge\Phi_a;\Gamma\vdash t_1\ominus t_1'\downarrow\tau,D\Rightarrow\Phi_1 \quad \Delta;\psi_a;\neg C\wedge\Phi_a;\Gamma\vdash t_2\ominus t_2'\downarrow\tau,D\Rightarrow\Phi_2 \quad \Delta\vdash C\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Gamma\vdash \text{split}\ (t_1,t_2)\ \text{with}\ C\ominus\text{split}\ (t_1',t_2')\ \text{with}\ C\downarrow\tau,D\Rightarrow C\to\Phi_1\wedge\neg C\to\Phi_2}$$

alg-r-split$\downarrow$

contra t

$$\frac{\Delta;\psi_a;\Phi_a\models\bot}{\Delta;\psi_a;\Phi_a;\Gamma\vdash\text{contra}\ t\downarrow A,L,U\Rightarrow\top}\ \textbf{alg-u-contra}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a\models\bot}{\Delta;\psi_a;\Phi_a;\Gamma\vdash\text{contra}\ t\ominus\text{contra}\ t'\downarrow\tau,D\Rightarrow\top}\ \textbf{alg-r-contra}\downarrow$$

$\uparrow\downarrow$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega\vdash t\uparrow A'\Rightarrow[\psi],L',U',\Phi_1 \quad \Delta;\psi,\psi_a;\Phi_a\models^{\mathsf{A}}A'\sqsubseteq A\Rightarrow\Phi_2}{\Delta;\psi_a;\Phi_a;\Omega\vdash t\downarrow A,L,U\Rightarrow\exists(\psi).\Phi_1\wedge\Phi_2\wedge U'\le U\wedge L\le L'}\ \textbf{alg-}\uparrow\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma\vdash t\ominus t'\uparrow\tau'\Rightarrow[\psi],D',\Phi_1 \quad \Delta;\psi,\psi_a;\Phi_a\models\tau'\equiv\tau\Rightarrow\Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma\vdash t\ominus t'\downarrow\tau,D\Rightarrow\exists(\psi).\Phi_1\wedge\Phi_2\wedge D'\le D}\ \textbf{alg-r-}\uparrow\downarrow$$

$(t:A,L,U)$

$$\frac{\Delta;\psi_a;\Phi_a;\Omega\vdash t\downarrow A,L,U\Rightarrow\Phi \quad \Delta;\Phi_a\vdash^{\mathsf{A}}A\ \mathsf{wf} \quad \text{FIV}(A,L,U)\in\Delta}{\Delta;\psi_a;\Phi_a;\Omega\vdash(t:A,L,U)\uparrow A\Rightarrow[\cdot],L,U,\Phi}\ \textbf{alg-u-anno-}\uparrow$$

$(t:\tau,t)$

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma\vdash t\ominus t'\downarrow\tau,D\Rightarrow\Phi \quad \Delta;\Phi_a\vdash\tau\ \mathsf{wf} \quad \text{FIV}(\tau,D)\in\Delta}{\Delta;\psi_a;\Phi_a;\Gamma\vdash(t:\tau,D)\ominus(t':\tau,D)\uparrow\tau\Rightarrow[\cdot],D,\Phi}\ \textbf{alg-r-anno-}\uparrow$$

Figure B.108: Algorithmic typing rules of RelCost, part 10.

Asynchronous rules

$$\frac{\Delta;\psi_a;\Phi_a;|\Gamma|1 \vdash t_1 \uparrow A_1 \Rightarrow [\psi], L_1, U_1, \Phi_1 \quad t_2 \in \text{fresh}(\mathbb{R}) \quad \Delta; t_2, \psi, \psi_a; \Phi_a; x : U(A_1, A_1), \Gamma \vdash t_2 \ominus t \downarrow \tau_2, D_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 \ominus t \downarrow \tau_2, D \Rightarrow \exists(\psi).(\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\Phi_2 \wedge U_1 + D_2 + c_{let} \doteq D)}\ \textbf{alg-r-let-t}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi_a;|\Gamma|2 \vdash t_1 \uparrow A_1 \Rightarrow [\psi], L_1, U_1, \Phi_1 \quad t_2 \in \text{fresh}(\mathbb{R}) \quad \Delta; t_2, \psi, \psi_a; \Phi_a; x : U(A_1, A_1), \Gamma \vdash t \ominus t_2 \downarrow \tau_2, D_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus \text{let } x = t_1 \text{ in } t_2 \downarrow \tau_2, D \Rightarrow \exists(\psi).(\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\Phi_2 \wedge D_2 - L_1 - c_{let} \doteq D)}\ \textbf{alg-r-t-let}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi;|\Gamma|1 \vdash t_1 \uparrow A_1 \xrightarrow{\text{exec}(L_e,U_e)} A_2 \Rightarrow [\psi], L_1, U_1, \Phi_1 \quad D_2 \in \text{fresh}(\mathbb{R}) \quad \Delta; D_2, \psi, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow U(A_1, A_2'), t_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_2' \downarrow U(A_2, A_2'), D \Rightarrow \exists(\psi).\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\Phi_2 \wedge U_1 + D_2 + U_e + c_{app} \doteq D}\ \textbf{alg-r-app-t}\downarrow$$

$$\frac{\Delta;\psi_a;\Phi;|\Gamma|2 \vdash t_1' \uparrow A_1' \xrightarrow{\text{exec}(L_e,U_e)} A_2' \Rightarrow [\psi], L_1, U_1, \Phi_1 \quad D_2 \in \text{fresh}(\mathbb{R}) \quad \Delta; D_2, \psi, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow U(A_2, A_1'), D_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_1'\ t_2' \downarrow U(A_2, A_2'), D \Rightarrow \exists(\psi).\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\Phi_2 \wedge D_2 - L_1 - L_e + c_{app} \doteq D}\ \textbf{alg-r-t-app}\downarrow$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;|\Gamma|2 \vdash t \uparrow A_1 + A_2 \Rightarrow [\psi], L_1, U_1, \Phi_1 \\ D_2 \in \text{fresh}(\mathbb{R}) \quad \Delta;\psi_a;\Phi_a; x : U(A_1, A_1), \Gamma \vdash t_1 \ominus t' \downarrow \tau, D_2 \Rightarrow \Phi_2 \\ \Delta;\psi_a;\Phi_a; y : U(A_2, A_2), \Gamma \vdash t_2 \ominus t' \downarrow \tau, t_2 \Rightarrow \Phi_3\end{array}}{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \\ \vdash \text{case } (t, x.t_1, y.t_2) \ominus t' \downarrow \tau, D \Rightarrow \exists(\phi).\Phi_1 \wedge (\exists t_2 :: \mathbb{R}.\Phi_2 \wedge U_1 + D_2 + c_{case} \doteq D)\end{array}}$$

alg-r-case-t↓

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;|\Gamma|2 \vdash t' \uparrow A_1 + A_2 \Rightarrow [\psi], L_1, U_1, \Phi_1 \\ D_2 \in \text{fresh}(\mathbb{R}) \quad \Delta;\psi_a;\Phi_a; x : U(A_1, A_1), \Gamma \vdash t \ominus t_1' \downarrow \tau, D_2 \Rightarrow \Phi_2 \\ \Delta;\psi_a;\Phi_a; y : U(A_2, A_2), \Gamma \vdash t \ominus t_2' \downarrow \tau, t_2 \Rightarrow \Phi_3\end{array}}{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Gamma \\ \vdash t \ominus \text{case } (t', x.t_1', y.t_2') \downarrow \tau, D \Rightarrow \exists(\phi).\Phi_1 \wedge (\exists D_2 :: \mathbb{R}.\Phi_2 \wedge D_2 - L_1 - c_{case} \doteq D)\end{array}}$$

alg-r-t-case↓

Figure B.109: Algorithmic typing rules of RelCost, part 11.

$\boxed{\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2 \Rightarrow \Phi}$ checks whether A_1 is subtype of A_2 and generates constraints Φ

$\boxed{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_2 \Rightarrow \Phi}$ checks whether τ_1 is equivalent to τ_2 and generates constraints Φ

$$\frac{}{\Delta;\psi_a;\Phi_a \models \text{int} \equiv \text{int} \Rightarrow \top}\ \textbf{alg-r-int} \qquad \frac{}{\Delta;\psi_a;\Phi_a \models \text{unit} \equiv \text{unit} \Rightarrow \top}\ \textbf{alg-r-unit}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a \models \tau_1 \xrightarrow{\text{diff}(D)} \tau_2 \equiv \tau_1' \xrightarrow{\text{diff}(D')} \tau_2' \Rightarrow \Phi_1 \wedge \Phi_2 \wedge D \doteq D'}\ \textbf{alg-r-fun}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a \models \tau_1 \times \tau_2 \equiv \tau_1' \times \tau_2' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-prod}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Delta;\psi_a;\Phi_a \models \tau_2 \equiv \tau_2' \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a \models \tau_1 + \tau_2 \equiv \tau_1' + \tau_2' \Rightarrow \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-sum}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \text{list}[n]^{\alpha}\,\tau \equiv \text{list}[n']^{\alpha'}\,\tau' \Rightarrow \Phi \wedge n \doteq n' \wedge \alpha \doteq \alpha'}\ \textbf{alg-r-list}$$

$$\frac{i, \Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \forall i \overset{\text{diff}(D)}{::} S.\tau \equiv \forall i \overset{\text{diff}(t')}{::} S.\tau' \Rightarrow \forall i :: S.\Phi \wedge D \doteq D'}\ \forall$$

$$\frac{i, \Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi \qquad i \notin FV(\Phi_a)}{\Delta;\psi_a;\Phi_a \models \exists i{::}S.\tau \equiv \exists i{::}S.\tau' \Rightarrow \forall i :: S.\Phi}\ \textbf{alg-r-}\exists \qquad \frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_2 \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models \Box\tau_1 \equiv \Box\tau_2 \Rightarrow \Phi}\ \textbf{B-}\Box$$

$$\frac{\begin{array}{ll}\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_1' \Rightarrow \Phi_1 & \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1' \\ \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2 & \Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2' \sqsubseteq A_2 \Rightarrow \Phi_2'\end{array}}{\Delta;\psi_a;\Phi_a \models U(A_1, A_2) \equiv U(A_1', A_2') \Rightarrow \Phi_1 \wedge \Phi_1' \wedge \Phi_2 \wedge \Phi_2'}\ \textbf{U}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models C \supset \tau \equiv C' \supset \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi}\ \textbf{c-impl}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a \models C \,\&\, \tau \equiv C' \,\&\, \tau' \Rightarrow C \leftrightarrow C' \wedge \Phi}\ \textbf{c-prod}$$

Figure B.110: Algortihmic type equivalence rules.

$$
\begin{array}{lcl}
|.| & : & \text{Expression} \to \text{Expression} \\
 & & \\
|\mathtt{n}| & = & \mathtt{n} \\
|()| & = & () \\
|x| & = & x \\
|\text{fix } f(x).t| & = & \text{fix } f(x).|t| \\
|\text{fix}_{NC}\ f(x).t| & = & \text{fix}_{NC}\ f(x).|t| \\
|t_1\ t_2| & = & |t_1|\,|t_2| \\
\vdots & & \\
|(t : A, k, t)| & = & |t| \\
|(t : \tau, t)| & = & |t|
\end{array}
$$

Figure B.111: Annotation erasure of RelCost.

B.5.1 Metatheory

Lemma 55 (Embedding of Binary Subtyping in RelCost)**.** *If* $\Delta;\Phi \models \tau \sqsubseteq \tau'$ *then* $\exists t \in$ *RelCostCore such that* $\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c \tau \xrightarrow{diff(0)} \tau'$.

Proof. Proof is by induction on the subtyping derivation. We denote the witness t of type $\tau \xrightarrow{\text{diff}(0)} \tau'$ as $\text{coerce}_{\tau,\tau'}$ for clarity.

Case

$$\frac{\Delta;\Phi_a \models \tau_1' \sqsubseteq \tau_1 \ (\star) \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2' \ (\diamond) \qquad \Delta;\Phi_a \models D \leq D'}{\Delta;\Phi_a \models \tau_1 \xrightarrow{\mathbf{diff}(D)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\mathbf{diff}(D')} \tau_2'} \rightarrow \mathbf{diff}$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau_1',\tau_1}\ .\ \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau_1',\tau_1} \ominus \text{coerce}_{\tau_1',\tau_1} \lesssim 0 :^c \tau_1' \xrightarrow{\text{diff}(0)} \tau_1$

By IH on $(\diamond)$, $\exists \text{coerce}_{\tau_2,\tau_2'}\ .\ \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau_2,\tau_2'} \ominus \text{coerce}_{\tau_2,\tau_2'} \lesssim 0 :^c \tau_2 \xrightarrow{\text{diff}(0)} \tau_2'$.

Then, using these two statements and $\Delta;\Phi \models t \leq t'$ with binary subeffecting rule (rule **c-r-**$\sqsubseteq$ in Figure B.96), we can construct the following derivation where

$t = \lambda x.\lambda y.\text{coerce}_{\tau_2,\tau_2'}\ (x\ (\text{coerce}_{\tau_1',\tau_1}\ y))$

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (\tau_1 \xrightarrow{\text{diff}(t)} \tau_2) \xrightarrow{\text{diff}(0)} \tau_1' \xrightarrow{\text{diff}(t')} \tau_2'$.

Case

$$\frac{}{\Delta;\Phi_a \models \mathbf{unit} \sqsubseteq \Box\mathbf{unit}}\ \mathbf{unit}$$

Then, we can immediately construct the derivation using the rule **c-nochange** in Figure B.92.

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\text{NC}\ () \ominus \lambda x.\text{NC}\ () \lesssim 0 :^c \text{unit} \xrightarrow{\text{diff}(0)} \Box\text{unit}}$$

Case

$$\frac{}{\Delta;\Phi_a \models \mathbf{int} \sqsubseteq \Box\mathbf{int}}\ \mathbf{int}\text{-}\Box$$

Then, we can construct the derivation using the primitive function:

$\text{box}_{\text{int}} : \text{int} \xrightarrow{\text{diff}(0)} \Box\text{int}.$

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\text{box}_{\text{int}}\, x \ominus \lambda x.\text{box}_{\text{int}}\, x \lesssim 0 :^{c} \text{int} \xrightarrow{\text{diff}(0)} \Box\text{int}}$$

Case

$$\frac{}{\Delta;\Phi_a \models \Box U\,(\textbf{int},\textbf{int}) \sqsubseteq \textbf{int}}\ \textbf{□U-int}$$

Then, we can construct the derivation using the primitive function:

$\text{box}_U : \Box U\,(\text{int},\text{int}) \xrightarrow{\text{diff}(0)} \text{int}$

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\text{box}_U\, x \ominus \lambda x.\text{box}_U\, x \lesssim 0 :^{c} \Box U\,(\text{int},\text{int}) \xrightarrow{\text{diff}(0)} \text{int}}$$

Case

$$\frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \tau}\ \textbf{T}$$

Then, we can immediately construct the derivation using the rule **c-der** in Figure B.92.

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\text{der}\, x \ominus \lambda x.\text{der}\, x \lesssim 0 :^{c} \Box\tau \xrightarrow{\text{diff}(0)} \tau}$$

Case

$$\frac{}{\Delta;\Phi_a \models \Box\tau \sqsubseteq \Box\Box\tau}\ \textbf{D}$$

Then, we can immediately construct the derivation using the rule **c-nochange** in Figure B.92.

$$\frac{}{\Delta;\Phi;\cdot \vdash \lambda x.\text{NC}\, x \ominus \lambda x.\text{NC}\, x \lesssim 0 :^{c} \Box\tau \xrightarrow{\text{diff}(0)} \Box\Box\tau}$$

Case

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2 (\star)}{\Delta;\Phi_a \models \Box\tau_1 \sqsubseteq \Box\tau_2}\ \textbf{B-}\Box$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau_1,\tau_2}\ .\ \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau_1,\tau_2} \ominus \text{coerce}_{\tau_1,\tau_2} \lesssim 0 :^c \tau_1 \xrightarrow{\text{diff}(0)} \tau_2$

Then, using $(\star)$ and rules **c-der** and **c-nochange** in Figure B.92, we can construct the derivation

$$\Delta;\Phi;\cdot \vdash \lambda x.\text{NC}\ (\text{coerce}_{\tau_1,\tau_2}\ (\text{der}\ x)) \ominus \lambda x.\text{NC}\ (\text{coerce}_{\tau_1,\tau_2}\ (\text{der}\ x)) \lesssim 0 :^c \Box\tau_1 \xrightarrow{\text{diff}(0)} \Box\tau_2$$

Case

$$\frac{}{\Delta;\Phi_a \models \tau \sqsubseteq U\,|\tau|}\ \textbf{W}$$

Then, we can immediately construct the derivation using the rule **c-switch** in Figure B.92.

$$\Delta;\Phi;\cdot \vdash \lambda x.\text{switch}\ x \ominus \lambda x.\text{switch}\ x \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} U\,(|\tau|1, |\tau|2)$$

Case

$$\frac{}{\Delta;\Phi_a \models \tau \sqsubseteq \tau}\ \textbf{refl}$$

Then, we can immediately construct the derivation

$$\Delta;\Phi;\cdot \vdash \lambda x.x \ominus \lambda x.x \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} \tau$$

Case

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_2\ (\star) \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_3\ (\diamond)}{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_3}\ \textbf{trans}$$

By IH on ($\star$), $\exists \text{coerce}_{\tau_1,\tau_2}\ .\ \ i :: S, \Delta; \Phi; \cdot \vdash \text{coerce}_{\tau_1,\tau_2} \ominus \text{coerce}_{\tau_1,\tau_2} \lesssim 0 :^c \tau_1 \xrightarrow{\text{diff}(0)} \tau_2$

By IH on ($\diamond$), $\exists \text{coerce}_{\tau_2,\tau_3}\ .\ \ i :: S, \Delta; \Phi; \cdot \vdash \text{coerce}_{\tau_2,\tau_3} \ominus \text{coerce}_{\tau_2,\tau_3} \lesssim 0 :^c \tau_2 \xrightarrow{\text{diff}(0)} \tau_3$.

Then, using ($\star$) and ($\diamond$), we can construct the derivation simply by function composition

$$\frac{}{\Delta; \Phi; \cdot \vdash \lambda x.\text{coerce}_{\tau_2,\tau_3}\ (\text{coerce}_{\tau_1,\tau_2}\ x) \ominus \lambda x.\text{coerce}_{\tau_2,\tau_3}\ (\text{coerce}_{\tau_1,\tau_2}\ x) \lesssim 0 :^c \tau_1 \xrightarrow{\text{diff}(0)} \tau_3}$$

Case

$$\frac{}{\Delta; \Phi_a \models \Box(\tau_1 \xrightarrow{\mathbf{diff}(D)} \tau_2) \sqsubseteq \Box\tau_1 \xrightarrow{\mathbf{diff}(0)} \Box\tau_2} \rightarrow \Box_{\mathbf{diff}}$$

Then, we can immediately construct the derivation where $t = \lambda x.\lambda y.\text{NC}\ (\text{der}\ x)\ (\text{der}\ y))$

$\Delta; \Phi; \cdot \vdash t \ominus t \lesssim 0 :^c \Box(\tau_1 \xrightarrow{\text{diff}(D)} \tau_2) \xrightarrow{\text{diff}(0)} \Box\tau_1 \xrightarrow{\text{diff}(0)} \Box\tau_2$.

Case

$$\frac{}{\Delta; \Phi_a \models U\,(A_1 \xrightarrow{\mathbf{exec}(L,U)} A_2) \sqsubseteq U\,A_1 \xrightarrow{\mathbf{diff}(U-L)} U\,A_2} \rightarrow \mathbf{execdiff}$$

Then, we can immediately construct the following derivation

where $t = \lambda x.\lambda y.\text{switch}\ (x\ y)$ using **c-switch** and **c-app** rules.

$\Delta; \Phi; \cdot$

$\vdash t \ominus t \lesssim 0 :^c (U\,(A_1 \xrightarrow{\text{exec}(L,U)} A_2, A_1' \xrightarrow{\text{exec}(L',U')} A_2')) \xrightarrow{\text{diff}(0)} U\,(A_1, A_1') \xrightarrow{\text{diff}(U-L')} U\,(A_2, A_2')$.

Case

$$\frac{i :: S, \Delta; \Phi_a \models \tau \sqsubseteq \tau'\ \ (\star) \qquad i :: S, \Delta; \Phi_a \models D \leq D' \qquad i \notin FV(\Phi_a)}{\Delta; \Phi_a \models \forall i \overset{\mathbf{diff}(D)}{::} S.\tau \sqsubseteq \forall i \overset{\mathbf{diff}(D')}{::} S.\tau'} \forall_{\mathbf{diff}}$$

By IH on ($\star$), $\exists \text{coerce}_{\tau,\tau'}\ .\ \ i :: S, \Delta; \Phi; \cdot \vdash \text{coerce}_{\tau,\tau'} \ominus \text{coerce}_{\tau,\tau'} \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} \tau'$.

Then, using this, the second premise and the **c-r-iLam** and **c-r-iApp** rules in RelCostCore, we can construct the following derivation:

$\Delta; \Phi; \cdot$

$\vdash \lambda x.\Lambda i.\text{coerce}_{\tau,\tau'}\ (x\,[i]) \ominus \lambda x.\Lambda i.\text{coerce}_{\tau,\tau'}\ (x\,[i]) \lesssim 0 :^c (\forall i \overset{\text{diff}(D)}{::} S.\tau) \to \forall i \overset{\text{diff}(D')}{::} S.\tau'.$

Case

$$\frac{}{\Delta;\Phi_a \models \Box(\forall i \overset{\textbf{diff}(D)}{::} S.\tau) \sqsubseteq \forall i \overset{\textbf{diff}(0)}{::} S.\Box\tau}\ \forall\Box$$

Then, we can immediately construct the following derivation using **c-der**, **c-nochange**, **c-r-iLam** and **c-r-iApp** rules in Figures B.92 and B.95.

$\Delta;\Phi;\cdot$

$\vdash \lambda x.\Lambda i.\text{NC}\ ((\text{der}\ x)\,[i]) \ominus \lambda x.\Lambda i.\text{NC}\ ((\text{der}\ x)\,[i]) \lesssim 0 :^c \Box(\forall i \overset{\text{diff}(D)}{::} S.\tau) \to \forall i \overset{\text{diff}(D')}{::} S.\Box\tau.$

Case

$$\frac{}{\Delta;\Phi_a \models U(\forall i \overset{\textbf{exec}(L,U)}{::} S.A, \forall i \overset{\textbf{exec}(L',U')}{::} S.A') \sqsubseteq \forall i \overset{\textbf{diff}(U-L')}{::} S.U(A,A')}\ \forall\textbf{U}$$

Then, we can immediately construct the following derivation

where $t = \lambda x.\Lambda i.\text{switch}\ (x\,[i])$ using **c-switch** and **c-iApp** rules in Figures B.92 and B.95.

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (U(\forall i \overset{\text{exec}(L,U)}{::} S.A, \forall i \overset{\text{exec}(L',U')}{::} S.A')) \to \forall i \overset{\text{diff}(U-L')}{::} S.U(A,A').$

Case

$$\frac{\Delta;\Phi_a \models \tau_1 \sqsubseteq \tau_1'\ (\star) \qquad \Delta;\Phi_a \models \tau_2 \sqsubseteq \tau_2'\ (\diamond)}{\Delta;\Phi_a \models \tau_1 \times \tau_2 \sqsubseteq \tau_1' \times \tau_2'}\ \times$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau_1,\tau_1'}\ .\ \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau_1,\tau_1'} \ominus \text{coerce}_{\tau_1,\tau_1'} \lesssim 0 :^c \tau_1 \xrightarrow{\text{diff}(0)} \tau_1'$.

By IH on $(\diamond)$, $\exists \text{coerce}_{\tau_2,\tau_2'}\ .\ \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau_2,\tau_2'} \ominus \text{coerce}_{\tau_2,\tau_2'} \lesssim 0 :^c \tau_2 \xrightarrow{\text{diff}(0)} \tau_2'$.

Then, using these two statements and the rules **c-prod** and **c-proj** in Figure B.93, we can show the following derivation where $e = \lambda x.\langle \text{coerce}_{\tau_1,\tau_1'}\ (\pi_1 x), \text{coerce}_{\tau_2,\tau_2'}\ (\pi_2 x)\rangle$.

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (\tau_1 \times \tau_2) \xrightarrow{\text{diff}(0)} \tau_1' \times \tau_2'.$

Case

$$\frac{}{\Delta;\Phi_a \models \Box\tau_1 \times \Box\tau_2 \equiv \Box(\tau_1 \times \tau_2)}\ \times\Box$$

We show the direction from right-to-left using the rules **c-der**, **c-nochange**, **c-r-proj**, **c-r-let** and **c-r-prod** in Figures B.92, B.93 and B.95 where the expression

$t = \lambda x.\text{let } a = \pi_1 x \text{ in}$

$\text{let } b = \pi_2 x \text{ in NC } (\langle \text{der } a, \text{der } b \rangle).$
$\Delta; \Phi; \cdot \vdash t \ominus t \lesssim 0 :^c \Box\tau_1 \times \Box\tau_2 \xrightarrow{\text{diff}(0)} \Box(\tau_1 \times \tau_2).$

Case

$$\frac{}{\Delta; \Phi_a \models U(A_1 \times A_2, A_1' \times A_2') \sqsubseteq U(A_1, A_1') \times U(A_2, A_2')} \times\mathbf{U}$$

Then, we can immediately construct the following derivation

where $t = \lambda x.(\langle \text{switch } \pi_1 x, \text{switch } \pi_2 x \rangle)$ using the **c-switch**, **c-r-prod** and **c-r-proj** rules in Figures B.92 and B.93.

$\Delta; \Phi; \cdot \vdash t \ominus t \lesssim 0 :^c (U(A_1 \times A_2, A_1' \times A_2')) \xrightarrow{\text{diff}(0)} U(A_1, A_1') \times U(A_2, A_2')$

Case

$$\frac{}{\Delta; \Phi_a \models \Box\tau_1 + \Box\tau_2 \sqsubseteq \Box(\tau_1 + \tau_2)} +\Box$$

We can construct the following derivation by using the rules **c-der**, **c-nochange**, **c-r-case**, **c-r-inl** and **c-r-inr** in Figure B.92 where the expression

$t = \lambda x.\ \text{case } (x, a.\text{NC } (\text{inl der } a), b.\text{NC } (\text{inr der } b)).$

$\Delta; \Phi; \cdot \vdash t \ominus t \lesssim 0 :^c \Box\tau_1 + \Box\tau_2 \xrightarrow{\text{diff}(0)} \Box(\tau_1 + \tau_2)$

Case

$$\frac{\Delta; \Phi_a \models n \doteq n' \ (\star) \qquad \Delta; \Phi_a \models \alpha \leq \alpha' \ (\diamond) \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau' \ (\dagger)}{\Delta; \Phi_a \models \mathbf{list}[n]^\alpha \tau \sqsubseteq \mathbf{list}[n']^{\alpha'} \tau'} \mathbf{l1}$$

By IH on (†), $\exists \text{coerce}_{\tau,\tau'}.\ \Delta; \Phi; \cdot \vdash \text{coerce}_{\tau,\tau'} \ominus \text{coerce}_{\tau,\tau'} \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} \tau'$

We first construct the more generic term for type:

$$\text{unit} \xrightarrow{\text{diff}(0)} \forall n{::}\mathbb{N}.\, \forall n'{::}\mathbb{N}.\, \forall \alpha{::}\mathbb{N}.\, \forall \alpha'{::}\mathbb{N}.\, ((n = n' \wedge \alpha \leq \alpha') \,\&\, \text{list}[n]^\alpha \tau) \xrightarrow{\text{diff}(0)} \text{list}[n']^{\alpha'} \tau' \tag{B.33}$$

and then instantiate the term for eq. (B.33) later.

It can be shown that such a derivation can be constructed for expression

$t' = \text{fix fList}(\_).\Lambda n.\Lambda n'.\Lambda\alpha.\Lambda\alpha'.\lambda x.\text{clet } x \text{ as } t \text{ in}$

$\text{case } t \text{ of}$

$\text{nil} \rightarrow \text{nil}$

$| \quad h \quad ::_N \quad tl \quad \rightarrow \quad \text{let } r \quad = \quad \text{fList} \quad () [n \; - \; 1] [n' \; -$

$1] [\alpha] [\alpha'] \; tl \text{ in } \text{cons}_{NC}(\text{NC } (\text{coerce}_{\tau,\tau'} \text{ der } h), r)$

$| \; h ::_C \; tl \rightarrow \text{let } r \; = \text{fList } () [n-1] [n'-1] [\alpha - 1] [\alpha' -$

$1] \; tl \text{ in } \text{cons}_C(\text{coerce}_{\tau,\tau'} \; h, r)$

Then, we can instantiate fList using ($\star$) and ($\diamond$) as follows where

$t'' = \lambda x.\text{fList } () \; [n][n'][\alpha][\alpha'] \; x$

$\Delta; \Phi; \cdot \vdash t'' \ominus t'' \lesssim 0 :^c \text{list}[n]^\alpha \tau \xrightarrow{\text{diff}(0)} \text{list}[n']^{\alpha'} \tau'.$

Case

$$\frac{}{\Delta; \Phi_a \models \mathbf{list}[n]^\alpha \Box \tau \sqsubseteq \Box (\mathbf{list}[n]^\alpha \tau)} \; \mathbf{l}\Box$$

We first construct the more generic term for type

$$\text{unit} \xrightarrow{\text{diff}(0)} \forall n{::}\mathbb{N}. \forall \alpha{::}\mathbb{N}. \text{list}[n]^\alpha \Box \tau \xrightarrow{\text{diff}(0)} \Box (\text{list}[n]^\alpha \tau) \tag{B.34}$$

and then instantiate the term for eq. (B.34) later. It can be shown that such a derivation can be constructed for expression

$t' = \text{fix fList}(\_).\Lambda n.\Lambda\alpha.\lambda x.$

$\text{case } t \text{ of}$

$\text{nil} \rightarrow \text{NC } (\text{nil })$

$| \; h ::_N \; tl \rightarrow \text{let } r = \text{fList } () \; [n-1] \; [\alpha] \; tl \text{ in}$

$\text{NC } (\text{cons}_{NC}(\text{der } h, \text{der } r))$

$| \; h ::_C \; tl \rightarrow \text{let } r = \text{fList } () \; [n-1] \; [\alpha - 1] \; tl \text{ in}$

$\text{NC } (\text{cons}_C(\text{der } h, \text{der } r))$

Then, we can instantiate fList with a concrete n and α as follows

where $t'' = \lambda x.\text{fList}\ ()\ [n][\alpha]\ x$

$\Delta;\Phi;\cdot \vdash t'' \ominus t'' \lesssim 0 :^c \text{list}[n]^{\alpha}\,\Box\,\tau \xrightarrow{\text{diff}(0)} \Box\,(\text{list}[n]^{\alpha}\,\tau)$.

Case

$$\frac{\Delta;\Phi_a \models \alpha \doteq 0}{\Delta;\Phi_a \models \mathbf{list}[n]^{\alpha}\,\tau \sqsubseteq \mathbf{list}[n]^{\alpha}\,\Box\,\tau}\ \mathbf{I2}$$

We first construct the more generic term for type

$$\text{unit} \xrightarrow{\text{diff}(0)} \forall n{::}\mathbb{N}.\,\forall \alpha{::}\mathbb{N}.\,(\alpha = 0\ \&\ \text{list}[n]^{\alpha}\,\tau) \xrightarrow{\text{diff}(0)} \text{list}[n]^{\alpha}\,\Box\,\tau \tag{B.35}$$

and then instantiate the term for eq. (B.35) later. It can be shown that such a derivation can be constructed for expression

$t' = \text{fix fList}(\_).\Lambda n.\Lambda\alpha.\lambda x.\text{clet}\ x\ \text{as}\ t\ \text{in}$

$\text{case}\ t\ \text{of}$

$\text{nil} \rightarrow \text{nil}$

$\mid h ::_N tl \rightarrow \text{let}\ r = \text{fList}\ ()\ [n-1][\alpha]\ tl\ \text{in}\ \text{cons}_{NC}(\text{NC}\ h, r)$

$\mid h ::_C tl \rightarrow \text{contra}$

Then, we can instantiate fList with a concrete n and α (note the premise $\alpha = 0$) as follows

where $t'' = \lambda x.\text{fList}\ ()\ [n][\alpha]\ x$

$\Delta;\Phi;\cdot \vdash t'' \ominus t'' \lesssim 0 :^c \text{list}[n]^{0}\,\tau \xrightarrow{\text{diff}(0)} \text{list}[n]^{0}\,\Box\,\tau$.

Case

$$\frac{i :: S, \Delta;\Phi_a \models \tau \sqsubseteq \tau'\ \ (\star) \qquad i \notin FV(\Phi_a)}{\Delta;\Phi_a \models \exists i{::}S.\,\tau \sqsubseteq \exists i{::}S.\,\tau'}\ \exists$$

By IH on $(\star)$, $\exists \text{coerce}_{\tau,\tau'}\ .\ \ i :: S, \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau,\tau'} \ominus \text{coerce}_{\tau,\tau'} \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} \tau'$.

Then, using this and the **c-r-pack** and **c-r-unpack** rules in RelCostCore in Figure B.95,

we can construct the following derivation

where $t = \lambda x.\text{unpack } x \text{ as } (y,i) \text{ in pack } (\text{coerce}_{\tau,\tau'}\ y) \text{ with } i$.

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (\exists i{::}S.\tau) \xrightarrow{\text{diff}(0)} \exists i{::}S.\tau'$.

Case

$$\frac{}{\Delta;\Phi_a \models \exists i{::}S.\Box\tau \sqsubseteq \Box(\exists i{::}S.\tau)}\ \exists\Box$$

Then, we can immediately construct the following derivation using **c-der**, **c-nochange**, **c-r-pack** and **c-r-unpack** rules in in Figures B.92 and B.95 where

$t = \lambda x.\text{unpack } x \text{ as } (y,i) \text{ in NC (pack der } y \text{ with } i)$.

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (\exists i{::}S.\Box\tau) \xrightarrow{\text{diff}(0)} \Box(\exists i{::}S.\tau)$.

Case

$$\frac{\Delta;\Phi_a \wedge C' \models C\ \ (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'\ \ (\diamond)}{\Delta;\Phi_a \models C \supset \tau \sqsubseteq C' \supset \tau'}\ \textbf{c-impl}$$

By IH on ($\diamond$), $\exists \text{coerce}_{\tau,\tau'}.\ \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau,\tau'} \ominus \text{coerce}_{\tau,\tau'} \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} \tau'$.

Then, using this and the premise ($\star$) along with the **c-r-c-implI** and **c-r-c-implE** rules in Figure B.96, we can construct the following derivation where $t = \lambda x.\text{coerce}_{\tau,\tau'}\ (\text{celim } x)$.

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (C \supset \tau) \xrightarrow{\text{diff}(0)} C' \supset \tau'$.

Case

$$\frac{}{\Delta;\Phi_a \models \Box(C \supset \tau) \sqsubseteq C \supset \Box\tau}\ \textbf{c-impl-}\Box$$

Then, we can immediately construct the following derivation using **c-der**, **c-nochange** and **c-r-c-implE** rules in RelCostCore where $t = \lambda x.\text{NC } (\text{celim der } x)$.

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c \Box(C \supset \tau) \xrightarrow{\text{diff}(0)} (C \supset \Box\tau)$.

Case

$$\frac{\Delta;\Phi_a \wedge C \models C'\ \ (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau'\ \ (\diamond)}{\Delta;\Phi_a \models C\ \&\ \tau \sqsubseteq C'\ \&\ \tau'}\ \textbf{c-and}$$

By IH on ($\diamond$), $\exists \text{coerce}_{\tau,\tau'}.\ \Delta;\Phi;\cdot \vdash \text{coerce}_{\tau,\tau'} \ominus \text{coerce}_{\tau,\tau'} \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} \tau'$.

Then, using this and the premise ($\star$) along with the **c-r-c-prodI** and **c-r-c-prodE** rules in Figure B.93, we can construct the following derivation

where $t = \lambda x.\text{clet } x \text{ as } y \text{ in coerce}_{\tau,\tau'}\ y$.

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (C \,\&\, \tau) \xrightarrow{\text{diff}(0)} C' \,\&\, \tau'$.

Case

$$\frac{}{\Delta;\Phi_a \models C \,\&\, \Box\tau \sqsubseteq \Box(C \,\&\, \tau)}\ \textbf{c-and-}\Box$$

Then, we can immediately construct the following derivation using **c-der**, **c-nochange**, **c-r-c-prodI** and **c-r-c-prodE** rules in Figures B.92, B.93 and B.95

where $t = \lambda x.\text{clet } x \text{ as } y \text{ in NC } (\text{der } y)$.

$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (C \,\&\, \Box\tau) \xrightarrow{\text{diff}(0)} \Box(C \,\&\, \tau)$.

□

Lemma 56 (Reflexivity of Algorithmic Binary Type Equivalence in RelCost)**.** $\Delta;\psi_a;\Phi_a \models \tau \equiv \tau \Rightarrow \Phi$ *and* $\Delta;\psi_a;\Phi_a \models \Phi$.

Proof. By induction on the binary type. □

Lemma 57 (Reflexivity of Unary Algorithmic Subtyping in RelCost)**.** $\Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A \Rightarrow \Phi$ *and* $\Delta;\Phi_a \models \Phi$.

Proof. By induction on the unary type. □

Lemma 58 (Transitivity of Unary Algorithmic Subtyping in RelCost)**.** *If* $\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2 \Rightarrow \Phi_1$ *and* $\Delta;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_3 \Rightarrow \Phi_2$ *and* $\Delta;\Phi_a \models \Phi_1 \wedge \Phi_2$*, then* $\Delta;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_3 \Rightarrow \Phi_3$ *for some* Φ_3 *such that* $\Delta;\Phi_a \models \Phi_3$.

Proof. By induction on the first subtyping derivation. □

Theorem 59 (Soundness of the Algorithmic Unary Subtyping in RelCost)**.** *Assume that*

1. $\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A' \sqsubseteq A \Rightarrow \Phi$

2. $FIV(\Phi_a, A, A') \subseteq \Delta, \psi_a$

3. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta; \Phi_a[\theta_a] \models^{\mathsf{A}} A'[\theta_a] \sqsubseteq A[\theta_a]$.

Proof. By induction on the algorithmic unary subtyping derivation. □

Theorem 60 (Completeness of the Unary Algorithmic Subtyping in RelCost)**.** *Assume that* $\Delta; \Phi_a \models^{\mathsf{A}} A' \sqsubseteq A$. *Then* $\exists \Phi$. *such that* $\Delta; \Phi_a \models^{\mathsf{A}} A' \sqsubseteq A \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the unary subtyping derivation. □

Theorem 61 (Soundness of the Algorithmic Binary Type Equality in RelCost)**.** *Assume that*

1. $\Delta; \psi_a; \Phi_a \models \tau' \equiv \tau \Rightarrow \Phi$

2. $FIV(\Phi_a, \tau, \tau') \subseteq \Delta, \psi_a$

3. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta; \Phi_a[\theta_a] \models \tau'[\theta_a] \equiv \tau[\theta_a]$.

Proof. By induction on the algorithmic binary type equivalence derivation. □

Theorem 62 (Completeness of the Binary Algorithmic Type Equivalence in RelCost)**.** *Assume that* $\Delta; \Phi_a \models \tau' \equiv \tau$. *Then* $\exists \Phi$. *such that* $\Delta; \Phi_a \models \tau' \equiv \tau \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the binary subtyping derivation. □

Theorem 63 (Soundness of ARel & Type Preservation of Embedding)**.** *The following holds.*

1. *If* $\Delta; \Phi; \Omega \vdash^t_k t \rightsquigarrow e^* : A$, *then* $\Delta; \Phi; \Omega \vdash^t_k e^* :^c A$ *and* $\Delta; \Phi; \Omega \vdash^t_k t : A$.

2. *If* $\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim t : \tau$, *then* $\Delta;\Phi;\Gamma \vdash t_1^* \ominus t_2^* \lesssim t :^c \tau$ *and* $\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim t : \tau$.

Proof. Proof is by simultaneous induction on the embedding derivations. The proof follows from the embedding rules presented in Figures B.92 to B.97. We show a few representative cases.

Proof of Theorem 63.2:

Case

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 \rightsquigarrow t_1^* : A \quad (\star) \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 \rightsquigarrow t_2^* : A \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow \mathbf{switch}\ t_1^* \ominus \mathbf{switch}\ t_2^* \lesssim U_1 - L_2 : U\ A}\ \textbf{e-switch}$$

By Theorem 63.1 on $(\star)$, we get $\Delta;\Phi;\Omega \vdash^{t_1}_{k_1} t_1^* :^c A_1 \quad (\star\star)$.

By Theorem 63.1 on $(\diamond)$, we get $\Delta;\Phi;\Omega \vdash^{t_2}_{k_2} t_2^* :^c A_2 \quad (\diamond\diamond)$.

Then, we conclude as follows: $\dfrac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 :^c A \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 :^c A}{\Delta;\Phi_a;\Gamma \vdash \text{switch}\ t_1 \ominus \text{switch}\ t_2 \lesssim U_1 - L_2 :^c U\ A}\ \textbf{c-switch}$

Case

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau \\ \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \mathbf{coerce}_{\tau,\tau'} \qquad \Delta;\Phi_a \models U \leq U'\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t'\ t_1^* \ominus t'\ t_2^* \lesssim D' : \tau'}\ \textbf{e-r-}\sqsubseteq[(\star)][(\diamond)][(\dagger)]$$

By Theorem 63.2 on $(\star)$, $\Delta;\Phi;\Gamma \vdash t_1^* \ominus t_2^* \lesssim t :^c \tau \quad (\star\star)$.

By Lemma 55 using $(\diamond)$, we know that $\Delta;\Phi;\cdot \vdash t' \ominus t' \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} \tau' \quad (\diamond\diamond)$.

By applying **c-r-app** rule in Figure B.94 tp $(\star\star)$ and $(\diamond\diamond)$,

we get $\Delta;\Phi;\Gamma \vdash t'\ t_1^* \ominus t'\ t_2^* \lesssim t :^c \tau' \quad (\spadesuit)$.

By reflexivity of binary type equivalence, we know $\Delta;\Phi_a \models \tau' \equiv \tau' \quad (\spadesuit\spadesuit)$.

Then, we conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t'\ t_1^* \ominus t'\ t_2^* \lesssim t :^c \tau' \quad (\spadesuit) \qquad \Delta;\Phi_a \models \tau' \equiv \tau' \quad (\spadesuit\spadesuit) \qquad \Delta;\Phi_a \models t \leq t' \quad (\dagger)}{\Delta;\Phi_a;\Gamma \vdash t'\ t_1^* \ominus t'\ t_2^* \lesssim t' :^c \tau'}\ \textbf{c-r-}\sqsubseteq$$

□

Theorem 64 (Completeness of RelCostCore)**.** *The following holds.*

1. *If* $\Delta;\Phi;\Omega \vdash^{U}_{L} t : A$ *then,* $\exists\, t^*$ *such that* $\Delta;\Phi;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A$.

2. *If* $\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ *then,* $\exists\, t_1^*$ *and* $\exists\, t_2^*$ *such that* $\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim D : \tau$.

Proof. Proof is by simultaneous induction on the typing derivations. The proof follows from the embedding rules presented in Figures B.82-B.84. We show a few representative cases.

Proof of Theorem 64.1:

Case

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A \quad (\star) \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \quad (\diamond) \qquad \Delta;\Phi_a \models k' \leq k \quad (\dagger) \qquad \Delta;\Phi_a \models t \leq t' \quad \dagger\dagger}{\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t : A'}\ \sqsubseteq\textbf{exec}$$

By Theorem 64.1 on $(\star)$, we get $\exists\, t^*$ such that $\Delta;\Phi;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A \quad (\star\star)$.

By **e-u-**$\sqsubseteq$ rule using $(\star\star)$, $(\diamond)$, $(\dagger)$ and $(\dagger\dagger)$, we conclude as follows

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \qquad \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'}{\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t \rightsquigarrow t^* : A'}\ \textbf{e-u-}\sqsubseteq$$

Proof of Theorem 64.2:

Case

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 : A \quad (\star) \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 : A \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim U_1 - L_2 : U\,A}\ \textbf{switch}$$

By Theorem 64.1 on $(\star)$, we get $\exists\, t_1^*$ such that $\Delta;\Phi;\Omega \vdash^{U_1}_{L_1} t_1 \rightsquigarrow {t_1}^* : A_1 \quad (\star\star)$.

By Theorem 64.1 on $(\diamond)$, we get $\exists\, t_2^*$ such that $\Delta;\Phi;\Omega \vdash^{U_2}_{L_2} t_2 \rightsquigarrow {t_2}^* : A_2 \quad (\diamond\diamond)$.

By **e-switch** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude as follows:

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 \rightsquigarrow t_1^* : A \quad (\star\star) \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 \rightsquigarrow t_2^* : A \quad (\diamond\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow \text{switch}\ t_1^* \ominus \text{switch}\ t_2^* \lesssim U_1 - L_2 : U\,A}\ \textbf{e-switch}.$$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t \lesssim D : \tau \quad (\star) \qquad \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x) \quad (\diamond)}{\Delta;\Phi_a;\Gamma,\Gamma';\Omega \vdash t \ominus t \lesssim 0 : \Box\tau}\ \textbf{nochange}$$

By Theorem 64.2 on ($\star$), we get $\exists\, t^*$ such that $\Delta;\Phi;\Gamma \vdash t \ominus t \rightsquigarrow t^* \ominus t^* \lesssim D : \tau$ (†).

By Lemma 55 on ($\diamond$), we get $\exists t_i = \text{coerce}_{\Gamma(x_i),\Box(\Gamma(x_i)}$ for all $x_i \in dom(\Gamma)$ (††).

By **t-nochange** embedding rule using (†) and (††), we can conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t \rightsquigarrow t^* \ominus t^* \lesssim D : \tau \quad (\dagger) \qquad \forall x_i \in dom(\Gamma),\ \ t_i = \text{coerce}_{\Gamma(x_i),\Box\Gamma(x_i)} \quad (\dagger\dagger)}{\Delta;\Phi_a;\Gamma,\Gamma' \vdash t \ominus t \rightsquigarrow \text{let } \overline{y_i = t_i\ x_i} \text{ in NC } t^*[\overline{y_i/x_i}] \ominus \text{let } \overline{y_i = t_i\ x_i} \text{ in NC } t^*[\overline{y_i/x_i}] \lesssim 0 : \Box\tau}$$

e-nochange.

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \textbf{list}[n]^\alpha\,\tau \quad (\star) \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \ominus t_1' \lesssim D' : \tau' \quad (\diamond)}{\begin{array}{c} i,\Delta;\Phi_a \wedge n = i+1;h:\Box\tau,tl:\textbf{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \ominus t_2' \lesssim D' : \tau' \quad (\dagger) \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h:\tau,tl:\textbf{list}[i]^\beta\,\tau,\Gamma \vdash t_2 \ominus t_2' \lesssim D' : \tau' \quad (\spadesuit) \end{array}}$$

$\Delta;\Phi_a;\Gamma \vdash$ **case** t **of nil** $\rightarrow t_1 \mid h :: tl \rightarrow t_2 \ominus$ **case** t' **of nil** $\rightarrow t_1' \mid h :: tl \rightarrow t_2' \lesssim D + D' : \tau'$
r-caseL

By Theorem 64.2 on ($\star$),

we get $\exists\, t^*$ and $\exists\, t'^*$ s.t. $\Delta;\Phi;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \text{list}[n]^\alpha\,\tau$ ($\star\star$).

By Theorem 64.2 on ($\diamond$),

we get $\exists\, t_1^*$ and $\exists\, t_1'^*$ s.t. $\Delta; n \doteq 0 \wedge \Phi;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow t_1^* \ominus t_1'^* \lesssim D : \tau'$ ($\diamond\diamond$).

By Theorem 64.2 on (†), we get $\exists\, t_2^*$ and $\exists\, t_2'^*$ s.t.

$i :: S,\Delta; n \doteq i+1 \wedge \Phi; h : \Box\tau, tl : \text{list}[i]^\alpha\,\tau,\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow t_2^* \ominus t_2'^* \lesssim D : \tau'$ (††).

By Theorem 64.2 on ($\spadesuit$), we get $\exists\, t_2^{**}$ and $\exists\, t_2'^{**}$ s.t.

$i :: S, \beta :: S, \Delta; n \doteq i+1 \wedge \alpha \doteq \beta+1 \wedge \Phi; h : \tau, tl : \text{list}[i]^\beta\,\tau,\Gamma$

$\vdash t_2 \ominus t_2' \rightsquigarrow t_2^{**} \ominus t_2'^{**} \lesssim D : \tau'$ ($\spadesuit\spadesuit$).

By **e-caseL** embedding rule using ($\star\star$), ($\diamond\diamond$), and ($\spadesuit\spadesuit$), we can conclude as follows

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \mathrm{list}[n]^{\alpha}\,\tau \\ \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D' : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1;h : \square\tau, tl : \mathrm{list}[i]^{\alpha}\,\tau,\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D' : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta+1;h : \tau, tl : \mathrm{list}[i]^{\beta}\,\tau,\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow t_3^* \ominus t_3'^* \lesssim D' : \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \begin{array}{l}\text{case } t \text{ of nil } \to t_1 \\ \mid h :: tl \to t_2\end{array} \ominus \begin{array}{l}\text{case } t \text{ of nil } \to t_1' \\ \mid h :: tl \to t_2'\end{array} \rightsquigarrow\lesssim D + D' : \tau'}$$

$$\begin{array}{l}\text{case } t^* \text{ of nil } \to \\ \quad t_1^* \\ \mid h ::_{NC} tl \to t_2^* \\ \mid h ::_{C} tl \to t_3^*\end{array} \ominus \begin{array}{l}\text{case } t'^* \text{ of nil } \to \\ \quad t_1'^* \\ \mid h ::_{NC} tl \to t_2'^* \\ \mid h ::_{C} tl \to t_3'^*\end{array}$$

e-r-caseL

Case

$$\frac{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\mathbf{diff}(D)} \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x : \tau_1, f : \square(\tau_1 \xrightarrow{\mathbf{diff}(D)} \tau_2),\Gamma \vdash t \ominus t \lesssim D : \tau_2 \quad (\star) \qquad \forall x \in dom(\Gamma).\ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \square\Gamma(x) \quad (\diamond)}{\Delta;\Phi_a;\Gamma \vdash \mathbf{fix}\ f(x).t \ominus \mathbf{fix}\ f(x).t \lesssim 0 : \square(\tau_1 \xrightarrow{\mathbf{diff}(D)} \tau_2)}\ \textbf{r-fixNC}$$

By Theorem 64.2 on $(\star)$, we get $\exists\, t^*$ such that

$\Delta;\Phi;x : \tau_1, f : \square(\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2),\Gamma \vdash t \ominus t \rightsquigarrow t^* \ominus t^* \lesssim D : \tau_2 \quad (\star\star)$.

By Lemma 55 on $(\diamond)$, we get $\exists t_i = \mathrm{coerce}_{\Gamma(x_i),\square(\Gamma(x_i)}$ for all $x_i \in dom(\Gamma)$ $(\diamond\diamond)$.

By **e-fixNC** embedding rule using $(\star\star)$ and $(\diamond\diamond)$, we can conclude as follows:

$$\frac{\Delta;\Phi_a \vdash \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2\ \mathsf{wf} \qquad \Delta;\Phi_a;x : \tau_1, f : \square(\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2),\Gamma \vdash t \ominus t \rightsquigarrow t^* \ominus t^* \lesssim D : \tau_2 \qquad \forall x_i \in dom(\Gamma),\ \ t_i = \mathrm{coerce}_{\Gamma(x_i),\square\Gamma(x_i)} \qquad t^{**} = \mathrm{let}\ \overline{y_i = t_i\ x_i}\ \mathrm{in}\ \mathrm{fix}_{NC}\ f(x).t^*[\overline{y_i/x_i}]}{\Delta;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).t \ominus \mathrm{fix}\ f(x).t \rightsquigarrow t^{**} \ominus t^{**} \lesssim 0 : \square(\tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2)}$$

e-r-fixNC.

Case

$$\frac{i :: S,\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D : \tau \quad (\star) \qquad i \notin \mathbf{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda.t \ominus \Lambda.t' \lesssim 0 : \forall i \overset{\mathbf{diff}(D)}{::} S.\tau}\ \textbf{r-iLam}$$

By Theorem 64.2 on $(\star)$, we get $\exists\, t^*$ and $\exists\, t'^*$

such that $i :: S,\Delta;\Phi;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \tau \quad (\star\star)$.

By **e-iLam** embedding rule using $(\star\star)$, we can conclude as follows:

$$\frac{i :: S, \Delta; \Phi_a; \Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \tau \qquad i \notin \mathrm{FIV}(\Phi_a; \Gamma)}{\Delta; \Phi_a; \Gamma \vdash \Lambda.t \ominus \Lambda.t' \rightsquigarrow \Lambda i.t^* \ominus \Lambda i.t'^* \lesssim 0 : \forall i \overset{\mathrm{diff}(D)}{::} S.\tau}\ \textbf{e-r-iLam}.$$

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \ominus t' \lesssim D : \forall i \overset{\mathbf{diff}(D')}{::} S.\tau \quad (\star) \qquad \Delta \vdash I : S \ (\diamond)}{\Delta; \Phi_a; \Gamma \vdash t[] \ominus t'[] \lesssim D + D'[I/i] : \tau\{I/i\}}\ \textbf{r-iApp}$$

By Theorem 64.2 on $(\star)$,

we get $\exists\, t^*$ such that $\Delta; \Phi; \Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \forall i \overset{\mathrm{exec}(t', \tau)}{::} S.$ $(\star\star)$.

By **e-iApp** embedding rule using $(\star\star)$ and $(\diamond)$, we can conclude as follows:

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \forall i \overset{\mathrm{diff}(U')}{::} S.\tau \qquad \Delta \vdash I : S}{\Delta; \Phi_a; \Gamma \vdash t[] \ominus t'[] \rightsquigarrow t^*[I] \ominus t'^*[I] \lesssim D + D'[I/i] : \tau\{I/i\}}\ \textbf{e-r-iApp}.$$

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \ominus t' \lesssim D : \tau\{I/i\} \quad (\star) \qquad \Delta \vdash I :: S \ (\diamond)}{\Delta; \Phi_a; \Gamma \vdash \mathbf{pack}\ t \ominus \mathbf{pack}\ t' \lesssim D : \exists i :: S.\tau}\ \textbf{r-pack}$$

By Theorem 64.2 on $(\star)$,

we get $\exists\, t^*$ and $\exists\, t'^*$ such that $\Delta; \Phi; \Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \tau\{I/i\}$ $(\star\star)$.

By **e-pack** embedding rule using $(\star\star)$ and $(\diamond)$, we can conclude as follows:

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D : \tau\{I/i\} \qquad \Delta \vdash I :: S}{\Delta; \Phi_a; \Gamma \vdash \mathrm{pack}\ t \ominus \mathrm{pack}\ t' \rightsquigarrow \mathrm{pack}\ t^* \text{ with } I \ominus \mathrm{pack}\ t'^* \text{ with } I \lesssim D : \exists i :: S.\tau}\ \textbf{e-r-pack}.$$

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \exists i :: S.\tau_1 \quad (\star) \qquad i :: S, \Delta; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \tau_2 \ (\diamond) \qquad i \notin FV(\Phi_a; \Gamma, \tau_2, t_2)}{\Delta; \Phi_a; \Gamma \vdash \mathbf{unpack}\ t_1\ \mathbf{as}\ x\ \mathbf{in}\ t_2 \ominus \mathbf{unpack}\ t_1'\ \mathbf{as}\ x\ \mathbf{in}\ t_2' \lesssim D_1 + D_2 : \tau_2}\ \textbf{r-unpack1}$$

By Theorem 64.2 on $(\star)$, we get $\exists\, t_1^*$ and $\exists\, t_1'^*$

such that $\Delta; \Phi; \Gamma \vdash t_1 \ominus t_1' \rightsquigarrow t_1^* \ominus t_1'^* \lesssim D : \exists i :: S.\tau_1$ $(\star\star)$.

By Theorem 64.2 on $(\diamond)$, we get $\exists\, t_2^*$ and $\exists\, t_2'^*$

such that $i :: S, \Delta; \Phi; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \rightsquigarrow t_2^* \ominus t_2'^* \lesssim D : \tau_2$ $(\diamond\diamond)$.

Proved by **e-r-unpack** embedding rule using $(\star\star)$ and $(\diamond\diamond)$.

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau \quad (\star) \qquad \Delta; \Phi_a \models \tau \sqsubseteq \tau' \ (\diamond) \qquad \Delta; \Phi_a \models t \leq t' \ \dagger}{\Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D' : \tau'}\ \textbf{r-}\sqsubseteq$$

By Theorem 64.2 on $(\star)$, we get $\exists\, t_1^*, t_2^*$ such that $\Delta; \Phi; \Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim t : \tau$ $(\star\star)$.

By Lemma 55 on ($\diamond$), we can show that $\exists t' = \text{coerce}_{\tau,\tau'}$ ($\diamond\diamond$).
By **e-r-$\sqsubseteq$** rule using ($\star\star$), ($\diamond\diamond$) and ($\dagger$), we conclude as follows

$$\frac{\Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \text{coerce}_{\tau,\tau'} \qquad \Delta;\Phi_a \models U \leq U' \qquad \Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1{}^* \ominus t_2{}^* \lesssim D : \tau}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t'\ t_1^* \ominus t'\ t_2^* \lesssim D' : \tau'}\ \textbf{e-r-}\sqsubseteq$$

□

Theorem 65 (Invariant of the Algorithmic Typechecking)**.** *We have the following.*

1. *Assume that* $\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, L, U \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Omega;A,L,U) \subseteq dom(\Delta,\psi_a)$. *Then* $FIV(\Phi) \subseteq dom(\Delta;\psi_a)$.

2. *Assume that* $\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A \Rightarrow [\psi], L, U, \Phi$ *and* $FIV(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$. *Then* $FIV(A,L,U,\Phi) \subseteq dom(\Delta,\psi;\psi_a)$.

3. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Gamma,\tau,D) \subseteq dom(\Delta,\psi_a)$.*Then* $FIV(\Phi) \subseteq dom(\Delta;\psi_a)$.

4. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau \Rightarrow [\psi], D, \Phi$ *and* $FIV(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$.*Then* $FIV(\tau,D,\Phi) \subseteq dom(\Delta;\psi;\psi_a)$.

Theorem 66 (Soundness of the Algorithmic Typechecking in RelCost)**.** *We have the following.*

1. *Assume that* $\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, L, U \Rightarrow \Phi$ *and*

   1.1. $FIV(\Phi_a,\Omega,A,L,U) \subseteq dom(\Delta,\psi_a)$

   1.2. $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable for some* θ_a *such that* $\Delta \rhd \theta_a : \psi_a$ *is derivable*

   Then $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a]$.

2. *Assume that* $\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A \Rightarrow [\psi], L, U, \Phi$ *and*

   2.1. $FIV(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$

2.2. $\forall\theta\ \forall\theta_a.\ \Delta;\Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *is provable s.t* $\Delta \rhd \theta : \psi$ *and* $\Delta \rhd \theta_a : \psi_a$ *are derivable*

Then $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta\,\theta_a]}_{L[\theta\,\theta_a]} |t| :^c A[\theta\,\theta_a]$.

3. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi$ *and*

3.1. $FIV(\Phi_a,\Gamma,\tau,D) \subseteq dom(\Delta,\psi_a)$

3.2. $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable for some* θ_a *such that* $\Delta \rhd \theta_a : \psi_a$ *is derivable*

Then $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta_a] :^c \tau[\theta_a]$.

4. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau \Rightarrow [\psi], D, \Phi$ *and*

4.1. $FIV(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$

4.2. $\forall\theta\ \forall\theta_a.\ \Delta;\Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *is provable s.t* $\Delta \rhd \theta : \psi$ *and* $\Delta \rhd \theta_a : \psi_a$ *are derivable*

Then $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta\,\theta_a] :^c \tau[\theta\,\theta_a]$.

Proof. Statements (1—4) follow from simultaneous structural induction on the algorithmic typing derivations. We present several cases below.

Proof of Thoerem 66.1:

Case

$$\frac{L_1,U_1,L_2,U_2 \in \mathbf{fresh}(\mathbb{R}) \qquad \Delta;L_1,U_1,\psi_a;\Phi_a;\Omega \vdash t_1 \downarrow A_1,L_1,U_1 \Rightarrow \Phi_1 \qquad \Delta;L_2,U_2,\psi_a;\Phi_a;\Omega \vdash t_2 \downarrow A_1,L_2,U_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash \langle t_1,t_2\rangle \downarrow A_1 \times A_2, L, U \Rightarrow \exists L_1,U_1 :: \mathbb{R}.\Phi_1 \wedge \exists L_2,U_2 :: \mathbb{R}.\Phi_2 \wedge U_1+U_2 \doteq U \wedge L \doteq L_1+L_2}$$

alg-u-prod-↓

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} \langle|t_1|,|t_2|\rangle :^c A_1[\theta_a] \times A_2[\theta_a]$.

By the main assumptions, we have $FIV(\Phi_a,\Omega,A,L,U) \subseteq dom(\Delta,\psi_a)$ $(\star)$

$\Delta;\Phi_a[\theta_a] \models (\exists L_1, U_1 :: \mathbb{R}.\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi_2 \wedge (U_1 + U_2) \doteq U \wedge (L_1 + L_2) \doteq L)[\theta_a]$ $(\star\star)$
Using $(\star)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \vdash L_1 :: \mathbb{R}$ and $\Delta \vdash U_1 :: \mathbb{R}$

b) $\Delta \vdash L_2 :: \mathbb{R}$ and $\Delta \vdash U_2 :: \mathbb{R}$

c) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta_a, L_1 \mapsto L_1, U_1 \mapsto U_1]$

d) $\Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a, L_2 \mapsto L_2, U_2 \mapsto U_2]$

e) $\Delta;\Phi_a[\theta_a] \models (U_1 + U_2) \doteq U[\theta_a] \wedge (L_1 + L_2) \doteq L[\theta_a]$

By Thoerem 66.1 on the first premise using $(\star)$ and c), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_1}_{L_1} |t_1| :^c A_1[\theta_a] \tag{B.36}$$

By Thoerem 66.1 on the second premise using $(\star)$ and d), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_2}_{L_2} |t_2| :^c A_2[\theta_a] \tag{B.37}$$

Combining eqs. (B.37) and (B.69) with **c-prod** rule, we get
$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_1+U_2}_{L_1+L_2} \langle |t_1|, |t_2| \rangle :^c A_1[\theta_a] \times A_2[\theta_a]$.
Then, by using e) with the **c-⊑** exec rule , we can conclude that
$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} \langle |t_1|, |t_2| \rangle :^c A_1[\theta_a] \times A_2[\theta_a]$.
Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A' \Rightarrow [\psi], L', t', \Phi_1 \qquad \Delta;\psi,\psi_a;\Phi_a \models^{\mathsf{A}} A' \sqsubseteq A \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, k, t \Rightarrow \exists(\psi).\Phi_1 \wedge \Phi_2 \wedge t' \leq t \wedge k \leq L'} \textbf{alg-}\uparrow\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a]$.
By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Omega, A, L, U) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta;\Phi_a[\theta_a] \models (\exists(\psi).\Phi_1 \wedge \Phi_2 \wedge U' \leq U \wedge L \leq L')[\theta_a]$ $(\star\star)$
By Theorem 82 using $(\star)$ and the first premise, we get
$\mathrm{FIV}(A', L', U', \Phi_1) \subseteq dom(\Delta, \psi; \psi_a)$ $(\diamond)$.
Using $(\star)$ and $(\diamond)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \rhd \theta_a : \psi_a$

b) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta_a, \theta_a]$

c) $\Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a, \theta_a]$

d) $\Delta; \Phi_a[\theta_a] \models U'[\theta\,\theta_a] \leq U[\theta_a] \wedge L[\theta_a] \leq L'[\theta\,\theta_a]$

By Thoerem 66.2 on the first premise using ($\star$), a) and b), we can show that

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U'[\theta\,\theta_a]}_{L'[\theta\,\theta_a]} |t| :^c A'[\theta\,\theta_a] \tag{B.38}$$

By Thoerem 74 using the second premise and c), we obtain

$$\Delta; \Phi_a[\theta_a] \models^{\mathsf{A}} A'[\theta\,\theta_a] \sqsubseteq A[\theta\,\theta_a] \tag{B.39}$$

Note that due to ($\star$), we have $A[\theta\,\theta_a] = A[\theta_a]$. Then we can conclude by the **c-⊑** exec rule using eqs. (B.38) and (B.39) and (d) that $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a]$.

Case

$$\frac{\Delta; \psi_a; \Phi_a; f : A_1 \xrightarrow{\mathbf{exec}(L',U')} A_2, x : A_1, \Omega \vdash t \downarrow A_2, L', U' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash \mathbf{fix}\ f(x).t \downarrow A_1 \xrightarrow{\mathbf{exec}(L',U')} A_2, L, U \Rightarrow \Phi \wedge L \doteq 0 \wedge 0 \doteq U}\ \textbf{alg-u-fix-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} \text{fix}\ f(x).|t| :^c A_1[\theta_a] \xrightarrow{\text{exec}(L'[\theta_a],U'[\theta_a])} A_2[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega, A_1 \xrightarrow{\text{exec}(L',U')}, L, U) \subseteq dom(\Delta, \psi_a)$ ($\star$) and $\Delta; \Phi_a[\theta_a] \models \Phi \wedge 0 \doteq L \wedge 0 \doteq U'[\theta_a]$ ($\star\star$)

Using ($\star$), we can show that

a) $\text{FIV}(\Phi_a, \Omega, A_1, A_1 \xrightarrow{\text{exec}(L',U')}, A_2, L', U') \subseteq dom(\Delta, \psi_a)$.

We also can show that ($\star\star$)'s derivation must be in a form such that we have

b) $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$

c) $\Delta; \Phi_a[\theta_a] \models 0 \doteq L[\theta_a]$

d) $\Delta; \Phi_a[\theta_a] \models 0 \doteq U[\theta_a]$

By Thoerem 66.1 on the first premise using a) and b), we can show that

$$\Delta; \Phi_a[\theta_a]; x : A_1[\theta_a], f : A_1[\theta_a] \xrightarrow{\text{exec}(L'[\theta_a],U'[\theta_a])} A_2[\theta_a], \Omega[\theta_a] \vdash^{U'[\theta_a]}_{L'[\theta_a]} |t| :^c A_2[\theta_a] \tag{B.40}$$

By the **c-fix** rule using eq. (B.40), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^0_0 \text{fix}\ f(x).|t| :^c A_1[\theta_a] \xrightarrow{\text{exec}(L'[\theta_a],U'[\theta_a])} A_2[\theta_a]$.
By **c-⊑** exec rule using (c) and (d), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} \text{fix}\ f(x).|t| :^c A_1[\theta_a] \xrightarrow{\text{exec}(L'[\theta_a],U'[\theta_a])} A_2[\theta_a]$.

Case

$$\frac{i :: S, \Delta; \psi_a; \Phi_a; \Omega \vdash t \downarrow A, L_e, U_e \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Omega \vdash \Lambda i.t \downarrow \forall i \overset{\mathbf{exec}(L_e, U_e)}{::} S.\, A, L, U \Rightarrow (\forall i :: S.\Phi) \wedge L \doteq 0 \wedge 0 \doteq U}\ \textbf{alg-u-iLam-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} \Lambda i.|t| :^c \forall i \overset{\text{exec}(L_e[\theta_a], U_e[\theta_a])}{::} S.\, A[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega, \forall i \overset{\text{exec}(L_e, t_e)}{::} S.\, A, L, U) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and

$\Delta; \Phi_a[\theta_a] \models ((\forall i :: S.\Phi) \wedge 0 \doteq L \wedge 0 \doteq U)[\theta_a]$ $(\star\star)$

Using $(\star)$, we can show that

a) $\text{FIV}(\Phi_a, \Omega, A, L_e, U_e) \subseteq i, dom(\Delta, \psi_a)$.

We can also show that $(\star\star)$'s derivation must be in a form such that we have

b) $i :: S, \Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$

c) $\Delta; \Phi_a[\theta_a] \models 0 \doteq L[\theta_a]$

d) $\Delta; \Phi_a[\theta_a] \models 0 \doteq U[\theta_a]$

By Thoerem 66.1 on the premise using a) and b), we can show that

$$i :: S, \Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U_e[\theta_a]}_{L_e[\theta_a]} |t| :^c A[\theta_a] \tag{B.41}$$

By the **c-iLam** rule using eq. (B.41),
we obtain $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^0_0 \Lambda i.|t| :^c \forall i \overset{\text{exec}(L_e[\theta_a], U_e[\theta_a])}{::} S.\, A[\theta_a]$.
By **c-**$\sqsubseteq_{\text{exec}}$ rule using (c) and (d),
we obtain $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} \Lambda i.|t| :^c \forall i \overset{\text{exec}(L_e[\theta_a], U_e[\theta_a])}{::} S.\, A[\theta_a]$.

Case

$$\frac{\Delta; \psi_a; \Phi_a; \Omega \vdash t \downarrow A\{I/i\}, L, U \Rightarrow \Phi \qquad \Delta \vdash I :: S}{\Delta; \psi_a; \Phi_a; \Omega \vdash \textbf{pack}\ t\ \textbf{with}\ I \downarrow \exists i{::}S.\, A, L, U \Rightarrow \Phi}\ \textbf{alg-u-pack-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} \text{pack}\ |t|\ \text{with}\ I :^c \exists i{::}S.\, A[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega, \exists i{::}S.\, A, L, U) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and

$\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$

Using $(\star)$ and the second premise, we can show that

a) $\text{FIV}(\Phi_a, \Omega, A\{I/i\}, L, U) \subseteq dom(\Delta, \psi_a)$.

By Thoerem 66.1 on the premise using a) and ($\star\star$), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a]\{I/i\} \tag{B.42}$$

By the **c-pack** rule using eq. (B.42) and the second premise, we obtain
$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]}$ pack $|t|$ with $I :^c \exists i{::}S.\, A[\theta_a]$.

Case

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow \exists i{::}S.\, A_1 \Rightarrow [\psi], L_1, U_1, \Phi_1 \qquad L_2, U_2 \in \textbf{fresh}(\mathbb{R}) \\ i :: S, \Delta; L_2, U_2, \psi, \psi_a; \Phi_a; x : A_1, \Omega \vdash t_2 \downarrow A_2, L_2, U_2 \Rightarrow \Phi_2 \qquad i \notin FV(\Phi_a;\Omega, A_2, L_2, U_2) \\ \Phi = \exists(\psi).(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\forall i :: S.\Phi_2 \wedge L \doteq L_1 + L_2 + c_{unpack} \wedge U_1 + U_2 + c_{unpack} \doteq U)\end{array}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \textbf{unpack } t_1 \textbf{ as } (x,i) \textbf{ in } t_2 \downarrow A_2, L, U \Rightarrow \Phi}$$

alg-u-unpack-$\downarrow$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]}$ unpack $|t_1|$ with (x,i) in $|t_2| :^c A_2[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega, A_2, L, U) \subseteq dom(\Delta, \psi_a)$ ($\star$) and

$\Delta;\Phi_a[\theta_a] \models$

$(\exists(\psi).(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\forall i :: S.\Phi_2 \wedge (U_1 + U_2 + c_{unpack}) \doteq U \wedge (L_1 + L_2 + c_{unpack}) \doteq L))[\theta_a]$

($\star\star$)

By Theorem 82 using the first premise and ($\star$), we get

$\text{FIV}(A_1, L_1, U_1, \Phi_1) \subseteq dom(\Delta, \psi; \psi_a)$ ($\diamond$).

Using ($\star$), ($\diamond$) and the 4th premise, ($\star\star$)'s derivation must be in a form such that we have

a) $\Delta \rhd \theta : \psi$

b) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta\,\theta_a]$

c) $i :: S, \Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a, \theta, L_2 \mapsto L_2, U_2 \mapsto U_2]$

d) $\Delta;\Phi_a[\theta_a] \models U_1[\theta\,\theta_a] + U_2 + c_{unpack} \doteq U[\theta_a]$

e) $\Delta;\Phi_a[\theta_a] \models L[\theta_a] \doteq L_1[\theta\,\theta_a] + L_2 + c_{unpack}$

By Thoerem 66.2 on the first premise using ($\star$), a) and b), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_1[\theta\,\theta_a]}_{L_1[\theta\,\theta_a]} |t_1| :^c \exists i{::}S.\, A_1[\theta\,\theta_a] \tag{B.43}$$

From ($\star$) and ($\diamond$), we can show that

f) $\text{FIV}(\Phi_a, A_1, \Omega, A_2, L_2, U_2) \subseteq i, L_2, t_2, dom(\Delta, \psi, \psi_a)$

By Thoerem 66.1 on the second premise using c), f), ($\star$) and ($\diamond$), we obtain

$$i :: S, \Delta; \Phi_a[\theta_a]; x : A_1[\theta\theta_a], \Omega[\theta_a] \vdash^{U_2}_{L_2} |t_2| :^c A_2[\theta\theta_a] \tag{B.44}$$

Note that due to ($\star$), we have $A_2[\theta\theta_a] = A_2[\theta_a]$. Then by **c-unpack** rule using eqs. (B.43) and (B.44), we can show that
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U_1[\theta\theta_a]+U_2+c_{unpack}}_{L_1[\theta\theta_a]+L_1+c_{unpack}}$ unpack $|t_1|$ with (x, i) in $|t_2| :^c A_2[\theta_a]$.
By $\sqsubseteq$ exec rule using (d) and (e), we obtain
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]}$ unpack $|t_1|$ with (x, i) in $|t_2| :^c A_2[\theta_a]$.
Case

$$\frac{\Delta; \psi_a; C \wedge \Phi_a; \Omega \vdash t_1 \downarrow A, L, U \Rightarrow \Phi_1 \qquad \Delta; \psi_a; \neg C \wedge \Phi_a; \Omega \vdash t_2 \downarrow A, L, U \Rightarrow \Phi_2 \qquad \Delta \vdash C \text{ wf}}{\Delta; \psi_a; \Phi_a; \Omega \vdash \textbf{split}\ (t_1, t_2)\ \textbf{with}\ C \downarrow A, L, U \Rightarrow C \rightarrow \Phi_1 \wedge \neg C \rightarrow \Phi_2}\ \textbf{alg-u-split}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]}$ split $(|t_1|, |t_2|)$ with $C :^c A[\theta_a]$.
By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega, A, L, U) \subseteq dom(\Delta, \psi_a)$ ($\star$) and
$\Delta; \Phi_a[\theta_a] \models (C \rightarrow \Phi_1 \wedge \neg C \rightarrow \Phi_2)[\theta_a]$ ($\star\star$)
Using ($\star$) and the third premise, we can show that

a) $\text{FIV}(C \wedge \Phi_a, \Omega, A, L, U) \subseteq dom(\Delta, \psi_a)$.

b) $\text{FIV}(\neg C \wedge \Phi_a, \Omega, A, L, U) \subseteq dom(\Delta, \psi_a)$.

Using ($\star\star$) and the third premise, we can show that

c) $\Delta; C \wedge \Phi_a[\theta_a] \models \Phi_1[\theta_a]$

d) $\Delta; \neg C \wedge \Phi_a[\theta_a] \models \Phi_2[\theta_a]$

By Thoerem 66.1 on the first premise using ($\star$) and c), we can show that

$$\Delta; C \wedge \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t_1| :^c A[\theta_a]\{I[\theta_a]/i\} \tag{B.45}$$

By Thoerem 66.1 on the second premise using ($\star$) and d), we can show that

$$\Delta; \neg C \wedge \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t_2| :^c A[\theta_a]\{I[\theta_a]/i\} \tag{B.46}$$

By the **c-split** rule using eqs. (B.45) and (B.46) and the third premise, we obtain $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} \text{split}\ (|t_1|,|t_2|)$ with $C :^c A[\theta_a]$.

Case

$$\frac{\Delta;\Phi \wedge C;\Omega \vdash t \downarrow A,L,U \Rightarrow \Phi}{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow C \supset A,L,U \Rightarrow C \to \Phi}\ \textbf{alg-u-c-impI}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c C[\theta_a] \supset A[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega,C \supset A,L,U) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models (C \to \Phi)[\theta_a]$ $(\star\star)$

Using $(\star)$, we can show that

a) $\text{FIV}(C \wedge \Phi_a,\Omega,A,L,U) \subseteq dom(\Delta,\psi_a)$.

By Thoerem 66.1 on the premise using $(\star)$ and a), we can show that

$$\Delta;C[\theta_a] \wedge \Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a] \tag{B.47}$$

By the **c-cimpI** rule using eq. (B.47), we obtain $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c C[\theta_a] \supset A[\theta_a]$.

Proof of Thoerem 66.2:

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A,L,U \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash^{\mathsf{A}} A\ \mathsf{wf} \qquad \textbf{FIV}(A,L,U) \in \Delta}{\Delta;\psi_a;\Phi_a;\Omega \vdash (t : A,L,U) \uparrow A \Rightarrow [\cdot],L,U,\Phi}\ \textbf{alg-u-anno-}\uparrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |(t : A,k,t)| :^c A[\theta_a]$.

Since by definition, $\forall t.\ |(t : \_,\_,\_)| = |t|$, STS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and

$\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ $(\star\star)$

Using the third premise, we can show that

a) $\text{FIV}(\Phi_a,\Omega,A,L,U) \subseteq dom(\Delta,\psi_a)$.

By Thoerem 66.1 on the first premise using $(\star\star)$ and a), we can conclude that $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a]$.

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow A_1 \xrightarrow{\mathbf{exec}(L_e,U_e)} A_2 \Rightarrow [\psi], L_1, U_1, \Phi_1 \qquad L_2, U_2 \in \mathbf{fresh}(\mathbb{R}) \qquad \Delta; L_2, U_2, \psi, \psi_a; \Phi_a; \Omega \vdash t_2 \downarrow A_1, L_2, U_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Omega \vdash t_1\ t_2 \uparrow A_2 \Rightarrow [L_2, L_2, \psi], L_1 + L_2 + L_e + c_{app}, U_1 + U_2 + U_e + c_{app}, \Phi_1 \wedge \Phi_2}\ \textbf{alg-u-app-}\uparrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{(U_1+U_2+U_e+c_{app})[\theta_a,\theta_2]}_{(L_1+L_2+L_e+c_{app})[\theta_a,\theta_2]} |t_1|\,|t_2| :^c A_2[\theta_a,\theta_2]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$ $(\star)$ and $\Delta;\Phi_a[\theta_a] \models (\Phi_1 \wedge \Phi_2)[\theta_a,\theta_2]$ $(\star\star)$ such that $\Delta \rhd \theta_2 : L_2, t_2, \psi$ $(\diamond)$ and $\Delta \rhd \theta_a : \psi_a$ are derivable.

By $(\diamond)$, we can show that $\theta_2 = L_2, t_2, \theta$ such that

a) $\theta(L_2) = L_2$ and $\theta(U_2) = U_2$ for some L_2 and U_2.

b) $\Delta \rhd \theta : \psi$

By Thoerem 66.2 on the first premise using $(\star)$ and (b), we obtain

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_1[\theta\theta_a]}_{L_1[\theta\theta_a]} |t_1| :^c A_1[\theta\theta_a] \xrightarrow{\text{exec}(L_e[\theta\theta_a],U_e[\theta\theta_a])} A_2[\theta\theta_a] \tag{B.48}$$

By Theorem 82.2 on the first premise and $(\star)$, we get

c) $\text{FIV}(A_1 \xrightarrow{\text{exec}(L_e,U_e)} A_2, L_1, U_1, \Phi_1) \subseteq dom(\Delta,\psi,\psi_a)$.

By $(\star)$ and c), we get

d) $\text{FIV}(\Phi_a,\Omega,A_2,L_2,U_2) \subseteq L_2, U_2, dom(\Delta,\psi,\psi_a)$.

By Thoerem 66.2 on the third premise using (c), (d) and $(\star\star)$, we obtain

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_2}_{L_2} |t_2| :^c A_1[\theta\theta_a] \tag{B.49}$$

Then, by using **c-app** rule using eqs. (B.48) and (B.49), we can show that $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_1[\theta\theta_a]+U_e[\theta\theta_a]+U_2}_{L_1[\theta\theta_a]+L_e[\theta\theta_a]+L_2} |t_1|\,|t_2| :^c A_2[\theta_a,\theta_2]$.

Note that we have $L_2[\theta_a,\theta_2] = L_2$ and $U_2[\theta_a,\theta_2] = U_2$. Moreover, $U_1[\theta_a,\theta_2] = U_1[\theta\theta_a]$ and $L_1[\theta_a,\theta_2] = L_1[\theta\theta_a]$ (similarly for L_e and U_e) since L_2, U_2 are fresh variables.

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow \forall i \overset{\mathbf{exec}(L_e,U_e)}{::} S.\, A' \Rightarrow [\psi], L, U, \Phi \qquad \Delta \vdash I :: S}{\Delta;\psi_a;\Phi_a;\Omega \vdash t\,[I] \uparrow A'\{I/i\} \Rightarrow [\psi], L + L_e[I/i], U + U_e[I/i], \Phi}\ \textbf{alg-u-iApp-}\uparrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{(U+U_e[I/i])[\theta\theta_a]}_{(L+L_e[I/i])[\theta\theta_a]} |t|\,[I] :^c (A'\{I/i\})[\theta\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Omega) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models \Phi_2[\theta\,\theta_a]$ $(\star\star)$ such that $\Delta \rhd \theta : \psi$ $(\diamond)$ and $\Delta \rhd \theta_a : \psi_a$ are derivable.
By Thoerem 66.2 on the first premise using $(\star)$ and $(\diamond)$, we obtain

$$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta\,\theta_a]}_{L[\theta\,\theta_a]} |t| :^c \forall i^{\text{exec}(L_e[\theta\,\theta_a], U_e[\theta\,\theta_a])}_{::} S.\, A'[\theta\,\theta_a] \tag{B.50}$$

Then, by **c-iApp** rule using eq. (B.50) and the second premise, we can conclude that
$\Delta; \Phi_a[\theta_a]; \Omega[\theta_a] \vdash^{U[\theta\,\theta_a]+U_e[\theta\,\theta_a][I/i]}_{L[\theta\,\theta_a]+L_e[\theta\,\theta_a][I/i]} |t|\ [I] :^c A'[\theta\,\theta_a]\{I/i\}$.

Proof of Thoerem 66.3:

Case

$$\frac{D' \in \textbf{fresh}(\mathbb{R}) \qquad \Delta; D', \psi_a; \Phi_a; \Box\Gamma \vdash t \ominus t \downarrow \tau, D' \Rightarrow \Phi}{\Delta; \psi_a; \Phi_a; \Gamma', \Box\Gamma \vdash \textbf{NC}\ t \ominus \textbf{NC}\ t \downarrow \Box\tau, D \Rightarrow 0 \doteq D \wedge (\exists D' :: \mathbb{R}.\Phi)}\ \textbf{alg-r-nochange-}\downarrow$$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma'[\theta_a], \Box\Gamma[\theta_a] \vdash \text{NC}\ t \ominus \text{NC}\ t \lesssim D[\theta_a] : \Box\tau[\theta_a]$.
By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma', \Box\Gamma, \tau, D) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta; \Phi_a[\theta_a] \models (0 \doteq D \wedge \exists D' :: \mathbb{R}.\Phi)[\theta_a]$ $(\star\star)$
Using $(\star)$ and the first premise, we can show that

a) $\text{FIV}(\Phi_a, \Gamma, \tau, D') \subseteq D', dom(\Delta, \psi_a)$.

Using $(\star)$, $(\star\star)$'s derivation must be in a form such that we have

b) $\Delta; \Phi_a[\theta_a] \models 0 \doteq U[\theta_a]$

c) $\Delta \vdash D' :: \mathbb{R}$ for some D'

d) $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a, D' \mapsto D']$

By Thoerem 66.3 on the premise using a), d) and $(\star)$, we can show that

$$\Delta; \Phi_a[\theta_a]; \Box\Gamma[\theta_a] \vdash |t| \ominus |t| \lesssim D' : \tau[\theta_a] \tag{B.51}$$

By the **c-nochange** rule using eq. (B.51), we obtain
$\Delta; \Phi_a[\theta_a]; \Gamma'[\theta_a], \Box\Gamma[\theta_a] \vdash \text{NC}\,|t| \ominus \text{NC}\,|t| \lesssim 0 : \Box\tau[\theta_a]$.
By the **c-r-⊑** rule using this and b), we obtain
$\Delta; \Phi_a[\theta_a]; \Gamma'[\theta_a], \Box\Gamma[\theta_a] \vdash \text{NC}\ t \ominus \text{NC}\ t \lesssim D[\theta_a] : \Box\tau[\theta_a]$.

Case

$$\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \mathbf{list}[n]^{\alpha}\,\tau \Rightarrow [\psi], D_1, \Phi_e$$
$$D_2 \in \mathbf{fresh}(\mathbb{R}) \qquad \Delta; D_2, \psi, \psi_a; n \doteq 0 \wedge \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \tau', D_2 \Rightarrow \Phi_1$$
$$i :: \mathbb{N}, \Delta; D_2, \psi, \psi_a; n \doteq i+1 \wedge \Phi_a; h : \square\tau, tl : \mathbf{list}[i]^{\alpha}\,\tau, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau', D_2 \Rightarrow \Phi_2$$
$$i :: \mathbb{N}, \beta :: \mathbb{N}, \Delta; D_2, \psi, \psi_a; n \doteq i+1 \wedge \alpha \doteq \beta + 1 \wedge \Phi_a; h : \tau, tl : \mathbf{list}[i]^{\beta}\,\tau, \Gamma$$
$$\vdash t_3 \ominus t_3' \downarrow \tau', D_2 \Rightarrow \Phi_3 \qquad \Phi_{body} =$$
$$(n \doteq 0 \to \Phi_1) \wedge (\forall i :: \mathbb{N}.(n \doteq i+1) \to (\Phi_2 \wedge \forall \beta :: \mathbb{N}.(\alpha \doteq \beta + 1) \to \Phi_3)) \wedge D_1 + D_2 \doteq D$$

$$\Delta;\psi_a;\Phi_a;\Gamma$$
$$\mathbf{case}\ t\ \mathbf{of\ nil}\ \to t_1 \qquad \mathbf{case}\ t'\ \mathbf{of\ nil}\ \to t_1'$$
$$\vdash \mid h ::_{NC} tl \to t_2 \quad \ominus \quad \mid h ::_{NC} tl \to t_2' \quad \downarrow \tau', D \Rightarrow \exists(\psi).(\Phi_e \wedge \exists D_2 :: \mathbb{R}.\Phi_{body})$$
$$\mid h ::_{C} tl \ \to t_3 \qquad \mid h ::_{C} tl \ \to t_3'$$

alg-r-caseL-↓

TS:

$$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash \begin{array}{l}\text{case } t \text{ of nil } \to |t_1| \\ \mid h ::_N tl \to |t_2| \\ \mid h ::_C tl \to |t_3|\end{array} \quad \ominus \quad \begin{array}{l}\text{case } t' \text{ of nil } \to |t_1'| \\ \mid h ::_N tl \to |t_2'| \\ \mid h ::_C tl \to |t_3'|\end{array} \quad \lesssim D[\theta_a] : \tau'[\theta_a]$$

By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma, \tau', t) \subseteq dom(\Delta, \psi_a)$ $(\star)$ and
$\Delta;\Phi_a[\theta_a] \models (\exists(\psi).\Phi_e \wedge \exists t_2 :: \mathbb{R}.\Phi_{body})[\theta_a]$ $(\star\star)$
By Theorem 82 using the first premise and $(\star)$, we get
$\text{FIV}(\text{list}[n]^{\alpha}\,\tau, t_1, \Phi_e) \subseteq dom(\Delta, \psi; \psi_a)$ $(\diamond)$.
Using $(\star)$ and $(\diamond)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \rhd \theta : \psi$
b) $\Delta;\Phi_a[\theta_a] \models \Phi_e[\theta\,\theta_a]$
c) $\Delta \vdash D_2 :: \mathbb{R}$
d) $\Delta; n[\theta\,\theta_a] \doteq 0 \wedge \Phi_a[\theta_a] \models \Phi_1[\theta\,\theta_a]$
e) $i :: S, \Delta; n[\theta\,\theta_a] \doteq i+1 \wedge \Phi_a[\theta_a] \models \Phi_2[\theta_a, \theta, D_2 \mapsto D_2]$
f) $i :: S, \beta :: S, \Delta; n[\theta\,\theta_a] \doteq i+1 \wedge \alpha[\theta\,\theta_a] \doteq \beta + 1 \wedge \Phi_a[\theta_a] \models \Phi_3[\theta_a, \theta, D_2 \mapsto D_2]$
g) $\Delta;\Phi_a[\theta_a] \models D_1[\theta\,\theta_a] + D_2 \doteq D[\theta_a]$

By Thoerem 66.4 on the first premise using b) and $(\star)$, we can show that

$$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D_1[\theta\,\theta_a] : \text{list}[n[\theta\,\theta_a]]^{\alpha[\theta\,\theta_a]}\,\tau[\theta\,\theta_a] \tag{B.52}$$

By Thoerem 66.3 on the second premise using d) and ($\star$), we can show that

$$\Delta; n[\theta\,\theta_a] \doteq 0 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_1| \ominus |t_1'| \lesssim D_2 : \tau'[\theta\,\theta_a] \tag{B.53}$$

By Thoerem 66.3 on the third premise using e) and ($\star$), we can show that

$$i :: S, \Delta; n[\theta\,\theta_a] \doteq i+1 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_2| \ominus |t_2'| \lesssim D_2 : \tau'[\theta\,\theta_a] \tag{B.54}$$

By Thoerem 66.3 on the fourth premise using f) and ($\star$), we can show that

$$i :: S, \beta :: S, \Delta; n[\theta\,\theta_a] \doteq i+1 \wedge \alpha[\theta\,\theta_a] \doteq \beta + 1 \wedge \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash |t_3| \ominus |t_3'| \lesssim D_2 : \tau'[\theta\,\theta_a] \tag{B.55}$$

Then by **c-r-caseL** rule using eqs. (B.52) to (B.55), we can show that

$$\begin{array}{lll} & \text{case } t \text{ of nil } \to |t_1| & \text{case } t' \text{ of nil } \to |t_1'| \\ \Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash & h ::_N tl \to |t_2| & \ominus h ::_N tl \to |t_2'| \qquad \lesssim D_1[\theta\,\theta_a] + D_2 : \tau'[\theta_a] \\ & \mid h ::_C tl \to |t_3| & \mid h ::_C tl \to |t_3'| \end{array}$$

We conclude by applying **c-r-$\sqsubseteq$** rule to this using g).

Case

$$\frac{\begin{array}{c} D_1, D_2 \in \textbf{fresh}(\mathbb{R}) \qquad i \in \textbf{fresh}(\mathbb{N}) \qquad \Delta; D_1, \psi_a; \Phi_a; \Gamma \vdash t_1 \ominus t_1' \downarrow \square\tau, D_1 \Rightarrow \Phi_1 \\ \Delta; i, D_2, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow \textbf{list}[i]^\alpha\,\tau, D_2 \Rightarrow \Phi_2 \qquad \Phi_2' = \Phi_2 \wedge n \doteq (i+1) \wedge D_1 + D_2 \doteq D \end{array}}{\begin{array}{c} \Delta; \psi_a; \Phi_a; \Gamma \\ \vdash \textbf{cons}_{NC}(t_1, t_2) \ominus \textbf{cons}_{NC}(t_1', t_2') \downarrow \textbf{list}[n]^\alpha\,\tau, D \Rightarrow \exists D_1 :: \mathbb{R}.(\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2') \end{array}}$$

alg-r-consNC-$\downarrow$

TS: $\Delta; \Phi_a[\theta_a]; \Gamma[\theta_a] \vdash \text{cons}_{NC}(|t_1|, |t_2|) \ominus \text{cons}_{NC}(|t_1'|, |t_2'|) \lesssim D[\theta_a] : \text{list}[n[\theta_a]]^{\alpha[\theta_a]}\,\tau[\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma, \text{list}[n]^\alpha\,\tau, D) \subseteq dom(\Delta, \psi_a)$ ($\star$) and

$\Delta; \Phi_a[\theta_a] \models (\exists D_1 :: \mathbb{R}.\Phi_1 \wedge \exists D_2 :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2')[\theta_a]$ ($\star\star$)

Using ($\star$), ($\star\star$)'s derivation must be in a form such that we have

a) $\Delta \vdash D_1 :: \mathbb{R}$

b) $\Delta \vdash D_2 :: \mathbb{R}$

c) $\Delta; \Phi_a[\theta_a] \models \Phi_1[\theta_a, D_1 \mapsto D_1]$

d) $\Delta \vdash I :: \mathbb{N}$

e) $\Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a, D_2 \mapsto D_2, i \mapsto I]$

f) $\Delta;\Phi_a[\theta_a] \models (I+1) \doteq n[\theta_a]$

g) $\Delta;\Phi_a[\theta_a] \models (D_1 + D_2) \doteq D[\theta_a]$

By Thoerem 66.3 on the third premise using ($\star$) and c), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t_1| \ominus |t_1'| \lesssim D_1 : \square\tau[\theta_a] \tag{B.56}$$

By Thoerem 66.3 on the fourth premise using ($\star$) and e), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t_2| \ominus |t_2'| \lesssim D_2 : \text{list}[I]^{\alpha[\theta_a]}\,\tau[\theta_a] \tag{B.57}$$

By **c-r-cons1** typing rule using eqs. (B.56) and (B.57), we obtain
$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash \text{cons}_{NC}(|t_1|,|t_2|) \ominus \text{cons}_{NC}(|t_1'|,|t_2'|) \lesssim D_1 + D_2 : \text{list}[I+1]^{\alpha[\theta_a]}\,\tau[\theta_a]$.
We conclude by applying **c-r-⊑** rule to this using f) and g).

Case

$$\frac{\Delta;\psi_a;\Phi;|\Gamma|1 \vdash t_1 \uparrow A_1 \xrightarrow{\textbf{exec}(L_e,U_e)} A_2 \Rightarrow [\psi], L_1, U_1, \Phi_1 \qquad D_2 \in \textbf{fresh}(\mathbb{R}) \qquad \Delta; D_2, \psi, \psi_a; \Phi_a; \Gamma \vdash t_2 \ominus t_2' \downarrow U(A_1, A_2'), t_2 \Rightarrow \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_2' \downarrow U(A_2, A_2'), D \Rightarrow \exists(\psi).\Phi_1 \wedge \exists t_2 :: \mathbb{R}.\Phi_2 \wedge U_1 + D_2 + U_e + c_{app} \doteq D}\ \textbf{alg-r-app-t}\downarrow$$

TS: $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_1|\,|t_2| \ominus |t_2'| \lesssim D[\theta_a] : U(A[\theta_a], A'[\theta_a])$.
By the main assumptions, we have $\text{FIV}(\Phi_a, \Gamma, U(A, A'), t) \subseteq dom(\Delta, \psi_a)$ ($\star$) and
$\Delta;\Phi_a[\theta_a] \models (\exists(\psi).\Phi_1 \wedge (\exists D_2 :: \mathbb{R}.\Phi_2 \wedge D_1 + D_2 + D_e + c_{app} \doteq D) \wedge \Phi_3)[\theta_a]$ ($\star\star$) such that
$\Delta \rhd \theta_a : \psi_a$ is derivable.
By Theorem 82 using ($\star$) and the first premise, we get
$\text{FIV}(A_1 \xrightarrow{\text{exec}(L_e,U_e)} A_2, L_1, U_1, \Phi_1) \subseteq dom(\Delta, \psi; \psi_a)$ ($\diamond$).
Using ($\star$) and ($\diamond$), ($\star\star$)'s derivation must be in a form such that we have

a) $\Delta \rhd \theta : \psi$

b) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta\,\theta_a]$

c) $\Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a, \theta, D_2 \mapsto D_2]$

d) $\Delta;\Phi_a[\theta_a] \models \Phi_3[\theta\,\theta_a]$

e) $\Delta;\Phi_a[\theta_a] \models D_1[\theta\,\theta_a] + D_e[\theta\,\theta_a] + D_2 + c_{app} \doteq D[\theta_a]$

By Thoerem 66.2 on the first premise using ($\star$), a) and b), we can show that

$$\Delta;\Phi_a[\theta_a];|\Gamma[\theta_a]|1 \vdash^{U_1[\theta\,\theta_a]}_{L_1[\theta\,\theta_a]} |t_1| :^c A_1[\theta\,\theta_a] \xrightarrow{\text{exec}(L_e[\theta\,\theta_a],U_e[\theta\,\theta_a])} A_2[\theta\,\theta_a] \tag{B.58}$$

From ($\star$) and ($\diamond$), we can show that

f) $\text{FIV}(\Phi_a,\Gamma,U\,(A_1,A'),t_2) \subseteq t_2, dom(\Delta,\psi;\psi_a)$

By Thoerem 66.3 on the third premise using c), ($\star$) and ($\diamond$), we obtain

$$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_2| \ominus |t_2'| \lesssim D_2 : U\,(A_1[\theta\,\theta_a], A'[\theta\,\theta_a]) \tag{B.59}$$

By Thoerem 74 on the fourth premise using ($\diamond$), ($\star$) and (d), we obtain

$$\Delta;\Phi_a[\theta_a] \models^{\mathsf{A}} A_2[\theta\,\theta_a] \sqsubseteq A[\theta_a] \tag{B.60}$$

By **U** subtyping rule using eq. (B.60) and **u-refl** subtyping rule, we obtain

$$\Delta;\Phi_a[\theta_a] \models U\,(A_2[\theta\,\theta_a], A'[\theta_a]) \sqsubseteq U\,(A[\theta_a], A'[\theta_a]) \tag{B.61}$$

Then by the **c-r-app-e** rule using eqs. (B.58) and (B.59), we can show that
$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_1|\,|t_2| \ominus |t_2'| \lesssim U_1[\theta\,\theta_a] + U_e[\theta\,\theta_a] + D_2 + c_{app} : U\,(A_2[\theta_a], A'[\theta_a])$.
Applying **c-r-$\sqsubseteq$** rule to this using (e) and eq. (B.61), we can conclude as
$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_1|\,|t_2| \ominus |t_2'| \lesssim D[\theta_a] : U\,(A[\theta_a], A'[\theta_a])$.

Proof of Thoerem 66.4:

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi \qquad \Delta;\Phi_a \vdash \tau\ \mathsf{wf} \qquad \mathbf{FIV}(\tau, D) \in \Delta}{\Delta;\psi_a;\Phi_a;\Gamma \vdash (t:\tau, D) \ominus (t':\tau, D) \uparrow \tau \Rightarrow [\cdot], D, \Phi}\ \textbf{alg-r-anno-}\uparrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |(t:\tau,t)| \ominus |(t':\tau,t)| \lesssim D[\theta_a] : \tau[\theta_a]$.
Since by definition, $\forall t.\ |(t:\_,\_)| = |t|$, STS: $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta_a] : \tau[\theta_a]$.
By the main assumptions, we have $\text{FIV}(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$ ($\star$) and
$\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ ($\star\star$)
Using the third premise, we can show that

a) $\text{FIV}(\Phi_a,\Gamma,\tau,L,U) \subseteq dom(\Delta,\psi_a)$.

By Thoerem 66.4 on the first premise using ($\star\star$) and a), we can conclude that $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta_a] : \tau[\theta_a]$.

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \uparrow \Box\tau \Rightarrow [\psi], D, \Phi}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathbf{der}\ t_1 \ominus \mathbf{der}\ t_2 \uparrow \tau \Rightarrow [\psi], D, \Phi}\ \textbf{alg-r-der-}\uparrow$$

TS: $\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta\,\theta_a] : \tau[\theta\,\theta_a]$.

By the main assumptions, we have $\mathrm{FIV}(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$ ($\star$) and $\Delta;\Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ ($\star\star$) such that $\Delta \rhd \theta : \psi$ ($\diamond$) and $\Delta \rhd \theta_a : \psi_a$ are derivable.

By Thoerem 66.4 on the first premise using ($\star$) and ($\diamond$), we obtain

$$\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta\,\theta_a] : \Box\tau[\theta\,\theta_a] \tag{B.62}$$

Then, by **c-der** rule using eq. (B.62) and the second premise, we can conclude that $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta\,\theta_a] : \tau[\theta\,\theta_a]$.

□

Theorem 67 (Completeness of the Algorithmic Typechecking in RelCost)**.** *We have the following.*

1. *Assume that* $\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^{c} A$. *Then,* $\exists e'$ *such that*
   - 1.1. $\Delta;\cdot;\Phi_a;\Omega \vdash t' \downarrow A, L, U \Rightarrow \Phi$
   - 1.2. $\Delta;\Phi_a \models \Phi$
   - 1.3. $|t'| = t$
2. *Assume that* $\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D :^{c} \tau$. *Then,* $\exists t'_1, t'_2$ *such that*
   - 2.1. $\Delta;\cdot;\Phi_a;\Gamma \vdash t'_1 \ominus t'_2 \downarrow \tau, D \Rightarrow \Phi$
   - 2.2. $\Delta;\Phi_a \models \Phi$
   - 2.3. $|t'_1| = t_1$ *and* $|t'_2| = t_2$

Proof. Proof is by simultaneous induction on the RelCostCore typing derivations.

Proof of Theorem 11.1:

Case

$$\frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} x :^{c} A}\ \textbf{c-var}$$

We can conclude as follows

$$\frac{\dfrac{\Omega(x) = A}{\Delta;\cdot;\Phi_a;\Omega \vdash x \uparrow A \Rightarrow [.], 0, 0, \top}\ \textbf{alg-u-var-}\uparrow \qquad \Delta;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A \Rightarrow \Phi \text{ by } \mathit{Lemma}\ 72}{\Delta;\psi_a;\Phi_a;\Omega \vdash x \downarrow A, 0, 0 \Rightarrow \top}$$

$$\textbf{alg-r-}\uparrow\downarrow$$

Case

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^{c} A \qquad \Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 :^{c} \textbf{list}[n]\ A}{\Delta;\Phi_a;\Omega \vdash^{U_1+U_2}_{L_1+L_2} \textbf{cons}_C(t_1, t_2) :^{c} \textbf{list}[n+1]\ A}\ \textbf{c-cons}$$

By Theorem 11.2 on the first premise, $\exists t_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Omega \vdash t_1' \downarrow A, L_1, U_1 \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1'| = t_1$

By a), we can show that for $L_1', U_1' \in \text{fresh}(\mathbb{R})$ where $\Phi_1' = \Phi_1 \wedge L_1 \doteq L_1' \wedge U_1 \doteq U_1'$

$$\Delta; L_1', U_1'; \Phi_a; \Omega \vdash t_1' \downarrow A, L_1', U_1' \Rightarrow \Phi_1' \tag{B.63}$$

By Theorem 11.2 on the second premise, $\exists t_2'$ such that

d) $\Delta;\cdot;\Phi_a;\Omega \vdash t_2' \downarrow \text{list}[n]\ A, L_2, t_2 \Rightarrow \Phi_2$

e) $\Delta;\Phi_a \models \Phi_2$

f) $|t_2'| = t_2$

By a), we can show that for $i, L_2', t_2' \in \text{fresh}(\mathbb{R})$ where $\Phi_2' = \Phi_2 \wedge L_2 \doteq L_2' \wedge U_2 \doteq U_2' \wedge i \doteq n$

$$\Delta; i, L_2', t_2'; \Phi_a; \Omega \vdash t_2' \downarrow \text{list}[i]\ A, L_2', t_2' \Rightarrow \Phi_2' \tag{B.64}$$

Then, we can conclude as follows

1.

$$\frac{\begin{array}{c} L_1', U_1', L_2', U_2' \in \text{fresh}(\mathbb{R}) \\ i \in \text{fresh}(\mathbb{N}) \qquad \Delta; L_1', t_1', \psi_a; \Phi_a; \Omega \vdash t_1' \downarrow A, L_1', U_1' \Rightarrow \Phi_1' \;\; eq.\ (\text{B.63}) \\ \Delta; i, L_2', t_2', \psi_a; \Phi_a; \Omega \vdash t_2' \downarrow \text{list}[i]\, A, L_2', U_2' \Rightarrow \Phi_2' \;\; eq.\ (\text{B.64}) \\ \Phi_2'' = (\Phi_2' \wedge n+1 \doteq (i+1) \wedge L_1 + L_2 \doteq L_1' + L_2' \wedge U_1' + U_2' \doteq U_1 + U_2) \end{array}}{\begin{array}{c} \Delta; \psi_a; \Phi_a; \Omega \\ \vdash \text{cons}_C(t_1', t_2') \downarrow \text{list}[n+1]\, A, L_1 + L_2, t_1 + t_2 \Rightarrow \exists L_1', U_1' :: \mathbb{R}.(\Phi_1' \wedge \exists L_2', U_2' :: \mathbb{R}.\exists i :: \mathbb{N}.\Phi_2'') \end{array}}$$

alg-u-cons-↓

2. Using b) and e) for the substitutions $L_i' = L_i$ and $t_i' = t_i$ for the fresh costs and $i = n$ for the size of the tail.
3. Using c) and f), $|\text{cons}_C(t_1', t_2')| = \text{cons}_C(t_1, t_2)$

Proof of Theorem 11.2:

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \square\tau}{\Delta; \Phi_a; \Gamma \vdash \textbf{der}\ t_1 \ominus \textbf{der}\ t_2 \lesssim D :^c \tau}\ \textbf{c-der}$$

By Theorem 11.2 on the premise, $\exists t_1', t_2'$ such that

a) $\Delta; \cdot; \Phi_a; \Gamma \vdash t_1' \ominus t_2' \downarrow \tau, t \Rightarrow \Phi$

b) $\Delta; \Phi_a \models \Phi$

c) $|t_1'| = t_1$ and $|t_2'| = t_2$

Then, we can conclude by using a), b) and c) as follows:

$$\frac{\Delta; \cdot; \Phi_a; \Gamma \vdash t_1' \ominus t_2' \downarrow \square\tau, D \Rightarrow \Phi}{\Delta; \cdot; \Phi_a; \Gamma \vdash \text{der}\ t_1' \ominus \text{der}\ t_2' \downarrow \tau, D \Rightarrow \Phi}\ \textbf{alg-r-der-↓}\ \text{and}$$

1.

$$\dfrac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \square\tau, D \Rightarrow \Phi}{\Delta;\cdot;\Phi_a;\Gamma \vdash \mathrm{der}\ t_1' \ominus \mathrm{der}\ t_2' \downarrow \tau, D \Rightarrow \Phi}\ \textbf{alg-r-der-}\downarrow}{\Delta;\cdot;\Phi_a;\Gamma \vdash (\mathrm{der}\ t_1' : \tau, D) \ominus (\mathrm{der}\ t_2' : \tau, D) \uparrow \tau \Rightarrow [\cdot], D, \Phi}\ \textbf{alg-r-anno-}\uparrow$$

$$\dfrac{\Delta;\Phi_a \models \tau \equiv \tau \Rightarrow \Phi' \ \text{ by } \mathit{Lemma}\ 70}{\Delta;\cdot;\Phi_a;\Gamma \vdash (\mathrm{der}\ t_1' : \tau, D) \ominus (\mathrm{der}\ t_2' : \tau, D) \downarrow \tau, D \Rightarrow \Phi \wedge \Phi' \wedge D \leq D}\ \textbf{alg-r-}\uparrow\downarrow$$

2. By c), $|(\mathrm{der}\ t_i' : \tau, D)| = \mathrm{der}\,|t_i'|$.

3. By b) and Lemma 70.

Case

$$\dfrac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 :^c A \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 :^c A}{\Delta;\Phi_a;\Gamma \vdash \textbf{switch}\ t_1 \ominus \textbf{switch}\ t_2 \lesssim U_1 - L_2 :^c U\ A}\ \textbf{c-switch}$$

By Theorem 11.1 on the first premise , $\exists t_1'$ such that

a) $\Delta;\cdot;\Phi_a;|\Gamma|1 \vdash t_1' \downarrow A_1, L_1, U_1 \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1'| = t_1$

By a), we can show that for $L_1', t_1' \in \mathrm{fresh}(\mathbb{R})$ where $\Phi_1' = \Phi_1 \wedge L_1 \doteq L_1' \wedge t_1 \doteq t_1'$

$$\Delta; L_1', t_1'; \Phi_a; |\Gamma|1 \vdash t_1' \downarrow A_1, L_1', t_1' \Rightarrow \Phi_1' \tag{B.65}$$

By Theorem 11.1 on the second premise , $\exists t_2'$ such that

d) $\Delta;\cdot;\Phi_a;|\Gamma|2 \vdash t_2' \downarrow A_2, L_2, U_2 \Rightarrow \Phi_2$

e) $\Delta;\Phi_a \models \Phi_2$

f) $|t_2'| = t_2$

By d), we can show that for $L_2', U_2' \in \mathrm{fresh}(\mathbb{R})$ where $\Phi_2' = \Phi_2 \wedge L_2 \doteq L_2' \wedge U_2 \doteq U_2'$

$$\Delta; L_2', U_2'; \Phi_a; |\Gamma|2 \vdash t_2' \downarrow A_2, L_2', U_2' \Rightarrow \Phi_2' \tag{B.66}$$

Then, we can conclude as follows

1. By using eqs. (B.65) and (B.66):

$$\frac{L_1', U_1', L_2', U_2' \in \text{fresh}(\mathbb{R}) \qquad \Delta; L_1', U_1', \psi_a; \Phi; |\Gamma| \vdash t_1' \downarrow A, L_1', U_1' \Rightarrow \Phi_1' \qquad \Delta; L_2', U_2', \psi_a; \Phi; |\Gamma| \vdash t_2' \downarrow A, L_2', U_2' \Rightarrow \Phi_2'}{\Delta; \psi_a; \Phi_a; \Gamma \vdash \text{switch } t_1' \ominus \text{switch } t_2' \downarrow D, U\,A \Rightarrow \exists L_1, U_1 :: \mathbb{R}.(\Phi_1' \wedge \exists L_2', U_2' :: \mathbb{R}.\Phi_2' \wedge U_1' - L_2' \doteq D)}$$

alg-r-switch$\downarrow$.

2. By using b) and e) and the substitutions $L_i' = L_i$ and $U_i' = U_i$ for the fresh costs where $U = U_1 - L_2$.

3. By c) and f), we get $|\text{switch } t_i'| = \text{switch } |t_i|$

Case

$$\frac{\Delta; \Phi_a; \Gamma \vdash t \ominus t' \lesssim D :^c \tau \qquad \Delta; \Phi_a \models \tau \equiv \tau' \qquad \Delta; \Phi_a \models t \leq U'}{\Delta; \Phi_a; \Gamma \vdash t \ominus t' \lesssim D' :^c \tau'}\ \textbf{c-r-}\equiv$$

By Theorem 11.2 on the first premise, $\exists t_1', t_2'$ such that

a) $\Delta; \cdot; \Phi_a; \Gamma \vdash t_1' \ominus t_2' \downarrow \tau, t \Rightarrow \Phi_1$

b) $\Delta; \Phi_a \models \Phi_1$

c) $|t_1'| = t$ and $|t_2'| = t'$

By Theorem 79 on the second premise,

d) $\Delta; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_2$

e) $\Delta; \Phi_a \models \Phi_2$.

Then, we can conclude as follows

1. By using a) and d)

$$\frac{\dfrac{\Delta; \cdot; \Phi_a; \Gamma \vdash t_1' \ominus t_2' \downarrow \tau, D \Rightarrow \Phi_1}{\Delta; \cdot; \Phi_a; \Gamma \vdash (t_1' : \tau, D) \ominus (t_2' : \tau, D) \uparrow \tau \Rightarrow [\cdot], D, \Phi_1}\ \textbf{alg-r-anno-}\uparrow \qquad \Delta; \Phi_a \models \tau \equiv \tau' \Rightarrow \Phi_2}{\Delta; \cdot; \Phi_a; \Gamma \vdash (t_1' : \tau, D) \ominus (t_2' : \tau, D) \downarrow \tau', D' \Rightarrow \Phi_1 \wedge \Phi_2 \wedge D \leq D'}$$

alg-r-$\uparrow\downarrow$

2. By using b), e) and the third premise, we can show that $\delta; \Phi_a \models \Phi_1 \wedge \Phi_2 \wedge D \leq D'$

3. By c), $|(t_1' : \tau, D)| = t$ and $(t_2' : \tau, D) = t'$.

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \tau_1 \xrightarrow{\mathbf{diff}(D)} \tau_2 \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau_1}{\Delta;\Phi_a;\Gamma \vdash t_1\ t_2 \ominus t_1'\ t_2' \lesssim t_1 + t_2 + t :^c \tau_2}\ \textbf{c-r-app}$$

By Theorem 11.2 on the first premise, $\exists t_1, t_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1 \xrightarrow{\mathrm{diff}(t)} \tau_2, t_1 \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1| = t_1$ and $|t_1'| = t_1'$

By Theorem 11.2 on the second premise, $\exists t_2, t_2'$ such that

d) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1, t_2 \Rightarrow \Phi_2$

e) $\Delta;\Phi_a \models \Phi_2$

f) $|t_2| = t_2$ and $|t_2'| = t_2'$

By d), we can show that for $t_2' \in \mathrm{fresh}(\mathbb{R})$ where $\Phi_2' = \Phi_2 \wedge t_2 \doteq t_2'$

$$\Delta; t_2';\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1, t_2' \Rightarrow \Phi_2' \tag{B.67}$$

Then, we can conclude as follows

1.

$$\frac{\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, t_1 \Rightarrow \Phi_1}{\Delta;\cdot;\Phi_a;\Gamma \vdash (t_1 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, D_1) \ominus (t_1' : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, D_1) \uparrow \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2 \Rightarrow [\cdot], D_1, \Phi_1} \qquad D_2' \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta; D_2';\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau_1, D_2' \Rightarrow \Phi_2'}{\Delta;\cdot;\Phi_a;\Gamma \vdash E_1 \ominus E_2 \uparrow \tau_2 \Rightarrow [D_2'], D_1 + D + D_2', \Phi_1 \wedge \Phi_2'}\ \textbf{c-r-app}}{\Delta;\cdot;\Phi_a;\Gamma \vdash E_1 \ominus E_2 \downarrow \tau_2, D_1 + D + D_2 \Rightarrow \exists D_2' :: \mathbb{R}.\Phi_1 \wedge \Phi_2' \wedge D_1 + D + D_2' \leq D_1 + D + D_2}\ \textbf{alg-r-}\uparrow\downarrow$$

where $E_1 = (t_1 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, D_1)\ t_2$ and $E_2 = (t_1' : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, D_1)\ t_2'$.

2. By using b) and e) and the substitution $D_2' = D_2$ for the fresh cost.
3. Using c) and f), $|(t_1 : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, D_1)\ t_2| = t_1\ t_2$ and $|(t_1' : \tau_1 \xrightarrow{\mathrm{diff}(D)} \tau_2, D_1)\ t_2'| = t_1'\ t_2'$.

Case

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \exists i{::}S.\tau_1 \\ i :: S, \Delta;\Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau_2 \qquad i \notin FV(\Phi_a;\Gamma,\tau_2,D_2)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \mathbf{unpack}\ t_1\ \mathbf{as}\ (x,i)\ \mathbf{in}\ t_2 \ominus \mathbf{unpack}\ t_1'\ \mathbf{as}\ (x,i)\ \mathbf{in}\ t_2' \lesssim D_1 + D_2 :^c \tau_2}\ \textbf{c-r-unpack1}$$

By Theorem 11.2 on the first premise, $\exists t_1, t_1'$ such that

a) $\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \exists i{::}S.\tau_1, t_1 \Rightarrow \Phi_1$

b) $\Delta;\Phi_a \models \Phi_1$

c) $|t_1| = t_1$ and $|t_1'| = t_1'$

By Theorem 11.2 on the second premise, $\exists t_2, t_2'$ such that

d) $i :: S, \Delta;\cdot;\Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2, D_2 \Rightarrow \Phi_2$

e) $i :: S, \Delta;\Phi_a \models \Phi_2$

f) $|t_2| = t_2$ and $|t_2'| = t_2'$

By d), we can show that for $D_2' \in \mathrm{fresh}(\mathbb{R})$ where $\Phi_2' = \Phi_2 \wedge D_2 \doteq D_2'$

$$i :: S, \Delta; D_2'; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2, D_2' \Rightarrow \Phi_2' \tag{B.68}$$

Then, we can conclude as follows

1.

$$\frac{\begin{array}{c}\dfrac{\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \exists i{::}S.\tau_1, D_1 \Rightarrow \Phi_1 \ \ (a)}{\Delta;\cdot;\Phi_a;\Gamma \vdash E_1 \ominus E_1' \uparrow \exists i{::}S.\tau_1 \Rightarrow [\cdot], D_1, \Phi_1}\ \textbf{alg-r-anno-}\uparrow \\ i :: S, \Delta; D_2'; \Phi_a; x : \tau_1, \Gamma \vdash t_2 \ominus t_2' \downarrow \tau_2, t_2' \Rightarrow \Phi_2' \ \ (eq.\ (B.68)) \\ \Phi' = \exists D_2' :: \mathbb{R}.\Phi_1 \wedge \Phi_2' \wedge D_1 + D_2' \doteq D_1 + D_2\end{array}}{\Delta;\cdot;\Phi_a;\Gamma \vdash \mathrm{unpack}\ E_1\ \mathrm{as}\ (x,i)\ \mathrm{in}\ t_2 \ominus \mathrm{unpack}\ E_1'\ \mathrm{as}\ (x,i)\ \mathrm{in}\ t_2' \downarrow \tau_2, D_1 + D_2 \Rightarrow \Phi'}$$

alg-r-unpack-↓

where $E_1 = (t_1 : \exists i{::}S.\tau_1, D_1,)$ and $E_2 = (t_1' : \exists i{::}S.\tau_1, D_1,)$

2. By using b) and e) and the substitution $D_2' = D_2$ for the fresh cost.
3. Using c) and f), $|\text{unpack } (t_1 : \exists i{::}S.\, \tau_1, D_1) \text{ as } (x,i) \text{ in } t_2| = \text{unpack } t_1 \text{ as } (x,i) \text{ in } t_2$ and $|\text{unpack } (t_1' : \exists i{::}S.\, \tau_1, D_1) \text{ as } (x,i) \text{ in } t_2'| = \text{unpack } t_1' \text{ as } (x,i) \text{ in } t_2'$.

□

B.6 BIAREL

Terms $t ::= x \mid l \mid n \mid r \mid () \mid \Lambda.t \mid t\,[] \mid \lambda x.t \mid t\,u \mid \mathsf{return}\,t \mid \mathsf{let}\{x\} = t\,\mathsf{in}\,t_1 \mid \mathrm{inr}\,t \mid$
$\mathsf{alloc}\,t\,t \mid \mathsf{alloc}_\mathsf{b}\,t\,t \mid \mathsf{updt}\,t\,t\,t \mid \mathsf{updt}_\mathsf{b}\,t\,t\,t \mid \mathsf{read}\,t\,t \mid \mathsf{read}_\mathsf{b}\,t\,t \mid \mathrm{inl}\,t$
$\mid \mathrm{case}\,(t, x.t_1, y.t_2) \mid \mathsf{pack}\,t \mid \mathsf{fix}\,f(x).t \mid \mathsf{unpack}\,t_1\,\mathsf{as}\,x\,\mathsf{in}\,t_2 \mid \mathsf{celim}\,t$
$\mathrm{switch}\,t \mid \mathrm{NC}\,t \mid \mathrm{split}\,t\,\mathrm{with}\,C \mid \mathrm{contra}\,t \mid \mathsf{FIXEXT}\,t\,\mathrm{with}\,U(A_1, A_2)$

Values $v ::= n \mid l \mid r \mid () \mid \lambda x.t \mid \Lambda.t \mid \mathsf{return}\,t \mid \mathsf{alloc}\,t\,t \mid \mathsf{alloc}_\mathsf{b}\,t\,t \mid \mathsf{updt}\,t\,t\,t \mid$
$\mathsf{updt}_\mathsf{b}\,t\,t\,t \mid \mathsf{read}\,t\,t \mid \mathsf{read}_\mathsf{b}\,t\,t \mid \mathrm{inl}\,v \mid \mathrm{inr}\,v \mid$
$\mathsf{let}\{x\} = t\,\mathsf{in}\,t_1 \mid \mathsf{pack}\,v \mid \Lambda.t \mid \mathsf{fix}\,f(x).t$

Figure B.112: Syntax of values and expressions of ArelCore.

Terms $t ::= x \mid l \mid n \mid r \mid () \mid \Lambda.t \mid t\,[] \mid \lambda x.t \mid t\,u \mid \mathsf{return}\,t \mid \mathsf{let}\{x\} = t\,\mathsf{in}\,t_1 \mid$
$\mathsf{alloc}\,t\,t \mid \mathsf{alloc}_\mathsf{b}\,t\,t \mid \mathsf{updt}\,t\,t\,t \mid \mathsf{updt}_\mathsf{b}\,t\,t\,t \mid \mathsf{read}\,t\,t \mid \mathsf{read}_\mathsf{b}\,t\,t \mid \mathrm{inl}\,t \mid \mathrm{inr}\,t$
$\mid \mathrm{case}\,(t, x.t_1, y.t_2) \mid \mathsf{pack}\,t \mid \mathsf{fix}\,f(x).t \mid \mathsf{unpack}\,t_1\,\mathsf{as}\,x\,\mathsf{in}\,t_2 \mid \mathsf{celim}\,t$
$\mathrm{switch}\,t \mid \mathrm{NC}\,t \mid \mathrm{split}\,t\,\mathrm{with}\,C \mid \mathrm{contra}\,t \mid (t : \tau, D) \mid (t : A, L, U)$
$\mid \mathsf{FIXEXT}\,t\,\mathrm{with}\,U(A_1, A_2)$

Values $v ::= n \mid l \mid r \mid () \mid \lambda x.t \mid \Lambda.t \mid \mathsf{return}\,t \mid \mathsf{alloc}\,t\,t \mid \mathsf{alloc}_\mathsf{b}\,t\,t \mid \mathsf{updt}\,t\,t\,t \mid$
$\mathsf{updt}_\mathsf{b}\,t\,t\,t \mid \mathsf{read}\,t\,t \mid \mathsf{read}_\mathsf{b}\,t\,t \mid \mathrm{inl}\,v \mid \mathrm{inr}\,v \mid$
$\mathsf{let}\{x\} = t\,\mathsf{in}\,t_1 \mid \mathsf{pack}\,v \mid \Lambda.t \mid \mathsf{fix}\,f(x).t$

Figure B.113: Syntax of values and expressions of BiARel.

$\boxed{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim D : \tau}$ Expressions $t_1 \ominus t_2$ are embedded into $t_1^* \ominus t_1^*$ with the relational type τ and the relational cost D.

$\boxed{\Sigma;\Delta;\Phi_a;\Omega \vdash_L^U t \rightsquigarrow t^* : A}$ Expression t is embedded into $t*$ with the unary type A and the minimum and maximum execution costs L and U, respectively.

General rules

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma| \vdash_{L_1}^{U_1} t_1 \rightsquigarrow t_1^* : A \qquad \Sigma;\Delta;\Phi_a;|\Gamma| \vdash_{L_2}^{U_2} t_2 \rightsquigarrow t_2^* : A}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow \text{switch } t_1^* \ominus \text{switch } t_2^* \lesssim U_1 - L_2 : U\ A}\ \textbf{e-switch}$$

Subsumption

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t \rightsquigarrow t^* : A \qquad \Delta;\Phi_a \models A \sqsubseteq A' \qquad \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'}{\Delta;\Phi_a;\Omega \vdash_{k'}^{t'} e \rightsquigarrow e^* : A'}\ \textbf{e-u-sub}$$

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau \\ \Delta;\Phi_a \models \tau \sqsubseteq \tau' \qquad t' = \text{coerce}_{\tau,\tau'} \qquad \Delta;\Phi_a \models D \leq D'\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t'\ t_1^* \ominus t'\ t_2^* \lesssim D' : \tau'}\ \textbf{e-r-sub}$$

Variables x

$$\frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash_0^0 x \rightsquigarrow x : A}\ \textbf{e-u-var} \qquad \frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \ominus x \rightsquigarrow x \ominus x \lesssim 0 : \tau}\ \textbf{e-r-var}$$

Figure B.114: The elaboration rules of ARel, part 1

alloc

$$\frac{\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 \rightsquigarrow t_1^* : \mathrm{int}[I] \qquad \Delta;\Phi_a;\Omega \vdash_{L_2}^{U_2} t_2 \rightsquigarrow t_2^* : A \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash_0^0 \mathsf{alloc}\ t_1\ t_2 \rightsquigarrow \mathsf{alloc}\ t_1^*\ t_2^* : \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \to \mathbb{N}\}}\ \textbf{e-u-alloc}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathrm{int}[I] \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \tau \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \rightsquigarrow \mathsf{alloc}\ t_1^*\ t_2^* \ominus \mathsf{alloc}\ t_1'^*\ t_2'^* \lesssim 0 : \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}}\ \textbf{e-r-alloc}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathrm{int}[I] \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \square\tau \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \rightsquigarrow \mathsf{alloc}_\mathsf{b}\ t_1^*\ t_2^* \ominus \mathsf{alloc}_\mathsf{b}\ t_1'^*\ t_2'^* \lesssim 0 : \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \emptyset\}}\ \textbf{e-r-allocb}$$

Figure B.115: The elaboration rules of ARel, part 2

read

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 \rightsquigarrow t_1^* : \mathsf{Array}_\gamma[I]\ A \qquad \Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 \rightsquigarrow t_2^* : \mathrm{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{read}\ t_1\ t_2 \rightsquigarrow \mathsf{read}\ t_1^*\ t_2^* : \overset{\mathrm{exec}(L_1+L_2+L_r,U_1+U_2+U_r)}{\{P\}\ \exists\gamma.A\ \{P\}}}\ \textbf{e-u-read}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \rightsquigarrow \mathsf{read}\ t_1^*\ t_2^* \ominus \mathsf{read}\ t_1'^*\ t_2'^* \lesssim 0 : \overset{\mathrm{diff}(D_1+D_2)}{\{P\}\ \exists\gamma.\tau\ \{P\}}}\ \textbf{e-r-read}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a \models I' \le I \qquad I' \notin \beta \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \rightsquigarrow \mathsf{read}\ t_1^*\ t_2^* \ominus \mathsf{read}\ t_1'^*\ t_2'^* \lesssim 0 : \overset{\mathrm{diff}(D_1+D_2)}{\{P \star \gamma \mapsto \beta\}\ \exists\gamma.\Box\tau\ \{P \star \gamma \mapsto \beta\}}}\ \textbf{e-r-readb}$$

Figure B.116: The elaboration rules of ARel, part 3

update

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 \rightsquigarrow t_1^* : \mathsf{Array}_\gamma[I]\ A \qquad \Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 \rightsquigarrow t_2^* : \mathrm{int}[I'] \qquad \Delta;\Phi_a;\Omega \vdash^{U_3}_{L_3} t_3 \rightsquigarrow t_3^* : A \qquad \Delta;\Phi_a \models I' \leq I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Delta;\Phi_a \models I' \in \beta}{\Delta;\Phi_a;\Omega \vdash^{0}_{0} \mathsf{updt}\ t_1\ t_2\ t_3 \rightsquigarrow \mathsf{updt}\ t_1^*\ t_2^*\ t_3^* : \{P \star \gamma \mapsto \beta\}\ \exists \gamma.A\ \{P \star \gamma \mapsto \beta\}}^{\mathrm{exec}(L_1+L_2+L_3+L_u, U_1+U_2+U_3+U_u)}\ \textbf{e-u-updt}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \rightsquigarrow {t_3}^* \ominus {t_3'}^* \lesssim D_3 : \tau \qquad \Delta;\Phi_a \models I' \leq I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \rightsquigarrow \mathsf{updt}\ t_1^*\ t_2^*\ t_3^* \ominus \mathsf{updt}\ t_1'^*\ t_2'^*\ t_3'^* \lesssim 0 : \{P \star \gamma \mapsto \beta\}\ \exists \gamma.\tau\ \{P \star \beta \cup \{I'\}\}}^{\mathrm{diff}(D_1+D_2+D_3)}$$

e-r-updt

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \mathrm{int}[I'] \qquad \Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \rightsquigarrow {t_3}^* \ominus {t_3'}^* \lesssim D_3 : \square\tau \qquad \Delta;\Phi_a \models I' \leq I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \rightsquigarrow \mathsf{updt}_\mathsf{b}\ t_1^*\ t_2^*\ t_3^* \ominus \mathsf{updt}_\mathsf{b}\ t_1'^*\ t_2'^*\ t_3'^* \lesssim 0 : \{P \star \gamma \mapsto \beta\}\ \exists \gamma.\tau\ \{P \star \beta \setminus \{I'\}\}}^{\mathrm{diff}(D_1+D_2+D_3)}$$

e-r-updtb

Figure B.117: The elaboration rules of ARel, part 4

return

$$\frac{\Delta;\Phi_a;\Omega \vdash_L^U t \rightsquigarrow t^* : A}{\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L,U)} \mathsf{return}\ t \rightsquigarrow \mathsf{return}\ t^* : \{P\}\ \exists\gamma.A\ \{P\}}\ \textbf{e-u-ret}$$

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \rightsquigarrow t^* \ominus t'^* \lesssim D_3 : \tau \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ t \ominus \mathsf{return}\ t' \rightsquigarrow \mathsf{return}\ t^* \ominus \mathsf{return}\ t'^* \lesssim 0 : \{P\}\ \exists\gamma.\tau\ \{P\}}\ \textbf{e-r-ret}$$

letm

$$\frac{\Delta;\Phi_a;\Omega \vdash_{L_1}^{U_1} t_1 \rightsquigarrow t_1^* : \{P\}\ \exists\vec{\gamma}_1.A\ \{P'\} \qquad \Sigma;\Delta,\vec{\gamma}_1;\Phi_a;\Omega,x:A \vdash_{L_2}^{U_2} t_2 \rightsquigarrow t_2^* : \{P'\}\ \exists\vec{\gamma}_2.A'\ \{Q\}}{\Delta;\Phi_a;\Omega \vdash_0^{\mathrm{exec}(L+L'+L_1+L_2+L_l,U+U'+U_1+U_2+U_l)} \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \rightsquigarrow \mathsf{let}\{x\} = t_1^*\ \mathsf{in}\ t_2^* : \{P\}\ \exists\_.A'\ \{Q\}}$$

e-u-letm

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \{P\}\ \exists\vec{\gamma}_1.\tau\ \{P'\}\ D_1 \\ \Sigma;\Delta,\vec{\gamma}_1:\vec{\mathbb{L}};\Phi_a;\Gamma,x:\tau \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \{P'\}\ \exists\vec{\gamma}_2.\tau'\ \{Q\}\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \rightsquigarrow \mathsf{let}\{x\} = t_1^*\ \mathsf{in}\ t_2^* \ominus \mathsf{let}\{x\} = t_1'^*\ \mathsf{in}\ t_2^* \lesssim 0 : \\ \{P\}\ \exists\_.\tau'\ \{Q\}\end{array}}$$

e-r-letm

Figure B.118: The elaboration rules of ARel, part 5

$\boxed{\Delta;\Phi_a \models \tau_1 \equiv \tau_2}$ checks whether τ_1 is equivalent to τ_2

$$\frac{\Delta;\psi_a \vDash I = I'}{\Delta;\psi_a;\Phi_a \models \mathrm{int}[I] \equiv \mathrm{int}[I']}\ \textbf{eq-int-I}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \qquad \vDash \gamma = \gamma' \qquad \Delta;\psi_a \vDash I = I'}{\Delta;\psi_a;\Phi_a \models \mathsf{Array}_{\gamma}[I]\ \tau \equiv \mathsf{Array}_{\gamma'}[I']\ \tau'}\ \textbf{eq-r-array}$$

$$\frac{\begin{array}{c}\Delta;\psi_a;\Phi_a \models \tau \equiv \tau' \qquad \Delta;\psi_a;\Phi_a \models P' \equiv P \\ \Delta;\psi_a;\Phi_a \models Q \equiv Q' \qquad \vDash \vec{\gamma}_1 = \vec{\gamma}_2\end{array}}{\Delta;\psi_a;\Phi_a \models \{P\}\ \exists \vec{\gamma}_1 \overset{\mathrm{diff}(D)}{:} \tau\ \{Q\} \equiv \{P'\}\ \exists \vec{\gamma}_2 \overset{\mathrm{diff}(D)}{:} \tau'\ \{Q'\}}\ \textbf{eq-r-monad}$$

$$\frac{\Delta;\psi_a;\Phi_a \models \tau_1 \equiv \tau_1' \qquad \Delta;\psi_a;\Phi_a \models \tau_2 \equiv \tau_2' \qquad \Delta;\psi_a \vDash D = D'}{\Delta;\psi_a;\Phi_a \models \tau_1 \xrightarrow{\mathrm{diff}\,D} \tau_2 \equiv \tau_1' \xrightarrow{\mathrm{diff}\,D'} \tau_2'}\ \textbf{eq-r-fun}$$

Figure B.119: Relational type equivalence rules of ARel.

$$\frac{\Delta;\psi_a;\Phi_a \models P \equiv P' \Rightarrow \Phi \qquad \Delta;\psi_a \vDash \beta_1 \doteq \beta_2}{\Delta;\psi_a;\Phi_a \models P \star \gamma \to \beta_1 \equiv P' \star \gamma \to \beta_2}\ \textbf{eq-r-predicate}$$

$$\frac{}{\Delta;\psi_a;\Phi_a \models \emptyset \equiv \emptyset}\ \textbf{eq-r-predicate-empty}$$

Figure B.120: Predicate equivalence rules of ARel.

$\boxed{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^{c} A}$ the unary core typing rules.

$\boxed{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D :^{c} \tau}$ the relational core typing rules.

General rules

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 :^{c} A \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 :^{c} A}{\Delta;\Phi_a;\Gamma \vdash \text{switch } t_1 \ominus \text{switch } t_2 \lesssim U_1 - L_2 :^{c} U\ A}\ \textbf{c-switch}$$

Unary Subtyping

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Omega \vdash^{U}_{L} t :^{c} A \qquad \Delta;\Phi_a \models A \sqsubseteq A' \\ \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'\end{array}}{\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t :^{c} A'}\ \textbf{c-}\sqsubseteq$$

Binary Subeffecting

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^{c} \tau \qquad \Delta;\Phi_a \models \tau \equiv \tau' \\ \Delta;\Phi_a \models D \leq D'\end{array}}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D' :^{c} \tau'}\ \textbf{c-r-}\equiv$$

Figure B.121: The typing rules of Arel-Core, part 1.

Alloc

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c \mathrm{int}[I] \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 :^c A \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{\mathrm{exec}(L_1+L_2+L_a,U_1+U_2+U_a)} \mathsf{alloc}\ t_1\ t_2 :^c \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \to \mathbb{N}\}}\ \textbf{c-alloc}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathrm{int}[I] \quad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau \\ \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \lesssim 0 :^{c}_{\mathrm{diff}(D_1+D_2)} \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}}\ \textbf{c-r-alloc}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathrm{int}[I] \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \square\tau \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}_\mathsf{b}\ t_1\ t_2 \ominus \mathsf{alloc}_\mathsf{b}\ t_1'\ t_2' \lesssim 0 :^{c}_{\mathrm{diff}(D_1+D_2)} \{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \to \emptyset\}}\ \textbf{c-r-allocB}$$

read

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c \mathsf{Array}_\gamma[I]\ A \\ \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 :^c \mathrm{int}[I'] \qquad \Delta;\Phi_a \vDash I' \le I \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{0}_{\mathrm{exec}(L_1+L_2+L_r,U_1+U_2+U_r)} \mathsf{read}\ t_1\ t_2 :^c \{P\}\ \exists\_.A\ \{P\}}\ \textbf{c-read}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathrm{int}[I'] \qquad \Delta;\Phi_a \vDash I' \le I \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \lesssim 0 :^{c}_{\mathrm{diff}(D_1+D_2)} \{P\}\ \exists\_.\tau\ \{P\}}\ \textbf{c-r-read}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathrm{int}[I'] \qquad \Delta;\Phi_a \vDash I' \le I \qquad I' \notin \beta \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}_\mathsf{b}\ t_1\ t_2 \ominus \mathsf{read}_\mathsf{b}\ t_1'\ t_2' \lesssim 0 :^{c}_{\mathrm{diff}(D_1+D_2)} \{P \star \gamma \mapsto \beta\}\ \exists\_.\square\tau\ \{P \star \gamma \mapsto \beta\}}\ \textbf{c-r-readb}$$

Figure B.122: The typing rules of Arel-Core, part 2.

update

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c \mathsf{Array}_\gamma[I]\ A \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_2}_{L_2} t_2 :^c \mathrm{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Omega \vdash^{U_3}_{L_3} t_3 :^c A \qquad \Delta;\Phi_a \vDash I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Delta;\Phi_a \vDash I' \in \beta}{\Sigma;\Delta;\Phi_a;\Omega \vdash^0_0 \mathsf{updt}\ t_1\ t_2\ t_3 :^c \{P \star \gamma \to \beta\}\ \exists\_.\mathrm{unit}\ \{P \star \gamma \to \beta\}}\ \begin{array}{l}\mathrm{U\text{-}U}\\ \mathrm{exec}(L_1+L_2+L_3+L_u, U_1+U_2+U_3+U_u)\end{array}$$

c-update

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathrm{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 :^c \tau \qquad \Delta;\Phi_a \vDash I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \lesssim 0 :^c \{P \star \gamma \mapsto \beta\}\ \exists\_.\mathrm{unit}\ \{P \star \gamma \mapsto \beta \cup \{I'\}\}}\ \mathrm{diff}(D_1+D_2+D_3)$$

c-r-update

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathrm{int}[I'] \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 :^c \Box\tau \qquad \Delta;\Phi_a \vDash I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}_\mathsf{b}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}_\mathsf{b}\ t_1'\ t_2'\ t_3' \lesssim 0 :^c \{P \star \gamma \mapsto \beta\}\ \exists\_.\mathrm{unit}\ \{P \star \gamma \mapsto \beta \setminus \{I'\}\}}\ \mathrm{diff}(D_1+D_2+D_3)$$

c-r-updateb

return

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^U_L t :^c A \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^0_0 \mathsf{return}\ t :^c \{P\}\ \exists\gamma.A\ \{P\}}\ \mathrm{exec}(L,U)\ \textbf{c-return}$$

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \tau \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Sigma;\Delta \vdash Q\ \mathsf{wf} \qquad \Sigma;\Delta \vDash P \subseteq Q}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ t_1 \ominus \mathsf{return}\ t_2 \lesssim 0 :^c \{P\}\ \exists\_.\tau\ \{Q\}}\ \mathrm{diff}(D)\ \textbf{c-r-return}$$

Figure B.123: The typing rules of Arel-Core, part 3.

letm

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U_1}_{L_1} t_1 :^c \{P\}\ \overset{\text{exec}(L,U)}{\exists \vec{\gamma}_1.A}\ \{P'\} \qquad \Sigma;\Delta,\vec{\gamma}_1;\Phi_a;\Omega,x:A \vdash^{U_2}_{L_2} t_2 :^c \{P'\}\ \overset{\text{exec}(L',U')}{\exists \vec{\gamma}_2.A'}\ \{Q\}}{\Sigma;\Delta;\Phi_a;\Omega \vdash^0_0 \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 :^c \overset{\text{exec}(L+L'+L_1+L_2+L_l,U+U'+U_1+U_2+U_l)}{\{P\}\ \exists \vec{\gamma}_1,\vec{\gamma}_2.A'}\ \{Q\}}\ \textbf{c-letm}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \{P\}\ \overset{\text{diff}(D)}{\exists \vec{\gamma}_1.\tau}\ \{P'\} \\ \Sigma;\Delta,\vec{\gamma}_1:\vec{\mathbb{L}};\Phi_a;\Gamma,x:\tau \vdash t_2 \ominus t_2' \lesssim D_2 :^c \{P'\}\ \overset{\text{diff}(D')}{\exists \vec{\gamma}_2.\sigma}\ \{Q\}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \lesssim 0 :^c \overset{\text{diff}(D+D'+D_1+D_2)}{\{P\}\ \exists \vec{\gamma}_1\vec{\gamma}_2.\sigma}\ \{Q\}}\ \textbf{c-r-letm}$$

Figure B.124: The typing rules of Arel-Core, part 4.

$\boxed{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \downarrow \tau, D \Rightarrow \Phi}$ Under the location environment Σ, the index variable environment Δ, the existential variable context Ψ_a, the current constraint environment Φ_a, the relational typing context Γ, terms t_1 and t_2 check against the input relational type τ and the relative cost D and generates the constraint Φ.

$\boxed{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \uparrow \tau \Rightarrow [\psi], D, \Phi}$ Under the location environment Σ, the index variable environment Δ, the existential variable context Ψ_a, the current constraint environment Φ_a, the relational typing context Γ, terms t_1 and t_2 synthesize the output relational type τ and the output relative cost D and generates the constraint Φ with all the new generated variables in ψ.

$\boxed{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, L, U \Rightarrow \Phi}$ Under the location environment Σ, the index variable environment Δ, the existential variable context Ψ_a, the current constraint environment Φ_a, the unary typing context Ω, term t checks against the input unary type A and its upper bound and lower bound of the execution cost specified by L and U, and generates the constraint Φ.

$\boxed{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A \Rightarrow [\psi], L, U, \Phi}$ Under the location environment Σ, the index variable environment Δ, the existential variable context Ψ_a, the current constraint environment Φ_a, the unary typing context Ω, terms t synthesizes the output unary type τ and the its upper bound and lower bound of the execution cost and generates the constraint Φ with all the new generated variables in ψ.

$\boxed{\Sigma;\Delta;\Psi_a;\Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2 \Rightarrow \Phi}$ Under the location environment Σ, the index variable environment Δ, the existential variable context Ψ_a, the current constraint environment Φ_a, checks whether A_1 is subtype of A_2 and generates constraints Φ

$\boxed{\Sigma;\Delta;\Psi_a;\Phi_a \models \tau_1 \equiv \tau_2 \Rightarrow \Phi}$ Under the location environment Σ, the index variable environment Δ, the existential variable context Ψ_a, the current constraint environment Φ_a, checks whether τ_1 is equivalent to τ_2 and generates constraints Φ

Figure B.125: Bidirectional algorithmic typing judgment explanation.

constant integer n

$$\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash \mathtt{n} \uparrow \mathrm{int} \Rightarrow [.], 0, 0, \top \ \textbf{alg-u-n-}\uparrow$$

$$\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathtt{n} \ominus \mathtt{n} \uparrow \mathrm{int} \Rightarrow [.], 0, \top \ \textbf{alg-r-n-}\uparrow$$

fix

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;f : A_1 \xrightarrow{\mathrm{exec}(L',U')} A_2, x : A_1, \Omega \vdash e \downarrow A_2, L', U' \Rightarrow \Phi}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash \mathrm{fix}\ f(x).e \downarrow A_1 \xrightarrow{\mathrm{exec}(L',U')} A_2, L, U \Rightarrow \Phi \wedge L \doteq 0 \wedge 0 \doteq U} \ \textbf{alg-u-fix-}\downarrow$$

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;f : \tau_1 \xrightarrow{\mathrm{diff}(t')} \tau_2, x : \tau_1, \Gamma \vdash t \ominus t' \downarrow \tau_2, D' \Rightarrow \Phi}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{fix}\ f(x).t \ominus \mathrm{fix}\ f(x).t' \downarrow \tau_1 \xrightarrow{\mathrm{diff}(D')} \tau_2, D \Rightarrow \Phi \wedge 0 \doteq D} \ \textbf{alg-r-fix-}\downarrow$$

variable x

$$\frac{\Omega(x) = A}{\Delta;\psi_a;\Phi_a;\Omega \vdash x \uparrow A \Rightarrow [.], 0, 0, \top} \ \textbf{alg-u-var-}\uparrow$$

$$\frac{\Gamma(x) = \tau}{\Delta;\psi_a;\Phi_a;\Gamma \vdash x \ominus x \uparrow \tau \Rightarrow [.], 0, \top} \ \textbf{alg-r-var-}\uparrow$$

switch

$$\frac{\Delta;\psi_a;\Phi_a;|\Gamma| \vdash t_1 \uparrow A \Rightarrow [\psi_1], \_, U_1, \Phi_1 \qquad \Delta;\psi_a;\Phi_a;|\Gamma| \vdash t_2 \uparrow A \Rightarrow [\psi_2], L_2, \_, \Phi_2}{\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathrm{switch}\ t_1 \ominus \mathrm{switch}\ t_2 \uparrow U\ A \Rightarrow [\psi_1, \psi_2], U_1 - L_2, \Phi_1 \wedge \Phi_2}$$

alg-r-switch↑

$$\frac{\begin{array}{c} L_1, U_1, L_2, U_2 \in \mathrm{fresh}(\mathbb{R}) \qquad \Delta; U_1, L_1, \psi_a;\Phi_a;|\Gamma| \vdash t_1 \downarrow A, L_1, U_1 \Rightarrow \Phi_1 \\ \Delta; U_2, L_2, \psi_a;\Phi_a;|\Gamma| \vdash t_2 \downarrow A, L_2, U_2 \Rightarrow \Phi_2 \end{array}}{\begin{array}{l} \Sigma;\Delta;\psi_a;\Phi_a;\Gamma \\ \vdash \mathrm{switch}\ t_1 \ominus \mathrm{switch}\ t_2 \downarrow D, U\ A \Rightarrow \exists L_1, U_1 :: \mathbb{R}.(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi_2 \wedge U_1 - L_2 \doteq D) \end{array}}$$

alg-r-switch↓

Figure B.126: Bidirectional algorithmic typing rules of ARel, part 1.

split

$$\frac{\Sigma;\Delta;\psi_a;C\wedge\Phi_a;\Omega\vdash t_1\downarrow A,L,U\Rightarrow\Phi_1 \quad \Sigma;\Delta;\psi_a;\neg C\wedge\Phi_a;\Omega\vdash t_1\downarrow A,L,U\Rightarrow\Phi_2 \quad \Delta\vdash C\ \mathsf{wf}}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega\vdash \text{split }(t_1)\text{ with }C\downarrow A,L,U\Rightarrow C\to\Phi_1\wedge\neg C\to\Phi_2}\ \textbf{alg-u-split}\downarrow$$

$$\frac{\Sigma;\Delta;\psi_a;C\wedge\Phi_a;\Gamma\vdash t_1\ominus t_1'\downarrow\tau,D\Rightarrow\Phi_1 \quad \Sigma;\Delta;\psi_a;\neg C\wedge\Phi_a;\Gamma\vdash t_1\ominus t_1'\downarrow\tau,D\Rightarrow\Phi_2 \quad \Delta\vdash C\ \mathsf{wf}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma\vdash \text{split }(t_1)\text{ with }C\ominus\text{split }(t_1')\text{ with }C\downarrow\tau,D\Rightarrow C\to\Phi_1\wedge\neg C\to\Phi_2}$$

alg-r-split$\downarrow$

contra

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a\models\bot}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega\vdash\text{contra }t\downarrow A,L,U\Rightarrow\top}\ \textbf{alg-u-contra}\downarrow$$

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a\models\bot}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma\vdash\text{contra }t\ominus\text{contra }t'\downarrow\tau,D\Rightarrow\top}\ \textbf{alg-r-contra}\downarrow$$

$\uparrow\downarrow$

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;\Omega\vdash t\uparrow A'\Rightarrow[\psi],L',U',\Phi_1 \quad \Sigma;\Delta;\psi,\psi_a;\Phi_a\models^{\mathsf{A}}A'\sqsubseteq A\Rightarrow\Phi_2}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega\vdash t\downarrow A,L,U\Rightarrow\exists(\psi).\Phi_1\wedge\Phi_2\wedge L'\le L\wedge U\le U'}\ \textbf{alg-}\uparrow\downarrow$$

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma\vdash t\ominus t'\uparrow\tau'\Rightarrow[\psi],D',\Phi_1 \quad \Sigma;\Delta;\psi,\psi_a;\Phi_a\models\tau'\equiv\tau\Rightarrow\Phi_2}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma\vdash t\ominus t'\downarrow\tau,D\Rightarrow\exists(\psi).\Phi_1\wedge\Phi_2\wedge D'\le D}\ \textbf{alg-r-}\uparrow\downarrow$$

annotation

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;\Omega\vdash t\downarrow A,L,U\Rightarrow\Phi \quad \Delta;\Phi_a\vdash^{\mathsf{A}}A\ \mathsf{wf} \quad \mathrm{FIV}(A,L,U)\in\Delta}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega\vdash(t:A,L,U)\uparrow A\Rightarrow[\cdot],L,U,\Phi}\ \textbf{alg-u-anno-}\uparrow$$

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma\vdash t\ominus t'\downarrow\tau,D\Rightarrow\Phi \quad \Delta;\Phi_a\vdash\tau\ \mathsf{wf} \quad \mathrm{FIV}(\tau,D)\in\Delta}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma\vdash(t:\tau,D)\ominus(t':\tau,D)\uparrow\tau\Rightarrow[\cdot],D,\Phi}\ \textbf{alg-r-anno-}\uparrow$$

Figure B.127: Bidirectional algorithmic typing rules of ARel, part 2.

Alloc algo

$$\frac{\begin{array}{c}\Sigma;\Delta \vdash L_1, U_1, L_2, U_2 \in \mathsf{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;L_1,U_1,\psi_a;\Phi_a;\Omega \vdash t_1 \downarrow \mathrm{int}[I], L_1, U_1 \Rightarrow \Phi_1 \\ \Sigma;\Delta;L_2,U_2,\psi_a;\Phi_a;\Omega \vdash t_2 \downarrow A, L_2, U_2 \Rightarrow \Phi_2 \\ \Phi = \Phi_2 \wedge L_1 + L_2 + L_a \doteq L \wedge U_1 + U_2 + U_a \doteq U \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Omega \\ \vdash \mathsf{alloc}\ t_1\ t_2 \downarrow \overset{\mathrm{exec}(L,U)}{\{P\}\ \exists\gamma : \mathsf{Array}_\gamma[I]\ A\ \{P \star \gamma \rightarrow \mathbb{N}\}, 0, 0 \Rightarrow \exists L_1, U_1 :: \mathbb{R}.(\Phi_1 \wedge \exists L_2, U_2 :: \mathbb{R}.\Phi)}\end{array}}$$

alg-u-alc-↓

$$\frac{\begin{array}{c}D_1, D_2 \in \mathsf{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_1,\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \mathrm{int}[I], D_1 \Rightarrow \Phi_1 \\ \Sigma;\Delta;D_2,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \tau, D_2 \Rightarrow \Phi_2 \\ \Phi = \Phi_2 \wedge D_1 + D_2 \doteq D \qquad \Sigma \vdash \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \\ \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \downarrow \overset{\mathrm{diff}(D)}{\{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \rightarrow \mathbb{N}\}, 0 \Rightarrow \exists D_1 :: \mathbb{R}.\Phi_1 \wedge (\exists D_2 :: \mathbb{R}.\Phi)}\end{array}}$$

alg-r-alc-↓

$$\frac{\begin{array}{c}D_1, D_2 \in \mathsf{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_1,\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \mathrm{int}[I], D_1 \Rightarrow \Phi_1 \\ \Sigma;\Delta;D_2,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \square\tau, D_2 \Rightarrow \Phi_2 \\ \Phi = \Phi_2 \wedge D_1 + D_2 \doteq D \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \\ \vdash \mathsf{alloc}_\mathsf{b}\ t_1\ t_2 \ominus \mathsf{alloc}_\mathsf{b}\ t_1'\ t_2' \downarrow \overset{\mathrm{diff}(D)}{\{P\}\ \exists\gamma.\ \mathsf{Array}_\gamma[I]\ \tau\ \{P \star \gamma \rightarrow \emptyset\}, 0 \Rightarrow \exists D_1 :: \mathbb{R}.\Phi_1 \wedge (\exists D_2 :: \mathbb{R}.\Phi)}\end{array}}$$

alg-r-alcB-↓

Figure B.128: Bidirectional algorithm-typing rules (alloc).

Read algo

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow \mathsf{Array}_\gamma[I]\ A \Rightarrow [\psi_1], L_1, U_1, \Phi_1 \\ \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Omega \vdash t_2 \uparrow \mathsf{int}[I'] \Rightarrow [\psi_2], L_2, U_2, \Phi_2 \qquad \Sigma;\Delta;\psi_a;\Phi_a \vDash I' \leq I \\ \Phi = \Phi_2 \wedge L_1 + L_2 + c_r \doteq L \wedge U_1 + U_2 + c_r \doteq U \qquad P = P' \star \gamma \rightarrow \_ \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash \mathsf{read}\ t_1\ t_2 \downarrow \{P\}\ \exists\_ : A\ \{P\}, 0, 0 \Rightarrow \exists(\psi_1).(\Phi_1 \wedge (\exists(\psi_2).\Phi))}\ \mathrm{exec}(L,U)$$

alg-u-read-↓

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1 \\ \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathsf{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2 \\ P = P' \star \gamma \rightarrow \_ \qquad \Sigma;\Delta;\psi_a;\Phi_a \vDash I' \leq I \qquad \Phi = \Phi_2 \wedge D_1 + D_2 \doteq D \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \downarrow \{P\}\ \exists\_.\tau\ \{P\}, 0 \Rightarrow \exists(\psi_1).\Phi_1 \wedge (\exists(\psi_2).\Phi)}\ \mathrm{diff}(D)$$

alg-r-read-↓

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1 \\ \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathsf{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2 \qquad P = P' \star \gamma \rightarrow \beta \\ \Sigma;\Delta;\psi_a;\Phi_a \vDash I' \leq I \wedge \neg(I' \in \beta) \qquad \Phi = \Phi_2 \wedge D_1 + D_2 \doteq D \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{read}_\mathsf{b}\ t_1\ t_2 \ominus \mathsf{read}_\mathsf{b}\ t_1'\ t_2' \downarrow \{P\}\ \exists\_.\tau\ \{P\}, 0 \Rightarrow \exists(\psi_1).\Phi_1 \wedge (\exists(\psi_2).\Phi)}\ \mathrm{diff}(D)$$

alg-r-readB-↓

Figure B.129: Bidirectional algorithm-typing rules (read).

Update algo

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow \mathsf{Array}_\gamma[I]\ A \Rightarrow [\psi_1], L_1, U_1, \Phi_1 \\ \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Omega \vdash t_2 \uparrow \mathrm{int}[I'] \Rightarrow [\psi_2], L_2, U_2, \Phi_2 \qquad \Sigma;\Delta;\psi_a;\Phi_a \vDash I' \le I \wedge I' \in \beta \\ L_3, U_3 \in \mathrm{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;L_3,U_3,\psi_2,\psi_1,\psi_a;\Phi_a;\Omega \vdash t_3 \downarrow A, L_3, U_3 \Rightarrow \Phi_3 \\ \Phi = \Phi_3 \wedge L_1 + L_2 + L_3 + c_r \doteq L \wedge U_1 + U_2 + U_3 + c_r \doteq U \\ P = P' \star \gamma \to \beta \qquad \Sigma;\Delta \vdash P' \ \mathsf{wf}\end{array}}{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Omega \\ \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \downarrow \{P\}\ \exists\_ : \mathrm{unit}\ \{P\}^{\mathrm{exec}(L,U)}, 0, 0 \Rightarrow \exists(\psi_1).(\Phi_1 \wedge (\exists(\psi_2).(\Phi_2 \wedge \exists L_3, U_3 :: \mathbb{R}.\Phi)))\end{array}}$$

alg-u-updt-↓

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1 \\ \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathrm{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2 \qquad \Sigma;\Delta;\psi_a;\Phi_a \vDash I' \le I \wedge \beta' = \beta \cup \{I'\} \\ D_3 \in \mathrm{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_3,\psi_2,\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \downarrow \tau, D_3 \Rightarrow \Phi_3 \\ P = P' \star \gamma \to \beta \qquad Q = P' \star \gamma \to \beta' \qquad \Phi = \Phi_2 \wedge D_1 + D_2 + D_3 \doteq D \qquad \Sigma;\Delta \vdash P' \ \mathsf{wf}\end{array}}{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \\ \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \downarrow \{P\}\ \exists\_.\mathrm{unit}\ \{Q\}^{\mathrm{diff}(D)}, 0 \Rightarrow \exists(\psi_1).(\Phi_1 \wedge (\exists(\psi_2).(\Phi_2 \wedge \exists D_3 :: \mathbb{R}.\Phi))\end{array}}$$

alg-r-updt-↓

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1 \\ \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathrm{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2 \qquad \Sigma;\Delta;\psi_a;\Phi_a \vDash I' \le I \wedge \beta' = \beta / \{I'\} \\ D_3 \in \mathrm{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_3,\psi_2,\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \downarrow \square\tau, D_3 \Rightarrow \Phi_3 \\ P = P' \star \gamma \to \beta \qquad Q = P' \star \gamma \to \beta' \qquad \Phi = \Phi_2 \wedge D_1 + D_2 + D_3 \doteq D \qquad \Sigma;\Delta \vdash P' \ \mathsf{wf}\end{array}}{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \\ \vdash \mathsf{updt}_\mathsf{b}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}_\mathsf{b}\ t_1'\ t_2'\ t_3' \downarrow \{P\}\ \exists\_.\mathrm{unit}\ \{Q\}^{\mathrm{diff}(D)}, 0 \Rightarrow \exists\psi_1.(\Phi_1 \wedge (\exists\psi_2.(\Phi_2 \wedge \exists D_3 :: \mathbb{R}.\Phi))\end{array}}$$

alg-r-updtB-↓

Figure B.130: Bidirectional algorithm-typing rules (update).

Let algo

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow^{\mathsf{exec}(L,U)} \{P\}\ \exists\vec{\gamma}_1 : A_1\ \{P'\} \Rightarrow [\psi_1], L_1, U_1, \Phi_1 \\ L_2, U_2 \in \mathsf{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;L_2,U_2,\psi_1,\psi_a;\Phi_a;\Omega \vdash t_2 \downarrow^{\mathsf{exec}(L',U')} \{P'\}\ \exists\vec{\gamma}_2 : A_2\ \{Q\}, L_2, U_2 \Rightarrow \Phi_2 \\ \Phi = \Phi_2 \wedge L + L' + L_1 + L_2 + c_l \doteq L'' \wedge U + U' + U_1 + U_2 + c_l \doteq U''\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash \mathsf{let_m}\{x\} = t_1 \ \mathsf{in}\ t_2 \downarrow^{\mathsf{exec}(L'',U'')} \{P\}\ \exists\_ : A_2\ \{Q\}, 0, 0 \Rightarrow \exists(\psi_1).\Phi_1 \wedge (\exists L_2, U_2 :: \mathbb{R}.\Phi)}$$

alg-u-let-↓

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow^{\mathsf{diff}(D)} \{P\}\ \exists\vec{\gamma}_1 : \tau_1\ \{P'\} \Rightarrow [\psi_1], D_1, \Phi_1 \\ D_2 \in \mathsf{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_2,\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow^{\mathsf{diff}(D')} \{P'\}\ \exists\vec{\gamma}_2 : \tau_2\ \{Q\}, D_2 \Rightarrow \Phi_2 \\ \Phi = \Phi_2 \wedge D_1 + D_2 + D + D' \doteq D''\end{array}}{\begin{array}{l}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \\ \vdash \mathsf{let}\{x\} = t_1 \ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1' \ \mathsf{in}\ t_2' \downarrow^{\mathsf{diff}(D'')} \{P\}\ \exists\_.\tau_2\ \{Q\}, 0 \Rightarrow \exists(\psi_1).(\Phi_1 \wedge (\exists D_2 :: \mathbb{R}.\Phi)\end{array}}$$

alg-r-let-↓

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow^{\mathsf{exec}(L,U)} \{P\}\ \exists\vec{\gamma}_1 : A_1\ \{P'\} \Rightarrow [\psi_1], L_1, U_1, \Phi_1 \\ \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Omega \vdash t_2 \uparrow^{\mathsf{exec}(L',U')} \{Q'\}\ \exists\vec{\gamma}_2 : A_2\ \{Q\} \Rightarrow [\psi_2], L_2, U_2, \Phi_2 \qquad P' = Q' \quad \Phi_3 = \Phi_1 \wedge \Phi_2\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash \mathsf{let_m}\{x\} = t_1 \ \mathsf{in}\ t_2 \uparrow^{\mathsf{exec}(L+L'+L_1+L_2+c_l, U+U'+U_1+U_2+c_l)} \{P\}\ \exists\_ : A_2\ \{Q\} \Rightarrow [\psi_1,\psi_2], 0, 0, \Phi_3}$$

alg-u-let-↑

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow^{\mathsf{diff}(D)} \{P\}\ \exists\vec{\gamma}_1 : \tau_1\ \{P'\} \Rightarrow [\psi_1], D_1, \Phi_1 \\ \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow^{\mathsf{diff}(D')} \{Q'\}\ \exists\vec{\gamma}_2 : \tau_2\ \{Q\} \Rightarrow [\psi_2], D_2, \Phi_2 \qquad Q' = P' \quad \Phi_3 = \Phi_1 \wedge \Phi_2\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{let_m}\{x\} = t_1 \ \mathsf{in}\ t_2 \ominus \mathsf{let_m}\{x\} = t_1' \ \mathsf{in}\ t_2' \uparrow^{\mathsf{diff}(D+D'+D_1+D_2)} \{P\}\ \exists\_.\tau_2\ \{Q\} \Rightarrow [\psi_1,\psi_2], 0, \Phi_3}$$

alg-r-let-↑

Figure B.131: Bidirectional algorithm-typing rules (let).

Return algo

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, L, U \Rightarrow \Phi \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash \mathsf{return}\ t \downarrow \{P\}\ \overset{\mathrm{exec}(L,U)}{\exists\_ : A_2}\ \{P\}, 0, 0 \Rightarrow \Phi}\ \textbf{alg-u-ret-}\downarrow$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \\ \Sigma;\Delta \vdash Q\ \mathsf{wf} \qquad \Sigma;\Delta \vDash P \subseteq Q\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{return}\ t \ominus \mathsf{return}\ t' \downarrow \{P\}\ \overset{\mathrm{diff}(D)}{\exists\_.\tau}\ \{Q\}, 0 \Rightarrow \Phi}\ \textbf{alg-r-ret-}\downarrow$$

Figure B.132: Bidirectional algorithmic typing rules (return).

Algorithmic Reltional type equivalence

$$\frac{}{\Sigma;\Delta;\psi_a;\Phi_a \vDash \mathrm{int}[I] \equiv \mathrm{int}[I'] \Rightarrow I \doteq I'}\ \textbf{alg-r-int-I}$$

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a \vDash \tau \equiv \tau' \Rightarrow \Phi \qquad \Sigma \vDash \gamma = \gamma'}{\Sigma;\Delta;\psi_a;\Phi_a \vDash \mathsf{Array}_{\gamma}[I]\ \tau \equiv \mathsf{Array}_{\gamma'}[I']\ \tau' \Rightarrow I \doteq I' \wedge \Phi}\ \textbf{alg-r-array}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a \vDash \tau \equiv \tau' \Rightarrow \Phi_1 \\ \Sigma;\Delta;\psi_a;\Phi_a \vDash P' \equiv P \Rightarrow \Phi_2 \qquad \Sigma;\Delta;\psi_a;\Phi_a \vDash Q \equiv Q' \Rightarrow \Phi_3 \qquad \Sigma \vDash \vec{\gamma}_1 = \vec{\gamma}_2\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a \vDash \{P\}\ \overset{\mathrm{diff}(D)}{\exists \vec{\gamma}_1 : \tau}\ \{Q\} \equiv \{P'\}\ \overset{\mathrm{diff}(D)}{\exists \vec{\gamma}_2 : \tau'}\ \{Q'\} \Rightarrow D \doteq D' \wedge \Phi_1 \wedge \Phi_2 \wedge \Phi_3}\ \textbf{alg-r-monad}$$

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a \vDash \tau_1 \equiv \tau_1' \Rightarrow \Phi_1 \qquad \Sigma;\Delta;\psi_a;\Phi_a \vDash \tau_2 \equiv \tau_2' \Rightarrow \Phi_2}{\Sigma;\Delta;\psi_a;\Phi_a \vDash \tau_1 \overset{\mathrm{diff}\,D}{\longrightarrow} \tau_2 \equiv \tau_1' \overset{\mathrm{diff}\,D'}{\longrightarrow} \tau_2' \Rightarrow D \doteq D' \wedge \Phi_1 \wedge \Phi_2}\ \textbf{alg-r-fun}$$

Figure B.133: Algorithmic relational equivalence rules of ARel.

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a \vDash P \equiv P' \Rightarrow \Phi}{\Sigma;\Delta;\psi_a;\Phi_a \vDash P \star \gamma \multimap \beta_1 \equiv P' \star \gamma \multimap \beta_2 \Rightarrow \Phi \wedge \beta_1 \doteq \beta_2}\ \textbf{predicate}$$

$$\frac{}{\Sigma;\Delta;\psi_a;\Phi_a \vDash \emptyset \equiv \emptyset \Rightarrow \top}\ \textbf{predicate-empty}$$

Figure B.134: Algorithmic predicate equivalence of ARel.

Algo Unary subtyping

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1' \sqsubseteq A_1 \Rightarrow \Phi_1 \qquad \Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_2' \Rightarrow \Phi_2}{\Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A_1 \overset{\mathrm{exec}(L,U)}{\longrightarrow} A_2 \sqsubseteq A_1' \overset{\mathrm{exec}(L',U')}{\longrightarrow} A_2' \Rightarrow \Phi_1 \wedge \Phi_2 \wedge L' < L \wedge U < U'}\ \textbf{alg-u-fun}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} A \sqsubseteq A' \Rightarrow \Phi_1 \\ \Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} P' \sqsubseteq P \Rightarrow \Phi_2 \qquad \Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} Q \sqsubseteq Q' \Rightarrow \Phi_3 \qquad \Sigma \vDash \vec{\gamma}_1 \sqsubseteq \vec{\gamma}_2\end{array}}{\Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} \{P\}\ \overset{\mathrm{exec}(L,U)}{\exists \vec{\gamma}_1 : A}\ \{Q\} \sqsubseteq \{P'\}\ \overset{\mathrm{exec}(L',U')}{\exists \vec{\gamma}_2 : A'}\ \{Q'\} \Rightarrow \Phi_1 \wedge \Phi_2 \wedge \Phi_3 \wedge L' \leq L \wedge U \leq U'}$$

$$\textbf{alg-u-monad}\quad \frac{\Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} P \sqsubseteq P' \Rightarrow \Phi}{\Sigma;\Delta;\psi_a;\Phi_a \models^{\mathsf{A}} P \star \gamma \to \beta_1 \sqsubseteq P' \star \gamma \to \beta_2 \Rightarrow \Phi \wedge \beta_1 \sqsubseteq \beta_2}\ \textbf{u-predicate}$$

Figure B.135: Bidirectional unary algorithmic subtyping rules of ARel.

$$\begin{array}{lcl}
|.| & : & \text{Expression} \to \text{Expression} \\
 & & \\
|\mathtt{n}| & = & \mathtt{n} \\
|()| & = & () \\
|x| & = & x \\
|\mathsf{FIXEXT}\ t \text{ with } U(A_1,A_2)| & = & \mathsf{FIXEXT}\ |t| \text{ with } U(A_1,A_2) \\
|\text{fix } f(x).e| & = & \text{fix } f(x).|t| \\
|t_1\ t_2| & = & |t_1|\,|t_2| \\
\vdots & & \\
|(t : A, L, U)| & = & |t| \\
|(t : \tau, D)| & = & |t|
\end{array}$$

Figure B.136: Annotation erasure of ARel.

B.6.1 Metatheory

B.6.1.1 Proof of Lemma 7

Lemma 68 (Embedding of Binary Subtyping in ARel)**.** *If* $\Delta;\Phi \models \tau \sqsubseteq \tau'$ *then* $\exists t \in \mathsf{ArelCore}$ *such that* $\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c \tau \xrightarrow{\mathit{diff}(0)} \tau'$.

Proof. Proof is by induction on the subtyping derivation. We denote the witness t of type $\tau \xrightarrow{\mathrm{diff}(0)} \tau'$ as $\mathrm{coerce}_{\tau,\tau'}$ for clarity.

Case

$$\frac{\Sigma;\Delta;\Phi_a \models \tau \sqsubseteq \tau' \quad \Delta;\Phi_a \models D \leq D' \qquad \Sigma,\vec{\gamma}_1;\Delta;\Phi_a \models Q \sqsubseteq Q' \qquad \Sigma;\Delta;\Phi_a \models P' \sqsubseteq P \qquad \vec{\gamma}_1 \subseteq \vec{\gamma}_2}{\Sigma;\Delta;\Phi_a \models \{P\}\ \exists \vec{\gamma}_1.\tau\ \{Q\} \overset{\mathrm{diff}(D)}{\sqsubseteq} \{P'\}\ \exists \vec{\gamma}_2.\tau'\ \{Q'\} \overset{\mathrm{diff}(D')}{}}\ \textbf{S-RM}$$

By IH on $(\star)$, $\exists \mathrm{coerce}_{\tau,\tau'}\ .\ \ i :: S,\Delta;\Phi;\cdot \vdash \mathrm{coerce}_{\tau,\tau'} \ominus \mathrm{coerce}_{\tau,\tau'} \lesssim 0 :^c \tau \xrightarrow{\mathrm{diff}(0)} \tau'$

We can construct the following derivation where $t = \lambda x.\mathsf{let}\ \{y\} = (x)\ \mathsf{in}\ \mathsf{return}\,\mathrm{coerce}_{\tau,\tau'}\ y$ using the **c-r-≡** ,**c-r-letm** and **c-r-return** rules.

$$\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c (\{P\}\ \exists \vec{\gamma}_1.\tau\ \{Q\}) \overset{\mathrm{diff}(D)}{\xrightarrow{\mathrm{diff}(0)}} (\{P'\}\ \exists \vec{\gamma}_2.\tau'\ \{Q'\}) \overset{\mathrm{diff}(D')}{}$$

Case

S-RUM

$$\frac{\beta'_i = \beta_i \cup T_i \cup T'_i}{\begin{array}{c}\Sigma;\Delta;\Phi_a \models U(\{\gamma_i \mapsto T_i\}\ \exists \vec{\gamma'_1}.A_1\ \{Q_1\}, \{\gamma_i \mapsto T'_i\}\ \exists \vec{\gamma'_2}.A_2\ \{Q_2\}) \\ \sqsubseteq \{\gamma_i \mapsto \beta_i\}\ \exists \vec{\gamma}_1,\vec{\gamma}_2.U(A_1,A_2)\ \{\gamma_i \mapsto \beta'_i\}\end{array}} \begin{array}{l}\mathrm{exec}(L,U) \\ \mathrm{exec}(L',U') \\ \mathrm{diff}(U-L')\end{array}$$

We can construct the following derivation where $t = \lambda x.\mathsf{let}\ \{y\} = (\mathsf{switch}\ x)\ \mathsf{in}\ \mathsf{return}\ y$ using the **c-switch** ,**c-r-letm** and **c-r-return** rules.

$$\begin{array}{l}\Delta;\Phi;\cdot \vdash t \ominus t \lesssim 0 :^c \\ U(\{\gamma_i \mapsto T_i\}\ \exists \vec{\gamma'_1}.A_1\ \{Q_1\}, \{\gamma_i \mapsto T'_i\}\ \exists \vec{\gamma'_2}.A_2\ \{Q_2\})) \to (\{\gamma_i \mapsto \beta_i\}\ \exists \vec{\gamma}_1,\vec{\gamma}_2.U(A_1,A_2)\ \{\gamma_i \mapsto \beta'_i\}\end{array} \begin{array}{l}\mathrm{exec}(L,U) \\ \mathrm{exec}(L',U') \\ \mathrm{diff}(U-L')\end{array}$$

□

Lemma 69 (Reflexivity of Predicate Equivalence in ARel)**.** $\Delta; \psi_a; \Phi_a \models P \equiv P \Rightarrow \Phi$ *and* $\Delta; \psi_a; \Phi_a \models \Phi$.

Proof. By induction on the definiton of the predicate P. □

Lemma 70 (Reflexivity of Algorithmic Binary Type Equivalence in ARel)**.** $\Delta; \psi_a; \Phi_a \models \tau \equiv \tau \Rightarrow \Phi$ *and* $\Delta; \psi_a; \Phi_a \models \Phi$.

Proof. By induction on the binary type. □

Lemma 71 (Reflexivity of Predicate Subtyping in ARel)**.** $\Delta; \Phi_a \models^{\mathsf{A}} P \sqsubseteq P \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the definiton of the predicate P. □

Lemma 72 (Reflexivity of Unary Algorithmic Subtyping in ARel)**.** $\Delta; \Phi_a \models^{\mathsf{A}} A \sqsubseteq A \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the unary type. □

Lemma 73 (Transitivity of Unary Algorithmic Subtyping in ARel)**.** *If* $\Delta; \Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_2 \Rightarrow \Phi_1$ *and* $\Delta; \Phi_a \models^{\mathsf{A}} A_2 \sqsubseteq A_3 \Rightarrow \Phi_2$ *and* $\Delta; \Phi_a \models \Phi_1 \wedge \Phi_2$*, then* $\Delta; \Phi_a \models^{\mathsf{A}} A_1 \sqsubseteq A_3 \Rightarrow \Phi_3$ *for some* Φ_3 *such that* $\Delta; \Phi_a \models \Phi_3$.

Proof. By induction on the first subtyping derivation. □

Theorem 74 (Soundness of the Algorithmic Unary Subtyping in ARel)**.** *Assume that*

1. $\Delta; \psi_a; \Phi_a \models^{\mathsf{A}} A' \sqsubseteq A \Rightarrow \Phi$
2. $FIV(\Phi_a, A, A') \subseteq \Delta, \psi_a$

3. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta; \Phi_a[\theta_a] \models^{\mathsf{A}} A'[\theta_a] \sqsubseteq A[\theta_a]$.

Proof. By induction on the algorithmic unary subtyping derivation. □

Theorem 75 (Completeness of the Unary Algorithmic Subtyping in ARel)**.** *Assume that* $\Delta; \Phi_a \models^{\mathsf{A}} A' \sqsubseteq A$. *Then* $\exists \Phi$. *such that* $\Delta; \Phi_a \models^{\mathsf{A}} A' \sqsubseteq A \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the unary subtyping derivation. □

Theorem 76 (Soundness of the Algorithmic Predicate Equality in ARel)**.** *Assume that*

1. $\Delta; \psi_a; \Phi_a \models P \equiv P' \Rightarrow \Phi$
2. $FIV(\Phi_a, P, P') \subseteq \Delta, \psi_a$
3. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta; \Phi_a[\theta_a] \models P[\theta_a] \equiv P'[\theta_a]$.

Proof. By induction on the algorithmic predicate equivalence derivation. □

Theorem 77 (Soundness of the Algorithmic Binary Type Equality in ARel)**.** *Assume that*

1. $\Delta; \psi_a; \Phi_a \models \tau' \equiv \tau \Rightarrow \Phi$
2. $FIV(\Phi_a, \tau, \tau') \subseteq \Delta, \psi_a$
3. $\Delta; \Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable s.t* $\Delta \rhd \theta_a : \psi_a$ *is derivable.*

Then $\Delta; \Phi_a[\theta_a] \models \tau'[\theta_a] \equiv \tau[\theta_a]$.

Proof. By induction on the algorithmic binary type equivalence derivation. □

Theorem 78 (Completeness of the Algorithmic Predicate Equivalence in ARel)**.** *Assume that* $\Delta; \Phi_a \models P \equiv P'$. *Then* $\exists \Phi$. *such that* $\Delta; \Phi_a \models P \equiv P' \Rightarrow \Phi$ *and* $\Delta; \Phi_a \models \Phi$.

Proof. By induction on the predicate equivalence derivation. □

Theorem 79 (Completeness of the Binary Algorithmic Type Equivalence in ARel)**.** *Assume that* $\Sigma;\Delta;\Phi_a \models \tau' \equiv \tau$. *Then* $\exists\,\Phi$. *such that* $\Delta;\Phi_a \models \tau' \equiv \tau \Rightarrow \Phi$ *and* $\Delta;\Phi_a \models \Phi$.

Proof. By induction on the binary subtyping derivation. □

B.6.1.2 Proof of Theorem 8

Theorem 80 (Soundness of ARel & Type Preservation of Embedding)**.** *The following holds.*

1. *If* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A$, *then* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t^* :^c A$ *and* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t : A$.
2. *If* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim D : \tau$, *then* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1^* \ominus t_2^* \lesssim D :^c \tau$ *and* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$.

Proof. Proof is by simultaneous induction on the embedding derivations. The proof follows from the embedding rules.

Proof of Theorem 80.2:

Case

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 \rightsquigarrow t_1^* : A \ (\star) \qquad \Sigma;\Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 \rightsquigarrow t_2^* : A \ (\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow \textbf{switch}\ t_1^* \ominus \textbf{switch}\ t_2^* \lesssim U_1 - L_2 : U\ A}\ \textbf{e-switch}$$

By Theorem 80.1 on ($\star$), we get $\Delta;\Phi;|\Gamma| \vdash^{U_1}_{L_1} t_1^* :^c A_1 \ \ (\star\star)$.
By Theorem 80.1 on ($\diamond$), we get $\Delta;\Phi;|\Gamma| \vdash^{U_2}_{L_2} t_2^* :^c A_2 \ \ (\diamond\diamond)$.
Then, we conclude as follows:

$$\frac{\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 :^c A \qquad \Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 :^c A}{\Delta;\Phi_a;\Gamma \vdash \text{switch}\ t_1 \ominus \text{switch}\ t_2 \lesssim U_1 - L_2 :^c U\ A}\ \textbf{c-switch}$$

.

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow {t_1}^* \ominus {t_2}^* \lesssim D : \tau \;\; (\star) \qquad \Delta;\Phi_a \models \tau \sqsubseteq \tau' \;(\diamond) \qquad t' = \mathbf{coerce}_{\tau,\tau'} \qquad \Delta;\Phi_a \models D \leq D' \;(\dagger)}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t'\ {t_1}^*_1 \ominus t'\ {t_2}^*_2 \lesssim D' : \tau'}\ \textbf{e-r-sub}$$

By Theorem 80.2 on $(\star)$, $\Delta;\Phi;\Gamma \vdash t_1^* \ominus t_2^* \lesssim D :^c \tau \;\; (\star\star)$.

By Lemma 55 using $(\diamond)$, we know that $\Delta;\Phi;\cdot \vdash t' \ominus t' \lesssim 0 :^c \tau \xrightarrow{\text{diff}(0)} \tau' \;\; (\diamond\diamond)$.

By applying **c-r-app** rule and $(\star\star)$ and $(\diamond\diamond)$, we get $\Delta;\Phi;\Gamma \vdash t'\ t_1^* \ominus t'\ t_2^* \lesssim D :^c \tau' \;\; (\spadesuit)$.

By reflexivity of binary type equivalence, we know $\Delta;\Phi_a \models \tau' \equiv \tau' \;\; (\spadesuit\spadesuit)$.

Then, we conclude as follows:

$$\frac{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D :^c \tau \;\; (\spadesuit) \qquad \Delta;\Phi_a \models \tau \equiv \tau' \;\; (\spadesuit\spadesuit) \qquad \Delta;\Phi_a \models D \leq D' \;\; (\dagger)}{\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim D' :^c \tau'}\ \textbf{c-r-} \equiv$$

Case

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathbf{int}[I] \;\; (\star) \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \tau \;\; (\diamond) \qquad \gamma\ \mathbf{fresh} \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \rightsquigarrow \mathsf{alloc}\ t_1^*\ t_2^* \ominus \mathsf{alloc}\ {t_1'}^*\ {t_2'}^* \lesssim 0 : \{P\}\ \exists\gamma.\ \overset{\text{diff}(D_1+D_2)}{\mathsf{Array}_\gamma}[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}}\ \textbf{e-r-alloc}$$

By Theorem 80.2 on $(\star)$, $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathrm{int}[I] \;\; (\star\star)$.

By Theorem 80.2 on $(\diamond)$, we know that $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau \;\; (\diamond\diamond)$.

By applying **c-r-alloc** rule and $(\star\star)$ and $(\diamond\diamond)$, we get:

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathrm{int}[I] \quad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \tau \qquad \gamma\ \mathsf{fresh} \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{alloc}\ t_1\ t_2 \ominus \mathsf{alloc}\ t_1'\ t_2' \lesssim 0 :^c \{P\}\ \exists\gamma.\ \overset{\text{diff}(D_1+D_2)}{\mathsf{Array}_\gamma}[I]\ \tau\ \{P \star \gamma \to \mathbb{N}\}}\ \textbf{c-r-alloc}$$

Case

$$\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \quad (\star)$$
$$\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \mathbf{int}[I'] \quad (\diamond)$$
$$\Delta;\Phi_a \models I' \le I \qquad I' \notin \beta \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}$$

$$\Delta;\Phi_a;\Gamma$$
$$\vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \rightsquigarrow \mathsf{read}\ t_1^*\ t_2^* \ominus \mathsf{read}\ t_1'^*\ t_2'^* \lesssim 0 : \{P \star \gamma \mapsto \beta\}\ \exists \gamma.\Box^{\mathrm{diff}(D_1+D_2)}\tau\ \{P \star \gamma \mapsto \beta\}$$

e-r-readb

By Theorem 80.2 on $(\star)$, $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \quad (\star\star)$.
By Theorem 80.2 on $(\diamond)$, we know that $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathrm{int}[I'] \quad (\diamond\diamond)$.
By applying **c-r-readb** rule and $(\star\star)$ and $(\diamond\diamond)$, we get:

$$\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \ (\star\star)$$
$$\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathrm{int}[I'] \ (\diamond\diamond) \qquad \Delta;\Phi_a \models I' \le I \qquad I' \notin \beta \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}$$

$$\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{read}_\mathsf{b}\ t_1\ t_2 \ominus \mathsf{read}_\mathsf{b}\ t_1'\ t_2' \lesssim 0 :^c \{P \star \gamma \mapsto \beta\}\ \exists \_.\Box^{\mathrm{diff}(D_1+D_2)}\tau\ \{P \star \gamma \mapsto \beta\}$$

Case

$$\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau \quad (\star)$$
$$\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \mathbf{int}[I'] \quad (\diamond)$$
$$\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \rightsquigarrow {t_3}^* \ominus {t_3'}^* \lesssim D_3 : \Box\tau \quad (\dagger) \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}$$

$$\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}\ t_1'\ t_2'\ t_3' \rightsquigarrow \mathsf{updt}_\mathsf{b}\ t_1^*\ t_2^*\ t_3^* \ominus \mathsf{updt}_\mathsf{b}\ t_1'^*\ t_2'^*\ t_3'^* \lesssim 0 :$$
$$\{P \star \gamma \mapsto \beta\}\ \exists \gamma.\tau^{\mathrm{diff}(D_1+D_2+D_3)}\ \{P \star \beta \setminus \{I'\}\}$$

e-r-updtb

By Theorem 80.2 on $(\star)$, $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \quad (\star\star)$.
By Theorem 80.2 on $(\diamond)$, we know that $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathrm{int}[I'] \quad (\diamond\diamond)$.
By Theorem 80.2 on $(\dagger)$, we know that $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 :^c \Box\tau \quad (\dagger\dagger)$.
By applying **c-r-updateb** rule and $(\star\star)$, $(\diamond\diamond)$ and $(\dagger,\dagger)$, we get:

$$\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \mathsf{Array}_\gamma[I]\ \tau \ (\star\star) \qquad \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 :^c \mathrm{int}[I'] \ (\diamond\diamond)$$
$$\Sigma;\Delta;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \lesssim D_3 :^c \Box\tau \ (\dagger\dagger) \qquad \Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}$$

$$\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{updt}_\mathsf{b}\ t_1\ t_2\ t_3 \ominus \mathsf{updt}_\mathsf{b}\ t_1'\ t_2'\ t_3' \lesssim 0 :^c \{P \star \gamma \mapsto \beta\}\ \exists \_.\mathrm{unit}^{\mathrm{diff}(D_1+D_2+D_3)}\ \{P \star \gamma \mapsto \beta \setminus \{I'\}\}$$

c-r-updateb

□

B.6.1.3 Proof of Theorem 9

Theorem 81 (Completeness of ARel)**.** *The following holds.*

1. *If* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t : A$ *then,* $\exists\, t^*$ *such that* $\Sigma;\Delta;\Phi;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A$.

2. *If* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim D : \tau$ *then,*

   $\exists\, t_1^*$ *and* $\exists\, t_2^*$ *such that* $\Sigma;\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow t_1^* \ominus t_2^* \lesssim D : \tau$.

Proof. Proof is by simultaneous induction on the typing derivations.

Proof of Theorem 81.1:

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U}_{L} t : A \;(\star) \qquad \Sigma;\Delta;\Phi_a \models A \sqsubseteq A' \qquad \Sigma;\Delta \models U \leq U' \qquad \Sigma;\Delta \models L' \leq L}{\Sigma;\Delta;\Phi_a;\Omega \vdash^{U'}_{L'} t : A'}\ \textbf{u-x}$$

By Theorem 81.1 on $(\star)$, we get $\exists\, t^*$ such that $\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A \;\; (\star\star)$.
By **e-u-sub** rule using $(\star\star)$,we conclude as follows.

$$\frac{\Delta;\Phi_a;\Omega \vdash^{U}_{L} t \rightsquigarrow t^* : A \;(\star\star) \qquad \Delta;\Phi_a \models A \sqsubseteq A' \qquad \Delta;\Phi_a \models L' \leq L \qquad \Delta;\Phi_a \models U \leq U'}{\Delta;\Phi_a;\Omega \vdash^{t'}_{k'} e \rightsquigarrow e^* : A'}\ \textbf{e-u-sub}$$

Proof of Theorem 81.2:

Case

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma|_1 \vdash^{U_1}_{L_1} t_1 : A_1 \;(\star) \qquad \Sigma;\Delta;\Phi_a;|\Gamma|_2 \vdash^{U_2}_{L_2} t_2 : A_2 \;(\diamond)}{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim U_1 - L_2 : U(A_1, A_2)}\ \textbf{r-s}$$

By Theorem 81.1 on $(\star)$, we get $\exists\, t_1^*$ such that $\Sigma;\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 \rightsquigarrow t_1^* : A_1 \;\; (\star\star)$.
By Theorem 81.1 on $(\diamond)$, we get $\exists\, t_2^*$ such that $\Sigma;\Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 \rightsquigarrow t_2^* : A_2 \;\; (\star\star)$.
By **e-switch** rule using $(\star\star)$ and $(\diamond\diamond)$,we conclude as follows.

$$\frac{\Sigma;\Delta;\Phi_a;|\Gamma| \vdash^{U_1}_{L_1} t_1 \rightsquigarrow t_1^* : A_1 \ \ (\star\star) \qquad \Sigma;\Delta;\Phi_a;|\Gamma| \vdash^{U_2}_{L_2} t_2 \rightsquigarrow t_2^* : A_2 \ \ (\diamond\diamond)}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \rightsquigarrow \text{switch } t_1^* \ominus \text{switch } t_2^* \lesssim U_1 - L_2 : U(A_1, A_2)} \textbf{ e-switch}$$

Case

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \textbf{int}[I] \ \ (\star) \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \square\tau \ \ (\diamond) \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \text{alloc } t_1\ t_2 \ominus \text{alloc } t_1'\ t_2' \lesssim 0 : \{P\}\ \exists\gamma.\ \text{Array}_\gamma[I]\ \tau\ \{P \star \gamma \rightarrow \emptyset\}^{\text{diff}(D_1+D_2)}} \textbf{ r-alloc-box}$$

By Theorem 81.2 on $(\star)$,

we get $\exists\, t_1^*$ such that $\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \text{int}[I] \ \ (\star\star)$.

By Theorem 81.2 on $(\diamond)$,

we get $\exists\, t_2^*$ such that $\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \square\tau \ \ (\diamond\diamond)$.

By **e-r-allocb** rule using $(\star\star)$ and $(\diamond\diamond)$,we conclude as follows.

$$\frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \text{int}[I] \ \ (\star\star) \\ \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \square\tau \ \ (\diamond\diamond) \qquad \gamma \text{ fresh} \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\begin{array}{c}\Delta;\Phi_a;\Gamma \\ \vdash \text{alloc } t_1\ t_2 \ominus \text{alloc } t_1'\ t_2' \rightsquigarrow \text{alloc}_\text{b}\ t_1^*\ t_2^* \ominus \text{alloc}_\text{b}\ t_1'^*\ t_2'^* \lesssim 0 : \{P\}\ \exists\gamma.\ \text{Array}_\gamma[I]\ \tau\ \{P \star \gamma \rightarrow \emptyset\}^{\text{diff}(D_1+D_2)}\end{array}} \textbf{ e-r-allocb}$$

Case

$$\frac{\begin{array}{c}\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \text{Array}_\gamma[I]\ \tau \\ \Sigma;\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim D_2 : \textbf{int}[I'] \qquad \Delta;\Phi_a \models I' \leq I \qquad \Sigma;\Delta \vdash P \text{ wf}\end{array}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \text{read } t_1\ t_2 \ominus \text{read } t_1'\ t_2' \lesssim 0 : \{P \star \gamma \rightarrow \beta\}\ \exists\_.\tau\ \{P \star \gamma \rightarrow \beta\}^{\text{diff}(D_1+D_2)}} \textbf{ r-r}$$

By Theorem 81.2 on $(\star)$,

we get $\exists\, t_1^*$ such that $\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \text{Array}_\gamma[I]\ \tau \ \ (\star\star)$.

By Theorem 81.2 on $(\diamond)$,

we get $\exists\, t_2^*$ such that $\Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \text{int}[I'] \ \ (\diamond\diamond)$.

By **e-r-read** rule using $(\star\star)$ and $(\diamond\diamond)$,we conclude as follows.

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \mathsf{Array}_\gamma[I]\ \tau\ (\star\star) \quad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \mathsf{int}[I']\ (\diamond\diamond) \quad \Delta;\Phi_a \models I' \le I \quad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\Phi_a;\Gamma \vdash \mathsf{read}\ t_1\ t_2 \ominus \mathsf{read}\ t_1'\ t_2' \rightsquigarrow \mathsf{read}\ t_1^*\ t_2^* \ominus \mathsf{read}\ t_1'^*\ t_2'^* \lesssim 0 : \{P\}\ \exists \gamma.\tau\ \{P\}}\ \textbf{e-r-read}$$

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 : \{P\}\ \exists^{\mathrm{diff}(D)} \vec{\gamma}_1.\tau\ \{P'\}\ (\star) \quad \Sigma;\Delta,\vec{\gamma}_1 : \vec{\mathbb{L}};\Phi_a;\Gamma,x:\tau \vdash t_2 \ominus t_2' \lesssim D_2 : \{P'\}\ \exists^{\mathrm{diff}(D')} \vec{\gamma}_2.\sigma\ \{Q\}\ (\diamond)}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \lesssim 0 : \{P\}\ \exists^{\mathrm{diff}(D+D'+D_1+D_2)} \vec{\gamma}_1\vec{\gamma}_2 : \sigma\ \{Q\}}\ \textbf{r-let}$$

By Theorem 81.2 on $(\star)$,

we get $\exists\, t_1^*$ such that $\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \{P_1\}\ \exists^{\mathrm{diff}(D)} \vec{\gamma}_1.\tau\ \{Q_1 \star Q_2\}\ D_1\ (\star\star)$.

By Theorem 81.2 on $(\diamond)$, we get $\exists\, t_2^*$ such that

$\Sigma;\Delta,\vec{\gamma}_1 : \vec{\mathbb{L}};\Phi_a;\Gamma,x:\tau \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \{Q_1 \star P_2\}\ \exists^{\mathrm{diff}(D')} \vec{\gamma}_2.\tau'\ \{Q\}\ (\diamond\diamond)$.

By **e-r-letm** rule using $(\star\star)$ and $(\diamond\diamond)$,we conclude as follows.

$$\frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \rightsquigarrow {t_1}^* \ominus {t_1'}^* \lesssim D_1 : \{P\}\ \exists^{\mathrm{diff}(D)} \vec{\gamma}_1.\tau\ \{P'\}\ D_1\ (\star\star) \quad \Sigma;\Delta,\vec{\gamma}_1 : \vec{\mathbb{L}};\Phi_a;\Gamma,x:\tau \vdash t_2 \ominus t_2' \rightsquigarrow {t_2}^* \ominus {t_2'}^* \lesssim D_2 : \{P'\}\ \exists^{\mathrm{diff}(D')} \vec{\gamma}_2.\tau'\ \{Q\}\ (\diamond\diamond)}{\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \rightsquigarrow \mathsf{let}\{x\} = t_1^*\ \mathsf{in}\ t_2^* \ominus \mathsf{let}\{x\} = t_1'^*\ \mathsf{in}\ t_2^* \lesssim 0 : \{P\}\ \exists_{\_}^{\mathrm{diff}(D_1+D_2+D+D')}.\tau'\ \{Q\}}\ \textbf{e-r-letm}$$

□

Theorem 82 (Invariant of the Algorithmic Typechecking)**.** *We have the following.*

1. *Assume that* $\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, L, U \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Omega;A,L,U) \subseteq dom(\Delta,\psi_a)$.

   Then $FIV(\Phi) \subseteq dom(\Delta;\psi_a)$.

2. *Assume that* $\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A \Rightarrow [\psi], L, U, \Phi$ *and* $FIV(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$.

   Then $FIV(A,L,U,\Phi) \subseteq dom(\Delta,\psi;\psi_a)$.

3. *Assume that* $\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi$ *and* $FIV(\Phi_a,\Gamma,\tau,D) \subseteq dom(\Delta,\psi_a)$. *Then* $FIV(\Phi) \subseteq dom(\Delta;\psi_a)$.

4. *Assume that* $\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau \Rightarrow [\psi], D, \Phi$ *and* $FIV(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$. *Then* $FIV(\tau,D,\Phi) \subseteq dom(\Delta;\psi;\psi_a)$.

B.6.1.4 Proof of Theorem 10

Theorem 83 (Soundness of the Algorithmic Typechecking in ARel)**.** *We have the following.*

1. *Assume that* $\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, L, U \Rightarrow \Phi$ *and*

   1.1. $FIV(\Phi_a,\Omega,A,L,U) \subseteq dom(\Delta,\psi_a)$

   1.2. $\Sigma;\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable for some* θ_a *such that* $\Delta \rhd \theta_a : \psi_a$ *is derivable*

   Then $\Sigma;\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a]$.

2. *Assume that* $\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t \uparrow A \Rightarrow [\psi], L, U, \Phi$ *and*

   2.1. $FIV(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$

   2.2. $\forall\theta\ \forall\theta_a.\ \Delta;\Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *is provable s.t* $\Delta \rhd \theta : \psi$ *and* $\Delta \rhd \theta_a : \psi_a$ *are derivable*

   Then $\Sigma;\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta\,\theta_a]}_{L[\theta\,\theta_a]} |t| :^c A[\theta\,\theta_a]$.

3. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \downarrow \tau, D \Rightarrow \Phi$ *and*

   3.1. $FIV(\Phi_a,\Gamma,\tau,D) \subseteq dom(\Delta,\psi_a)$

   3.2. $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$ *is provable for some* θ_a *such that* $\Delta \rhd \theta_a : \psi_a$ *is derivable*

   Then $\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta_a] :^c \tau[\theta_a]$.

4. *Assume that* $\Delta;\psi_a;\Phi_a;\Gamma \vdash t \ominus t' \uparrow \tau \Rightarrow [\psi], D, \Phi$ *and*

   4.1. $FIV(\Phi_a,\Gamma) \subseteq dom(\Delta,\psi_a)$

   4.2. $\forall\theta\ \forall\theta_a.\ \Delta;\Phi_a[\theta_a] \models \Phi[\theta\,\theta_a]$ *is provable s.t* $\Delta \rhd \theta : \psi$ *and* $\Delta \rhd \theta_a : \psi_a$ *are derivable*

Then $\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t| \ominus |t'| \lesssim D[\theta\,\theta_a] :^c \tau[\theta\,\theta_a]$.

Proof. Statements (1—4) follow from simultaneous structural induction on the algorithmic typing derivations. We present several cases below.

Proof of Theorem 83.1:

Case

$$\frac{\Delta;\psi_a;\Phi_a;\Omega \vdash t \downarrow A, L, U \Rightarrow \Phi\ (\diamond) \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Delta;\psi_a;\Phi_a;\Omega \vdash \mathsf{return}\ t \downarrow \{P\}\ \overset{\mathrm{exec}(L,U)}{\exists\_ : A}\ \{P\}, 0, 0 \Rightarrow \Phi}\ \textbf{alg-u-ret-}\downarrow$$

TS: $\Sigma;\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |\,\mathsf{return}\ t\,| :^c \{P\}\ \overset{\mathrm{exec}(L[\theta_a],U[\theta_a])}{\exists\_ : A}\ \{P\}\,[\theta_a]$.

By the main assumptions, we have $FIV(\Phi_a,\Omega,(\{P\}\ \overset{\mathrm{exec}(L,U)}{\exists\_ : A}\ \{P\}),L,U) \subseteq dom(\Delta,\psi_a)$ $(\star)$

$\Delta;\Phi_a[\theta_a] \models (\Phi)[\theta_a]$ $(\star\star)$

Using $(\star)$, $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta;\Phi_a[\theta_a] \models \Phi[\theta_a]$

By Theorem 83.1 on $(\diamond)$ using $(\star)$ and b), we can show that

$$\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a] \tag{B.69}$$

Using rule **c-ret** rule, we get

$$\frac{\Sigma;\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{u[\theta_a]}_{L[\theta_a]} |t| :^c A[\theta_a] \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}}{\Sigma;\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{0}_{0} |\,\mathsf{return}\ t\,| :^c \overset{\mathrm{exec}(L[\theta_a],U[\theta_a])}{\{P\}\ \exists\gamma.A}\ \{P\}}\ \textbf{c-return}$$

Proof of Theorem 83.2:

Case

$$\frac{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash t_1 \uparrow \{P\}\ \exists \vec{\gamma}_1 : A_1\ \{P'\} \overset{\text{exec}(L,U)}{\Rightarrow} [\psi_1], L_1, U_1, \Phi_1\ (\dagger) \quad \Sigma;\Delta;\psi_1,\psi_a;\Phi_a;\Omega, x : A_1 \vdash t_2 \uparrow \{Q'\}\ \exists \vec{\gamma}_2 : A_2\ \{Q\} \overset{\text{exec}(L',U')}{\Rightarrow} [\psi_2], L_2, U_2, \Phi_2\ (\spadesuit) \quad P' = Q'}{\Sigma;\Delta;\psi_a;\Phi_a;\Omega \vdash \mathsf{let_m}\{x\} = t_1 \text{ in } t_2 \uparrow \overset{\text{exec}(L+L'+L_1+L_2+C_L,U+U'+U_1+U_2+c_l)}{\{P\}\ \exists \_ : A_2\ \{Q\}} \Rightarrow [\psi_1,\psi_2], 0, 0, \Phi_1 \wedge \Phi_2}$$

alg-u-let-↑

TS: $\Sigma;\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{(U_1+U_2+c_l)[\theta\theta_a]}_{(L_1+L_2+c_l)[\theta\theta_a]} |\,\mathsf{let_m}\{x\} = t_1 \text{ in } t_2\,| :^c \overset{\text{exec}((L+L')[\theta\theta_a],(U+U')[\theta\theta_a])}{\{P\}\ \exists \_ : A\ \{P\}}\ [\theta\theta_a]$.

By the main assumptions, we have $\text{FIV}(\Phi_a,\Omega) \subseteq dom(\Delta,\psi_a)$ $(\star)$

$\Delta;\Phi_a[\theta_a] \models (\Phi_1 \wedge \Phi_2)[\theta\theta_a]$ $(\star\star)$

From $(\star\star)$, we know : $\Delta;\Phi_a[\theta_a] \models (\Phi_1)[\theta\theta_a]$ $(\diamond)$ and $\Delta;\Phi_a[\theta_a] \models (\Phi_2)[\theta\theta_a]$ $(\diamond\diamond)$ By Theorem 83.2 on $(\dagger)$ using $(\star)$ and $(\diamond)$, we can show that

$\Sigma;\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_1[\theta\theta_a]}_{L_1[\theta\theta_a]} |t_1| :^c \{P\}\ \overset{\text{exec}(L,U)}{\exists \vec{\gamma}_1 : A_1}\ \{P'\}[\theta\theta_a]$ $(\dagger\dagger)$.

From Theorem 82, we know that $\text{FIV}(A_1) \subseteq dom(\psi_1)$ which implies $\text{FIV}(\Phi_a,\Omega,x : A_1) \subseteq dom(\Delta,(\psi_a,\psi_1))$ $(\heartsuit)$.

By Theorem 83.2 on $(\spadesuit)$ using $(\heartsuit)$ and $(\diamond\diamond)$, we can show that

$\Sigma;\Delta;\Phi_a[\theta_a];\Omega, x : A_1[\theta_a] \vdash^{U_2[\theta\theta_a]}_{L_2[\theta\theta_a]} |t_2| :^c \{Q'\}\ \overset{\text{exec}(L',U')}{\exists \vec{\gamma}_2 : A_2}\ \{Q\}[\theta\theta_a]$ $(\spadesuit\spadesuit)$.

$$\frac{\Sigma;\Delta;\Phi_a[\theta_a];\Omega[\theta_a] \vdash^{U_1[\theta\theta_a]}_{L_1[\theta\theta_a]} |t_1| :^c \{P\}\ \overset{\text{exec}(L,U)}{\exists \vec{\gamma}_1 : A_1}\ \{P'\}[\theta\theta_a]\ (\dagger\dagger) \quad \Sigma;\Delta;\Phi_a[\theta_a];\Omega, x : A_1[\theta_a] \vdash^{U_2[\theta\theta_a]}_{L_2[\theta\theta_a]} |t_2| :^c \{Q'\}\ \overset{\text{exec}(L',U')}{\exists \vec{\gamma}_2 : A_2}\ \{Q\}[\theta\theta_a]\ (\spadesuit\spadesuit)}{\Sigma;\Delta;\Phi_a;\Omega \vdash^0_0 \mathsf{let}\{x\} = t_1 \text{ in } t_2 :^c \overset{\text{exec}((L+L'+L_1+L_2+c_l)[\theta\theta_a],(U+U'+U_1+U_2+c_l)[\theta\theta_a])}{\{P\}\ \exists \vec{\gamma}_1, \vec{\gamma}_2 . A_2\ \{Q\}}}\ \textbf{c-letm}$$

Proof of Theorem 83.3:

Case

$$\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \uparrow \mathsf{Array}_\gamma[I]\ \tau \Rightarrow [\psi_1], D_1, \Phi_1\ (a)$$
$$\Sigma;\Delta;\psi_a,\psi_1;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \uparrow \mathbf{int}[I'] \Rightarrow [\psi_2], D_2, \Phi_2\ (b)$$
$$\Sigma;\Delta;\psi_a;\Phi_a \vDash I' \leq I \wedge \beta' = \beta/\{I'\}$$
$$D_3 \in \mathbf{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_3,\psi_2,\psi_1,\psi_a;\Phi_a;\Gamma \vdash t_3 \ominus t_3' \downarrow \Box\tau, D_3 \Rightarrow \Phi_3\ (c)$$
$$P = P' \star \gamma \rightarrow \beta \qquad Q = P' \star \gamma \rightarrow \beta' \qquad \Phi = \Phi_2 \wedge D_1 + D_2 + D_3 \doteq D \qquad \Sigma;\Delta \vdash P'\ \mathsf{wf}$$

$$\Sigma;\Delta;\psi_a;\Phi_a;\Gamma \vdash \mathsf{updt_b}\ t_1\ t_2\ t_3 \ominus \mathsf{updt_b}\ t_1'\ t_2'\ t_3' \downarrow\ \{P\}\ \exists^{\mathrm{diff}(D)}\_.\mathbf{unit}\ \{Q\}, 0 \Rightarrow$$
$$\exists(\psi_1).(\Phi_1 \wedge (\exists(\psi_2).(\Phi_2 \wedge \exists D_3 :: \mathbb{R}.\Phi)))$$

alg-r-updtB-↓

TS: $\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |\,\mathsf{updt_b}\ t_1\ t_2\ t_3\,| \ominus |\,\mathsf{updt_b}\ t_1'\ t_2'\ t_3'\,| \lesssim D[\theta_a] :^c \{P\}\ \exists^{\mathrm{diff}(D)}\_.\mathrm{unit}\ \{Q\}[\theta_a]$.

By the main assumptions, we have $\mathrm{FIV}(\Phi_a, \Gamma, (\{P\}\ \exists^{\mathrm{diff}(D)}\_.\mathrm{unit}\ \{Q\}), D) \subseteq dom(\Delta, \psi_a)$ $(\star)$

$\Delta;\Phi_a[\theta_a] \models (\exists(\psi_1).(\Phi_1 \wedge (\exists(\psi_2).(\Phi_2 \wedge \exists D_3 :: \mathbb{R}.\Phi))))[\theta_a]$ $(\star\star)$

By Theorem 82 on $(\star)$ and the first premise,

then we get $\mathrm{FIV}(\mathsf{Array}_\gamma[I]\ \tau, D_1, \Phi_1) \subseteq dom(\Delta, \psi_a, \psi_1)$ (1).

Similarly on the second premise, we have $\mathrm{FIV}(\mathrm{int}[I'], D_2, \Phi_2) \subseteq dom(\Delta, \psi_a, \psi_1, \psi_2)$ (2)

Using $(\star)$, (1), (2), $(\star\star)$'s derivation must be in a form such that we have

a) $\Delta \vdash d_3 :: \mathbb{R}$

b) $\Delta;\Phi_a[\theta_a] \models \Phi_1[\theta_a]$

c) $\Delta;\Phi_a[\theta_a] \models \Phi_2[\theta_a]$

d) $\Delta;\Phi_a[\theta_a] \models \Phi_3[\theta_a, D_3 \mapsto d_3]$

e) $\Delta;\Phi_a[\theta_a] \models (D_1[\theta_a] + D_2[\theta_a] + d_3) \doteq D[\theta_a]$

By Theorem 83.4 on (a) using (1) and b), we know:

$\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_1| \ominus |t_1'| \lesssim D_1[\theta\,\theta_a] :^c \mathsf{Array}_\gamma[I]\ \tau[\theta\,\theta_a]$ $(\star)$

By Theorem 83.4 on (b) using (2) and c), we know:

$\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_2| \ominus |t_2'| \lesssim D_2[\theta\,\theta_a] :^c \mathrm{int}[I'][\theta\,\theta_a]$ $(\diamond)$

By Theorem 83.3 on (c) using (1)(2) and d), we know:

$\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_3| \ominus |t_3'| \lesssim d_3 :^c \Box\tau[\theta_a]$ $(\dagger)$

Using the above statements and the premises, we conclude using the rule c-r-updateb.

$$\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_1| \ominus |t_1'| \lesssim D_1[\theta\,\theta_a] :^c \mathsf{Array}_\gamma[I]\ \tau[\theta\,\theta_a] \quad (\star)$$
$$\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_2| \ominus |t_2'| \lesssim D_2[\theta\,\theta_a] :^c \mathrm{int}[I'][\theta\,\theta_a] \quad (\diamond)$$
$$\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a] \vdash |t_3| \ominus |t_3'| \lesssim d_3 :^c \Box\tau[\theta_a] \quad (\dagger)$$
$$\Delta;\Phi_a \models I' \le I \qquad \Sigma;\Delta \vdash P\ \mathsf{wf}$$

$$\Sigma;\Delta;\Phi_a[\theta_a];\Gamma[\theta_a]$$
$$\vdash |\,\mathsf{updt_b}\ t_1\ t_2\ t_3\,| \ominus |\,\mathsf{updt_b}\ t_1'\ t_2'\ t_3'\,| \lesssim 0 :^c \{P \star \gamma \mapsto \beta\}\ \exists\_.\mathrm{unit}^{\mathrm{diff}(D[\theta_a])}\ \{P \star \gamma \mapsto \beta \setminus \{I'\}\}$$

c-r-updateb

□

B.6.1.5 Proof of Theorem 11

Theorem 84 (Completeness of the Algorithmic Typechecking in Arel)**.** *We have the following.*

1. *Assume that* $\Sigma;\Delta;\Phi_a;\Omega \vdash^U_L t :^c A$*. Then,* $\exists t'$ *such that*

   1.1. $\Sigma;\Delta;\cdot;\Phi_a;\Omega \vdash t' \downarrow A, L, U \Rightarrow \Phi$

   1.2. $\Delta;\Phi_a \models \Phi$

   1.3. $|t'| = t$

2. *Assume that* $\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \tau$*. Then,* $\exists t_1', t_2'$ *such that*

   2.1. $\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau, D \Rightarrow \Phi$

   2.2. $\Delta;\Phi_a \models \Phi$

   2.3. $|t_1'| = t_1$ *and* $|t_2'| = t_2$

Proof. Proof is by simultaneous induction on the ARel typing derivations.

Proof of Theorem 11.2:

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim D :^c \tau \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Sigma;\Delta \vdash Q\ \mathsf{wf} \qquad \Sigma;\Delta \models P \subseteq Q}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{return}\ t_1 \ominus \mathsf{return}\ t_2 \lesssim 0 :^c \{P\}\ \overset{\mathrm{diff}(D)}{\exists}\_.\tau\ \{Q\}}\ \textbf{c-r-return}$$

By IH on the first premise, we know that exists t_1', t_2' such that:

$\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau, D \Rightarrow \Phi$ $(\star)$

$\Delta;\Phi_a \models \Phi$ $(\diamond)$

$|t_1'| = t_1$ and $|t_2'| = t_2$ $(\dagger)$

By using $(\star)$ and the premises, we can conclude using the algorithmic return rule.

$$\frac{\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash t_1' \ominus t_2' \downarrow \tau, D \Rightarrow \Phi \qquad \Sigma;\Delta \vdash P\ \mathsf{wf} \qquad \Sigma;\Delta \vdash Q\ \mathsf{wf} \qquad \Sigma;\Delta \models P \subseteq Q}{\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash \mathsf{return}\ t_1' \ominus \mathsf{return}\ t_2' \downarrow\ \{P\}\ \overset{\mathrm{diff}(D)}{\exists}\_.\tau\ \{Q\}, 0 \Rightarrow \Phi}\ \textbf{alg-r-ret-}\downarrow$$

Case

$$\frac{\Sigma;\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim D_1 :^c \{P\}\ \overset{\mathrm{diff}(D)}{\exists}\vec{\gamma}_1.\tau\ \{P'\} \qquad \Sigma;\Delta,\vec{\gamma}_1 : \vec{\mathbb{L}};\Phi_a;\Gamma, x:\tau \vdash t_2 \ominus t_2' \lesssim D_2 :^c \{P'\}\ \overset{\mathrm{diff}(D')}{\exists}\vec{\gamma}_2.\sigma\ \{Q\}}{\Sigma;\Delta;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1\ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1'\ \mathsf{in}\ t_2' \lesssim 0 :^c \{P\}\ \overset{\mathrm{diff}(D+D'+D_1+D_2)}{\exists}\vec{\gamma}_1\vec{\gamma}_2.\sigma\ \{Q\}}\ \textbf{c-r-letm}$$

By IH on the first premise, we know that exists $t_1^*, t_1'^*$ such that:

$\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash t_1^* \ominus t_1'^* \downarrow\ \{P\}\ \overset{\mathrm{diff}(D)}{\exists}\vec{\gamma}_1.\tau\ \{P'\}, D_1 \Rightarrow \Phi_1$ $(\star)$

$\Delta;\Phi_a \models \Phi_1$ $(\diamond)$

$|t_1^*| = t_1$ and $|t_1'^*| = t_1'$ $(\dagger)$

Choose $t_1^* = (t_1 :\{P\}\ \overset{\mathrm{diff}(D)}{\exists}\vec{\gamma}_1.\tau\ \{P'\}, D_1)$ and $t_1'^* = (t_1' :\{P\}\ \overset{\mathrm{diff}(D)}{\exists}\vec{\gamma}_1.\tau\ \{P'\}, D_1)$.

By the rule alg-r-anno-↑, we have:

$$\frac{\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \downarrow \{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma}_1.\tau\ \{P'\}, D_1 \Rightarrow \Phi_1\ (\star) \qquad \Delta;\Phi_a \vdash \{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma}_1.\tau\ \{P'\}\ \mathsf{wf} \qquad \mathrm{FIV}(\{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma}_1.\tau\ \{P'\}, D_1) \in \Delta}{\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash t_1^* \ominus t_1'^* \uparrow \{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma}_1.\tau\ \{P'\} \Rightarrow [\cdot], D_1, \Phi}\ \textbf{alg-r-anno-}\uparrow$$

By IH on the second premise, we know that exists $t_2^*, t_2'^*$ such that:

$\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash t_2^* \ominus t_2'^* \downarrow \{P'\}\ \exists^{\mathrm{diff}(D')}\vec{\gamma}_2.\sigma\ \{Q\}, D_2 \Rightarrow \Phi_2\ (\star\star)$

$\Delta;\Phi_a \models \Phi_2\ (\diamond\diamond)$

$|t_2^*| = t_2$ and $|t_2'^*| = t_2'\ (\dagger\dagger)$

By $(\star\star)$, we can show that for $D_2' \in \mathrm{fresh}(\mathbb{R})$ where $\Phi_2' = \Phi_2 \wedge D_2 \doteq D_2'$

$$\Sigma;\Delta;D_2',\cdot;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \{P'\}\ \exists^{\mathrm{diff}(D')}\vec{\gamma}_2 : \tau_2\ \{Q\}, D_2' \Rightarrow \Phi_2' \tag{B.70}$$

By the algorithmic rule alg-r-let-↓, we can conclude that:

$$\frac{\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash t_1^* \ominus t_1'^* \uparrow \{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma}_1 : \tau_1\ \{P'\} \Rightarrow [\psi_1], D_1, \Phi_1 \qquad D_2' \in \mathrm{fresh}(\mathbb{R}) \qquad \Sigma;\Delta;D_2';\Phi_a;\Gamma \vdash t_2 \ominus t_2' \downarrow \{P'\}\ \exists^{\mathrm{diff}(D')}\vec{\gamma}_2 : \tau_2\ \{Q\}, D_2' \Rightarrow \Phi_2' \qquad \Phi = \Phi_2' \wedge D_1 + D_2' + D + D' \doteq D''}{\Sigma;\Delta;\cdot;\Phi_a;\Gamma \vdash \mathsf{let}\{x\} = t_1^* \ \mathsf{in}\ t_2 \ominus \mathsf{let}\{x\} = t_1'^* \ \mathsf{in}\ t_2' \downarrow \{P\}\ \exists^{\mathrm{diff}(D'')}\_.\tau_2\ \{Q\}, 0 \Rightarrow \Phi_1 \wedge (\exists D_2 :: \mathbb{R}.\Phi)}\ \textbf{alg-r-let-}\downarrow$$

In this case, we know that there exists

$t = \mathsf{let}\{x\} = (t_1 : \{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma}_1.\tau\ \{P'\}, D_1)\ \mathsf{in}\ t_2$ and $t' = \mathsf{let}\{x\} = (t_1' : \{P\}\ \exists^{\mathrm{diff}(D)}\vec{\gamma}_1.\tau\ \{P'\}, D_1)\ \mathsf{in}\ t_2'$.

□

APPENDIX C

Appendix for Adaptive analysis

C.1 LOOP LANGUAGE

C.1.1 Syntax and Semantics

Expressions can be either arithmetic expressions or boolean expressions. An arithmetic expression can be a constant n denoting integer, a variable x from some countable set Var, the empty list [], a list $[a_1, \ldots, a_k]$ of arithmetic expressions, A boolean expression can be as usual `true` or `false`, the negation of a boolean expression, or a combination of boolean expressions by means of an operation $\oplus_b$ or the result of a comparison $\sim$ between arithmetic expressions.

Commands in the loop language are labelled — we assume that labels are unique and corresponds to the line of code where they appear, implicitly this gives us a control flow graph representation for the program. A command can be either a labelled assignment $[x \leftarrow e]^l$, or a labelled query request $[x \leftarrow q\ (e_q)]^l$, a labelled skip $[\texttt{skip}]^l$, the composition of two labelled commands $[c_1; c_2]^l$, a labelled if statement if $([b]^l, c_1, c_2)$, a labelled loop statement $\texttt{loop}\ [a]^l\ \texttt{do}\ c$.

A memory is a partial map from the labelled variables to values.

A trace is a list which only tracks the queries called and the results of the queries.

For memory and traces we will use notation: we will write ++ for concatenation of lists in traces, and we will use the standard notation $m[x \to v]$ for the update of the memory.

For the query request, we have $q(e_q)$ so that e_q represents the actual query. The query expressions e_q will be evaluated into the normal form, we call them query values v_q, which can be a natural number, the χ, and its access $\chi[n]$ as well as arithmetic

Arithmetic Operators	$\oplus_a$	::=	$+ \mid - \mid \times \mid \div$
Boolean Operators	$\oplus_b$	::=	$\vee \mid \wedge \mid \neg$
Relational Operators	$\sim$	::=	$< \mid \leq \mid ==$
Label	l	:=	$\mathbb{N}$
Loop Maps	w	$\in$	Label $\times \mathbb{N}$
Arithmetic Expressions	a	::=	$n \mid x \mid a \oplus_a a$
Boolean Expressions	b	::=	true | false | $\neg b \mid b \oplus_b b \mid a \sim a$
Expressions	e	::=	$a \mid b \mid [] \mid [e, \ldots, e]$
Values	v	::=	$n \mid$ true | false | $[] \mid [v, \ldots, v]$
Query expressions	e_q	::=	$a \mid \chi \mid \chi[a] \mid e_q \oplus_a e_q$
Query Values	v_q	::=	$n \mid \chi \mid \chi[n] \mid v_q \oplus_a v_q$
Labelled commands	c	::=	$[x \leftarrow e]^l \mid [x \leftarrow q\,(e_q)]^l \mid \texttt{loop}\ [a]^l\ \texttt{do}\ c \mid c; c$
			$\mid$ if $([b]^l, c, c) \mid [\texttt{skip}]^l$
Memory	m	::=	$[] \mid m[x \rightarrow v]$
Trace	t	::=	$[] \mid (q(v_q)^{(l,w)}) :: t$
Annotated Query	$\mathscr{AQ}$	::=	$\{q(v_q)^{(l,w)}\}$

Figure C.1: Syntax of loop language

operations over query values.

We are working on the database X^n, where X is the universe of the database. We define that two query expressions are the same in certain memory m by $=_q^m$.

$e_{q1} =_q^m e_{q2}$ iff $\forall k \in X.\exists v.\langle m, e_{q1}[k/\chi]\rangle \xrightarrow{*} \langle m, v\rangle \wedge \langle m, e_{q1}[k/\chi]\rangle \xrightarrow{*} \langle m, v\rangle$.

When we compare two v_q, the memory m can be empty. So we can define: $v_{q1} =_q^{[]} v_{q2}$.

We use $=_q$ as the shorthand for $=_q^{[]}$.

We define the inclusion relation $\in_q$ of one annotated query $q(v_q)^{l,w}$in the trace t:

$q(v_q)^{l,w} \in_q t$ iff $\exists q(v'_q)^{l,w} \in t.(v_q) =_q^{[]} (v'_q)$.

$q(v_q)^{l,w} \notin_q t$ iff $\forall q(v'_q)^{l,w} \in t, ((v_q) \neq_q^{[]} (v'_q)$

$$\boxed{\langle m, a\rangle \to_a \langle m, a'\rangle}$$

$$\frac{n = m(x)}{\langle m, x\rangle \to_a \langle m, n\rangle}\ \text{a-var} \qquad \frac{\langle m, a_1\rangle \to_a \langle m, a_1'\rangle}{\langle m, a_1 \oplus_a a_2\rangle \to_a \langle m, a_1' \oplus_a a_2\rangle}\ \text{a-aop1}$$

$$\frac{\langle m, a_2\rangle \to_a \langle m, a_2'\rangle}{\langle m, n_1 \oplus_a a_2\rangle \to_a \langle m, n_1 \oplus_a a_2'\rangle}\ \text{a-aop2} \qquad \frac{n = n_1 \oplus n_2}{\langle m, n_1 \oplus_a n_2\rangle \to_a \langle m, n\rangle}\ \text{a-aop3}$$

$$\boxed{\langle m, b\rangle \to_b \langle m, b'\rangle}$$

$$\frac{\langle m, a_1\rangle \to_a \langle m, a_1'\rangle}{\langle m, a_1 \sim a_2\rangle \to_b \langle m, a_1' \sim a_2\rangle}\ \text{b-rop1} \qquad \frac{\langle m, a_2\rangle \to_a \langle m, a_2'\rangle}{\langle m, n_1 \sim a_2\rangle \to_b \langle m, n_1 \sim a_2'\rangle}\ \text{b-rop2}$$

$$\frac{\text{true} = n_1 \sim n_2}{\langle m, n_1 \sim n_2\rangle \to_a \langle m, \text{true}\rangle}\ \text{b-rop3} \qquad \frac{\text{false} = n_1 \sim n_2}{\langle m, n_1 \sim n_2\rangle \to_a \langle m, \text{false}\rangle}\ \text{b-rop4}$$

$$\boxed{\langle m, e_q\rangle \to_q \langle m, e_q'\rangle}$$

$$\frac{\langle m, a\rangle \to_a \langle m, a'\rangle}{\langle m, a\rangle \to_q \langle m, a'\rangle}\ \text{q-a} \qquad \frac{\langle m, a\rangle \to_a \langle m, a'\rangle}{\langle m, \chi[a]\rangle \to_q \langle m, \chi[a']\rangle}\ \text{q-chi}$$

$$\frac{\langle m, e_{q_1}\rangle \to_q \langle m, e_{q_1}'\rangle}{\langle m, e_{q_1} \oplus_a e_{q_2}\rangle \to_q \langle m, e_{q_1}' \oplus_a e_{q_2}\rangle}\ \text{q-aop1}$$

$$\frac{\langle m, e_{q_2}\rangle \to_a \langle m, e_{q_2}'\rangle}{\langle m, v_{q_1} \oplus_a e_{q_2}\rangle \to_q \langle m, v_{q_1} \oplus_a e_{q_2}'\rangle}\ \text{q-aop2}$$

Figure C.2: Operational semantics of loop language, part 1

$$\boxed{\langle m,c,t,w\rangle \to \langle m',c',t',w'\rangle}\ \frac{\langle m,e\rangle \to \langle m,e'\rangle}{\langle m,[x \leftarrow e]^l,t,w\rangle \to \langle m,[x \leftarrow e']^l,t,w\rangle}\ \textbf{l-assn1}$$

$$\frac{}{\langle m,[x \leftarrow v]^l,t,w\rangle \to \langle m[v/x],[\texttt{skip}]^l,t,w\rangle}\ \textbf{l-assn2}$$

$$\frac{\langle m,a\rangle \to_a \langle m,a'\rangle}{\langle m,\texttt{loop}\ [a]^l\ \texttt{do}\ c,t,w\rangle \to \langle m,\texttt{loop}\ [a']^l\ \texttt{do}\ c,t,(w+l)\rangle}\ \textbf{l-loop-a}$$

$$\frac{v_N > 0}{\langle m,\texttt{loop}\ [v_N]^l\ \texttt{do}\ c,t,w\rangle \to \langle m,c;\texttt{loop}\ [(v_N-1)]^l\ \texttt{do}\ c,t,(w+l)\rangle}\ \textbf{l-loop}$$

$$\frac{v_N = 0}{\langle m,\texttt{loop}\ [v_N]^l\ \texttt{do}\ c,t,w\rangle \to \langle m,[\texttt{skip}]^l,t,(w \setminus l)\rangle}\ \textbf{l-loop-exit}$$

$$\frac{\langle m,e_q\rangle \to_q \langle m,e_q'\rangle}{\langle m,[x \leftarrow q(e_q)]^l,t,w\rangle \to \langle m,[x \leftarrow q(e_q')]^l,t,w\rangle}\ \textbf{l-query-e}$$

$$\frac{q(v_q) = v}{\langle m,[x \leftarrow q(v_q)]^l,t,w\rangle \to \langle m[v/x],\texttt{skip},t +\!\!+ [q(v_q)^{(l,w)}],w\rangle}\ \textbf{l-query-v}$$

$$\frac{\langle m,c_1,t,w\rangle \to \langle m',c_1',t',w'\rangle}{\langle m,c_1;c_2,t,w\rangle \to \langle m',c_1';c_2,t',w'\rangle}\ \textbf{l-seq1} \qquad \frac{}{\langle m,[\texttt{skip}]^l;c_2,t,w\rangle \to \langle m,c_2,t,w\rangle}\ \textbf{l-seq2}$$

$$\frac{\langle m,b\rangle \to_b b'}{\langle m,\text{if } ([b]^l,c_1,c_2),t,w\rangle \to \langle m,\text{if } ([b']^l,c_1,c_2),t,w\rangle}\ \textbf{l-if}$$

$$\frac{}{\langle m,\text{if } ([\text{true}]^l,c_1,c_2),t,w\rangle \to \langle m,c_1,t,w\rangle}\ \textbf{l-if-t}$$

$$\frac{}{\langle m,\text{if } ([\text{false}]^l,c_1,c_2),t,w\rangle \to \langle m,c_2,t,w\rangle}\ \textbf{l-if-f}$$

Figure C.3: Operational semantics of loop language, part 2

Operational Semantics. We define the operations on the loop ma w, where w_l refers to a map w without the key l.

$$\begin{array}{lll} w \setminus l & = w & l \notin Keys(w) \\ & = w_l & Otherwise \\ w + l & = w[l \rightarrow 0] & l \notin Keys(w) \\ & w_l[l \rightarrow w(l) + 1] & Otherwise \end{array}$$

C.1.2 Adaptivity of Programs in Loop Language

Definition 11 (Label Order)

$<_w \; and \; =_w$.

$$w_1 =_w w_2 \triangleq Keys(w_1) = Keys(w_2) \wedge \forall k \in Keys(w_1).w_1(k) = w_2(k)$$

$$\emptyset =_w \emptyset$$

$$mk(w_i) = MinKey(w_i)$$

$$\begin{aligned} w_1 <_w w_2 &\triangleq && w_1 = \emptyset \\ &\triangleq && mk(w_1) < mk(w_2) && w_1, w_2 ! = \emptyset \\ &\triangleq && w_1(mk(w_1)) < w_2(mk(w_2)) && mk(w_1) = mk(w_2) \\ &\triangleq && (w_1 \setminus mk(w_1)) <_w (w_2 \setminus mk(w_2)) && Otherwise \end{aligned}$$

Definition 12 (Query may dependency)

One query $q(v_{q_1})$ may depend on another query $q(v_{q_2})$ in a program c, with a starting loop maps w, a starting memory m, hidden database D, denoted as

$\mathsf{DEP}(q(v_{q_1})^{(l_1,w_1)}, q(v_{q_2})^{(l_2,w_2)}, c, w, m, D)$ is defined as below.

$$
\begin{array}{l}
\forall t.\exists m_1, m_3, t_1, t_3, c_2. \\
\left(\begin{array}{l}
\langle m, c, t, w\rangle \rightarrow^* \langle m_1, [x \leftarrow q(v_{q_1})]^{l_1}; c_2, t_1, w_1\rangle \rightarrow \\
\langle m_1[q(v_{q_1})(D)/x], c_2, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle m_3, \mathtt{skip}, t_3, w_3\rangle \\
\wedge \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \wedge q(v_{q_2})^{(l_2,w_2)} \in_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), m_3', t_3', w_3'. \\
\langle m_1[v/x], c_2, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle m_3', \mathtt{skip}, t_3', w_3'\rangle \\
\wedge (q(v_{q_2})^{(l_2,w_2)}) \notin_q (t_3' - t_1)\Big) \\
\wedge \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \wedge q(v_{q_2})^{(l_2,w_2)} \notin_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), m_3', t_3', w_3'. \\
\langle m_1[v/x], c_2, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle m_3', \mathtt{skip}, t_3', w_3'\rangle \\
\wedge (q(v_{q_2})^{(l_2,w_2)}) \in_q (t_3' - t_1)\Big)
\end{array}\right)
\end{array}
$$

Definition 13 (Query-based Dependency Graph)

Given a program c, a database D, a starting memory m, an initial loop maps w, the query-based dependency graph $G(c, D, m, w) = (V, E)$ is defined as:

$V = \{q(v_q)^{l,w} \in \mathscr{AQ} \mid \forall t.\exists m', w', t'.\langle m, c, t, w\rangle \rightarrow^* \langle m', \mathtt{skip}, t', w'\rangle \wedge q(v_q)^{l,w} \in (t' - t)\}$.

$E = \left\{(q(v_q)^{(l,w)}, q(v_q')^{(l',w')}) \in \mathscr{AQ} \times \mathscr{AQ} \;\middle|\; \mathsf{DEP}(q(v_q')^{(l',w')}, q(v_q)^{(l,w)}, c, w, m, D)\right\}$.

Definition 14 (Adaptivity in loop language)

Given a program c, and a memory m, a database D, a starting loop maps w, the adaptivity of the dependency graph $G(c, D, m, w) = (V, E)$ is the length of the longest path in this graph. We denote the path from $q(v_q)^{(l,w)}$ to $q(v_q')^{(l',w')}$ as $p(q(v_q)^{(l,w)}, q(v_q')^{(l',w')})$.

The adaptivity denoted as $A(c, D, m, w)$.

$$A(c, D, m, w) = \max_{q(v_q)^{(l,w)}, q(v'_q)^{(l',w')} \in V} |p(q(v_q)^{(l,w)}, q(v'_q)^{(l',w')})|$$

C.2 THE SSA LANGUAGE

Arithmatic Operators	$\oplus_a$	::=	$+ \mid - \mid \times \mid \div$
Boolean Operators	$\oplus_b$	::=	$\vee \mid \wedge \mid \neg$
Relational Operators	$\sim$	::=	$< \mid \leq \mid ==$
Label	l	:=	$\mathbb{N}$
Loop Maps	w	$\in$	Label $\times \mathbb{N}$
Arithmetic Expressions	$\mathbf{a}$	::=	$n \mid x \mid \mathbf{a} \oplus_a \mathbf{a} \mid [] \mid [\mathbf{a}, \ldots, \mathbf{a}]$
Boolean Expressions	b	::=	$\text{true} \mid \text{false} \mid \neg\mathbf{b} \mid \mathbf{b} \oplus_b \mathbf{b} \mid \mathbf{a} \sim \mathbf{a}$
Expressions	$\mathbf{e}$	::=	$\mathbf{a} \mid \mathbf{b}] \mid [\mathbf{e}, \ldots, \mathbf{e}]$
Values	v	::=	$n \mid \text{true} \mid \text{false} \mid [] \mid [v, \ldots, v]$
Query expressions	$\mathbf{e_q}$	::=	$\mathbf{a} \mid \chi \mid \chi[\mathbf{a}] \mid \mathbf{e_q} \oplus_a \mathbf{e_q}$
Query Values	v_q	::=	$n \mid \chi \mid \chi[n] \mid v_q \oplus_a v_q$
Labelled commands	$\mathbf{c}$	::=	$[\mathbf{x} \leftarrow \mathbf{e}]^l \mid [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l \mid ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}') \mid [\texttt{skip}]^l$
			$\texttt{loop}\ [\mathbf{a}]^l, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c} \mid$
			$\mathbf{c}; \mathbf{c} \mid \text{if } ([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], \mathbf{c}, \mathbf{c})$
Memory	m	::=	$\emptyset \mid (\mathbf{x^l} \rightarrow \mathbf{v}) :: \mathbf{m}$
Trace	t	::=	$[] \mid (q(v_q)^{(l,w)}) :: t$
Annotated Query	$\mathscr{AQ}$	::=	$\{q(v_q)^{(l,w)}\}$
SSA Variables	$\mathscr{SV}$	::=	$\{\mathbf{x}\}$
Labelled SSA Variables	$\mathscr{LV}$	::=	$\{\mathbf{x}^{(l,w)}\}$

Figure C.4: Syntax of the SSA loop language

C.2.1 SSA Transformation Rules

We use a translation environment δ, to map variables x in the low level language to those $\mathbf{x}$ in ssa-form language. We use a name environment denoted as Σ, a set of ssa variables so we can get a fresh variable by $fresh(\Sigma)$. We define $\delta_1 \bowtie \delta_2$ in a similar way as Vekris et al. (2016b).

$$\delta_1 \bowtie \delta_2 = \{(x, \mathbf{x_1}, \mathbf{x_2}) \in \mathcal{VAR} \times \mathcal{SV} \times \mathcal{SV} \mid x \mapsto \mathbf{x_1} \in \delta_1, x \mapsto \mathbf{x_2} \in \delta_2, \mathbf{x_1} \neq \mathbf{x_2}\}$$

$$\delta_1 \bowtie \delta_2 / \bar{x} = \{(x, \mathbf{x_1}, \mathbf{x_2}) \in \mathcal{VAR} \times \mathcal{SV} \times \mathcal{SV} \mid x \notin \bar{x} \wedge x \mapsto \mathbf{x_1} \in \delta_1, x \mapsto \mathbf{x_2} \in \delta_2, \mathbf{x_1} \neq \mathbf{x_2}\}$$

We call a list of variables $\bar{x}$.

$$[\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \{(x, x_1, x_2) | \forall 0 \leq i < |\bar{x}|, x = \bar{x}[i] \wedge x_1 = \bar{x_1}[i] \wedge x_2 = \bar{x_2}[i] \wedge |\bar{x}| = |\bar{x_1}| = |\bar{x_2}|\}$$

$$\boxed{\delta; e \hookrightarrow \mathbf{e}}$$

$$\frac{}{\delta; x \hookrightarrow \delta(x)}\ \textbf{S-VAR}$$

$$\boxed{\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'}$$

$$\frac{\begin{array}{c}\delta; b \hookrightarrow \mathbf{b} \qquad \Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma_1; \delta; c_2 \hookrightarrow \mathbf{c_2}; \delta_2; \Sigma_2 \\ [\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \delta_1 \bowtie \delta_2 \qquad [\bar{y}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}] = \delta \bowtie \delta_1 / \bar{x} \qquad [\bar{z}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}] = \delta \bowtie \delta_2 / \bar{x} \\ \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'][\bar{y} \mapsto \bar{\mathbf{y}}'][\bar{z} \mapsto \bar{\mathbf{z}}'] \qquad \bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}'\ fresh(\Sigma_2) \qquad \Sigma' = \Sigma_2 \cup \{\bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}'\}\end{array}}{\Sigma; \delta; [\text{if } (b, c_1, c_2)]^l \hookrightarrow [\text{if } (\mathbf{b}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}', \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}', \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^l; \delta'; \Sigma'}\ \textbf{S-IF}$$

$$\frac{\delta; e \hookrightarrow \mathbf{e} \qquad \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{\mathbf{x}\}}{\Sigma; \delta; [x \leftarrow e]^l \hookrightarrow [\mathbf{x} \leftarrow \mathbf{e}]^l; \delta'; \Sigma'}\ \textbf{S-ASSN}$$

$$\frac{\delta; e_q \hookrightarrow \mathbf{e_q} \qquad \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{\mathbf{x}\}}{\Sigma; \delta; [x \leftarrow q(e_q))]^l \hookrightarrow [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l; \delta'; \Sigma'}\ \textbf{S-QUERY}$$

$$\frac{\begin{array}{c}\delta; a \hookrightarrow \mathbf{a} \qquad \Sigma; \delta; c \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma; \delta_1; c \hookrightarrow \mathbf{c_2}; \delta_1; \Sigma_1 \\ [\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \delta \bowtie \delta_1 \qquad \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'] \\ \bar{\mathbf{x}}'\ fresh(\Sigma_1) \qquad \mathbf{c}' = \mathbf{c_1}[\bar{\mathbf{x}}'/\bar{\mathbf{x_1}}] = \mathbf{c_2}[\bar{\mathbf{x}}'/\bar{\mathbf{x_2}}]\end{array}}{\Sigma; \delta; [\texttt{loop}\ a\ \texttt{do}\ c]^l \hookrightarrow [\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}']^l; \delta_1[\bar{x} \to \bar{\mathbf{x}}']; \Sigma \cup \{\bar{\mathbf{x}}'\}}\ \textbf{S-LOOP}$$

$$\frac{\Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma_1; \delta_1; c_2 \hookrightarrow \mathbf{c_2}; \delta'; \Sigma'}{\Sigma; \delta; c_1; c_2 \hookrightarrow \mathbf{c_1}; \mathbf{c_2}\ ; \delta'; \Sigma'}\ \textbf{S-SEQ}$$

Figure C.5: SSA transformation rules

$$
\begin{array}{lll}
 & \left[x \leftarrow q(\chi[1])\right]^1; & \left[\mathbf{x_1} \leftarrow q(\chi[1])\right]^1; \\
 & \text{if } (x == 0)^2 & \text{if } (\mathbf{x_1} == 0)^2, [\mathbf{y_3}, \mathbf{y_1}, \mathbf{y_2}], [], [] \\
 & \text{then } \left[y \leftarrow q(\chi[2])\right]^3 & \text{then } \left[\mathbf{y_1} \leftarrow q(\chi[2])\right]^3 \\
c_1 \triangleq & \text{else } \left[y \leftarrow 0\right]^4; & \hookrightarrow \quad \text{else } \left[\mathbf{y_2} \leftarrow 0\right]^4; \\
 & \text{if } (x == 1)^5 & \text{if } (\mathbf{x_1} == 1)^5, [\mathbf{y_6}, \mathbf{y_4}, \mathbf{y_5}] \\
 & \text{then } \left[y \leftarrow 0\right]^6 & \text{then } \left[\mathbf{y_4} \leftarrow 0\right]^6 \\
 & \text{else } \left[y \leftarrow q(\chi[2])\right]^7 & \text{else } \left[\mathbf{y_5} \leftarrow q(\chi[2])\right]^7
\end{array}
$$

$$
\begin{array}{lll}
 & \left[x \leftarrow q(\chi[1])\right]^1; & \left[\mathbf{x_1} \leftarrow q(\chi[1])\right]^1; \\
 & \left[y \leftarrow q(x + \chi[1])\right]^2; & \left[\mathbf{y_1} \leftarrow q(\mathbf{x_1} + \chi[1])\right]^2; \\
c_2 \triangleq & \text{if } (x + y == 5)^3 & \text{if } (\mathbf{x_1} + \mathbf{y_1} == 5)^3, [], [], [] \\
 & \text{then } \left[z \leftarrow q(\chi[3])\right]^4 & \hookrightarrow \quad \text{then } \left[\mathbf{z_1} \leftarrow q(\chi[3])\right]^4 \\
 & \text{else } \left[\texttt{skip}\right]^5; & \text{else } \left[\texttt{skip}\right]^5; \\
 & \left[w \leftarrow q(\chi[4])\right]^6; & \left[\mathbf{w_1} \leftarrow q(\chi[4])\right]^6;
\end{array}
$$

$$
\begin{array}{lll}
 & \left[x \leftarrow q(\chi[1])\right]^1; & \left[\mathbf{x_1} \leftarrow q(\chi[1])\right]^1; \\
 & \texttt{loop}\ 100^2\ \texttt{do} & \texttt{loop} 100^2, 0, [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]\ \texttt{do} \\
c_3 \triangleq & \left(\left[y \leftarrow q(\chi[2] + x)\right]^3; \right. & \hookrightarrow \quad \left(\left[\mathbf{y_1} \leftarrow q(\chi[2] + \mathbf{x_3})\right]^3; \right. \\
 & \left[z \leftarrow q(\chi[3])\right]^4; & \left[\mathbf{z_1} \leftarrow q(\chi[3])\right]^4; \\
 & \left. \left[x \leftarrow y + z + x\right]^5 \right); & \left. \left[\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{z_1} + \mathbf{x_3}\right]^5 \right);
\end{array}
$$

Figure C.6: SSA transformation examples

Definition 15 (Well defined memory) 1. $m \vDash \delta \triangleq \forall x \in \texttt{dom}(\delta), \exists v, (x, v) \in m$.

2. $\mathbf{m} \vDash_{ssa} \delta \triangleq \forall \mathbf{x} \in \texttt{codom}(\delta), \exists v, (\mathbf{x}, v) \in \mathbf{m}$.

Definition 16 (Inverse of transformed memory)

$m = \delta^{-1}(\mathbf{m}) \triangleq \forall x \in \texttt{dom}(\delta), (\delta(x), m(x)) \in \mathbf{m}$.

Lemma 85 (value remains during trans)**.** *Given* $\delta; e \hookrightarrow \mathbf{e}$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *then* $\langle m, e\rangle \rightarrow v$ *and* $\langle \mathbf{m}, \mathbf{e}\rangle \rightarrow \mathbf{v}$ *and* $v = \mathbf{v}$.

C.2.1.1 Proof of Theorem 16

Theorem 86 (Soundness of transformation)**.** *Given* $\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'$,
$\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, *if there exist an execution of* c *in the loop language, starting with a trace* t *and loop maps* w, $\langle m, c, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle$, *then there also exists a corresponding execution of* $\mathbf{c}$ *in the ssa language so that* $\langle \mathbf{m}, \mathbf{c}, t, w\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t', w'\rangle$ *and* $m' = \delta'^{-1}(\mathbf{m}')$.

Proof. We assume that q is the same when transformed to $\mathbf{q}$, as the primitive in both languages. And the value remains the same during the transformation. It is proved by induction on the transformation rules.

- **Case** $\dfrac{\Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma_1; \delta_1; c_2 \hookrightarrow \mathbf{c_2}; \delta'; \Sigma'}{\Sigma; \delta; c_1; c_2 \hookrightarrow \mathbf{c_1}; \mathbf{c_2}; \delta'; \Sigma'}$ **S-SEQ**

  We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w.

  $$\frac{\langle m, c_1, t, w\rangle \rightarrow^* \langle m_1, \texttt{skip}, t_1, w_1\rangle \qquad \langle m_1, c_2, t_1, w_1\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle}{\langle m, c_1; c_2, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle}$$

  We choose the transformed memory $\mathbf{m}$ so that $m = \delta^{-1}(\mathbf{m})$.

To show: $\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t'.w'\rangle$ and $m' = \delta'^{-1}(\mathbf{m}')$.

By induction hypothesis on the first premise, we have:

$$\langle \mathbf{m}, \mathbf{c_1}, t, w\rangle \rightarrow^* \langle \mathbf{m_1}, \texttt{skip}, t_1, w_1\rangle \wedge m_1 = \delta_1^{-1}(\mathbf{m_1})$$

By induction hypothesis on the second premise, using conclusion $m_1 = \delta_1^{-1}(\mathbf{m_1})$. We have:

$$\langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t', w'\rangle \wedge m' = \delta'^{-1}(\mathbf{m}')$$

So we know that

$$\frac{\langle \mathbf{m}, \mathbf{c_1}, t, w\rangle \rightarrow^* \langle \mathbf{m_1}, \texttt{skip}, t_1, w_1\rangle \qquad \langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t', w'\rangle}{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t', w'\rangle}$$

- **Case** $\dfrac{\delta; e \hookrightarrow \mathbf{e} \qquad \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{x\}}{\Sigma; \delta; [x \leftarrow e]^l \hookrightarrow [\mathbf{x} \leftarrow \mathbf{e}]^l; \delta'; \Sigma'}$ **S-ASSN**

We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w, we know that the resulting trace is still t from its evaluation rule **l-assn** when we suppose $m, e \rightarrow v$.

$$\frac{}{\langle m, [x \leftarrow v]^l, t, w\rangle \rightarrow \langle m[v/x], [\texttt{skip}]^l, t, w\rangle}\ \textbf{l-assn}$$

We choose the transformed memory $\mathbf{m}$ so that $m = \delta^{-1}(\mathbf{m})$.

To show: $\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{e}]^l, t, w\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t, w\rangle$ and $m' = \delta'^{-1}(\mathbf{m}')$.

From the rule **ssa-assn**, we assume $\mathbf{m}, \mathbf{e} \rightarrow \mathbf{v}$, we know that

$$\frac{}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{v}]^{\mathbf{l}}, t, w \rangle \rightarrow \langle \mathbf{m}[\mathbf{x} \mapsto \mathbf{v}], [\texttt{skip}]^{\mathbf{l}}, t, w \rangle} \textbf{ ssa-assn}$$

We also know that $\delta' = \delta[x \mapsto \mathbf{x}]$ and $m = \delta^{-1}(\mathbf{m})$, $m' = m[v/x]$. We can show that $m[v/x] = \delta[x \mapsto \mathbf{x}]^{-1}(\mathbf{m}[\mathbf{x} \mapsto v])$.

- **Case** $\dfrac{\begin{array}{c} \delta; e_q \hookrightarrow \mathbf{e_q} \\ \delta' = \delta[x \mapsto \mathbf{x}] \qquad \mathbf{x}\ fresh(\Sigma) \qquad \Sigma' = \Sigma \cup \{x\} \end{array}}{\Sigma; \delta; [x \leftarrow q(e_q)]^l \hookrightarrow [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l; \delta'}$ **S-QUERY**

We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w, we know that when we suppose $\langle m, e_q \rangle \rightarrow_q \langle m, v_q \rangle$.

$$\frac{q(v_q)(D) = v}{\langle m, [x \leftarrow q(v_q)]^l, t, w \rangle \rightarrow \langle m[v/x], \texttt{skip}, t \mathbin{+\!\!+} [q(v_q)^{(l,w)}], w \rangle} \textbf{ l-query}$$

We choose the transformed memory $\mathbf{m}$ so that $m = \delta^{-1}(\mathbf{m})$.

To show: $\langle \mathbf{m}, [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l, t, w \rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t, w \rangle$ and $m' = \delta'^{-1}(\mathbf{m}')$.

From the rule **ssa-query**, we know that

$$\frac{q(v_q)(D) = v}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{q}(\mathbf{e_q})]^{\mathbf{l}}, t, w \rangle \rightarrow^* \langle \mathbf{m}[\mathbf{x} \mapsto v], \texttt{skip}, t \mathbin{+\!\!+} [q(v_q)^{(l,w)}], w \rangle}$$

We also know that $\delta' = \delta[x \mapsto \mathbf{x}]$ and $m = \delta^{-1}(\mathbf{m})$, $m' = m[v/x]$. We can show that $m[v/x] = \delta[x \mapsto \mathbf{x}]^{-1}(\mathbf{m}[\mathbf{x} \mapsto v])$.

- **Case** $\dfrac{\begin{array}{c}\delta; b \hookrightarrow \mathbf{b} \qquad \Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1 \qquad \Sigma_1; \delta; c_2 \hookrightarrow \mathbf{c_2}; \delta_2; \Sigma_2 \\ [\bar{x}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] = \delta_1 \bowtie \delta_2 \qquad [\bar{y}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}] = \delta \bowtie \delta_1 / \bar{x} \qquad [\bar{z}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}] = \delta \bowtie \delta_2 / \bar{x} \\ \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'][\bar{y} \mapsto \bar{\mathbf{y}}'][\bar{z} \mapsto \bar{\mathbf{z}}'] \qquad \bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}' \; fresh(\Sigma_2) \qquad \Sigma' = \Sigma_2 \cup \{\bar{\mathbf{x}}', \bar{\mathbf{y}}', \bar{\mathbf{z}}'\}\end{array}}{\Sigma; \delta; [\textbf{if } (b, c_1, c_2)]^l \hookrightarrow [\textbf{if } (\mathbf{b}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}', \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}', \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^l; \delta'; \Sigma'}$ **S-IF**

We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w. There are two possible evaluation rules depending on the the condition b, we choose the case when b = true, we know there is an execution in ssa language so that $\mathbf{b}$ = true, we use the rule **l-if-t**.

$$\frac{}{\langle m, [\text{if } (\text{true}, c_1, c_2)]^l, t, w\rangle \rightarrow \langle m, c_1, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle}$$

We choose the transformed memory $\mathbf{m}$ so that $m = \delta^{-1}(\mathbf{m})$.

TS: $\langle \mathbf{m}, [\text{if } (\text{true}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}', \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}', \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], c_1, c_2)]^l, t, w\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t', w'\rangle$ and $m' = \delta'^{-1}(\mathbf{m}')$.

We use the corresponding rule **SSA-IF-T**.

$$\frac{}{\begin{array}{l}\langle \mathbf{m}, [\text{if } (\text{true}, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}', \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}', \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{\mathbf{l}}, t, w\rangle \rightarrow \\ \quad \langle \mathbf{m}, \mathbf{c_1}; ifvar(\bar{\mathbf{x}}', \bar{\mathbf{x_1}}); ifvar(\bar{\mathbf{y}}', \bar{\mathbf{y_2}}); ifvar(\bar{\mathbf{z}}', \bar{\mathbf{z_1}}), t, w\rangle\end{array}} \textbf{ssa-if-t}$$

By induction hypothesis on $\Sigma; \delta; c_1 \hookrightarrow \mathbf{c_1}; \delta_1; \Sigma_1$, and we know that $\langle m, c_1, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle$, from our assumption that $m = \delta^{-1}(\mathbf{m})$, we know that

$$\langle \mathbf{m}, \mathbf{c_1}, t, w\rangle \rightarrow^* \langle \mathbf{m_1}, \texttt{skip}, t', w'\rangle \wedge m' = \delta_1^{-1}(\mathbf{m_1})$$

and we then have:

$$\frac{\langle \mathbf{m}, \mathbf{c_1}, t, w\rangle \rightarrow^* \langle \mathbf{m_1}, \texttt{skip}, t', w'\rangle}{\begin{array}{c}\langle \mathbf{m}, \mathbf{c_1}; ifvar(\bar{\mathbf{x}}', \bar{\mathbf{x}_1}); ifvar(\bar{\mathbf{y}}', \bar{\mathbf{y}_1}); ifvar(\bar{\mathbf{z}}', \bar{\mathbf{z}_1}), t, w\rangle \rightarrow^* \\ \langle \mathbf{m_1}[\bar{\mathbf{x}}' \mapsto \mathbf{m_1}(\bar{\mathbf{x}_1}), \bar{\mathbf{y}}' \mapsto \mathbf{m_1}(\bar{\mathbf{y}_2}), \bar{\mathbf{z}}' \mapsto \mathbf{m_1}(\bar{\mathbf{z}_1})], \texttt{skip}, t', w'\rangle\end{array}}$$

Now, we want to show that

$m' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}', \bar{y} \mapsto \bar{\mathbf{y}}', \bar{z} \mapsto \bar{\mathbf{z}}']^{-1}(\mathbf{m_1}[\bar{\mathbf{x}}' \mapsto \mathbf{m_1}(\bar{\mathbf{x}_1}), \bar{\mathbf{y}}' \mapsto \mathbf{m_1}(\bar{\mathbf{y}_2}), \bar{\mathbf{z}}' \mapsto \mathbf{m_1}(\bar{\mathbf{z}_1})])$.

Unfold the definition, we want to show that $\forall x \in (\texttt{dom}(\delta) \cup \bar{x} \cup \bar{y} \cup \bar{z}), (\delta[\bar{x} \mapsto \bar{\mathbf{x}}', \bar{y} \mapsto \bar{\mathbf{y}}', \bar{z} \mapsto \bar{\mathbf{z}}'](x), m'(x))$
$\in \mathbf{m_1}[\bar{\mathbf{x}}' \mapsto \mathbf{m_1}(\bar{\mathbf{x}_1}), \bar{\mathbf{y}}' \mapsto \mathbf{m_1}(\bar{\mathbf{y}_2}), \bar{\mathbf{z}}' \mapsto \mathbf{m_1}(\bar{\mathbf{z}_1})]$.

1. For variable x in $\bar{x}$, we can find a corresponding ssa variable $\mathbf{x} \in \bar{\mathbf{x}}'$, so that $(\mathbf{x}, m'(x)) \in \mathbf{m_1}[\bar{\mathbf{x}}' \mapsto \mathbf{m_1}(\bar{\mathbf{x}_1})]$. It is because we know $[x \mapsto \mathbf{x_1}]$ for certain $\mathbf{x_1} \in \bar{\mathbf{x}_1}$ in δ_1, then by unfolding $m' = \delta_1^{-1}(\mathbf{m_1})$ and $\bar{\mathbf{x}_1} \in \texttt{codom}(\delta_1)$, we know $(\mathbf{x_1}, m'(x)) \in \mathbf{m_1}$ so that $m'(x) = \mathbf{m_1}(\mathbf{x_1})$.

2. For variable $y \in \bar{y}$, we know that $y \in \texttt{dom}(\delta_1)$, then $[y \mapsto \mathbf{y_2}]$ for certain $\mathbf{y_2} \in \bar{\mathbf{y}_2}$ in δ_1. So we know that $(\delta_1(y), m'(y)) \in \mathbf{m_1}$, and then $m'(y) = \mathbf{m_1}(\mathbf{y_2})$. We can show $(\mathbf{y}, m'(y)) \in \mathbf{m_1}[\bar{\mathbf{y}}' \mapsto \mathbf{m_1}(\bar{\mathbf{y}_2})]$.

3. For variable $z \in \bar{z}$, we know that $z \notin \texttt{dom}(\delta_1)$ by the definition (otherwise z will appear in $\bar{x}$), then $[z \mapsto \mathbf{z_1}]$ for certain $\mathbf{z_1} \in \bar{\mathbf{z}_1}$ in δ. We know $(\delta(z), m(z)) \in \mathbf{m}$ from our assumption, so we have $m(z) = \mathbf{m}(\mathbf{z_1})$. Because z is not modified in c_1, so that $m(z) = m'(z)$. Also $\mathbf{m}$ will not shrink during execution and $\mathbf{z_1}$ will not be written in $\mathbf{c_1}$, so $(\mathbf{z_1}, m'(z)) \in \mathbf{m_1}$. Then we can show that $(\mathbf{z}, m'(z)) \in \mathbf{m_1}[\bar{\mathbf{z}}' \mapsto \mathbf{m_1}(\bar{\mathbf{z}_1})]$.

4. For variable $k \in \texttt{dom}(\delta) - \bar{x} - \bar{y} - \bar{z}$. From our assumption $m = \delta^{-1}(\mathbf{m})$, we can show $(\delta(k), m(k)) \in \mathbf{m}$. We know that k is not written in either branch from

our definition, so $(\delta(k), m'(k)) \in \mathbf{m_1}$.

- **Case** $\dfrac{\begin{array}{c}\delta; a \hookrightarrow \mathbf{a} \qquad \Sigma;\delta;c \hookrightarrow \mathbf{c_1};\delta_1;\Sigma_1 \qquad \Sigma;\delta_1;c \hookrightarrow \mathbf{c_2};\delta_1;\Sigma_1 \\ [\bar{x},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}] = \delta \bowtie \delta_1 \qquad \delta' = \delta[\bar{x} \mapsto \bar{\mathbf{x}}'] \\ \bar{\mathbf{x}}'\ fresh(\Sigma_1) \qquad \mathbf{c}' = \mathbf{c_1}[\bar{\mathbf{x}}'/\bar{\mathbf{x}_1}] = \mathbf{c_2}[\bar{\mathbf{x}}'/\bar{\mathbf{x}_2}]\end{array}}{\delta; [\texttt{loop}\ a\ \texttt{do}\ c]^l \hookrightarrow [\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \texttt{do}\ \mathbf{c}']^l; \delta[\bar{x} \to \bar{\mathbf{x}}']; \Sigma_1 \cup \{\bar{\mathbf{x}}'\}}$ **S-LOOP**

We choose an arbitrary memory m so that $m \vDash \delta$, we choose a trace t and a loop maps w. Suppose $\langle m, a\rangle \to v_N$. There are two cases, when $v_N = 0$, the loop body is not executed so we can easily show that the trace is not modified.

When the loop is still running ($v_N > 0$), we have the following the evaluation:

$$\frac{v_N > 0}{\langle m, [\texttt{loop}\ v_N\ \texttt{do}\ c]^l, t, w\rangle \to \langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w + l)\rangle}\ \textbf{l-loop}$$

We then have the following evaluation:

$$\frac{\langle m, c, t, (w + l)\rangle \to^* \langle m', \texttt{skip}, t_1', w'\rangle}{\langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w + l)\rangle \to^* \langle m', [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t_1', w'\rangle}$$

We can have the following evaluation of the ssa-form using the rule **ssa-loop**, when we assume , $m = \delta^{-1}(\mathbf{m})$.

$$\frac{v_N > 0 \qquad n = 0 \implies i = 1 \qquad n > 0 \implies i = 2}{\begin{array}{c}\langle \mathbf{m}, [\texttt{loop}\ v_N, n, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t, w\rangle \to \\ \langle \mathbf{m}, \mathbf{c}'[\bar{\mathbf{x}_i}/\bar{\mathbf{x}}']; [\texttt{loop}\ (v_N - 1), n + 1, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \texttt{do}\ \mathbf{c}', t, (w + l)\rangle\end{array}}\ \textbf{ssa-loop}$$

Depending on if the counter n is equal to 0 or not, there are two possible execution paths (the variables $\bar{\mathbf{x}}$ is replaced by the $\bar{\mathbf{x}_1}$ or $\bar{\mathbf{x}_2}$). We start from the first iteration (when $n = 0$) when $v_N > 0$.

By induction hypothsis on the premise $\Sigma;\delta;c \hookrightarrow \mathbf{c_1};\delta_1;\Sigma_1$, we know that

$$\langle \mathbf{m}, \mathbf{c}'[\bar{\mathbf{x}_1}/\bar{\mathbf{x}}'], t, (w+l)\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t_i', w'\rangle \wedge m' = \delta_1^{-1}(\mathbf{m}')$$

Hence we can conclude that:

$$\frac{\langle \mathbf{m}, \mathbf{c}'[\bar{\mathbf{x}_1}/\bar{\mathbf{x}}'], t, (w+l)\rangle \rightarrow^* \langle \mathbf{m}', \texttt{skip}, t_1', w'\rangle}{\begin{array}{c}\langle \mathbf{m}, \mathbf{c}'[\bar{\mathbf{x}_1}/\bar{\mathbf{x}}']; [\texttt{loop } (\mathbf{v_N}-\mathbf{1}), \mathbf{n}+\mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}] \texttt{ do } \mathbf{c}']^{\mathbf{l}}, t, (w+l)\rangle \rightarrow^* \\ \langle \mathbf{m}', [\texttt{loop } (\mathbf{v_N}-\mathbf{1}), \mathbf{n}+\mathbf{1}, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}] \texttt{ do } \mathbf{c}']^{\mathbf{l}}, t_1', w'\rangle\end{array}}$$

Then there are two cases,

1. when $v_N - 1 = 0$ so that the loop exits. The l-level execution as follows by the evaluation rule **l-loop-exit**.

$$\frac{v_N - 1 = 0}{\langle m', [\texttt{loop } v_N - 1 \texttt{ do } c]^l, t_i', w'\rangle \rightarrow \langle m', [\texttt{skip}]^l, t_1', (w' \setminus l)\rangle}\ \textbf{l-loop-exit}$$

   The corresponding ssa-form evaluation as follows:

$$\frac{v_N - 1 = 0}{\begin{array}{c}\langle \mathbf{m}', [\texttt{loop } v_N - 1, n, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}] \texttt{ do } \mathbf{c}]^{\mathbf{l}}, t_1', w'\rangle \rightarrow \\ \langle \mathbf{m}'[\bar{\mathbf{x}} \mapsto \mathbf{m}'(\bar{\mathbf{x}_2})], [\texttt{skip}]^{\mathbf{l}}, t_1', (w' \setminus l)\rangle\end{array}}\ \textbf{ssa-loop-exit}$$

   We can see that both traces are not changed during the exit of the loop. We need to show that $m' = \delta^{-1}(\mathbf{m}'[\bar{\mathbf{x}} \mapsto \mathbf{m}'(\bar{\mathbf{x}_2})])$. We know that $[\bar{x} \mapsto \bar{x_2}]$ in δ_1 from the definition, so we can show that for any variable $\mathbf{x_2} \in \bar{x_2}$, $(\mathbf{x_2}, m'(x)) \in \mathbf{m}'$. For variables $x \in \texttt{dom}(\delta) - \bar{x}$, the variable is not modified during the execution of c so that we know $m(x) = m'(x)$, and then we can show that

$(\delta(x), m'(x)) \in \mathbf{m}'$ because $\delta(x)$ is not written in $\mathbf{c}'[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}']$.

2. when $v_n - 1 > 0$ so that the loop is still running.

   We want to show that : assuming in the $i-th$ ($i < v_N$) iteration, starting with t_i and w_i and $m_i = \delta_1^{-1}(\mathbf{m_i})$ so that the low level evaluation as follows $\langle m_i, [\texttt{loop } v_N - i \texttt{ do } c]^l, t_i, w_i\rangle \rightarrow^* \langle m_{i+1}, [\texttt{loop } (v_N - i - 1) \texttt{ do } c]^l, t_{i+1}, w_{i+1}\rangle$. Then the corresponding ssa form evaluation as follows :

$$\langle \mathbf{m_i}, [\texttt{loop } (v_N - i), n, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}']^{\mathbf{l}}, t_i, (w_i + l)\rangle \rightarrow^*$$
$$\langle \mathbf{m_{i+1}}, [\texttt{loop } (v_N - i - 1), n+1, [\bar{\mathbf{x}}', \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}']^{\mathbf{l}}, t_{i+1}, w_{i+1}\rangle$$

   and $m_i = \delta^{-1}(\mathbf{m_i})$.

   We then have the low level evauation:

$$\frac{v_N - i > 0}{\begin{array}{c}\langle m_i, [\texttt{loop } v_N - i \texttt{ do } c]^l, t_i, w_i\rangle \rightarrow \\ \langle m_i, c; [\texttt{loop } (v_N - i - 1) \texttt{ do } c]^l, t_i, (w_i + l)\rangle\end{array}} \textbf{ l-loop}$$

   We then have the following evaluation:

$$\frac{\langle m_i, c, t_i, (w_i + l)\rangle \rightarrow^* \langle m_{i+1}, \texttt{skip}, t_{i+1}, w_{i+1}\rangle}{\begin{array}{c}\langle m_i, c; [\texttt{loop } (v_N - i - 1) \texttt{ do } c]^l, t_i, (w_i + l)\rangle \rightarrow^* \\ \langle m_{i+1}, [\texttt{loop } (v_N - i - 1) \texttt{ do } c]^l, t_{i+1}, w_{i+1}\rangle\end{array}}$$

   By induction hypothsis on the premise $\Sigma; \delta_1; c \hookrightarrow \mathbf{c_2}; \delta_1; \Sigma_1$, we know that

$$\langle \mathbf{m_i}, \mathbf{c}'[\bar{\mathbf{x_2}}/\bar{\mathbf{x}}'], t_i, (w_i + l)\rangle \rightarrow^* \langle \mathbf{m_{i+1}}, \texttt{skip}, t_{i+1}, w_{i+1}\rangle \wedge m_{i+1} = \delta_1^{-1}(\mathbf{m_{i+1}})$$

Hence we can conclude that:

$$\frac{\langle \mathbf{m_i}, \mathbf{c}'[\bar{\mathbf{x}_2}/\bar{\mathbf{x}}'], t_i, (w_i + l)\rangle \rightarrow^* \langle \mathbf{m_{i+1}}, \texttt{skip}, t_{i+1}, w_{i+1}\rangle}{\begin{array}{c}\langle \mathbf{m_i}, \mathbf{c}'[\bar{\mathbf{x}_2}/\bar{\mathbf{x}}']; [\texttt{loop}\ (v_N - i - 1), n+1, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t_i, (w_i + l)\rangle \rightarrow^* \\ \langle \mathbf{m_{i+1}}, [\texttt{loop}\ (v_N - i - 1), n+1, [\bar{\mathbf{x}}', \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \texttt{do}\ \mathbf{c}']^{\mathbf{l}}, t_{i+1}, w_{i+1}\rangle\end{array}$$

So we can show that before the exit of the loop after $(v_N = n)$ iterations, we have $t_n = t_n$ and $m_n = \delta_1^{-1}(\mathbf{m_n})$. This proof is similar when it comes to the exit as in case 1.

□

Definition 17 (Query may dependency in SSA)

One query $q(v_{q_2})$ *may depend* on another query $q(v_{q_1})$ in a program $\mathbf{c}$, with a starting loop maps w, denoted as $\mathsf{DEP}_{\mathsf{ssa}}(q(v_{q_1})^{(l_1,w_1)}, q(v_{q_2})^{(l_2,w_2)}, \mathbf{c}, w, \mathbf{m}, D)$.

is defined as below.

$$
\begin{array}{l}
\forall t.\exists \mathbf{m_1}, \mathbf{m_3}, t_1, t_3, c_2. \\
\left(\begin{array}{l}
\langle \mathbf{m}, c, t, w\rangle \rightarrow^* \langle \mathbf{m_1}, [\mathbf{x} \leftarrow q(v_{q_1})]^{l_1}; \mathbf{c_2}, t_1, w_1\rangle \rightarrow \\
\langle \mathbf{m_1}[q(v_{q_1})(D)/\mathbf{x}], \mathbf{c_2}, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle \mathbf{m_3}, \mathtt{skip}, t_3, w_3\rangle \\
\wedge \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \wedge q(v_{q_2})^{(l_2,w_2)} \in_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), \mathbf{m_3'}, t_3', w_3'. \\
\langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle \mathbf{m_3'}, \mathtt{skip}, t_3', w_3'\rangle \\
\wedge (q(v_{q_2})^{(l_2,w_2)}) \notin_q (t_3' - t_1)\Big) \\
\wedge \\
\Big(q(v_{q_1})^{(l_1,w_1)} \in_q (t_3 - t) \wedge q(v_{q_2})^{(l_2,w_2)} \notin_q (t_3 - t_1) \\
\implies \exists v \in \mathtt{codom}(q(v_{q_1})), \mathbf{m_3'}, t_3', w_3'. \\
\langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 + +[q(v_{q_1})^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle \mathbf{m_3'}, \mathtt{skip}, t_3', w_3'\rangle \\
\wedge (q(v_{q_2})^{(l_2,w_2)}) \in_q (t_3' - t_1)\Big)
\end{array} \right)
\end{array}
$$

Definition 18 (Dependency Graph in SSA)

.

Given a program $\mathbf{c}$, a database D, a starting memory $\mathbf{m}$, an initial loop maps w, the dependency graph $G_s(\mathbf{c}, D, \mathbf{m}, w) = (V, E)$ is defined as:

$V = \{q(v_q)^{l,w} \in \mathscr{AQ} \mid \forall t.\exists \mathbf{m}', w', t'.\langle \mathbf{m}, \mathbf{c}, t, w\rangle \rightarrow^* \langle \mathbf{m}', \mathtt{skip}, t', w'\rangle \wedge q(v_q)^{l,w} \in (t' - t)\}$.

$E = \left\{ (q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')}) \in \mathscr{AQ} \times \mathscr{AQ} \;\middle|\; \mathsf{DEP}_{\mathsf{ssa}}(q(v_q{}')^{(l',w')}, q(v_q)^{(l,w)}, \mathbf{c}, w, \mathbf{m}, D) \right\}$.

Definition 19 (Adaptivity in SSA language)

Given a program $\mathbf{c}$, and a meory $\mathbf{m}$, a database D, a starting loop maps w, the adaptivity of the dependency graph $G_s(\mathbf{c}, D, \mathbf{m}, w) = (V, E)$ is the length of the longest path in this graph. We denote the path from $q(v_q)^{(l,w)}$ to $q(v_q{}')^{(l',w')}$ as $p(q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')})$.

The adaptivity denoted as $A_s(\mathbf{c}, D, \mathbf{m}, w)$.

$$A_s(\mathbf{c}, D, \mathbf{m}, w) = \max_{q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')} \in V} \{|p(q(v_q)^{(l,w)}, q(v_q{}')^{(l',w')})|\}$$

Theorem 87 (Same graph)**.** *Given* $\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'$*,*
$\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$*, starting with a trace* t *and loop maps* w*, then* $G(c, D, m, w) = G_s(\mathbf{c}, D, \mathbf{m}, w)$.

Proof. The nodes in the two graphs are the same according to soundness of the transformation theorem. We will show that the edges are the same. Unfold the definition of DEP, when we find some v in the codomain of one query that may change the appearance of another query, we can always show that $v = \mathbf{v}$ works for $\mathsf{DEP_s}$ as well. In the other direction, for any $\mathbf{v}$ that may change the appearance of another query and lead to a different trace t_3' in the ssa execution, there is the same $v = \mathbf{v}$ so that the trace t_3' is generated in the loop language execution, according to the transformation theorem. □

Corollary 87.1

Given $\Sigma; \delta; c \hookrightarrow \mathbf{c}; \delta'; \Sigma'$, $\forall m.m \vDash \delta.\forall \mathbf{m}, \mathbf{m} \vDash_{ssa} \delta \wedge m = \delta^{-1}(\mathbf{m})$, starting with a trace t and loop maps w, then $A(c, D, m, w) = A_s(\mathbf{c}, D, \mathbf{m}, w)$.

C.2.2 SSA Operational Semantics

the command ifvar$(\bar{x}, \bar{x}')$ stores the variable map during the run time, which is a map from ssa variable $\mathbf{x}$ to variable $\mathbf{x_i}$. This map is designed for if condition, when the variable may comes from two branches and this command records which branch the variable comes from.

$$\boxed{\langle \mathbf{m}, \mathbf{c}, \mathbf{t}, \mathbf{w} \rangle \rightarrow \langle \mathbf{m}', \mathbf{c}', \mathbf{t}', \mathbf{w}' \rangle}$$

$$\frac{q(v_q)(D) = v}{\langle \mathbf{m}, [\mathbf{x} \leftarrow q(v_q)]^l, t, w \rangle \rightarrow \langle \mathbf{m}[\mathbf{x} \mapsto v], \texttt{skip}, t +\!\!+ [q(v_q)^{(l,w)}], w \rangle} \textbf{ ssa-query}$$

$$\frac{}{\langle \mathbf{m}, [\mathbf{x} \leftarrow v]^l, t, w \rangle \rightarrow \langle \mathbf{m}[\mathbf{x} \mapsto v], [\texttt{skip}]^l, t, w \rangle} \textbf{ ssa-assn}$$

$$\frac{\langle \mathbf{m}, \mathbf{b} \rangle \rightarrow_b \mathbf{b}'}{\begin{array}{l}\langle \mathbf{m}, [\text{if } (\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{\mathbf{l}}, t, w \rangle \rightarrow \\ \langle \mathbf{m}, [\text{if } (\mathbf{b}', [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{\mathbf{l}}, t, w \rangle\end{array}} \textbf{ ssa-if}$$

$$\frac{}{\begin{array}{l}\langle \mathbf{m}, [\text{if } (\text{true}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{\mathbf{l}}, t, w \rangle \rightarrow \\ \langle m, c_1;\ ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x_1}}); ifvar(\bar{\mathbf{y}}, \bar{\mathbf{y_2}}); ifvar(\bar{\mathbf{z}}, \bar{\mathbf{z_1}}), t, w \rangle\end{array}} \textbf{ ssa-if-t}$$

$$\frac{}{\begin{array}{l}\langle \mathbf{m}, [\text{if } (\text{false}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{\mathbf{l}}, t, w \rangle \rightarrow \\ \langle m, c_2; ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x_2}}); ifvar(\bar{\mathbf{y}}, \bar{\mathbf{y_1}}); ifvar(\bar{\mathbf{z}}, \bar{\mathbf{z_2}}), t, w \rangle\end{array}} \textbf{ ssa-if-f}$$

$$\frac{}{\langle \mathbf{m}, ifvar(\bar{\mathbf{x}}, \bar{\mathbf{x}}'), t, w \rangle \rightarrow \langle \mathbf{m}[\bar{\mathbf{x}} \rightarrow \mathbf{m}(\bar{\mathbf{x}}')], \texttt{skip}, t, w \rangle} \textbf{ ssa-ifvar}$$

Figure C.7: SSA operational semantics, part 1

$$\frac{v_N > 0 \qquad n = 0 \implies i = 1 \qquad n > 0 \implies i = 2}{\begin{array}{c}\langle \mathbf{m}, [\texttt{loop}\ v_N, n, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, w\rangle \rightarrow \\ \langle \mathbf{m}, \mathbf{c}[\bar{\mathbf{x}_i}/\bar{\mathbf{x}}]; [\texttt{loop}\ (v_N - 1), n + 1, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, (w + l)\rangle\end{array}}\ \textbf{ssa-loop}$$

$$\frac{v_N = 0 \qquad n = 0 \implies i = 1 \qquad n > 0 \implies i = 2}{\langle \mathbf{m}, [\texttt{loop}\ v_N, n, [\bar{\mathbf{x}}, \bar{\mathbf{x}_1}, \bar{\mathbf{x}_2}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, w\rangle \rightarrow \langle \mathbf{m}[\bar{\mathbf{x}} \mapsto \mathbf{m}(\bar{\mathbf{x}_i})], [\texttt{skip}]^{\mathbf{l}}, t, (w \setminus l)\rangle}\ \textbf{ssa-loop-exit}$$

Figure C.8: SSA operational semantics, part 2

C.3 THE ANALYSIS ALGORITHM

C.3.1 Analysis Rules/Algorithms of ARel

There are two steps for our algorithm to get the estimation of the adaptivity of a program **c** in the ssa form.

1. Estimate the variables that are new generated (via assignment) and store these variables in a global list G. We have the algorithm of the form : $\mathsf{VE}(G; w; \mathbf{c}) \rightarrow (G'; w')$.

2. We start to track the dependence between variables in a matrix M, whose size is $|G| \times |G|$, and track whether arbitrary variable is assigned with a query result in a vector V with size $|G|$. The algorithm to fill in the matrix is of the form: $\mathsf{GG}(\Gamma; \mathbf{c}; i_1) \rightarrow (M; V; i_2)$. Γ is a vector records the variables the current program **c** depends on, the index i_1 is a pointer which refers to the position of the first new-generated variable in **c** in the global list G, and i_2 points to the first new variable that is not in **c** (if exists).

We give an example of M and V of the program c.

$$c = \begin{array}{c} \mathbf{x_1} \leftarrow \mathbf{q}\,(\chi[\mathbf{1}]); \\ \mathbf{x_2} \leftarrow \mathbf{x_1} + \mathbf{1}; \\ \mathbf{x_3} \leftarrow \mathbf{x_2} + \mathbf{2} \end{array} \qquad M = \begin{bmatrix} & (x_1) & (x_2) & (x_3) \\ (x_1) & 0 & 0 & 0 \\ (x_2) & 1 & 0 & 0 \\ (x_3) & 1 & 1 & 0 \end{bmatrix}, V = \begin{bmatrix} (x_1) & 1 \\ (x_2) & 0 \\ (x_3) & 0 \end{bmatrix}$$

Still use the program c as the example, the global list G is now : $[x_1, x_2, x_3]$. The function Left and Right is used to generate the corresponding vector of the left side and right side

of an assignment. Take $x_2 \leftarrow x_1 + 1$ as an example, the result is shown as follows.

$$\mathsf{L}(1) = \begin{bmatrix} 0 & (x_1) \\ 1 & (x_2) \\ 0 & (x_3) \end{bmatrix} \qquad \mathsf{R}(\mathsf{x}_1 + 1, 1) = \begin{bmatrix} 1 & 0 & 0 \\ (x_1) & (x_2) & (x_3) \end{bmatrix}$$

Now let us think about the loop.

$$\begin{aligned} \mathbf{c_3} \triangleq\ & \left[\mathbf{x_1} \leftarrow \mathbf{q}(\chi[\mathbf{1}])\right]^1; \\ & \texttt{loop}\ [2]^2, \mathbf{0}, \\ & [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]\ \texttt{do} \\ & \Big(\left[\mathbf{y_1} \leftarrow \mathbf{q}(\chi[\mathbf{2}] + \mathbf{2})\right]^3; \\ & \left[\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{x_3}\right]^5\Big); \\ & [z_1 \leftarrow x_3 + 2]^6 \end{aligned}$$

$$M = \begin{bmatrix} & (x_1) & (x_3^1) & (y_1^1) & (x_3^1) & (x_3^2) & (y_1^2) & (x_2^2) & (x_3^f) & (z_1) \\ (x_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3^1) & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (y_1^1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2^1) & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3^2) & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ (y_1^2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2^2) & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ (x_3^f) & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ (z_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, V = \begin{bmatrix} (x_1) & 1 \\ (x_3^1) & 0 \\ (y_1^1) & 1 \\ (x_2^1) & 0 \\ (x_3^2) & 0 \\ (y_1^2) & 1 \\ (x_2^2) & 0 \\ (x_3^f) & 0 \\ (z_1) & 0 \end{bmatrix}$$

$$
\begin{array}{rl}
 & \left[\mathbf{x_1} \leftarrow \mathbf{q}(\chi[\mathbf{1}])\right]^1; \\
 & \texttt{loop}\ [0]^2, 0, \\
\mathbf{c_3'} \triangleq & [\mathbf{x_3}, \mathbf{x_1}, \mathbf{x_2}]\ \texttt{do} \\
 & \Big(\left[\mathbf{y_1} \leftarrow \mathbf{q}(\chi[\mathbf{2}])\right]^3; \\
 & \left[\mathbf{x_2} \leftarrow \mathbf{y_1} + \mathbf{x_3}\right]^5\Big); \\
 & [z_1 \leftarrow x_3 + 2]^6
\end{array}
\qquad
M = \begin{bmatrix}
 & (x_1) & (x_3^f) & (z_1) \\
(x_1) & 0 & 0 & 0 \\
(x_3^f) & 1 & 0 & 0 \\
(z_1^2) & 0 & 1 & 0
\end{bmatrix}, V = \begin{bmatrix}
(x_1) & 1 \\
(x_3^f) & 0 \\
(z_1) & 0
\end{bmatrix}
$$

We can now look at the if statement.

$$
\begin{array}{rl}
 & \left[x \leftarrow q(\chi[1])\right]^1; \\
 & \left[y \leftarrow q(\chi[2])\right]^2; \\
 & \text{if}\ (x + y == 5)^3 \\
c_4 \triangleq & \text{then}\ \left[x \leftarrow q(\chi[1] + 3)\right]^4 \\
 & \text{else}(\ \left[x \leftarrow q(\chi[4])\right]^5; \\
 & y \leftarrow 2); \\
 & \left[z \leftarrow x + y\right]^6;
\end{array}
\quad \hookrightarrow \quad
\begin{array}{rl}
 & \left[\mathbf{x_1} \leftarrow \mathbf{q}(\chi[\mathbf{1}])\right]^1; \\
 & \left[\mathbf{y_1} \leftarrow \mathbf{q}(\chi[\mathbf{2}])\right]^2; \\
 & \text{if}\ (\mathbf{x_1} + \mathbf{y_1} == \mathbf{5})^3, [x_4, x_2, x_3], [], [y_3, y_1, y_2] \\
\mathbf{c_4} \triangleq & \text{then}\ \left[\mathbf{x_2} \leftarrow \mathbf{q}(\chi[\mathbf{1}] + \mathbf{3})\right]^4 \\
 & \text{else}(\ \left[\mathbf{x_3} \leftarrow \mathbf{q}(\chi[\mathbf{4}])\right]^5; \\
 & \mathbf{y_2} \leftarrow \mathbf{2}); \\
 & \left[\mathbf{z_1} \leftarrow \mathbf{x_4} + \mathbf{y_3}\right]^6;
\end{array}
$$

$$M_{c4} = \begin{bmatrix} & (x_1) & (y_1) & (x_2) & (x_3) & (y_2) & (x_4) & (y_3) & (z_1) \\ (x_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (y_1) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_3) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (y_2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ (x_4) & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ (y_3) & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ (z_1) & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, V_{c4} = \begin{bmatrix} (x_1) & 1 \\ (y_1) & 1 \\ (x_2) & 1 \\ (x_3) & 1 \\ (y_2) & 0 \\ (x_4) & 0 \\ (y_3) & 0 \\ (z_1) & 0 \end{bmatrix}$$

C.3.2 The Matrix-Vector Algorithm

C.3.2.1 Variable Estimation

We first generate a list of variables G that will be assigned with values (via the command $x \leftarrow e$ or $x \leftarrow q(e_q)$).

$$\frac{}{\mathsf{VE}(G; w; [\mathbf{x} \leftarrow \mathbf{e}]^{\mathbf{l}}) \rightarrow (G + + [\mathbf{x}^{(l,w)}]; w)}\ \textbf{ag-asgn}$$

$$\frac{}{\mathsf{VE}(G; w; [\mathbf{x} \leftarrow q(\mathbf{e_q})]^{l}) \rightarrow (G + + [\mathbf{x}^{(l,w)}]; w)}\ \textbf{ag-query}$$

$$\frac{\begin{array}{c}\mathsf{VE}(G; w; \mathbf{c_1}) \rightarrow (G_1; w_1) \qquad \mathsf{VE}(G_1; w; \mathbf{c_2}) \rightarrow (G_2; w_2) \\ G_3 = G_2 + + [\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w})}] + + [\bar{\mathbf{y}}^{(\mathbf{l},\mathbf{w})}] + + [\bar{\mathbf{z}}^{(\mathbf{l},\mathbf{w})}]\end{array}}{\mathsf{VE}(G; w; [\text{if } (\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^{l}) \rightarrow (G_3; w)}\ \textbf{ag-if}$$

$$\frac{\mathsf{VE}(G; w; \mathbf{c_1}) \rightarrow (G_1; w_1) \qquad \mathsf{VE}(G_1; w_1; \mathbf{c_2}) \rightarrow (G_2; w_2)}{\mathsf{VE}(G; w; (\mathbf{c_1}; \mathbf{c_2})) \rightarrow (G_2; w_2)}\ \textbf{ag-seq}$$

$$\frac{\begin{array}{c}G_0 = G \quad w_0 = w \qquad \forall 0 \le z < N. \mathsf{VE}(G_z + + [\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w_z}+\mathbf{l})}]; (w_z + l); \mathbf{c}) \rightarrow (G_{z+1}; w_{z+1}) \\ G_f = G_N + + [\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w_N}\setminus\mathbf{l})}] \qquad \mathbf{a} = N\end{array}}{\mathsf{VE}(G; w; [\texttt{loop } \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^{l}) \rightarrow (G_f; w_N \setminus l)}\ \textbf{ag-loop}$$

Figure C.9: The variable estimation algorithm

C.3.2.2 Graph Generation Rules

Γ is a matrix of one row and N columns, $N = |G| = |V|$.

$\mathsf{L}(\mathsf{i})$ generates a matrix of one column, N rows, where the $i-th$ row is 1, all the other rows are 0.

$\mathsf{R}(\mathsf{e},\mathsf{i})$ generates a matrix of one row and N columns, where the locations of variables in e is marked as 1. To handle loop, for instance, the variable y appears many times in G, the argument i helps to find the location of the current living variable y in the expression e, which is the latest y with the largest location $i_y < i$ in our global variable list G.

$$\forall 0 \leq z < |\bar{x}|.\bar{x}(z) = x_z, \bar{x_1}(z) = x_{1z}, \bar{x_2}(z) = x_{2z}$$

$$\Gamma \vdash^{i,i+|\bar{\mathbf{x}}|}_{M,v_\emptyset} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \triangleq \forall 0 \leq z < |\bar{x}|.\Gamma \vdash^{i+z,i+z+1}_{M_{x_z},V_\emptyset} x_z \leftarrow x_{1z} + x_{2z} \text{ where } M = \sum_{z\in[|\bar{x}|]} M_{x_z}$$

$$\begin{aligned} M_1; M_2 &:= M_2 \cdot M_1 + M_1 + M_2 \\ V_1 \uplus V_2 &:= \begin{cases} 1 & (V_1[i] = 1 \vee V_2[i] = 1) \wedge i = 1, \cdots, N \wedge |V_1| = |V_2| \\ 0 & o.w. \end{cases} \end{aligned}$$

Definition 20 (Valid matrix)

For a global list G, $G \vDash (M, V)$ iff the cardinality of G equals to the one of V, $|G| = |V|$ and the matrix M is of size $|V| \times |V|$.

Definition 21 (Valid index)

For a global list G, a loop maps w, $G; w \vDash (\mathbf{c}, i_1, i_2)$ iff $G' = G[0, \ldots, i_1 - 1], G'; w; \mathbf{c} \rightarrow G''; w' \wedge G'' = G[0, \ldots, i_2 - 1]$.

$$\boxed{\Gamma \vdash^{i_1,i_2}_{M,V} \; c = \mathsf{GG}(\Gamma; c; i_1) \rightarrow (M, V, i_2)}$$

$$\frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e}, i) + \Gamma)}{\mathsf{GG}(\Gamma; [\mathbf{x} \leftarrow \mathbf{e}]^l; i) \rightarrow (M; V_\emptyset; i+1)}\ \textbf{ad-asgn}$$

$$\frac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e_q}, i) + \Gamma) \qquad V = \mathsf{L}(i)}{\mathsf{GG}(\Gamma; [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l; i) \rightarrow (M; V; i+1)}\ \textbf{ad-query}$$

$$\frac{\begin{array}{c}\mathsf{GG}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_1}; i_1) \rightarrow (M_1; V_1; i_2) \qquad \mathsf{GG}(\Gamma + \mathsf{R}(\mathbf{b}, i_1); \mathbf{c_2}; i_2) \rightarrow (M_2; V_2; i_3) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i_3) \rightarrow (M_x; V_\emptyset; i_3 + |\bar{\mathbf{x}}|) \qquad \mathsf{GG}(\Gamma; [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}]; i_3 + |\bar{\mathbf{x}}|) \rightarrow (M_y; V_\emptyset; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}]; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}|) \rightarrow (M_y; V_\emptyset; i_3 + |\bar{\mathbf{x}}| + |\bar{\mathbf{y}}| + |\bar{\mathbf{z}}|) \\ M = (M_1 + M_2) + M_x + M_y + M_z\end{array}}{\mathsf{GG}(\Gamma; \text{if } ([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}); i_1) \rightarrow (M; V_1 \uplus V_2; i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)}\ \textbf{ad-if}$$

$$\frac{\mathsf{GG}(\Gamma; \mathbf{c_1}; i_1) \rightarrow (M_1; V_1; i_2) \qquad \mathsf{GG}(\Gamma; \mathbf{c_2}; i_2) \rightarrow (M_2; V_2; i_3)}{\mathsf{GG}(\Gamma; (\mathbf{c_1}; \mathbf{c_2}); i_1) \rightarrow ((M_1; M_2); V_1 \uplus V_2; i_3)}\ \textbf{ad-seq}$$

$$\frac{\begin{array}{c}B = |\bar{\mathbf{x}}| \qquad A = |\mathbf{c}| \\ \forall 0 \le j < N. \mathsf{GG}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + j * (B + A)) \rightarrow (M_{1j}; V_{1j}; i + B + j * (B + A)) \\ \mathsf{GG}(\Gamma; \mathbf{c}; i + B + j * (B + A)) \rightarrow (M_{2j}; V_{2j}; i + B + A + j * (B + A)) \\ \mathsf{GG}(\Gamma; [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]; i + N * (B + A)) \rightarrow (M; V; i + N * (B + A) + B) \\ \mathbf{a} = N \qquad M' = M + \sum_{0 \le j < N} (M_{1j} + M_{2j}) \qquad V' = V \uplus \sum_{0 \le j < N} (V_{1j} \uplus V_{2j})\end{array}}{\mathsf{GG}(\Gamma; \texttt{loop}\ [\mathbf{a}]^l,\ 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}, i) \rightarrow (M'; V'; i + N * (B + A) + B)}\ \textbf{ad-loop}$$

Figure C.10: The graph generation algorithm

$$\begin{array}{lcl}
|[\texttt{loop}\ \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l|_{low} & = & [\texttt{loop}\ |\mathbf{a}|_{low},\ \texttt{do}\ |\mathbf{c}|_{low}]^l \\
|\mathbf{c_1}; \mathbf{c_2}|_{low} & = & |\mathbf{c_1}|_{low}; |\mathbf{c_2}|_{low} \\
|[\text{if } (\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})]^l|_{low} & = & [\text{if } (|\mathbf{b}|_{low}, |\mathbf{c_1}|_{low}, |\mathbf{c_2}|_{low})]^l \\
|[\mathbf{x} \leftarrow \mathbf{e}]^l|_{low} & = & [|\mathbf{x}|_{low} \leftarrow |\mathbf{e}|_{low}]^l \\
|[\mathbf{x} \leftarrow q(\mathbf{e_q})]^l|_{low} & = & [|\mathbf{x}|_{low} \leftarrow q(|\mathbf{e_q}|_{low})]^l \\
|x_i|_{low} & = & x \\
|n|_{low} & = & n \\
|\mathbf{a_1} \oplus_a \mathbf{a_2}|_{low} & = & |\mathbf{a_1}|_{low} \oplus_a |\mathbf{a_2}|_{low} \\
|\mathbf{b_1} \oplus_b \mathbf{b_2}|_{low} & = & |\mathbf{b_1}|_{low} \oplus_b |\mathbf{b_2}|_{low}
\end{array}$$

Figure C.11: The erasure of SSA

Definition 22 (Valid gamma)

$\Gamma \vDash i_1$ iff $\forall i \geq i_1, \Gamma(i_1) = 0$.

Definition 23 (Adaptivity of piece of program)

Given a program **c**, the global list G, s.t. $\Gamma \vdash^{i_1,i_2}_{M,V} \mathbf{c}$,

the Graph $G_{ssa}(M, V, G, i_1, i_2) = (Nodes, Edges, Weights)$ is. defined as:

Nodes $Vt = \{G(j) \in \mathscr{LV} \mid i_1 \leq j < i_2\}$

Edges $E = \{(G(j_1), G(j_2)) \in \mathscr{LV} \times \mathscr{LV} \mid M[j_1][j_2] \geq 1 \wedge i_1 \leq j_1, j_2 < i_2\}$

Weights $Wt = \{(G(j), 1) \in \mathscr{LV} \times \mathscr{N} | i_1 \leq j < i_2 \wedge V[j] = 1\} \cup \{(G(j), 0) \in \mathscr{LV} \times \mathscr{N} | i_1 \leq j < i_2 \wedge V[j] = 0\}$.

Adaptivity of the program defined according to the logic is as:

$$Adapt(M, V, i_1, i_2) := \max_{vt_1, vt_2 \in Vt} \mathsf{Weight}(p(vt_1, vt_2), Wt),$$

where $p(k, l)$ is the path in graph $G_{ssa}(M, V, i_1, i_2)$ starting from k to l,

$\mathsf{Weight}(p(vt_1, vt_2), Wt)$ get the total sum of weights along the path $p(vt_1, vt_2)$.

C.3.3 Soundness of ADAPT

Definition 24 (Subgraph)

Given two graphs $G_s = (V_1, E_1)$, $G_{ssa} = (V_2, E_2, Wt)$, $G_s \subseteq G_{ssa}$ iff:

$\exists f, g$ so that

1. for every $v \in V_1$, $f(v) \in V_2 \wedge f$ is injective, and $wt(f(v)) = 1$.

2. $\forall e = (v_i, v_j) \in E_1$, there exists a path $g(e)$ from $f(v_i)$ to $f(v_2)$ in G_{ssa}.

Definition 25 (Mapping of vertices from G_s to G_{ssa} graph)

Let us define a map $f : \mathscr{A}\mathscr{Q} -> \mathscr{LV}$ as follow: $f(q(v_q)^{l,w}) = \mathbf{x}^{l,w}$

Lemma 88. *$\forall t, m, w, c, \langle m, c, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle$, every node in $t' - t$ has unique label.*

Proof. By induction on the evaluation of program c.

Case $\langle m, [\texttt{loop}\ v_N\ \texttt{do}\ c]^l, t, w\rangle \rightarrow^* \langle m_N, \texttt{skip}, t_N, w_N\rangle$

We want to show every node in $t_N - t$ has unique label. Suppose $v_N = N$,so we have N iteration. When $N = 0$, there is no new generated trace, it is trivially true. When $N > 0$, we have the following evaluation.

$$\frac{v_N > 0}{\langle m, [\texttt{loop}\ v_N\ \texttt{do}\ c]^l, t, w\rangle \rightarrow \langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w + l)\rangle}\ \textbf{l-loop}$$

$$\frac{\langle m, c, t, (w + l)\rangle \rightarrow^* \langle m_1, \texttt{skip}, t_1, w_1\rangle}{\langle m, c; [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t, (w + l)\rangle \rightarrow^* \langle m_1, [\texttt{loop}\ (v_N - 1)\ \texttt{do}\ c]^l, t_1, w_1\rangle}$$

By induction hypothesis on $\langle m, c, t, (w+l)\rangle \rightarrow^* \langle m_1, \texttt{skip}, t_1, w_1\rangle$, we know that for every node in $t_1 - t$ has unique label.

Unfold the loop, for the $2nd$, $3rd$, until the $N - th$ iteration, we can also conclude that every node in $t_2 - t_1$, $t_3 - t_2$, until $t_N - t_{N-1}$ has unique label.

Because $w \neq w+l \neq w_1+l \neq w_2+l \neq \ldots, \neq w_{N-1}+l$, we know that node from different iteration $(i \neq j)$ $t_i - t_{i-1}$ and $t_j - t_{j-1}$ has different loop maps.

Case $\langle m, c_1; c_2, t, w\rangle \rightarrow^* \langle m'', \texttt{skip}, t'', w''\rangle$

$$\frac{\langle m, c_1, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle \qquad \langle m', c_2, t', w'\rangle \rightarrow^* \langle m'', \texttt{skip}, t'', w''\rangle}{\langle m, c_1; c_2, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w'\rangle}$$

By induction on premises, we know that : every node in $t' - t$ and $t'' - t'$ has unique label. Also, all the program lines $[l_1, l_2, \ldots, l_n]$ in c_1 is smaller than the minimal program line in c_2, so every node in $t' - t$ has different line number with node in $t'' - t'$. □

Lemma 89 (Label consistency in the dependency graph over queries in the loop lan-

guage). *Given a program c, for a database D, a memory m, a loop maps w, every node $q(v_q)^{(l,w)}$ in the graph $(V_1, E_1) = G(c, D, m, w)$ has unique label (l, w).*

Proof. From the definition of $G(c, D, m, w)$, the vertices comes from the trace generated in the low level evaluation $\langle m, c, [], w\rangle \rightarrow^* \langle m', \texttt{skip}, t, w'\rangle$. This can be proved using Lemma 2. □

Lemma 90. *Given $\Gamma \vdash^{(i_1,i_2)}_{M,V} \mathbf{c}$, for any global list G and a loop maps w, such that $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V)$. Assume $G' = G[0, \ldots, i_1 - 1]$, so that $G'; w; \mathbf{c} \hookrightarrow G''; w'$. For any database D and memory m and trace t, $\langle m, |\mathbf{c}|_{low}, t, w\rangle \rightarrow^* \langle m', \texttt{skip}, t', w''\rangle$. Then $w' = w''$.*

C.3.3.1 Proof of Lemma 18

Lemma 91 ($f_{D,m}$ is well defined)**.** *Given $\Gamma \vdash^{(i_1,i_2)}_{M,V} \mathbf{c}$, for any global list G and a loop maps w, such that $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V)$ and $\Gamma \vdash i_1$. For any database D and memory $\mathbf{m}$, the function $f_{D,m} : \mathscr{AQ} \rightarrow \mathscr{LV}$ maps all the nodes in query-based graph $(V_1, E_1) = G_s(\mathbf{c}, D, \mathbf{m}, w)$ to the unique node in the variable-based graph $(V_2, E_2, Wt) = G_{ssa}(M, V, G, i_1, i_2)$, defined as :*

$f_{D,m}(q^{l,w}) = \mathbf{x}^{(l,w)}$ *where* $q^{(l,w)} \in V_1$ *and* $\mathbf{x}^{(l,w)} \in V_2$ *is injective*

and $\forall q(v_q)^{(l,w)} \in V_1, wt(f(q(v_q)^{(l,w)})) = 1$.

Proof. To show $f_{D,m}$ is injective.

We want to show that:

for every node $q^{(l,w)}$ in V_1(in the new generated trace by executing c), there exist only one corresponding labelled variable $\mathbf{x}^{(l,w)} \in \mathscr{LV}$ in V_2(also know as $G[i_1, \ldots, (i_2 - 1)]$) with the same label (l, w).

By induction on $\Gamma \vdash^{(i_1,i_2)}_{M,V} \mathbf{c}$.

$$\textbf{Case}\ \frac{M = (\mathsf{L}(i) + R(e_q, i)) * \Gamma \qquad V = \mathsf{L}(i)}{\Gamma \vdash^{(i,i+1)}_{M,V} \mathbf{x} \leftarrow q(e_q)}\ \textbf{query}$$

We assume $G; w \vDash ([\mathbf{x} \leftarrow q(e_q)]^l, i, i+1)$ and $G \vDash (M, V)$. We have the loop program $|[\mathbf{x} \leftarrow q(e_q)]^l|_{low} = [x \leftarrow q(e_q)]^l$. We choose the database D and memory m, a starting trace t, from the definition of $(V_1, E_1) = G([x \leftarrow q(e_q)]^l, D, m, w)$, we know v_1 comes from the trace $t' - t$ generated in the following ssa evaluation.

$$\frac{q(v_q)(D) = v}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{q}(\mathbf{v_q})]^{\mathbf{l}} \rangle, t, w \rightarrow^* \langle \mathbf{m}[\mathbf{v}/\mathbf{x}], \texttt{skip}, t + +[q(v_q)^{(l,w)}], w \rangle}$$

We unfold $G; w \vDash ([\mathbf{x} \leftarrow q(e_q)]^l, i, i+1)$ so we know that

$G' = G[0, \ldots, (i-1)] \wedge G'; w; [\mathbf{x} \leftarrow q(e_q)]^l \rightarrow G' + +[\mathbf{x}^{(l,w)}]; w$

Considering every node in V_2 has the unique label. we know that there is the unique labelled variable $\mathbf{x}^{(l,w)}$ in V_2 with the same label as the one $q^{(l,w)}$ in V_1. From the analysis rules, we also know that $Wt(\mathbf{x}^{(l,w)}) = 1$ because $[\mathbf{x} \leftarrow q(e_q)]^l$ associates $\mathbf{x}$ with the query result, and $G(i) = \mathbf{x}^{(l,w)}\ V(i) = 1$.

$$\textbf{Case}\ \frac{\Gamma \vdash^{(i_1,i_2)}_{M_1,V_1} \mathbf{c_1} \qquad \Gamma \vdash^{(i_2,i_3)}_{M_2,V_2} \mathbf{c_2}}{\Gamma \vdash^{(i_1,i_3)}_{M_1;M_2,V_1 \uplus V_2} \mathbf{c_1}; \mathbf{c_2}}\ \textbf{seq}$$

We assume $G; w \vDash (\mathbf{c_1}; \mathbf{c_2}, i_1, i_3)$ and denote $c_1 = |\mathbf{c_1}|_{low}$ and $c_2 = |\mathbf{c_2}|_{low}$. We choose the memory m, trace t and hidden database D, we have the ssa evaluation:

$$\frac{\langle m, c_1, t, w \rangle \rightarrow^* \langle m_1, \texttt{skip}, t_1, w_1 \rangle \qquad \langle m_1, c_2, t_1, w_1 \rangle \rightarrow^* \langle m_2, \texttt{skip}, t_2, w_2 \rangle}{\langle m, c_1; c_2, t, w \rangle \rightarrow \langle m_2, \texttt{skip}, t_2, w_2 \rangle}$$

From our assumption $G; w \vDash (\mathbf{c_1}; \mathbf{c_2}, i_1, i_3)$, we denote $G' = G[0, \ldots, (i_1 - 1)]$. We have the

following :

$$\frac{G'; w; \mathbf{c_1} \to G_1; w_1' \qquad G_1; w_1'; \mathbf{c_2} \to G_2; w_2'}{G'; w; \mathbf{c_1}; \mathbf{c_2} \to G_2; w_2'}$$

We know that $\mathbf{c_1}, \mathbf{c_2}$ are the subterm of $\mathbf{c_1}; \mathbf{c_2}$ so there exist i_2 such that $G; w \vDash (\mathbf{c_1}, i_1, i_2)$ and $G; w_1' \vDash (\mathbf{c_2}, i_2, i_3)$. By Lemma 90, we conclude that $w_1 = w_1'$.

By induction hypothesis on the first premise, we know : every node $q^{(l,w)} \in \mathscr{AQ}$ in $t_1 - t$ has a unique mapping $\mathbf{x}^{(l,w)}$ with the same label in $G[i_1, \ldots, (i_2 - 1)]$.

By IH on the second premise, we conclude that : every node $q^{(l,w)} \in \mathscr{AQ}$ in $t_2 - t_1$ has a unique mapping $\mathbf{x}^{(l,w)}$ with the same label in $G[i_2, \ldots, (i_3 - 1)]$.

Now we can conclude that : every node $q^{(l,w)} \in \mathscr{AQ}$ in $t_2 - t$ has a unique mapping $\mathbf{x}^{(l,w)}$ with the same label in $G[i_1, \ldots, (i_3 - 1)]$.

For all the query assignments $[\mathbf{x} \leftarrow q(e_q)]^l$ in $\mathbf{c_1}$, we know $\exists i_1 \le i < i_2, G(i) = \mathbf{x}^{(l,w)} \wedge V(i) = 1$. Similarly, for all the query assignments $[\mathbf{x} \leftarrow q(e_q)]^l$ in $\mathbf{c_2}$, we know $\exists i_2 \le i \le i_3, G(i) = \mathbf{x}^{(l,w)} \wedge V_2(i) = 1$. So we know that for all the query assignments $[\mathbf{x} \leftarrow q(e_q)]^l$ in $\mathbf{c_1}; \mathbf{c_2}$, $\exists i_1 \le i \le i_3, G(i) = \mathbf{x}^{(l,w)} \wedge V_1 \uplus V_2(i) = 1$. From the definition, we know $wt(\mathbf{x}^{(l,w)}) = 1$ for all the query assignments in $\mathbf{c_1}; \mathbf{c_2}$ and $f_{D,m}(q(v_q)^{(l,w)}) = \mathbf{x}^{(l,w)}$

$$\textbf{Case}\ \frac{\begin{array}{c} B = |\bar{\mathbf{x}}| \qquad \Gamma \vdash^{(i,i+B)}_{M_{10},V_{10}} \qquad \Gamma \vdash^{(i+B,i+B+A)}_{M_{20},V_{20}} \mathbf{c} \\ \forall 1 \le j < N. \Gamma \vdash^{(i+j*(B+A),i+B+j*(B+A))}_{M_{1j},V_{1j}} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \\ \Gamma \vdash^{(i+B+j*(B+A),i+B+A+j*(B+A))}_{M_{2j},V_{2j}} \mathbf{c} \\ \Gamma \vdash^{(i+N*(B+A),i+N*(B+A)+B)}_{M,V} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \qquad \mathbf{a} = \lceil N \rceil \\ M' = M + \sum_{0 \le j < N} M_{1j} + M_{2j} \qquad V' = V \uplus \sum_{0 \le j < N} V_{1j} \uplus V_{2j} \end{array}}{\Gamma \vdash^{(i,i+N*(B+A)+B)}_{M',V'} [\texttt{loop}\ \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l}\ \textbf{loop}$$

We assume $G; w \vDash ([\texttt{loop}\ \mathbf{a}, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, i, i + N * (B + A) + B)$ and denote $c = |\mathbf{c}|_{low}$ and $a = |\mathbf{a}|_{low}$. We choose the memory m, trace t and hidden database D, we have

the ssa evaluation supposing $\langle m, a\rangle \rightarrow v_N$:

$$\frac{v_N > 0}{\langle m, [\texttt{loop } v_N \texttt{ do } c]^l, t, w\rangle \rightarrow \langle m, c; [\texttt{loop } (v_N - 1) \texttt{ do } c]^l, t, (w + l)\rangle}\ \textbf{l-loop}$$

$$\frac{\langle m, c, t, (w + l)\rangle \rightarrow^* \langle m_1, \texttt{skip}, t_1, w_1\rangle}{\langle m, c; [\texttt{loop } (v_N - 1) \texttt{ do } c]^l, t, (w + l)\rangle \rightarrow^* \langle m_1, [\texttt{loop } (v_N - 1) \texttt{ do } c]^l, t_1, w_1\rangle}$$

We assume $G' = G[0, \ldots, i-1]$, from our assumption, we have:

$$\frac{\begin{array}{c} G_0 = G' \quad w_0 = w \qquad \forall 0 \le z < N. G_z + +[\tilde{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_z} + \mathbf{l})}]; (w_z + l); \mathbf{c} \rightarrow G_{z+1}; w_{z+1} \\ G_f = G_N + +[\tilde{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_N} \setminus \mathbf{l})}] \qquad \mathbf{e} = \lceil \mathbf{N} \rceil \end{array}}{G'; w; [\texttt{loop } (e), n, [\tilde{\mathbf{x}}, \tilde{\mathbf{x}_1}, \tilde{\mathbf{x}_2}] \texttt{ do } \mathbf{c}]^l \rightarrow G_f; w_N \setminus l}\ \textbf{loop}$$

By induction hypothesis on $\Gamma \vdash^{(i+B, i+B+A)}_{M_{20}, V_{20}} \mathbf{c}$ using the conclusion $\langle m, c, t, (w + l)\rangle \rightarrow^* \langle m_1, \texttt{skip}, t_1, w_1\rangle$ and $G_z + +[\tilde{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_z} + \mathbf{l})}]; (w_z + l); \mathbf{c} \rightarrow G_{z+1}; w_{z+1}$ where $z = 0$. We can conclude the following about the first iteration.

Every node $q^{(l,w)}$ in $t_1 - t$, there is a unique labelled variable $\mathbf{x}^{(l,w)}$ with the same label in. $G[i + B, \ldots, i + B + A - 1]$.

For other iteration $1 \le j < N$, we can show similarly by induction hypothesis on $\Gamma \vdash^{(i+B+j*(B+A), i+B+A+j*(B+A))}_{M_{2j}, V_{2j}} \mathbf{c}$. □

C.3.3.2 Proof of Theorem 19

Theorem 92. *[Soundness of ADAPT] Given* $\Gamma \vdash^{(i_1, i_2)}_{M,V} \mathbf{c}$*, for any global list G, loop maps w such that* $G; w \vDash (\mathbf{c}, i_1, i_2) \wedge G \vDash (M, V), \Gamma \vdash i_1$*. K is the number of queries inquired during the execution of the piece of program* $\mathbf{c}$ *and* $|V|$ *gives the number of non-zeros in V. Then,*

$$K \le |V| \wedge \forall D, \mathbf{m}. G_s(\mathbf{c}, D, \mathbf{m}, w) \subseteq G_{ssa}(M, V, G, i_1, i_2)$$

Proof. By induction on the judgment $\Gamma \vdash^{i_1,i_2}_{M,V} \mathbf{c}$.

- **Case** $\dfrac{M = \mathsf{L}(i) * (\mathsf{R}(\mathbf{e}, i) + \Gamma)}{\Gamma \vdash^{(i,i+1)}_{M,V_\emptyset} \mathbf{x} \leftarrow \mathbf{e}}$ **asgn**

  Given a memory m, database D, trace t, loop maps w, then from the following operational semantics:

  $$\frac{}{\langle \mathbf{m}, [\mathbf{x} \leftarrow \mathbf{e}]^l, t, w\rangle \rightarrow^* \langle \mathbf{m}, [\mathbf{x} \leftarrow v]^l, t, w\rangle \rightarrow \langle \mathbf{m}[\mathbf{x} \mapsto v], [\texttt{skip}]^l, t, w\rangle}$$

  We need to show:

  1. $G_s(\mathbf{x} \leftarrow \mathbf{e}, D, m, w) \subseteq G_{ssa}(M, V_\emptyset, i, i+1)$.
  2. $K \leq |V_\emptyset|$.

  The first goal is shown because the new generated trace t' is empty, based on the definition of G_s, we know that G_s has no vertices.

  The second goal is proved because $K = 0$.

- **Case** $\dfrac{M = \mathsf{L}(i) * (\Gamma + \mathsf{R}(\mathbf{e_q}, i)) \qquad V = \mathsf{L}(i)}{\Gamma \vdash^{(i,i+1)}_{M,V} \mathbf{x} \leftarrow q(\mathbf{e_q})}$ **query**

  Given a memory m, database D, trace t, loop maps w, then from the following operational semantics:

  $$\frac{q(v_q)(D) = v}{\langle \mathbf{m}, [\mathbf{x} \leftarrow q(\mathbf{e_q})]^l, t, w\rangle \rightarrow \langle \mathbf{m}[v/\mathbf{x}], \texttt{skip}, t +\!\!+ [(q(v_q)^{(l,w)})], w\rangle}$$

  We need to show:

  1. $G_s([\mathbf{x} \leftarrow q(\mathbf{e_q})]^l, D, m, w) \subseteq G_{ssa}(M, V, i, i+1)$.

2. $K \leq |V|$.

The first goal is shown as follows: because the new generated trace $t' = [q(v)^{(l,w)}]$ only has one element. By the definition of $G_s = (V_s, E_s)$, we know there is only one vertex vx in V and no edge in E, i.e., $V_s = \{q(v)^{(l,w)}\}$ and $E_s = \{\}$.

From Lemma 91,we know that there exists a function f so that $f(vx)$ (equivalent to $q(v)^{(l,w)}$ in G_s) exists in the vertices V_{ssa} of $G_{ssa}(M, v, i, i+1) = (V_{ssa}, E_{ssa})$. Since $V_s \subseteq V_{ssa}$ and $E_s \subseteq E_{ssa}$, it is proved that $G_s([\mathbf{x} \leftarrow q(\mathbf{e_q})]^l, D, m, w) \subseteq G_{ssa}(M, V, i, i+1)$. So we have the first goal proved.

The second goal is proved because $V = \mathsf{L}(i)$ is not empty so that $|V| \geq 1$, and $K = 1$.

- **Case** $\dfrac{\begin{array}{c}\Gamma + \mathsf{R}(b, i_1) \vdash^{(i_1,i_2)}_{M_1,V_1} \mathbf{c_1} \qquad \Gamma + \mathsf{R}(b, i_1) \vdash^{(i_2,i_3)}_{M_2,V_2} \mathbf{c_2} \\ \forall 0 \leq j < |\bar{x}|.\bar{x}(j) = x_j, \bar{x_1}(j) = x_{1j}, \bar{x_2}(j) = x_{2j} \\ \forall 0 \leq j < |\bar{x}|.\Gamma \vdash^{i_3+j, i_3+j+1}_{M_{x_j},V_\emptyset} x_j \leftarrow x_{1j} + x_{2j} \\ \forall 0 \leq j < |\bar{y}|.\Gamma \vdash^{i_3+|\bar{x}|+j, i_3+|\bar{x}|+j+1}_{M_{y_j},V_\emptyset} y_j \leftarrow y_{1j} + y_{2j} \\ \forall 0 \leq j < |\bar{z}|.\Gamma \vdash^{i_3+|\bar{x}|+|\bar{y}|+j, i_3+|\bar{x}|+|\bar{y}|+j+1}_{M_{z_j},V_\emptyset} z_j \leftarrow z_{1j} + z_{2j} \\ M = (M_1 + M_2) + \sum_{j\in[|\bar{x}|]} M_{x_j} + \sum_{j\in[|\bar{y}|]} M_{y_j} + \sum_{j\in[|\bar{z}|]} M_{z_j}\end{array}}{\Gamma \vdash^{(i_1, i_3+|\bar{x}|+|\bar{y}|+|\bar{z}|)}_{M, V_1 \uplus V_2} \mathbf{if}\ ([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2})}\ \mathbf{if}$

Given a memory $\mathbf{m}$, database D, trace t, loop maps w and a global list G. We assume that $G \vDash M, V_1 \uplus V_2, G; w \vDash (\mathbf{if}\ (\mathbf{b}, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_1}, \mathbf{c_2}), i_1, i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)$. There are two possible executions. We choose one branch and the other will be similar.

From the assumption, denote $G' = G[0,\ldots,(i_1-1)]$ we know that

$$\frac{G';w;\mathbf{c_1}\rightarrow G_1;w_1 \qquad G_1;w;\mathbf{c_2}\rightarrow G_2;w_2 \qquad G_3 = G_2++[\bar{\mathbf{x}}^{(\mathbf{l},\mathbf{w})}]++[\bar{\mathbf{y}}^{(\mathbf{l},\mathbf{w})}]++[\bar{\mathbf{z}}^{(\mathbf{l},\mathbf{w})}]}{G';w;\text{if }([\mathbf{b}]^l,[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}],[\bar{\mathbf{y}},\bar{\mathbf{y_1}},\bar{\mathbf{y_2}}],[\bar{\mathbf{z}},\bar{\mathbf{z_1}},\bar{\mathbf{z_2}}],\mathbf{c_1},\mathbf{c_2})\rightarrow G_3;w}\ \textbf{if}$$

$\mathbf{c_1}$ is the sub term of if $(\mathbf{b},[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}],[\bar{\mathbf{y}},\bar{\mathbf{y_1}},\bar{\mathbf{y_2}}],[\bar{\mathbf{z}},\bar{\mathbf{z_1}},\bar{\mathbf{z_2}}],\mathbf{c_1},\mathbf{c_2})$. so we have: $G \vDash M_1, V_1$ and $G; w \vDash (\mathbf{c_1}, i_1, i_2)$, and $\Gamma + \mathsf{R}(b, i_1) \vDash i_1$.

By induction hypothesis on the first premise $\Gamma+\mathsf{R}(b,i_1)\vdash^{(i_1,i_2)}_{M,V}\mathbf{c_1}$, we can conclude that :

$$K_{\mathbf{c_1}} \leq |V_1| \wedge G_s(\mathbf{c_1}, D, m, w) \subseteq G_{ssa}(M_1, V_1, i_1, i_2).$$

By the definition of K, we have $K_{\text{if }(\mathbf{b},[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}],[\bar{\mathbf{y}},\bar{\mathbf{y_1}},\bar{\mathbf{y_2}}],[\bar{\mathbf{z}},\bar{\mathbf{z_1}},\bar{\mathbf{z_2}}],\mathbf{c_1},\mathbf{c_2})} = K_{\mathbf{c_1}}$ when $b = \text{true}$.

Next, we want to show : $G_s(\text{if }([\mathbf{b}]^l,[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}],[\bar{\mathbf{y}},\bar{\mathbf{y_1}},\bar{\mathbf{y_2}}],[\bar{\mathbf{z}},\bar{\mathbf{z_1}},\bar{\mathbf{z_2}}],\mathbf{c_1},\mathbf{c_2}),D,\mathbf{m},w)$ $\subseteq G_{ssa}(M, V_1 \uplus V_2, i_1, i_3 + |\bar{x}| + |\bar{y}| + |\bar{z}|)$. By the definition of G_s, we know that $G_s(\text{if }([\mathbf{b}]^l,[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}],[\bar{\mathbf{y}},\bar{\mathbf{y_1}},\bar{\mathbf{y_2}}],[\bar{\mathbf{z}},\bar{\mathbf{z_1}},\bar{\mathbf{z_2}}],\mathbf{c_1},\mathbf{c_2}),D,\mathbf{m},w)$ is the same graph as $G_s(\mathbf{c_1},D,\mathbf{m},w)$. We can also show that $G_{ssa}(M_1,V_1,G,i_1,i_2)$ is a subgraph of $G_{ssa}(M,V_1\uplus V_2,i_1,i_3+|\bar{x}|+|\bar{y}|+|\bar{z}|)$ because all the vertices in the former also appears in the later($[i_1,i_2]$ is a smaller range of $[i_1,i_3+i_3+|\bar{x}|+|\bar{y}|+|\bar{z}|]$) and same for the edges(M contains M_1).

- **Case** $\dfrac{\Gamma\vdash^{(i_1,i_2)}_{M_1,V_1}\mathbf{c_1} \qquad \Gamma\vdash^{(i_2,i_3)}_{M_2,V_2}\mathbf{c_2}}{\Gamma\vdash^{(i_1,i_3)}_{M_1;M_2,V_1\uplus V_2}\mathbf{c_1};\mathbf{c_2}}\ \textbf{seq}$

Given a memory $\mathbf{m}$, database D, trace t, loop maps w ,a global list G. We assume $G; w \vDash (\mathbf{c_1};\mathbf{c_2}, i_1, i_3)$, $G \vDash M_1; M_2, V_1 \uplus V_2$. From the following ssa operational semantics:

$$\frac{\langle \mathbf{m}, \mathbf{c_1}, t, w\rangle \rightarrow^* \langle \mathbf{m_1}, \texttt{skip}, t_1, w_1\rangle \qquad \langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1\rangle \rightarrow^* \langle \mathbf{m_2}, \texttt{skip}, t_2, w_2\rangle}{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w\rangle \rightarrow \langle \mathbf{m_2}, \texttt{skip}, t_2, w_2\rangle}$$

We need to show:

1. $G_s(\mathbf{c_1}; \mathbf{c_2}, D, \mathbf{m}, w) \subseteq G_{ssa}(M_1; M_2, V_1 \uplus V_2, i_1, i_3)$.
2. $K_{\mathbf{c_1};\mathbf{c_2}} \leq |V_1 \uplus V_2|$.

From our assumption, we denote $G' = G[0, \ldots, (i_1 - 1)]$. We have the following :

$$\frac{G'; w; \mathbf{c_1} \rightarrow G_1; w_1' \qquad G_1; w_1'; \mathbf{c_2} \rightarrow G_2; w_2'}{G'; w; \mathbf{c_1}; \mathbf{c_2} \rightarrow G_2; w_2'}$$

Because $\mathbf{c_1}, \mathbf{c_2}$ are subterms of $\mathbf{c_1}; \mathbf{c_2}$, we know that there exists an index $i_1 \leq i_2 \leq i_3$ so that $G, w \vDash (\mathbf{c_1}, i_1, i_2)$ and $G, w_1' \vDash (\mathbf{c_2}, i_2, i_3)$.

By induction hypothesis, we have:

$$K_{\mathbf{c_1}} \leq |V_1| \wedge G_s(\mathbf{c_1}, D, \mathbf{m}, w) \subseteq G_{ssa}(M_1, V_1, G, i_1, i_2)$$

$$K_{\mathbf{c_2}} \leq |V_2| \wedge G_s(\mathbf{c_2}, D, \mathbf{m_1}, w_1') \subseteq G_{ssa}(M_2, V_2, G, i_2, i_3)$$

We can easily show that $K_{\mathbf{c_1};\mathbf{c_2}} \leq |V_1 \uplus V_2|$ by the definition of $K_{\mathbf{c_1};\mathbf{c_2}} = K_{\mathbf{c_1}} + K_{\mathbf{c_2}}$, and $|V_1 \uplus V_2| = |V_1| + |V_2|$ because the non-zeros range in $V_1([i_1, i_2))$ is disjoint with that in $V_2([i_2, i_3))$.

By the definition of G_s, we know that vertices of $(VX_1, E_1) = G_s(\mathbf{c_1}, D, \mathbf{m}, w)$ come from the trace $t_1 - t$ and $(VX_2, E_2) = G_{low}(c_2)$ from $t_2 - t_1$.

Unfold the definition of subgraph $\subseteq$, we need to show two cases.

1. The vertices in $G_s(c_1; c_2)$ can be mapped to the vertices in $G_{ssa}(M_1; M_2, V_1 \uplus V_2, i_1, i_3)$ by $f_{D,m}$.

   By induction hypothesis, we know that there exist functions f_{D,m,c_1} for $G_s(\mathbf{c_1}, D, \mathbf{m}, w)$ and f_{D,m_1,c_2} for $G_s(\mathbf{c_2}, D, \mathbf{m}, w_1')$. Then, we construct the mapping function

   $$f_{D,m,\mathbf{c_1};\mathbf{c_2}}(vx) = \begin{cases} f_{D,m,c_1}(vx) & (vx) \in (t_1 - t) \\ f_{D,m_1,c_2}(vx) & (vx) \in (t_2 - t_1) \end{cases},$$

   which maps vertices in $G_s(c_1; c_2)$ to the vertices in $G_{ssa}(M_1; M_2, V_1 \uplus V_2, i_1, i_3)$.

2. Every edge in $G_s(\mathbf{c_1}; \mathbf{c_2}, D, \mathbf{m}, w)$ can be mapped to a corresponding path in $G_{ssa}(M_1; M_2, G, V_1 \uplus V_2, i_1, i_3)$. The edges in $G_s(\mathbf{c_1}; \mathbf{c_2}, D, \mathbf{m}, w)$ comes from three sources:

   2.1. Edges in $G_s(\mathbf{c_1}, D, \mathbf{m}, w)$. We can find a path in $G_{ssa}(M_1; M_2, V_1 \uplus V_2, i_1, i_3)$ for edge of this kind because we can find one path in a smaller graph $G_{ssa}(M_1 V_1, i_1, i_2)$

   2.2. Edges in $G_s(\mathbf{c_2}, D, \mathbf{m_1}, w_1')$. This can be proved in a similar way as in previous case.

   2.3. Edges (vx_1, vx_2) where $vx_1 = q_1(v_1)^{(l_1, w_1)}$ comes from $G_s(\mathbf{c_1}, D, \mathbf{m}, w)$ and $vx_2 = q_2(v_2)^{(l_2, w_2)}$ comes from $G_s(\mathbf{c_2}, D, \mathbf{m_1}, w_1')$. We unfold the definition of one edge so that we have:

   $$\mathsf{DEP_{ssa}}(vx_1, vx_2, \mathbf{c_1}; \mathbf{c_2}, w, \mathbf{m})$$

   Since we know that $q_1(v_1)^{(l_1, w_1)}$ comes from $t_1 - t$, so we can assume that $\mathbf{c_1} = \mathbf{c_1''}; \mathbf{c_1'}$. The last command of c_1'' is $[\mathbf{x} \leftarrow q_1(\mathbf{e_1})]^{l_1}$. We have the

following SSA evaluation:

$$\frac{\langle \mathbf{m}, \mathbf{c_1}, t, w\rangle \rightarrow^* \langle \mathbf{m_1'}, [\mathbf{x} \leftarrow q_1(v_1)]^{l_1}; \mathbf{c_1'}, t_1', w_1'\rangle \rightarrow \langle \mathbf{m_1}, \texttt{skip}, t_1, w_1\rangle \quad \langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1\rangle \rightarrow^* \langle \mathbf{m_2}, \texttt{skip}, t_2, w_2\rangle}{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w\rangle \rightarrow \langle \mathbf{m_2}, \texttt{skip}, t_2, w_2\rangle}$$

Then unfold the $\mathsf{DEP}_{\mathsf{ssa}}$:

$$\begin{aligned}
&\forall \mathbf{m_1}, \mathbf{m_3}, D, t, t_1, t_3. \langle \mathbf{m}, \mathbf{c}, t, w\rangle \rightarrow^* \langle \mathbf{m_1}, [\mathbf{x} \leftarrow q_1(v_1)]^{l_1}; \mathbf{c_2}, t_1, w_1\rangle \rightarrow \\
&\langle \mathbf{m_1}[v_q/\mathbf{x}], \mathbf{c_2}, t_1 + +[q_1(v_1)^{(l_1,w_1)}], w_1\rangle \rightarrow^* \langle \mathbf{m_3}, \texttt{skip}, t_3, w_3\rangle \\
&\wedge \Big(q_1(v_1)^{(l_1,w_1)} \in (t_3 - t) \wedge q_2(v_2)^{(l_2,w_2)} \in (t_3 - t_1) \implies \\
&\exists v \in \texttt{codom}(q_1(v_1)(D)). \langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 + +[q_1(v_1)^{(l_1,w_1)}], w_1\rangle \\
&\rightarrow^* \langle \mathbf{m_3'}, \texttt{skip}, t_3', w_3'\rangle \wedge (q_2(v_2)^{(l_2,w_2)}) \notin (t_3' - t_1) \Big) \\
&\wedge \Big(q_1(v_1)^{(l_1,w_1)} \in (t_3 - t) \wedge q_2(v_2)^{(l_2,w_2)} \notin (t_3 - t_1) \implies \\
&\exists v \in \texttt{codom}(q_1(v_1)(D)). \langle \mathbf{m_1}[v/\mathbf{x}], \mathbf{c_2}, t_1 + +[q_1(v_1)^{(l_1,w_1)}], w_1\rangle \\
&\rightarrow^* \langle \mathbf{m_3'}, \texttt{skip}, t_3', w_3'\rangle \wedge q_2(v_2)^{(l_2,w_2)} \in (t_3' - t_1) \Big)
\end{aligned}$$

We already know that $q_1(v_1)^{(l_1,w_1)} \in (t_1 - t), q_2(v_2)^{(l_2,w_2)} \in (t_2 - t_1)$. So from DEP, we know that: $\exists v \in \texttt{codom}(q_1(v_1)(D)). \langle \mathbf{m_1'}[v/\mathbf{x}], \mathbf{c_1'}; \mathbf{c_2}, t_1' + +[(q_1(v_1)^{(l_1,w_1)})], w_1'\rangle \rightarrow^* \langle \mathbf{m_2'}, \texttt{skip}, t_2', w_2'\rangle \wedge (q_2(v_2)^{(l_2,w_2)}) \notin (t_2' - t)$.

We have:

$$\frac{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w\rangle \rightarrow^* \langle \mathbf{m_1'}, [\mathbf{x} \leftarrow q_1(v_1)]^{l_1}; \mathbf{c_1'}; \mathbf{c_2}, t_1', w_1'\rangle \rightarrow \langle \mathbf{m_1}, \texttt{skip}, t_1, w_1\rangle \quad \langle \mathbf{m_1}, \mathbf{c_2}, t_1, w_1\rangle \rightarrow^* \langle \mathbf{m_2}, \texttt{skip}, t_2, w_2\rangle}{\langle \mathbf{m}, \mathbf{c_1}; \mathbf{c_2}, t, w\rangle \rightarrow^* \langle \mathbf{m_2}, \texttt{skip}, t_2, w_2\rangle}$$

We know there exists a value v of $q_1(D)$, assigned to variable $\mathbf{x}^{(l_1,w_1')}$, such that $q_2(v_2)^{(l_2,w_2)} \notin t_2 - t_1$.

Since $q_2(v_2)^{(l_2,w_2)}$ comes from some certain query request

We assume it comes from $c_2' = [\mathbf{y} \leftarrow q_2(\mathbf{e_2})]^{l_2}$. We know that the execution of $\mathbf{c_2'}$ depends on $\mathbf{x}^{(l_1,w_1')}$. We want to show there is a path from $y^{(l_2,w_2)}$ to $x^{(l_1,w_1)}$ in G_{ssa}.

There are two situations that $\mathbf{c_2'}$ depends on $\mathbf{x}^{(l_1,w_1)}$.

i. $\mathbf{c_2'}$ directly depends on the variable $\mathbf{x}$. In this case, there are two possible cases:

   A. $\mathbf{c_2'}$ in either branch of if $([\mathbf{b}]^l, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}], [\bar{\mathbf{y}}, \bar{\mathbf{y_1}}, \bar{\mathbf{y_2}}], [\bar{\mathbf{z}}, \bar{\mathbf{z_1}}, \bar{\mathbf{z_2}}], \mathbf{c_a}, \mathbf{c_b})$. And $\mathbf{x}$ is used in the conditional $\mathbf{b}$.

   B. $c_2' = [\mathbf{y} \leftarrow q(\mathbf{e_2})]^{l_2}$ and variable $\mathbf{x}$ is used in $\mathbf{e_2}$.

   In both situations, $M[\mathbf{y}^{(l_2,w_2)}][\mathbf{x}^{(l_1,w_1)}] = 1$ so we know there is a direct edge in G_{ssa}.

ii. $\mathbf{c_2'}$ indirectly depends on $\mathbf{x}^{(l_1,w_1)}$. Then there exists a sequence of labelled variables $\mathbf{z_1}^{(l^1,w^1)}, \mathbf{z_2}^{(l^2,w^2)}, \ldots, \mathbf{z_n}^{(l^n,w^n)}$ for some $l_1 < l^1, l^2 \ldots, l^n < l_2$ and some $w^1, w^2, \ldots, w^n$ such that :1. $\mathbf{z_1}^{(l^1,w^1)}$ directly depends on $\mathbf{x}$. 2. $\mathbf{z_i}^{(l^i,w^i)}$ directly depends on $\mathbf{z_j}^{(l^j,w^j)}$ when $i = j+1 \wedge 1 < i < n$. 3. $\mathbf{c_2'}$ directly depends on $\mathbf{z_n}^{(l^n,w^n)}$.

   We now show that direct dependence between labelled variables $\mathbf{y}^{(l,w)}$ and $\mathbf{x}^{(l',w')}$ in ssa language is tracked by our algorithm.

   There are three possible situations:

   A. The assignment statement $[\mathbf{y} \leftarrow \mathbf{e}]$, and $\mathbf{x}$ appears in $\mathbf{e}$.

   B. The assignment of $\mathbf{y}$ (either $[\mathbf{y} \leftarrow \mathbf{e}]^l$ or $[\mathbf{y} \leftarrow q(e)]^l$) appears in either branch of an if statement. $\mathbf{x}$ is used in b.

   C. $[\mathbf{y} \leftarrow q(\mathbf{e})]^l$ and the variable $\mathbf{x}$ is used in $\mathbf{e}$.

   In the above three situations, our algorithm shows that $M[\mathbf{y}^{(l,w)}][\mathbf{x}^{(l',w')}] = 1$ so that there is an edge from $\mathbf{z}^{(l,w)}$ to $\mathbf{x}^{(l',w')}$.

So we can show that there is path from the labelled variable in $\mathbf{c}'_{\mathbf{2}}$ to $\mathbf{x}^{(l_1,w_1)}$ when $\mathbf{c}'_{\mathbf{2}}$ indirectly depends on $\mathbf{x}^{(l_1,w_1)}$.

- **Case** $$\frac{\begin{array}{c} B = |\bar{\mathbf{x}}| \qquad \Gamma \vdash^{(i,i+B)}_{M_{10},V_{10}} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \qquad \Gamma \vdash^{(i+B,i+B+A)}_{M_{20},V_{20}} \mathbf{c} \\ \forall 1 \le j < N. \Gamma \vdash^{(i+j*(B+A),i+B+j*(B+A))}_{M_{1j},V_{1j}} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \\ \Gamma \vdash^{(i+B+j*(B+A),i+B+A+j*(B+A))}_{M_{2j},V_{2j}} \mathbf{c} \qquad \Gamma \vdash^{(i+N*(B+A),i+N*(B+A)+B)}_{M,V} [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \\ \mathbf{a} = \lceil N \rceil \qquad M' = M + \sum_{0 \le j < N} M_{1j} + M_{2j} \qquad V' = V \uplus \sum_{0 \le j < N} V_{1j} \uplus V_{2j} \end{array}}{\Gamma \vdash^{(i,i+N*(B+A)+B)}_{M',V'} [\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l}$$ **loop**

Given a memory $\mathbf{m}$, database D, trace t, loop maps w, a global list G. Suppose we have $G \vDash M', V'$ and $G; w \vDash (\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}), i, i + N * (B + A) + B$, suppose $\langle \mathbf{m}, \mathbf{a} \rangle \rightarrow \langle \mathbf{m}, v_N \rangle$, then from the following SSA operational semantics:

$$\frac{v_N > 0 \qquad n = 0 \implies i = 1 \qquad n > 0 \implies i = 2}{\begin{array}{c} \langle \mathbf{m}, [\texttt{loop}\ v_N, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, w \rangle \rightarrow \\ \langle \mathbf{m}, \mathbf{c}[\bar{\mathbf{x_i}}/\bar{\mathbf{x}}]; [\texttt{loop}\ (v_N - 1), n + 1, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, (w + l) \rangle \end{array}}\ \textbf{ssa-loop}$$

$$\frac{v_N = 0 \qquad n = 0 \implies i = 1 \qquad n > 0 \implies i = 2}{\langle \mathbf{m}, [\texttt{loop}\ v_N, n, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^{\mathbf{l}}, t, w \rangle \rightarrow \langle \mathbf{m}[\bar{\mathbf{x}} \mapsto \mathbf{m}(\bar{\mathbf{x_i}})], [\texttt{skip}]^{\mathbf{l}}, t, (w \setminus l) \rangle}$$

When $v_N = 0$, the body is not executed and the new generated trace is empty, this case is trivially proved.

When $e_N = N$ for a positive integer N, there will be N iterations. The execution as follows :

$$\begin{array}{l} \langle \mathbf{m}, [\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, t, w \rangle \rightarrow \\ \langle \mathbf{m}, \mathbf{c}[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}]; [\texttt{loop}\ (v_N - 1), 1, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, t, (w + l) \rangle \rightarrow \\ \langle \mathbf{m_1}, [\texttt{loop}\ v_N - 1, 1, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, t_1, w_1 \rangle \ldots \langle \mathbf{m_N}, [\texttt{skip}]^l, t_N, (w_N \setminus l) \rangle \end{array}$$

We need to show:

1. $G_s([\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l, D, \mathbf{m}, w) \subseteq G_{ssa}(M', V', i, i + N * (B + A) + B)$.
2. $K_{[\texttt{loop}\ v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l} \leq |V'|$.

From our assumption $G; w \vDash (\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}), i, i + N * (B + A) + B$, we denote $G' = G[0, \ldots, (i-1)]$, so we have:

$$\frac{G_0 = G' \quad w_0 = w \qquad \forall 0 \leq z < N. G_z + +[\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_z} + \mathbf{l})}]; (w_z + l); \mathbf{c} \rightarrow G_{z+1}; w_{z+1} \qquad G_f = G_N + +[\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_N} \setminus \mathbf{l})}] \qquad \mathbf{a} = \lceil \mathbf{N} \rceil}{G'; w; [\texttt{loop}\ \mathbf{a}, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}]\ \texttt{do}\ \mathbf{c}]^l \rightarrow G_f; w_N \setminus l}\ \textbf{loop}$$

We can show that $G[0, \ldots, (i + B - 1)] = G_0 + +[\bar{\mathbf{x}}^{(\mathbf{l}, \mathbf{w_z} + \mathbf{l})}]$ because $B = |\bar{x}|$, and $G[0, \ldots, (i + B + A)] = G_1$
because A is the number of variables assigned in $\mathbf{c}$. So we have $G; w \vDash (\mathbf{c}, i + B, i + B + A)$, $G; w_1 \vDash (\mathbf{c}, i + B + 1 * (B + A), i + B + A + 1 * (B + A))$, until $G, w_{N-1} \vDash (\mathbf{c}, i + B + (N-1) * (B + A), i + B + A + (N-1) * (B + A))$.

By induction hypothesis on premise $\Gamma \vdash^{(i+B, i+B+A)}_{M_{20}, V_{20}} \mathbf{c}$, we conclude about the first iteration.

$$G_s(\mathbf{c}[\bar{\mathbf{x_1}}/\bar{\mathbf{x}}], D, \mathbf{m}, w) \subseteq G_{ssa}(M_{20}, V_{20}, i + B, i + B + A) \wedge K_c \leq |V_{20}|$$

By induction hypothesis on premise $\Gamma \vdash^{(i+B+j*(B+A), i+B+A+j*(B+A))}_{M_{2j}, V_{2j}} \mathbf{c}$ for $0 < j < N$, we have:

$$G_s(\mathbf{c}[\bar{\mathbf{x_2}}/\bar{\mathbf{x}}], D, \mathbf{m_j}, w_j) \subseteq G_{ssa}(M_{2j}, V_{2j}, i + B + j * (B + A), i + B + A + j * (B + A))$$

$$\wedge K_c \leq |V_{2j}|$$

We can show the first goal: $K_{[\texttt{loop } v_N,0,[\bar{\mathbf{x}},\bar{\mathbf{x_1}},\bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l} = N * K_c \leq \sum_{0 \leq j < N}(|V_{2j}|)$. Because the non-zeros of any two V_{2j}, V_{2i} $(i \neq j)$ are disjoint.

Then we need to show:

$$G_s([\texttt{loop } v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l, D, \mathbf{m}, w) \subseteq G_{ssa}(M', V', i, i + N * (B + A) + B)$$

Unfold the definition of subgraph $\subseteq$, we need to show two cases.

1. The vertices in $G_s([\texttt{loop } v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l, D, \mathbf{m}, w)$ can be mapped to the vertices in $G_{ssa}(M', V', i, i+N*(B+A)+B)$ by $f_{D,m}$. We know that there exists function $f_{D,m_j,c}$ for $G_s(\mathbf{c}, D, \mathbf{m_j}, w_j)$ when $0 \leq j < N$. We can have

$$f_{D,m}(vx) = \quad f_{D,m_j,c}(vx) \quad vx = q(v)^{(l,w_j)}$$

2. Every edge in $G_s([\texttt{loop } v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l, D, \mathbf{m}, w)$ can be mapped to a corresponding path in $G_{ssa}(M', V', i, i + N * (B + A) + B)$. The edges (vx_1, vx_2) can be divided into two categories.

   2.1. vx_1, vx_2 appear in the same iteration $(t_{j+1} - t_j)$ when $0 \leq j < N$ and $t_0 = t$. We know that $(vx_1, vx_2) \in G_s(\mathbf{c}, D, \mathbf{m_j}, w_j)$ so we can find a path $p(f_{D,M}(vx_1), f_{D,m}(vx_2))$ in $G_{ssa}(M_{2j}, V_{2j}, i + B + j * (B + A), i + B + A + j * (B + A))$. We can easily show that $G_{ssa}(M_{2j}, V_{2j}, i + B + j * (B + A), i + B + A + j * (B + A))$ is a subgraph of $G_{ssa}(M', V', i, i + N * (B + A) + B)$.

   2.2. vx_1, vx_2 comes from different iterations $j_1(t_{j_1+1} - t_{j_1}), j_2(t_{j_2+1} - t_{j_2})$ for $0 \leq j_1 < j_2 < N$. We assume $vx_1 = q_1(v_1)^{(l'_1, w'_{j_1})}$ and $vx_2 = q_2(v_2)^{(l'_2, w'_{j_2})}$. We unfold the definition of one edge so that we have:

$$\mathsf{DEP}(vx_1, vx_2, [\texttt{loop } v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l, w) \wedge \mathsf{To}(vx_1, vx_2)$$

We know that $q_1(v_1)^{(l_1', w_{j_1}')}$ in the trace $t_{j_1+1} - t_{j_1}$ comes from a command $\mathbf{c_1} = [\mathbf{x} \leftarrow q_1(\mathbf{e_1})]^{l_1'}$ We assume the loop body $\mathbf{c} = \mathbf{c_0}; \mathbf{c_1}; \mathbf{c_2}$ then we have:

$$\frac{\begin{array}{c}\langle \mathbf{m}, [\texttt{loop } v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l, t, w\rangle \rightarrow^* \\ \langle \mathbf{m'_{j_1}}, [\mathbf{x} \leftarrow q_1(v_1)]^{l_1'}; \mathbf{c_2}; [\texttt{loop } (v_N - j_1), j_1, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l, t'_{j_1}, w'_{j_1}\rangle \rightarrow^* \\ \langle \mathbf{m_{j_1}}, [\texttt{loop } (v_N - j_1), j_1, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l, t_{j_1}, w_{j_1}\rangle \rightarrow^* \\ \langle \mathbf{m_N}, [\texttt{skip}]^l, (t_N, w_N/l)\rangle\end{array}}{\langle \mathbf{m}, [\texttt{loop } v_N, 0, [\bar{\mathbf{x}}, \bar{\mathbf{x_1}}, \bar{\mathbf{x_2}}] \texttt{ do } \mathbf{c}]^l, t, w\rangle \rightarrow^* \langle \mathbf{m_N}, [\texttt{skip}]^l, (t_N, w_N/l)\rangle}$$

From the definition of DEP, we conclude that there exists a value v of $q_1(D)$, assigned to variable $\mathbf{x}^{(l_1', w_{j_1}')}$, such that $q_2(v_2)^{(l_2', w_{j_2}')} \notin t_{j_2+1} - t_{j_2}$.

Similarly, we know that $q_2^{(l_2', w_{j_2}')}$ in the trace comes from a command $\mathbf{c'_2} = [\mathbf{y} \leftarrow q_2(e_2)]^{l_2'}$. We know that the execution of $\mathbf{c'_2}(\mathbf{y}^{(l_2', w_{j_2}')})$ depends on $\mathbf{x}^{(l_1', w_{j_1}')}$.

Similar as we show in sequence case, we now need to show there exists a path in G_{ssa} from $\mathbf{y}^{(l_2', w_{j_2}')}$ to $\mathbf{x}^{(l_1', w_{j_1}')}$ if the former affect the appearance of the later.

We first show that $\mathbf{y}^{(l_2', w_{j_2}')}$ will not directly depend on $\mathbf{x}^{(l_1', w_{j_1}')}$ based on the position of $\mathbf{c'_2}$ in $\mathbf{c}$, there are three situations:

i. $l_2' < l_1'$ so that $\mathbf{c'_2} \in \mathbf{c_0}$. $\mathbf{y}^{(l_2', w_{j_2}')}$ will not directly depend on $\mathbf{x}^{(l_1', w_{j_1}')}$ because c_2 appears before the assignment of $\mathbf{x}$ at line l_1'.

ii. $l_2' = l_1'$ so that $q_1 = q_2$ and $\mathbf{x} = \mathbf{y}$. It is the same reason as previous case that $\mathbf{x}^{(l_1', w_{j_2}')}$ does not directly depend on $\mathbf{x}^{(l_1', w_{j_1}')}$.

iii. $l_2' > l_1'$ so that $\mathbf{c'_2} \in \mathbf{c_2}$. In this case, $\mathbf{y}^{(l_2', w_{j_2}')}$ still does not directly depend on $\mathbf{x}^{(l_1', w_{j_1}')}$ because there is only a direct edge between variables in the same iteration, while according to our assumption, it is

not true.

Then, we show $\mathbf{y}^{(l_2', w'_{j_2})}$ indirectly depends on $\mathbf{x}^{(l_1', w'_{j_1})}$. There exists a sequence of labelled variables $lv_1, lv_2, \ldots, lv_n$ for such that :1. lv_1 directly depends on $\mathbf{x}^{(l_1', w'_{j_1})}$. 2. lv_i directly depends on lv_j when $i = j+1 \wedge 1 < i < n$. 3. $\mathbf{y}^{(l_2', w'_{j_2})}$ directly depends on lv_n.

We now show that direct dependence between any two labelled variables $\mathbf{y}^{(l,w)}$ and $\mathbf{x}^{(l',w')}$ in ssa language is tracked by our algorithm.

There are three possible situations:

i. The assignment statement $[\mathbf{y} \leftarrow \mathbf{e}]$, and $\mathbf{x}$ appears in $\mathbf{e}$. This case also covers the variables in $\bar{\mathbf{x}}$ of our loop statement, because we treat $[\bar{x}, \bar{x_1}, \bar{x_2}]$ as assignment $\mathbf{x} = \mathbf{x_1} + \mathbf{x_2}$ to allow the values of variables passed to next iteration.

ii. The assignment of $\mathbf{y}$ (either $[\mathbf{y} \leftarrow \mathbf{e}]^l$) appears in either branch of an if statement. $\mathbf{x}$ is used in b.

iii. $[\mathbf{y} \leftarrow q(\mathbf{e})]^l$ and the variable $\mathbf{x}$ is used in $\mathbf{e}$.

In the above three situations, our algorithm shows that
$M[\mathbf{y}^{(l,w)}][\mathbf{x}^{(l',w')}] = 1$ so that there is an edge from $\mathbf{z}^{(l,w)}$ to $\mathbf{x}^{(l',w')}$.

So we can show that :
there is path from the labelled variable in $\mathbf{y}^{(l_2', w'_{j_2})}$ to $\mathbf{x}^{(l_1', w'_{j_1})}$ when $\mathbf{y}^{(l_2', w'_{j_2})}$ indirectly depends on $\mathbf{x}^{(l_1', w'_{j_1})}$.

□

BIBLIOGRAPHY

Abadi, M., Cardelli, L., & Curien, P. (1993). Formal parametric polymorphism. In *POPL '93: Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 157–170). https://doi.org/10.1145/158511.158622

Abel, A., & Scherer, G. (2012). On irrelevance and algorithmic equality in predicative type theory. *Logical Methods in Computer Science, 8*(1). https://doi.org/10.2168/LMCS-8(1:29)2012

Abel, A., Vezzosi, A., & Winterhalter, T. (2017). Normalization by evaluation for sized dependent types. *Proceedings of the ACM on Programming Languages, 1*(ICFP), article 33, pp. 1–30. https://doi.org/10.1145/3110277

Aguirre, A., Barthe, G., Birkedal, L., Bizjak, A., Gaboardi, M., & Garg, D. (2018). Relational reasoning for Markov chains in a probabilistic guarded lambda calculus. In: Ahmed, A. (ed.) *Programming Languages and Systems. ESOP 2018*. Lecture Notes in Computer Science, vol. 10801, pp. 214–241. Springer, Cham. https://doi.org/10.1007/978-3-319-89884-1_8

Aguirre, A., Barthe, G., Gaboardi, M., Garg, D., Katsumata, S., & Sato, T. (2021). Higher order probabilistic adversarial computations: Categorical semantics and program logics. *CoRR, abs/2107.01155*.

Aguirre, A., Barthe, G., Gaboardi, M., Garg, D., & Strub, P. (2017). A relational logic for higher-order programs. *Proceedings of the ACM on Programming Languages, 1*(ICFP), article 21, pp. 1–29. https://doi.org/10.1145/3110265

Ahmed, A. (2006). Step-indexed syntactic logical relations for recursive and quantified types. In: Sestoft, P. (ed.) *Programming Languages and Systems. ESOP 2006*. Lecture Notes in Computer Science, vol. 3924, pp. 69–83. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11693024_6

Ahmed, A., Dreyer, D., & Rossberg, A. (2009). State-dependent representation independence. In *POPL '09: Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 340–353). https://doi.org/10.1145/1480881.1480925

Ahmed, A. J. (2004). *Semantics of types for mutable state*. Doctoral dissertation – Princeton University. https://www.cs.princeton.edu/techreports/2004/713.pdf

Appel, A.W., &McAllester, D. A. (2001). An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and*

Systems, 23(5), 657–683.

Arlt, S., Podelski, A., Bertolini, C., Schäf, M., Banerjee, I., & Memon, A. M. (2012). Lightweight static analysis for GUI testing. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering, ISSRE 2012*, Dallas, TX, USA (pp. 301–310). IEEE Computer Society. https://doi.org/10.1109/ISSRE.2012.25

Austin, T. M., & Sohi, G. S. (1992). Dynamic dependency analysis of ordinary programs. In A. Gottlieb (Ed.) *ISCA '92: Proceedings of the 19th Annual International Symposium on Computer Architecture*, (pp. 342–351). ACM. https://doi.org/10.1145/139669.140395

Barbosa, M., Barthe, G., Grégoire, B., Koutsos, A., & Strub, P. (2021). Mechanized proofs of adversarial complexity and application to universal composability. *IACR Cryptology*. ePrint Archive, 2021, 156. https://eprint.iacr.org/2021/156

Barthe, G., Fournet, C., Grégoire, B., Strub, P., Swamy, N., & Béguelin, S. Z. (2014). Probabilistic relational verification for cryptographic implementations. In *POPL '14: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. (pp. 193–205). https://doi.org/10.1145/2535838.2535847

Barthe, G., Gaboardi, M., Arias, E. J. G., Hsu, J., Roth, A., & Strub, P. (2015). Higher order approximate relational refinement types for mechanism design and differential privacy. In *POPL '15: Proceedings of the 42st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. (pp. 55–68). https://doi.org/10.1145/2676726.2677000

Bassily, R., Nissim, K., Smith, A. D., Steinke, T., Stemmer, U., & Ullman, J. R. (2016). Algorithmic stability for adaptive data analysis. In D. Wichs, & Y. Mansour (Eds.) *STOC '16: Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing* (pp. 1046–1059). ACM. https://doi.org/10.1145/2897518.2897566

Benton, N. (2004). Simple relational correctness proofs for static analyses and program transformations. In *POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. (pp. 14–25). https://doi.org/10.1145/964001.964003

Benton, N., Hofmann, M., & Nigam, V. (2014). Abstract effects and proof-relevant logical relations. In *POPL '14: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. (pp. 619–631). https://doi.org/10.1145/2535838.2535869

Benton, N., Hofmann, M., & Nigam, V. (2016). Effect-dependent transformations for concurrent programs. In *PPDP '16: Proceedings of the 18th International Symposium on*

Principles and Practice of Declarative Programming, (pp. 188–201). https://doi.org/10.1145/2967973.2968602

Bierman, G. M.,Meijer, E., & Torgersen, M. (2007). Lost in translation: Formalizing proposed extensions to C#. In *OOPSLA '07: Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, (pp. 479–498). https://doi.org/10.1145/1297027.1297063

Bilardi, G., & Pingali, K. (1996). A framework for generalized control dependence. In *PLDI '96: Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation*, (pp. 291–300). https://doi.org/10.1145/231379.231435

Bobot, F., Conchon, S., Contejean, E., Iguernelala, M., Lescuyer, S., & Mebsout, A. (2013). *The alt-ergo automated theorem prover*, 2008.

Bracha, G., Odersky, M., Stoutamire, D., & Wadler, P. (1998). Making the future safe for the past: Adding genericity to the java programming language. In *OOPSLA '98: Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, (pp. 183–200). https://doi.org/10.1145/286936.286957

Breazu-Tannen, V., Coquand, T., Gunter, C. A., & Scedrov, A. (1991). Inheritance as implicit coercion. *Information and Computation*, *93*(1), 172–221. https://doi.org/10.1016/0890-5401(91)90055-7

Brunel, A., Gaboardi, M.,Mazza, D., & Zdancewic, S. (2014). A core quantitative coeffect calculus. In: Shao, Z. (eds.) *Programming Languages and Systems. ESOP 2014*. Lecture Notes in Computer Science, vol. 8410, pp. 351–370. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-54833-8_19

Charguéraud, A., & Pottier, F. (2015). Machine-checked verification of the correctness and amortized complexity of an efficient union-find implementation. In: Urban, C., Zhang, X. (eds.) *Interactive Theorem Proving. ITP 2015*. Lecture Notes in Computer Science, vol. 9236, pp. 137–153. Springer, Cham. https://doi.org/10.1007/978-3-319-22102-1_9

Çiçek, E. (2018). *Relational Cost Analysis*. Ph.D. dissertation, Max Planck Institute for Software Systems and Saarland University. https://www.mpi-sws.org/tr/2018-006.pdf

Çiçek, E., Barthe, G., Gaboardi, M., Garg, D., & Hoffmann, J. (2017). Relational cost analysis. In *POPL '17: Proceedings of the 44st ACM SIGPLAN Symposium on Principles of Programming Languages*, (pp. 316–329). https://doi.org/10.1145/3009837.3009858

Çiçek, E., Garg, D., & Acar, U. A. (2015). Refinement types for incremental computational complexity. In: Vitek, J. (eds) *Programming Languages and Systems. ESOP 2015*. Lecture Notes in Computer Science, vol. 9032, pp. 406–431. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-46669-8_17

Çiçek, E., Paraskevopoulou, Z., & Garg, D. (2016). A type theory for incremental computational complexity with control flow changes. In *ICPF 2016: Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming* (pp. 132–145). https://doi.org/10.1145/2951913.2951950

Çiçek, E., Qu, W., Barthe, G., Gaboardi, M., & Garg, D. (2019). Bidirectional type checking for relational properties. In *PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, (pp. 533–547). https://doi.org/10.1145/3314221.3314603

Cooley, J.W., & Tukey, J.W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, *19*(90), 297–301. https://doi.org/10.2307/2003354

Coquand, T. (1996). An algorithm for type-checking dependent types. *Science of Computer Programming, 26*(1), 167–177. https://doi.org/10.1016/0167-6423(95)00021-6

Crary, K. (2000). Typed compilation of inclusive subtyping. In *ICPF '00: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, (pp. 68–81). ACM. https://doi.org/10.1145/351240.351247

Cutler, J. W., Licata, D. R., & Danner, N. (2020). Denotational recurrence extraction for amortized analysis. *Proceedings of the ACM on Programming Languages*, *4*(ICFP), article 97, 1–29. https://doi.org/10.1145/3408979

Cytron, R., Ferrante, J., Rosen, B. K., Wegman, M. N., & Zadeck, F. K. (1991). Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *13*(4), 451–490. https://doi.org/10.1145/115372.115320

Dal Lago, U., & Gaboardi, M. (2011). Linear dependent types and relative completeness. In *2011 26th Annual IEEE Symposium on Logic in Computer Science*, (pp. 133–142). Piscataway, NJ: IEEE. https://doi.org/10.1109/LICS.2011.22

Dal Lago, U., & Petit, B. (2013). The geometry of types. In *POPL '13: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL),* (pp. 167–178). https://doi.org/10.1145/2429069.2429090

Danielsson, N. A. (2008). Lightweight semiformal time complexity analysis for purely functional data structures. In *POPL '08: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (pp. 133–144). https://doi.org/10.1145/1328438.1328457

Danner, N., Licata, D. R., & Ramyaa, R. (2015). Denotational cost semantics for functional languages with inductive types. In *ICPF 2015: Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*, (pp. 140–151). https://doi.org/10.1145/2784731.2784749

Davies, R., & Pfenning, F. (2000). Intersection types and computational effects. In *ICPF '00: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, (pp. 198–208). https://doi.org/10.1145/351240.351259

de Amorim, A. A., Gaboardi, M., Arias, E. J. G., & Hsu, J. (2014). Really natural linear indexed type checking. In *IFL '14: Proceedings of the 26th 2014 International Symposium on Implementation and Application of Functional Languages*, article 5, (pp. 1–12). https://doi.org/10.1145/2746325.2746335

Dunfield, J., & Krishnaswami, N. R. (2013). Complete and easy bidirectional type checking for higher-rank polymorphism. In *ICFP '13: Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming* (pp. 429–442). https://doi.org/10.1145/2500365.2500582

Dunfield, J., & Krishnaswami, N. R. (2019). Sound and complete bidirectional typechecking for higher-rank polymorphism with existentials and indexed types. *Proceedings of the ACM on Programming Languages*, *3*(POPL), article 9, (pp. 1–28). https://doi.org/10.1145/3290322

Dunfield, J., & Pfenning, F. (2003). Type assignment for intersections and unions in call-by-value languages. In: Gordon, A.D. (ed.) *Foundations of Software Science and Computation Structures. FoSSaCS 2003*. Lecture Notes in Computer Science, vol. 2620, pp. 250–266. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-36576-1_16

Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., & Roth, A. (2015a). Generalization in adaptive data analysis and holdout reuse. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, & R. Garnett (Eds.) *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, December 7–12, 2015, Montreal, Quebec, Canada, (pp. 2350–2358). https://proceedings.neurips.cc/paper/2015/hash/bad5f33780c42f2588878a9d07405083-Abstract.html.

Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., & Roth, A. (2015b). The reusable holdout: Preserving validity in adaptive data analysis. *Science, 349*(6248), 636–638. https://doi.org/10.1126/science.aaa9375

Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., & Roth, A. L. (2015c). Preserving statistical validity in adaptive data analysis. In R. A. Servedio, & R. Rubinfeld (Eds.) *STOC '15: Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, (pp. 117–126). ACM. https://doi.org/10.1145/2746539.2746580

Feldman, V., & Steinke, T. (2017). Generalization for adaptively-chosen estimators via stable median. In S. Kale, & O. Shamir (Eds.) Proceedings of the 30th Conference on Learning Theory, COLT 2017, vol. 65 of *Proceedings of Machine Learning Research*, (pp. 728–757). PMLR. http://proceedings.mlr.press/v65/feldman17a.html

Felleisen, M. (1991). On the expressive power of programming languages. *Science of Computer Programming*, *17*(1–3), 35–75. https://doi.org/10.1016/0167-6423(91)90036-W

Ferrante, J., Ottenstein, K. J., &Warren, J. D. (1987). The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS), 9*(3), 319–349. https://doi.org/10.1145/24039.24041

Filliâtre, J.-C., & Paskevich, A. (2013). Why3: Where programs meet provers. In: Felleisen, M., Gardner, P. (eds) *Programming Languages and Systems. ESOP 2013*. Lecture Notes in Computer Science, vol. 7792, pp. 125–128. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37036-6_8

Gaboardi, M., Haeberlen, A., Hsu, J., Narayan, A., & Pierce, B. C. (2013). Linear dependent types for differential privacy. In *POPL '13: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (pp. 357–370). https://doi.org/10.1145/2429069.2429113

Gaboardi, M., Katsumata, S., Orchard, D., & Sato, T. (2021). Graded Hoare logic and its categorical semantics. In: Yoshida, N. (ed.) *Programming Languages and Systems. ESOP 2021*. Lecture Notes in Computer Science, vol. 12648, pp. 234–263. Springer, Cham. https://doi.org/10.1007/978-3-030-72019-3_9

Gelman, A., & Loken, E. (2014). The statistical crisis in science. *American Scientist, 102*(6), 460–465. https://www.americanscientist.org/article/the-statistical-crisis-in-science

Gundry, A. (2013). *Type Inference, Haskell and Dependent Types*. Doctoral dissertation – University of Strathclyde. http://adam.gundry.co.uk/pub/thesis/thesis-2013-12-03.pdf

Hammer, C., Grimme, M., & Krinke, J. (2006). Dynamic path conditions in dependence graphs. In J. Hatcliff, & F. Tip (Eds.) *PEPM '06: Proceedings of the 2006 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*, (pp. 58–67). ACM. https://doi.org/10.1145/1111542.1111552

Hardt, M., & Ullman, J. R. (2014). Preventing false discovery in interactive data analysis is hard. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, (pp. 454–463). IEEE Computer Society. https://doi.org/10.1109/FOCS.2014.55

Hoffmann, J., Aehlig, K., & Hofmann, M. (2012). Multivariate Amortized Resource Analysis. *ACM Transactions on Programming Languages and Systems (TOPLAS), 34*(3), article 14, pp. 1–62. https://doi.org/10.1145/2362389.2362393

Ioannidis, J. P. (2005). Why most published research findings are false. *PLoS Medicine*, *2*(8), e124. https://doi.org/10.1371/journal.pmed.0020124

Jung, C., Ligett, K., Neel, S., Roth, A., Sharifi-Malvajerdi, S., & Shenfeld, M. (2020). A new analysis of differential privacy's generalization guarantees. In T. Vidick (ed.) *11th Innovations in Theoretical Computer Science Conference (ITCS 2020).* Leibniz International Proceedings in Informatics (LIPIcs), Vol. 151, (pp. 31:1–31:17). Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.ITCS.2020.31

Kavvos, G., Morehouse, E., Licata, D. R., & Danner, N. (2019). Recurrence extraction for functional programs through call-by-push-value. *Proceedings of the ACM on Programming Languages*, *4*(POPL), article 15, pp. 1–31. https://doi.org/10.1145/3371083

Lahiri, S. K., Vaswani, K., & Hoare, C. A. R. (2010). Differential static analysis: opportunities, applications, and challenges. In G. Roman, & K. J. Sullivan (Eds.) *FoSER '10: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, (pp. 201–204). https://doi.org/10.1145/1882362.1882405

Lichtman, B., & Hoffmann, J. (2017). Arrays and references in resource aware ML. In *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*. Leibniz International Proceedings in Informatics (LIPIcs), Vol. 84, (pp. 26:1–26:20). Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2017) https://doi.org/10.4230/LIPIcs.FSCD.2017.26

Lindley, S., McBride, C., & McLaughlin, C. (2017). Do be do be do. In *POPL '17: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages,* (pp. 500–514). https://doi.org/10.1145/3009837.3009897

Lobo-Vesga, E., Russo, A., & Gaboardi, M. (2021). A programming language for data privacy with accuracy estimations. *ACM Transactions on Programming Languages and*

Systems (TOPLAS), 43(2), article 6, pp. 1–42. https://doi.org/10.1145/3452096

Lucassen, J. M., & Gifford, D. K. (1988). Polymorphic effect systems. In *POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (pp. 47–57). https://doi.org/10.1145/73560.73564

Memon, A. M. (2007). An event-flow model of GUI-based applications for testing. *Software Testing, Verification & Reliability*, *17*(3), 137–157. https://dl.acm.org/doi/10.5555/1286486.1286487

Nagar, K., & Jagannathan, S. (2018). Automated detection of serializability violations under weak consistency. In *29th International Conference on Concurrency Theory (CONCUR 2018)*. Leibniz International Proceedings in Informatics (LIPIcs), Vol. 118, (pp. 41:1-41:18). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.CONCUR.2018.41

Nanevski, A., Banerjee, A., & Garg, D. (2011). Verification of information flow and access control policies with dependent types. In *SP '11: Proceedings of the 2011 IEEE Symposium on Security and Privacy*, (pp. 165–179). https://doi.org/10.1109/SP.2011.12

Nanevski, A., Banerjee, A., & Garg, D. (2013). Dependent type theory for verification of information flow and access control policies. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *35*(2), article 6, pp. 1–41. https://doi.org/10.1145/2491522.2491523

Nanevski, A., Morrisett, J. G., & Birkedal, L. (2008). Hoare type theory, polymorphism and separation. *Journal of Functional Programming, 18*(5–6), 865–911.

Neis, G., Dreyer, D., & Rossberg, A. (2011). Non-parametric parametricity. *Journal of Functional Programming, 21*(4–5), 497–562.

Ngo, V. C., Dehesa-Azuara, M., Fredrikson, M., & Hoffmann, J. (2017). Verifying and synthesizing constant-resource implementations with types. In *2017 IEEE Symposium on Security and Privacy (SP)*, (pp. 710–728). doi: 10.1109/SP.2017.53

Nielson, F., & Nielson, H. (1999). Type and effect systems. In: Olderog, ER., Steffen, B. (eds.) *Correct System Design*. Lecture Notes in Computer Science, vol. 1710, pp. 114–136. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48092-7_6

Odersky, M., Zenger, M., & Zenger, C. (2001). Colored local type inference. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (pp. 41–53). https://doi.org/10.1145/360204.360207

Petricek, T., Orchard, D. A., & Mycroft, A. (2013). Coeffects: Unified static analysis of context-dependence. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) *Automata, Languages, and Programming. ICALP 2013*. Lecture Notes in Computer Science, vol. 7966, pp. 385–397. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39212-2_35

Peyton Jones, S. L., Vytiniotis, D., Weirich, S., & Shields, M. (2007). Practical type inference for arbitrary-rank types. *Journal of Functional Programming*, *17*(1), 1–82. https://doi.org/10.1017/S0956796806006034

Pientka, B. (2008). A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *POPL '08: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (pp. 371–382). https://doi.org/10.1145/1328438.1328483

Pierce, B. C., & Turner, D. N. (2000). Local type inference. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *22*(1), 1–44. https://doi.org/10.1145/345099.345100

Pollock, L. L., & Soffa, M. L. (1989). An incremental version of iterative data flow analysis. *IEEE Transactions on Software Engineering, 15*(12), 1537–1549. https://doi.org/10.1109/32.58766

Pottier, F., & Simonet, V. (2003). Information flow inference for ML. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *25*(1), 117–158. https://doi.org/10.1145/596980.596983

Qu, W., Gaboardi, M., & Garg, D. (2019). Relational cost analysis for functional imperative programs. *Proceedings of the ACM on Programming Languages*, *3*(ICFP), article 92, pp. 1–29.

Radicek, I., Barthe, G., Gaboardi, M., Garg, D., & Zuleger, F. (2018b). Monadic refinements for relational cost analysis. *Proceedings of the ACM on Programming Languages*, *2*(POPL), article 36, pp. 1–32. https://doi.org/10.1145/3158124

Reistad, B., & Gifford, D. K. (1994). Static dependent costs for estimating execution time. In *LFP '94: Proceedings of the 1994 ACM Conference on LISP and Functional Programming*, (pp. 65–78). https://doi.org/10.1145/182409.182439

Rogers, R., Roth, A., Smith, A. D., Srebro, N., Thakkar, O., & Woodworth, B. E. (2020). Guaranteed validity for empirical approaches to adaptive data analysis. In S. Chiappa, & R. Calandra (Eds.) The 23rd International Conference on Artificial Intelligence and

Statistics, AISTATS 2020, vol. 108 of *Proceedings of Machine Learning Research*, (pp. 2830–2840). PMLR. http://proceedings.mlr.press/v108/rogers20a.html

Ryder, B. G., & Paull, M. C. (1988). Incremental data-flow analysis algorithms. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *10*(1), 1–50. https://doi.org/10.1145/42192.42193

Steinke, T., & Zakynthinou, L. (2020). Reasoning about generalization via conditional mutual information. In J. D. Abernethy, & S. Agarwal (Eds.) Conference on Learning Theory, COLT 2020, vol. 125 of *Proceedings of Machine Learning Research*, (pp. 3437–3452). PMLR. http://proceedings.mlr.press/v125/steinke20a.html

Swamy, N., Chen, J., Fournet, C., Strub, P., Bhargavan, K., & Yang, J. (2011). Secure distributed programming with value-dependent types. In *ICPF '11: Proceeding of the 16th ACM SIGPLAN International Conference on Functional Programming*, (pp. 266–278). https://doi.org/10.1145/2034773.2034811

Swamy, N., Hritcu, C., Keller, C., Rastogi, A., Delignat-Lavaud, A., Forest, S., Bhargavan, K., Fournet, C., Strub, P., Kohlweiss, M., Zinzindohoue, J. K., & Béguelin, S. Z. (2016). Dependent types and multi-monadic effects in F. In *POPL '16: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (pp. 256–270). https://doi.org/10.1145/2837614.2837655

Toro, M., & Tanter, E. (2015). Customizable gradual polymorphic effects for scala. In *OOPSLA 2015: Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, (pp. 935–953). New York, NY, USA: ACM. https://doi.org/10.1145/2814270.2814315

Turon, A. J., Thamsborg, J., Ahmed, A., Birkedal, L., & Dreyer, D. (2013). Logical relations for fine-grained concurrency. In *POPL '13: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (pp. 343–356). https://doi.org/10.1145/2429069.2429111

Ullman, J. R., Smith, A. D., Nissim, K., Stemmer, U., & Steinke, T. (2018). The limits of post-selection generalization. In S. Bengio, H. M.Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.) *Advances in Neural Information Processing Systems 31: NeurIPS 2018*, (pp. 6402–6411). https://proceedings.neurips.cc/paper/2018/hash/77ee3bc58ce560b86c2b59363281e914-Abstract.html.

Unno, H., Torii, S., & Sakamoto, H. (2017). Automating induction for solving horn clauses. In: Majumdar, R., Kunčak, V. (eds.) *Computer Aided Verification. CAV 2017*.

Lecture Notes in Computer Science, vol 10427, pp. 571–591. Springer, Cham. https://doi.org/10.1007/978-3-319-63390-9_30

Vazou, N., Seidel, E. L., & Jhala, R. (2014). Liquidhaskell: experience with refinement types in the real world. In *Haskell '14: Proceedings of the 2014 ACM SIGPLAN Symposium on Haskell*, (pp. 39–51). https://doi.org/10.1145/2633357.2633366

Vekris, P., Cosman, B., & Jhala, R. (2016). Refinement types for typescript. In C. Krintz, & E. Berger (Eds.) *PLDI '16: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, (pp. 310–325). ACM. https://doi.org/10.1145/2908080.2908110

Wang, P., Wang, D., & Chlipala, A. (2017). TiML: A functional language for practical complexity analysis with invariants. *Proceedings of the ACM on Programming Languages, 1*(OOPSLA), article 79, pp. 1–26. https://doi.org/10.1145/3133903

Winograd-Cort, D., Haeberlen, A., Roth, A., & Pierce, B. C. (2017). A framework for adaptive differential privacy. *Proceedings of the ACM on Programming Languages*, *1*(ICFP), article 10, pp. 1–29. https://doi.org/10.1145/3110254

Xi, H. (1998). *Dependent Types in Practical Programming*. Doctoral dissertation – Carnegie Mellon University. https://www.cs.cmu.edu/~rwh/students/xi.pdf

Xi, H., & Pfenning, F. (1999). Dependent types in practical programming. In *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (pp. 214–227). https://doi.org/10.1145/292540.292560

Yeom, S., Giacomelli, I., Fredrikson, M., & Jha, S. (2018). Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium, CSF*, (pp. 268–282). IEEE Computer Society. https://doi.org/10.1109/CSF.2018.00027

Zhang, D., Wang, Y., Suh, G. E., & Myers, A. C. (2015). A hardware design language for timing-sensitive information-flow security. In *ASPLOS '15: Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, (pp. 503–516). https://doi.org/10.1145/2694344.2694372

CURRICULUM VITAE