2012-02-17

Algebraic characterizations of flow-network typings

Kfoury, Assaf

Computer Science Department, Boston University


https://hdl.handle.net/2144/11392

Boston University
Algebraic Characterizations of Flow-Network Typings

Assaf Kfoury*
Boston University
Boston, Massachusetts, USA
kfoury@bu.edu

May 2, 2012

Abstract

A flow network $\mathcal{N}$ is a capacited finite directed graph, with multiple input ports/arcs and multiple output ports/arcs. A flow $f$ in $\mathcal{N}$ assigns a non-negative real number to every arc and is feasible if it satisfies flow conservation at every node and respects lower-bound/upper-bound capacities at every arc. We develop an algebraic theory of feasible flows in such networks with several beneficial consequences.

We define algorithms to infer, from a given flow network $\mathcal{N}$, an algebraic classification, which we call a typing for $\mathcal{N}$, of all assignments $f_0$ of values to the input and output arcs of $\mathcal{N}$ that can be extended to a feasible flow $f$. We then establish necessary and sufficient conditions on an arbitrary typing $T$ guaranteeing that $T$ is a valid typing for some flow network $\mathcal{N}$. Based on these necessary and sufficient conditions, we define operations on typings that preserve their validity (to be typings for flow networks), and examine the implications for a typing theory of flow networks.

*Partially supported by NSF award CCF-0820138
## Contents

1 Introduction 1

2 Flow Networks 3  
   2.1 Constant Input/Output Arcs vs. Producer/Consumer Nodes 4  
   2.2 Flow Conservation, Capacity Constraints, Type Satisfaction 4  
   2.3 A Simplifying Assumption 5

3 Examples 8

4 Flow-Network Typings 11  
   4.1 Uniqueness and Redundancy in Typings 12  
   4.2 Valid Typings and Principal Typings 13

5 Flows Versus Path Assignments 16  
   5.1 Typings for Path-Assignments and Their Satisfaction 21  
   5.2 From Typings for Flows to Typings for Path Assignments – and Back 22

6 Inferring Typings that Are Total, Tight, and Principal 24

7 Necessary and Sufficient Conditions 29

8 Special Flow Networks 31

9 Proof of Sufficiency 33  
   9.1 Assigning Lower-Bound Capacities 33  
   9.2 Assigning Upper-Bound Capacities 35  
   9.3 Assigning Remaining Capacities 45

10 Operating on Network Typings 49  
   10.1 The Lattice of Tight, Total, and Valid Typings 50  
   10.2 Compositionality of Valid Typings 54

11 Open Problems and Future Work 58  
   11.1 More Efficient Algorithms 58  
   11.2 Alternative or Improved Algebraic Formulations 60  
   11.3 Beyond Efficiency and Alternative Algebraic Formulations 63
1 Introduction

The work we report herein is a little off the beaten track. So we briefly explain the background that led to it. It starts with the modeling and analysis of large systems that are assembled in an incremental and modular way, while preserving desirable safety properties and other system requirements.

Background and motivation. Many large-scale, safety-critical systems can be viewed as inter-connections of subsystems, or modules, each of which is a producer, consumer, or regulator of flows. These flows are characterized by a set of variables and a set of constraints thereof, reflecting inherent or assumed properties or rules governing how the modules operate and what constitutes safe operation. Our notion of flow encompasses streams of physical entities (e.g., vehicles on a road, fluid in a pipe), data objects (e.g., sensor network packets, video frames), or consumable resources (e.g., electric energy, compute cycles).

Traditionally, the design and implementation of such flow networks follows a bottom-up approach, enabling system designers to certify desirable safety invariants of the system as a whole: Properties of the full system depend on a complete determination of the underlying properties of all subsystems. For example, the development of real-time applications necessitates the use of real-time kernels so that timing properties at the application layer (top) can be established through knowledge and/or tweaking of much lower-level system details (bottom), such as worst-case execution or context-switching times [6, 10, 12], specific scheduling and power parameters [1, 11, 13, 16], among many others.

While justifiable in some instances, this vertical approach does not lend itself well to emerging practices in the assembly of complex large-scale systems – namely, the integration of various subsystems into a whole by system integrators who may not possess the requisite expertise or knowledge of the internals of these subsystems [9]. This latter alternative can be viewed as a horizontal and incremental approach to system design and implementation, which has significant merits with respect to scalability and modularity. However, it also poses a major and largely unmet challenge with respect to verifiable trustworthiness – namely, how to formally certify that the system as a whole will satisfy specific safety invariants and to determine formal conditions under which it will remain so, as it is augmented, modified, or subjected to local component failures.

Further elaboration on this background can be found in a series of companion reports and articles over the last three years [3, 4, 5, 7, 8, 15].

Our proposed framework. In support of this broader agenda, we make a foray into a well-established area of combinatorial algorithms in this report – flow networks and their connections to linear programming – but from a different angle. Starting from a network \( \mathcal{N} \) with multiple input arcs/sources and output arcs/sinks, we want to derive an algebraic characterization (what we call a typing \( T \)) of all feasible flows from inputs to outputs in \( \mathcal{N} \). More precisely, we want an algebraic characterization of all assignments of values to the inputs and outputs of \( \mathcal{N} \), herein called input/output functions, extendable to feasible flows in \( \mathcal{N} \).

Moreover, we want this characterization to satisfy a modularity property in the sense that, if \( \mathcal{N}' \) is another network with typing \( T' \), and if we connect \( \mathcal{N} \) and \( \mathcal{N}' \) by linking some of their outputs to some of their inputs to obtain a new network denoted \( \mathcal{N} \oplus \mathcal{N}' \), then the typing of \( \mathcal{N} \oplus \mathcal{N}' \) is obtained by direct (and relatively easy) algebraic operations on \( T \) and \( T' \) – without any need to re-examine the two components \( \mathcal{N} \) and \( \mathcal{N}' \). Put differently, an analysis (to produce a typing) for the assembled network \( \mathcal{N} \oplus \mathcal{N}' \) can be directly obtained from the analysis of \( \mathcal{N} \) and the analysis of \( \mathcal{N}' \).

And we want more. The desired characterization should also satisfy a compositionality property, in the sense that neither of the two typings \( T \) and \( T' \) depends on the other; that is, the analysis (to produce \( T \)) for \( \mathcal{N} \) and the analysis (to produce \( T' \)) for \( \mathcal{N}' \) can be carried out independently of each other without knowledge that the two will be subsequently assembled together.\(^1\)

\(^1\)In the study of programming languages, there are syntax-directed, inductively defined, type systems that are modular but not compositional in our sense. A case in point is the so-called Hindley-Milner type system for ML-like functional languages, where the
A complementary view of the preceding is to start from an already defined typing $T$ and use it as a specification, or system requirement, against which we design a network or test the behavior of an existing one. In this dual sense, we certify the safe behavior of an already-designed network $\mathcal{N}$, or we use $T$ to guide the process of designing a network $\mathcal{N}$ satisfying $T$.

The first view of a typing theory for flow networks is one of analysis, and the second view is one of synthesis. Both are supported by our examination in this report; several examples will illustrate them.

**Main results.** What we call a flow-network typing $T$ turns out to be a bounded convex polyhedron or polytope, in the Cartesian space $\mathbb{R}^n$ for some integer $n \geq 0$, subject to appropriately defined restrictions. If $T$ is a typing for a flow network $\mathcal{N}$, then the dimension $n$ is the total number of sources and sinks in $\mathcal{N}$. Our main results in this report are:

- Theorem 54, in Section 6, certifies the correctness of algorithms for inferring what we call a principal typing for an arbitrarily given flow network.
- Theorem 57 and its Corollary 58, in Section 7, give necessary and sufficient conditions for a polytope to be a principal flow-network typing.
- Section 10, where we examine operations on principal typings, which in turn support the modularity and compositionality described above.

All of our major results heavily depend on ideas and methods from linear algebra and linear programming.

**Organization of the report.** Section 2 introduces our formulation of flow networks, as capacitated directed graphs with multiple inputs (source nodes) and outputs (sink nodes). Section 3 presents four relatively simple flow networks, carefully defined to exhibit various features of interest for the later examination. Section 4 precisely defines typings of flow networks as polytopes. Sections 2, 3, and 4, are essential background for the rest of the report.

To reach the first of our two main results, Theorem 54, there is a fair amount of preliminary work, consisting in developing transformations on flow networks that will make them easier to analyze. This material is in Sections 5 and 6.

For our second main result, specifically the sufficiency part of Theorem 57, we establish algebraic results for the particular polytopes inferred from flow networks. This is done in Sections 8 and 9.

The characterization provided by Theorem 57 and its Corollary 58, of flow-network typings as polytopes, support modularity and compositionality, which we briefly present in Section 10. We conclude with several open problems, conjectures, and discussion of follow-up work, in Section 11.

This is a long report. Some of the technical developments are fairly detailed and arduous. It is perhaps best to start with the background material in Sections 2, 3, and 4, and – before delving deep into proofs and technical details – to consider how the basic examples in Section 3 are used throughout to justify typing notions and constructions of flow networks. Particularly useful places for informal explanations, where these examples are repeatedly modified and examined, are: the very end of Section 2, the beginning of Section 6 right after Procedure 45, the very end of Section 9, the very end of Section 10, and Section 11.

---

2 In the literature, a polytope may or may not be convex. Throughout this report, we take a polytope to mean a polyhedron which is both convex and bounded.
Acknowledgments. The work reported hereinafter is a fraction of a collective effort involving several people, under the umbrella of the iBench Initiative at Boston University, co-directed by Azer Bestavros and myself. The website https://sites.google.com/site/ibenchbu/ gives a list of current and past participants, and research activities. Several iBench participants were a captive audience for partial presentations of the included material, in several sessions over the last two years. Special thanks are due to them all.

2 Flow Networks

We take a flow network $\mathcal{N}$ as a pair $\mathcal{N} = (\mathbf{N}, \mathbf{A})$ where $\mathbf{N}$ is a set of nodes and $\mathbf{A}$ a set of directed arcs. Capacities on arcs are determined by a lower-bound $\text{LC} : \mathbf{A} \to \mathbb{R}^+$ and an upper-bound $\text{UC} : \mathbf{A} \to \mathbb{R}^+$ satisfying the conditions $0 \leq \text{LC}(a) \leq \text{UC}(a)$ and $\text{UC}(a) \neq 0$ for every $a \in \mathbf{A}$. We write $\mathbb{R}$ and $\mathbb{R}^+$ for the sets of all reals and all non-negative reals, respectively.

We identify the two ends of an arc $a \in \mathbf{A}$ by writing $\text{tail}(a)$ and $\text{head}(a)$, with the understanding that flow moves from $\text{tail}(a)$ to $\text{head}(a)$. The set $\mathbf{A}$ of arcs is the disjoint union of three sets: the set $\mathbf{A}_{\#}$ of internal arcs, the set $\mathbf{A}_{\text{in}}$ of input arcs, and the set $\mathbf{A}_{\text{out}}$ of output arcs:

$$\begin{align*}
\mathbf{A} &= \mathbf{A}_{\#} \cup \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \\
\mathbf{A}_{\#} &= \{ a \in \mathbf{A} \mid \text{head}(a) \in \mathbf{N} \text{ and } \text{tail}(a) \in \mathbf{N} \} \\
\mathbf{A}_{\text{in}} &= \{ a \in \mathbf{A} \mid \text{head}(a) \in \mathbf{N} \text{ and } \text{tail}(a) \notin \mathbf{N} \} \\
\mathbf{A}_{\text{out}} &= \{ a \in \mathbf{A} \mid \text{head}(a) \notin \mathbf{N} \text{ and } \text{tail}(a) \in \mathbf{N} \}
\end{align*}$$

The tail of any input arc is not attached to any node, and the head of an output arc is not attached to any node. A few things are simplified later if we exclude self-loops; that is, for all $a \in \mathbf{A}_{\#}$, we assume that $\text{head}(a) \neq \text{tail}(a)$.

We do not assume that $\mathcal{N}$ is connected as a directed graph – an assumption often made in studies of network flows, which is sensible when there is only one input arc (or one “source node”) and only one output arc (or one “sink node”).

We assume that $\mathbf{N} \neq \emptyset$, i.e., there is at least one node in $\mathbf{N}$, without which there would be no input arc, no output arc, and nothing to say.

A flow $f$ in $\mathcal{N}$ is a function that assigns a non-negative real number to every $a \in \mathbf{A}$. Formally, a flow is a function $f : \mathbf{A} \to \mathbb{R}^+$ which, if feasible, satisfies “flow conservation” and “capacity constraints” (below).

We call a bounded, closed interval $[r, r']$ of real numbers (possibly negative) a type, and we call a typing a partial map $T$ (possibly total) that assigns types to subsets of the input and output arcs. Formally, $T$ is of the following form, where $\mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}}$:

$$T : \mathcal{P}(\mathbf{A}_{\text{in,out}}) \to \mathcal{I}(\mathbb{R})$$

where $\mathcal{P}(\ )$ is the power-set operator, $\mathcal{P}(\mathbf{A}_{\text{in,out}}) = \{ A \mid A \subseteq \mathbf{A}_{\text{in,out}} \}$, and $\mathcal{I}(\mathbb{R})$ is the set of bounded, closed intervals of reals:

$$\mathcal{I}(\mathbb{R}) = \left\{ [r, r'] \mid r, r' \in \mathbb{R} \text{ and } r \leq r' \right\}.$$

As a function, $T$ is not totally arbitrary and satisfies certain conditions, discussed in Section 4, which qualify it as a network typing.

Henceforth, we use the term “network” to mean “flow network” in the sense just defined.

---

1Our notion of a “typing” as an assignment of types/intervals to members of a powerset is different from a notion by the same name in the study of type systems for programming languages. In the latter, a typing refers to a derivable “typing judgment” consisting of a program expression $M$, a type assigned to $M$, and a type environment that includes a type for every variable occurring free in $M$. 

3
2.1 Constant Input/Output Arcs vs. Producer/Consumer Nodes

We make the notion of a network a little more complicated in order to simplify some of the constructions. It is often more convenient to deal with an outer arc \( a \in A_{\text{in, out}} \) differently if \( a \) is a constant arc, i.e., if \( LC(a) = UC(a) = d \). For such a constant arc \( a \), the value assigned to \( a \) by a feasible flow cannot vary and is always the same \( d \). It can also be omitted from the set \( a \in A_{\text{in, out}} \) of outer arcs altogether, provided we introduce a distinction between “producer nodes”, “consumer nodes”, and “nodes that are neither producer nor consumer”, as we explain next.

We use the letter \( \nu \) (“nu” for “node”) throughout to denote members of \( N \). We call a function \( \kappa : N \rightarrow \mathbb{R} \) a producer/consumer assignment for the network \( \mathcal{N} \). As defined at the beginning of Section 2, \( \kappa(\nu) = 0 \) for every \( \nu \in N \), and there are no producer and consumer nodes in \( \mathcal{N} \).

If arc \( a \in A_{\text{in}} \) is a constant input arc with \( LC(a) = UC(a) = d > 0 \) and \( head(a) = \nu \), we can turn \( \nu \) into a producer node of \( d \) units by adding \( d \) to \( \kappa(\nu) \) and then exclude \( a \) from \( A_{\text{in}} \).

Similarly, if arc \( a \in A_{\text{out}} \) is a constant output arc with \( LC(a) = UC(a) = d > 0 \) and \( tail(a) = \nu \), we can turn \( \nu \) into a consumer node of \( d \) units by subtracting \( d \) from \( \kappa(\nu) \) and then exclude \( a \) from \( A_{\text{out}} \).

There may be two (or more) input arcs, say \( a_1, a_2 \in A_{\text{in}} \), with the same node \( \nu \) as common head, i.e., \( \nu = head(a_1) = head(a_2) \). If both \( a_1 \) and \( a_2 \) are constant arcs, with \( LC(a_1) = UC(a_1) = d_1 \) and \( LC(a_2) = UC(a_2) = d_2 \), we may choose to exclude them both from \( A_{\text{in}} \), which in turn necessitates incrementing \( \kappa(\nu) \) by \( d_1 + d_2 \) units.

Similarly, if there are two (or more) constant output arcs \( a_1, a_2 \in A_{\text{out}} \) with the same tail node \( \nu = tail(a_1) = tail(a_2) \), we may choose to exclude them both from \( A_{\text{out}} \) by decrementing \( \kappa(\nu) \) by an appropriate amount.

Just as we can exclude an outer arc \( a \) from \( A_{\text{in, out}} \) whenever \( a \) is a constant arc, by appropriately adjusting the producer/consumer assignment, we can also carry out the reverse operation: Introduce a constant input (resp., output) arc \( a \) with \( LC(a) = UC(a) = d \) by decrementing (resp., incrementing) \( \kappa(\nu) \) by \( d \), where \( \nu = head(a) \) (resp. \( \nu = tail(a) \)).

Based on the preceding, \( \kappa(\nu) \) may be a positive number or a negative number. If it is a positive number, \( \nu \) is a producer, and if it is a negative number, \( \nu \) is a consumer. A node \( \nu \) is either a producer, or a consumer, or neither; \( \nu \) cannot be both a producer and a consumer.

If there are no producers and no consumers in \( \mathcal{N} \), i.e., \( \kappa(\nu) = 0 \) for every \( \nu \in N \), we typically omit mention of \( \kappa \) altogether. We use the convenience of explicitly designating some nodes as producer/consumer, and removing (some) constant input/output arcs from consideration in Sections 2.2, 2.3, 9, and 11. All of the remaining sections can be read by assuming \( \kappa(\nu) = 0 \) for every node \( \nu \). Whenever we say “constant input/output arc \( a \)”, we mean \( LC(a) = UC(a) > 0 \), avoiding the longer and less convenient “non-zero constant input/output arc \( a \)”.

2.2 Flow Conservation, Capacity Constraints, Type Satisfaction

Though obvious, we precisely state fundamental concepts underlying our entire examination and introduce some of our notational conventions, in Definitions 1, 2, 3, and 4.

**Definition 1 (Flow Conservation).** If \( A \) is a subset of arcs in \( \mathcal{N} \) and \( f \) a flow in \( \mathcal{N} \), we write \( \sum f(A) \) to denote the sum of the flows assigned to all the arcs in \( A \):

\[
\sum f(A) = \sum \{ f(a) \mid a \in A \}
\]

By convention, \( \sum \emptyset = 0 \). If \( A = \{a_1, \ldots, a_p\} \) is the set of arcs entering a node \( \nu \), and \( B = \{b_1, \ldots, b_q\} \) the set of arcs exiting \( \nu \), conservation of flow at \( \nu \) is expressed by the linear equation:

\[
(1) \quad \kappa(\nu) + \sum f(A) = \sum f(B)
\]

There is one such equation \( E_\nu \) for every node \( \nu \in N \) and \( \mathcal{E} = \{ E_\nu \mid \nu \in N \} \) is the collection of all equations enforcing flow conservation in \( \mathcal{N} \).
Definition 2 (Capacity Constraints). A flow $f$ satisfies the capacity constraints at arc $a \in A$ if:

$$(2) \quad LC(a) \leq f(a) \leq UC(a)$$

There are two such inequalities $C_a$ for every $a \in A$ and $\mathcal{C} = \{ C_a \mid a \in A \}$ is the collection of all inequalities enforcing capacity constraints in $\mathcal{N}$. 

Definition 3 (Feasible Flows). A flow $f$ is feasible iff two conditions:

- for every node $\nu \in N$, the equation in (1) is satisfied,
- for every arc $a \in A$, the two inequalities in (2) are satisfied,

following standard definitions of network flows.

Definition 4 (Type Satisfaction). Let $\mathcal{N}$ be a network with input/output arcs $A_{in, out} = A_{in} \cup A_{out}$, and let $T : \mathcal{P}(A_{in, out}) \to I(\mathbb{R})$ be a typing over $A_{in, out}$. We say the flow $f$ satisfies $T$ if, for every $A \in \mathcal{P}(A_{in, out})$ for which $T(A)$ is defined and $T(A) = [r, r']$, it is the case:

$$(3) \quad r \leq \sum f(A \cap A_{in}) - \sum f(A \cap A_{out}) \leq r'$$

We often denote a typing $T$ for $\mathcal{N}$ by simply writing $\mathcal{N} : T$.

Notation 5. We use mostly the letter $\mathcal{N}$ possibly decorated (with a prime, double prime, tilde, etc.), and occasionally the letter $M$, to range over the set of networks. To denote particular example networks, we exclusively use the letter $\mathcal{N}$ appropriately subscripted (as in $\mathcal{N}_1, \mathcal{N}_2$, etc.).

We use mostly the letters $A_{in}$ and $A_{out}$, and occasionally the letters $B_{in}$ and $B_{out}$, to denote the sets of input arcs and output arcs. We also write $A_{in, out}$ for their disjoint unions $A_{in} \cup A_{out}$ and $B_{in} \cup B_{out}$; although this notation is a little ambiguous (not indicating which arc in $A_{in, out}$ is an input and which is an output), the context will always disambiguate.

We use exclusively the letters $\mathcal{E}$ and $\mathcal{C}$ to range over sets of flow-conservation equations and sets of capacity-constraint inequalities. To denote particular examples of such sets, we appropriately subscript them (as in $\mathcal{E}_1$ and $\mathcal{E}_2$, etc.).

We use mostly the letter $T$ possibly decorated (with a prime, double prime, tilde, etc.), and occasionally the letters $S$ and $U$, to range over typings. To denote particular typings, in all the examples, we exclusively use the letter $T$ appropriately subscripted (as in $T_1, T_2$, etc.).

If $\mathcal{N}_i$ is a particular example network, with particular conservation equations $\mathcal{E}_j$ and constraint inequalities $\mathcal{C}_j$, and $T_k$ is a particular example typing inferred from $\mathcal{E}_j$ and $\mathcal{C}_j$ or is related to them in some way, we make the subscripts $i, j$, and $k$, all the same.

### 2.3 A Simplifying Assumption

We need to restrict the form of networks for technical reasons that will simplify our later analysis. Proposition 7 shows that this restriction does not make our examination any less general. The usefulness of this restriction, which we call Property (†), is demonstrated by Proposition 11: If (†) is satisfied, then we can assume flows from input to output move along acyclic paths and thus avoid all cycles.

Restriction 6. Let $\mathcal{N} = (N, A)$ be a network, with $A = A_\# \cup A_{in} \cup A_{out}$, lower-bound function $LC : A \to \mathbb{R}^+$ and upper-bound function $UC : A \to \mathbb{R}^+$, and producer/consumer assignment $\kappa : N \to \mathbb{R}$. Unless explicitly stated otherwise, we assume Property (†) holds throughout, which is:

(†) For every $a \in A_\#$, it holds that $LC(a) = 0$. 

We thus assume that the lower-bound LC(a) on every internal arc in A#, but not on any of the outer arcs in A_in ∪ A_out, is zero. We place no restriction on the upper-bound capacity UC(a).

**Proposition 7.** Let N = (N, A) be a network not necessarily satisfying Property (†). We can construct another network N' = (N', A') satisfying Property (†) such that:

1. N = N', A_in = A'_in, A_out = A'_out, and A# = A'#.

2. For every feasible flow f : A → R⁺ in N, there is a uniquely defined feasible flow f' : A' → R⁺ in N' — and, conversely, for every feasible flow f' : A' → R⁺ in N', there is a uniquely defined feasible flow f : A → R⁺ in N — such that:

   \[
   f'(a) = \begin{cases} 
   f(a) - LC(a) & \text{if } a \in A# \text{ and } LC(a) > 0, \\
   f(a) & \text{otherwise.}
   \end{cases}
   \]

As directed graphs, N and N' are identical except for differences in their lower-bound and upper-bound capacities and, as shown in the proof below, in their producer/consumer assignments.

Note that the flows f and f' in part 2 above assign the same values to all outer arcs A_in,out. Hence, if T : P(A_in,out) → T(R) is a typing, f satisfies T if and only if f' satisfies T.

**Proof.** We can assume that for every a ∈ A we have 0 ≤ LC(a) ≤ UC(a), otherwise if LC(a) > UC(a) for some a ∈ A, there is no feasible flow in N.

Initially, we take the capacity bounds and the producer/consumer assignment of N' to be the same as those of N, i.e., LC₀ = LC, UC₀ = UC and κ₀ = κ, and we adjust them in stages, one stage for every internal arc a ∈ A# such that LC₀(a) ≠ 0.

Consider an arbitrary a ∈ A# such that LC₀(a) = d > 0. Let ν' = tail(a) and ν'' = head(a). For the first stage, we adjust the producer/consumer assignment as follows:

\[
κ₁(ν) = \begin{cases} 
κ₀(ν') - d & \text{if } ν = ν', \\
κ₀(ν'') + d & \text{if } ν = ν'', \\
κ₀(ν) & \text{otherwise.}
\end{cases}
\]

If κ(ν') = κ(ν'') = 0, i.e., nodes ν' and ν'' in N are neither producer nor consumer, then in N', node ν' becomes a consumer and node ν'' a producer. The adjustment from κ₀ to κ₁ induces an adjustment in the capacity bounds:

\[
LC₁(a) = LC₀(a) - d = 0 \quad \text{and} \quad UC₁(a) = UC₀(a) - d.
\]

After stage 1, there is one less internal arc a ∈ A# such that LC₀(a) ≠ 0. We proceed in the same way to produce LCᵢ₊₁, UCᵢ₊₁ and κᵢ₊₁ from LCᵢ, UCᵢ and κᵢ for every i > 0. Since A# is finite, this process is bound to terminate in n ≥ 1 stages, at the end of which we have a network N' where LCᵢ(a) = LCᵢₙ(a) = 0 for every internal arc a ∈ A#. It is now straightforward to check that the conclusion of the proposition is satisfied.

**Definition 8 (Subnetworks Induced by Feasible Flows).** Let N = (N, A) be a network and f : A → R⁺ a feasible flow in N. The subnetwork N' = (N', A') induced by f is given by:

- A' = \{ a ∈ A | f(a) ≠ 0 \},
- N' = \{ ν ∈ N | ν = head(a) or ν = tail(a) for some a ∈ A' \},

6
• \( A'_{\text{in}} = A_{\text{in}} \cap A' \), \( A'_{\text{out}} = A_{\text{out}} \cap A' \), and \( A'_{\#} = A_{\#} \cap A' \).

• \( LC'(a) = LC(a) \) and \( UC'(a) = UC(a) \) for every \( a \in A' \).

In words, \( N' \) consists of all the arcs \( a \in N \) such that \( f(a) \neq 0 \). \( \square \)

**Proposition 9.** Let \( N' = (N, A') \) be a network, \( f : A \to \mathbb{R}^+ \) a feasible flow in \( N' \), and \( N = (N', A') \) the subnetwork induced by \( f \). Then \( f' = [f]_{A'} \) is a feasible flow in \( N' \), where \( [f]_{A'} \) denotes the restriction of \( f : A \to \mathbb{R}^+ \) to the subset \( A' \subseteq A \).

**Proof.** This is straightforward from the definitions. All details omitted. \( \square \)

**Definition 10 (Paths).** A path \( \pi \) in a network \( N \) is a sequence of arcs \( b_1 b_2 \cdots b_k \) such that \( \text{head}(b_j) = \text{tail}(b_{j+1}) \) for every \( 1 \leq j < k \). We write \( \text{first}(\pi) \) to denote the first arc \( b_1 \), and \( \text{last}(\pi) \) to denote the last arc \( b_k \), in \( \pi \). If \( b = b_j \) for some \( 1 \leq j \leq k \), we say arc \( b \) occurs in \( \pi \) and write \( b \in \pi \). We write \( \bar{\pi} \) for the multiset \( \{b_1, b_2, \ldots, b_k\} \) and \( \bar{\pi} \) for the corresponding set, i.e., \( \bar{\pi} \) is obtained from \( \bar{\pi} \) by making the multiplicity of every entry equal to 1. In general \( |\bar{\pi}| \geq |\bar{\pi}| \). We recall some standard terminology, for later reference:

- A path \( \pi \) is acyclic if \( \pi \) does not visit the same node twice.
- A path \( \pi \) is a cycle if \( \text{head}(\text{last}(\pi)) = \text{tail}(\text{first}(\pi)) \).
- A cycle \( \pi \) is simple if omitting \( \text{last}(\pi) \) from \( \pi \) gives an acyclic path.
- A path \( \pi_1 \) is a subpath of a path \( \pi_2 \) if \( \pi_1 \subseteq \pi_2 \). In such a case, we may say \( \pi_2 \) includes or contains \( \pi_1 \).
- A path \( \pi \) is full if \( \text{first}(\pi) \in A_{\text{in}} \) and \( \text{last}(\pi) \in A_{\text{out}} \).

If \( \pi \) is an acyclic path or a simple cycle, then \( \bar{\pi} = \bar{\pi} \) and \( \text{length}(\pi) = \bar{\pi} = \bar{\pi} \). If \( \pi \) is a cycle, then \( \text{length}(\pi) \geq 2 \), because there are no self-loops. We distinguish two special subsets, \( \Gamma \) and \( \Delta \), of paths in \( N' \):

\[
\begin{align*}
\Gamma &= \{ \gamma \mid \gamma \text{ is a full acyclic path in } N' \} \\
\Delta &= \{ \delta \mid \delta \text{ is a simple cycle in } N' \}
\end{align*}
\]

Since \( N' \) is finite, both \( \Gamma \) and \( \Delta \) are finite sets. Let \( \Pi = \Gamma \cup \Delta \), which we call the set of good paths.

Every full path \( \pi \) in \( N' \) can be uniquely decomposed into a full acyclic path \( \gamma \in \Gamma \) and finitely many, possibly overlapping, simple cycles \( \delta_1, \ldots, \delta_\ell \in \Delta \). In such a case, we write \( \pi = \gamma \oplus \delta_1 \oplus \cdots \oplus \delta_\ell \). \( \square \)

**Proposition 11.** Let \( N = (N, A) \) be a network that satisfies Property (\( \dagger \)) and let \( f : A \to \mathbb{R}^+ \) be a feasible flow in \( N \). Then there is a feasible flow \( g : A \to \mathbb{R}^+ \) in \( N' \) such that:

1. The subnetwork of \( N \) induced by \( g \) is acyclic.
2. \( f(a) = g(a) \) for every \( a \in A_{\text{in}} \cup A_{\text{out}} \).

In words, assuming that every producer/consumer node is turned into the head/tail of an input/output arc, all feasible flows can be restricted to move along full acyclic paths in \( N' \) from input to output arcs.

**Proof.** Let us say that a simple cycle \( \delta \) in \( N' \) is active relative to a feasible flow \( f \) if \( f(b) \neq 0 \) for at least one arc \( b \in \delta \). Starting from the given \( f \), we want to define a feasible flow \( g \) such that every simple cycle \( \delta \) in \( N' \) is not active according to \( g \). We obtain \( g \) from \( f_0 = f \) in as many stages as there remain simple cycles that are active, with one simple cycle becoming non-active after each stage. The number of simple cycles being finite, this process is bound to terminate, i.e., we define feasible flows \( f_0, f_1, \ldots, f_n \) in succession, with the last \( f_n \) being the desired \( g \) because all simple cycles are non-active relative to \( f_n \).
It suffices to explain how $f_{i+1}$ is obtained from $f_i$, where $0 \leq i < n$. Suppose $\delta$ is a simple cycle which is active relative to $f_i$. Let
\[
    r = \min \{ f_i(b) \mid b \in \delta \} > 0
\]
$r$ is not zero because $\delta$ is active relative to $f_i$. We define $f_{i+1}$ as follows:
\[
    f_{i+1}(b) = \begin{cases} 
        f_i(b) - r & \text{if } b \in \delta, \\
        f_i(b) & \text{if } b \notin \delta.
    \end{cases}
\]
It is readily checked that, if $f_i$ is feasible, then so is $f_{i+1}$ and $\delta$ is now no longer active because, for at least one $b \in \delta$, it must be that $f_{i+1}(b) = 0$. Finally, the subnetwork of $\mathcal{N}$ induced by $g$ is acyclic, because there is no simple cycle which is active relative to $g$. 

Proposition 11 does not hold if the network $\mathcal{N}$ does not satisfy Property ($\dagger$), as illustrated by Example 12 in the next section.

3 Examples

There are four examples of networks in this section. They set our graphical conventions for the rest of the report and will be used repeatedly to illustrate various notions.

Unless explicitly stated otherwise, there are no producer and consumer nodes in our examples, i.e., $\kappa(\nu) = 0$ for every node $\nu$.

Example 12. The network $\mathcal{N}_1$ on the left in Figure 1 does not satisfy Property ($\dagger$) of Section 2.3, if $t \neq 0$. This network contains exactly one simple cycle $\delta = a_4 a_5 a_6$.

We leave the lower bounds $r$, $s$, and $t$, on arcs $a_2$, $a_3$, and $a_6$, unspecified for the moment. Later, we choose $r$, $s$, and $t$, to illustrate other concepts. We use the arc names $\{a_1, \ldots, a_6\}$ as variables in the equations and inequalities below. There are 3 equations enforcing flow conservation in $\mathcal{N}_1$:
\[
    \mathcal{E}_1 = \{ a_1 + a_5 = a_4, \quad a_2 + a_6 = a_5, \quad a_3 + a_6 = a_4 \}
\]
There are 12 inequalities, with 2 for each of the 6 arcs, enforcing lower-bound and upper-bound constraints:
\[
    \mathcal{G}_1 = \{ \ 0 \leq a_1 \leq 15, \quad r \leq a_2 \leq 50, \quad s \leq a_3 \leq 35, \quad 0 \leq a_4 \leq 40, \quad 0 \leq a_5 \leq 40, \quad t \leq a_6 \leq 40 \} \]
Every feasible flow $f : \{a_1, \ldots, a_6\} \to \mathbb{R}^+$ in $\mathcal{N}_1$ must assign a value $f(a_6) \geq t$. Hence, if $t \neq 0$, the simple cycle $\delta$ is necessarily active for all feasible flows.

The network $\mathcal{N}_1'$ on the right in Figure 1 satisfies Property ($\dagger$), after turning one node into a producer and one node into a consumer, each of $t$ units, according to Proposition 7. The equations in $\mathcal{E}_1$ and inequalities in $\mathcal{G}_1$ have to be adjusted accordingly:
\[
    \mathcal{E}_1' = \{ a_1 + a_5 = a_4, \quad a_2 + a_6 + t = a_5, \quad a_3 + a_6 + t = a_4 \}
\]
\[
    \mathcal{G}_1' = \{ \ 0 \leq a_1 \leq 15, \quad r \leq a_2 \leq 50, \quad s \leq a_3 \leq 35, \quad 0 \leq a_4 \leq 40, \quad 0 \leq a_5 \leq 40, \quad 0 \leq a_6 \leq 40 - t \} \]
In $\mathcal{N}_1'$, the simple cycle $\delta = a_4 a_5 a_6$ is no longer active for all feasible flows.

8
We can compute the values of a maximum feasible flow and a minimum feasible flow using linear programming, e.g., the network simplex method. Alternatively, we can use standard algorithms on capacitated graphs, e.g., the min-cut/max-flow theorem and the max-cut/min-flow theorem.

The upper-bound capacity of a min-cut $\Phi$ is the value of a feasible max flow in $\mathcal{N}_2$, and the lower-bound capacity of a max-cut $\Psi$ is the value of a feasible min flow in $\mathcal{N}_2$. (In general, there are more than one of each, but not in the network $\mathcal{N}_2$.) The upper-bound capacity 14 of the min-cut $\Phi$ is obtained according to the formula:

$$\sum\{\text{upper-bounds of forward arcs in } \Phi\} - \sum\{\text{lower-bounds of backward arcs in } \Phi\} = 14$$

and the lower-bound capacity 4 of the max-cut $\Psi$ is obtained according to the formula:

$$\sum\{\text{lower-bounds of forward arcs in } \Psi\} - \sum\{\text{upper-bounds of backward arcs in } \Psi\} = 7$$

$\Phi$ and $\Psi$ are shown in Figure 3. Hence, the value of any feasible flow in $\mathcal{N}_2$ will be in the interval $[7, 14]$ and the types assigned by a typing $T$ will have to enforce these limits, among other things.

The preceding formulas for $\Phi$ and $\Psi$ can be in fact simplified because $\mathcal{N}_2$ satisfies Property (†). Namely, there are three useful consequences of (†): (1) the lower-bound of every backward arc in a min-cut (such as $\Phi$ here) is 0, (2) a max-cut consists of all arcs in $A_{in}$ or (such as $\Psi$ here) all arcs in $A_{out}$, and (3) there are no backward arcs in a max-cut.

Example 13. We choose a network $\mathcal{N}_2$ which satisfies Property (†) of Section 2.3. $\mathcal{N}_2$ is shown in Figure 2. We first list the collection $\mathcal{E}_2$ of 8 equations enforcing flow conservation in $\mathcal{N}_2$:

$$\mathcal{E}_2 = \left\{ a_1 = a_6 + a_7 , \quad a_2 = a_8 + a_9 , \quad a_3 + a_{18} = a_{10} + a_{11} , \quad a_6 + a_8 = a_{12} , \quad a_7 + a_9 + a_{10} = a_{13} + a_{14} , \quad a_{14} + a_{15} = a_{17} + a_{18} , \quad a_{12} + a_{13} + a_{17} = a_4 + a_{16} , \quad a_{11} + a_{16} = a_5 + a_{15} \right\}$$

We use the arc names $\{a_1, \ldots, a_{18}\}$ as variables in the preceding equations, and again in the collection $\mathcal{E}_2$ of 2\cdot 18 = 36 inequalities enforcing lower-bound and upper-bound constraints:

$$\mathcal{E}_2 = \left\{ 2 \leq a_1 \leq 15 , \quad 0 \leq a_2 \leq 20 , \quad 4 \leq a_3 \leq 25 , \quad 3 \leq a_4 \leq 8 , \quad 4 \leq a_5 \leq 15 , \quad 0 \leq a_6 \leq 5 , \quad 0 \leq a_7 \leq 5 , \quad 0 \leq a_8 \leq 2 , \quad 0 \leq a_9 \leq 10 , \quad 0 \leq a_{10} \leq 10 , \quad 0 \leq a_{11} \leq 4 , \quad 0 \leq a_{12} \leq 5 , \quad 0 \leq a_{13} \leq 3 , \quad 0 \leq a_{14} \leq 2 , \quad 0 \leq a_{15} \leq 3 , \quad 0 \leq a_{16} \leq 10 , \quad 0 \leq a_{17} \leq 7 , \quad 0 \leq a_{18} \leq 6 \right\}$$

We can compute the values of a maximum feasible flow and a minimum feasible flow using linear programming, e.g., the network simplex method. Alternatively, we can use standard algorithms on capacitated graphs, e.g., the min-cut/max-flow theorem and the max-cut/min-flow theorem.

Figure 1: Network $\mathcal{N}_1$ on the left does not satisfy Property (†) of Section 2.3, if $t \neq 0$, network $\mathcal{N}_1'$ on the right does, after introducing a producer node and a consumer node, each of $t$ units, indicated by heavy arrow heads. If only one capacity is shown for an arc, it is an upper bound; all omitted lower bounds are 0. The lower bounds $r$ and $s$ are specified in follow-up examples.
Figure 2: Network $\mathcal{N}_2$ satisfies Property (†), with its named arcs (on the left) and its lower-bound and upper-bound capacities (on the right). If only one capacity is shown, it is an upper bound; omitted lower bounds are 0.

Figure 3: The min cut $\Phi = \{a_{11}, a_{12}, a_{13}, a_{14}, a_{18}\}$ and the max cut $\Psi = \{a_4, a_5\}$ in $\mathcal{N}_2$.

Example 14. Network $\mathcal{N}_3$ is shown on the left in Figure 4. There are 6 equations in $\mathcal{E}_3$ enforcing flow conservation, one for each node in $\mathcal{N}_3$, and $2 \cdot 11 = 22$ inequalities in $\mathcal{C}_3$ enforcing lower-bound and upper-bound constraints, two for each arc in $\mathcal{N}_3$. We omit inclusion of $\mathcal{E}_3$ and $\mathcal{C}_3$, which are straightforward.

In Figure 4, all omitted lower-bound capacities are 0 and all omitted upper-bound capacities are $K$. $K$ is an unspecified "very large number".

By easy inspection, a minimum flow in $\mathcal{N}_3$ pushes 0 units through, and a maximum flow in $\mathcal{N}_3$ pushes 30 units. The value of every feasible flow in $\mathcal{N}_3$ will therefore be in the interval $[0, 30]$.

An appropriate typing for $\mathcal{N}_3$ will specify a permissible interval at each of the outer arcs $/\point{a_1}, /\point{a_2}, /\point{a_3}, /\point{a_4}/\point{}$ so that the total flow pushed through $\mathcal{N}_3$ remains within the interval $[0, 30]$. $\square$

Example 15. Network $\mathcal{N}_4$ is shown on the right in Figure 4. There are 8 equations in $\mathcal{E}_4$ enforcing flow conservation, one for each node in $\mathcal{N}_4$, and $2 \cdot 16 = 32$ inequalities in $\mathcal{C}_4$ enforcing lower-bound and upper-bound constraints, two for each arc in $\mathcal{N}_4$. We omit inclusion of $\mathcal{E}_4$ and $\mathcal{C}_4$, which are straightforward.

By inspection, a minimum flow in $\mathcal{N}_4$ pushes 0 units through, and a maximum flow in $\mathcal{N}_4$ pushes 30 units. The value of all feasible flows in $\mathcal{N}_4$ will therefore be in the interval $[0, 30]$, the same as for $\mathcal{N}_3$ in Example 14.

However, as we will note when we re-visit $\mathcal{N}_3$ and $\mathcal{N}_4$ in Examples 47 and 48, an appropriate typing for the first will not be necessarily appropriate for the second, nor vice-versa. This will imply, among other things, there are maximum-value flows in $\mathcal{N}_3$ assigning values to the outer arcs $\{a_1, a_2, a_3, a_4\}$ which are different from those implied by a maximum-value flow in $\mathcal{N}_4$. $\square$
4 Flow-Network Typings

Let \( A = A_\# \cup A_{\text{in}} \cup A_{\text{out}} \) be the set of arcs in a network, with \( A_{\text{in}} = \{a_1, \ldots, a_m\} \) and \( A_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\} \), where \( m, n \geq 1 \). As before, we abbreviate \( A_{\text{in}} \cup A_{\text{out}} \) by writing \( A_{\text{in,out}} \), and call a partial map \( T \) of the form:

\[
T : \mathcal{P}(A_{\text{in,out}}) \rightarrow \mathcal{I}(\mathbb{R})
\]

a typing over \( A_{\text{in,out}} \). A typing \( T \) over \( A_{\text{in,out}} \) defines a convex polyhedron, which we denote \( \text{Poly}(T) \), in the Euclidean hyperspace \( \mathbb{R}^{m+n} \), as we explain next. We think of the \( m + n \) arcs in \( A_{\text{in,out}} \) as the dimensions of the space \( \mathbb{R}^{m+n} \), and we thus use the arc names as variables to which we assign values in \( \mathbb{R} \). \( \text{Poly}(T) \) is the non-empty intersection of at most \( 2 \cdot (2^{m+n} - 1) \) halfspaces, because there are \( (2^{m+n} - 1) \) non-empty subsets in \( \mathcal{P}(A_{\text{in,out}}) \) and each induces two inequalities. Let \( \emptyset \neq A \subseteq A_{\text{in,out}} \) with:

\[
A \cap A_{\text{in}} = \{a'_1, \ldots, a'_k\} \quad \text{and} \quad A \cap A_{\text{out}} = \{a'_{k+1}, \ldots, a'_{\ell}\}
\]

Suppose \( T(A) \) is defined and let \( T(A) = [r, r'] \). Corresponding to \( A \), there are two linear inequalities in the variables \( \{a'_1, \ldots, a'_k\} \), denoted \( T_\#(A) \) and \( T_\varepsilon(A) \):

\[
(4) \quad T_\#(A) : \quad a'_1 + \cdots + a'_k - a'_{k+1} - \cdots - a'_{\ell} \geq r \quad \text{or, more succintly,} \quad \sum (A \cap A_{\text{in}}) - \sum (A \cap A_{\text{out}}) \geq r
\]

\[
T_\varepsilon(A) : \quad a'_1 + \cdots + a'_k - a'_{k+1} - \cdots - a'_{\ell} \leq r' \quad \text{or, more succintly,} \quad \sum (A \cap A_{\text{in}}) - \sum (A \cap A_{\text{out}}) \leq r'
\]

and, therefore, two halfspaces \( \text{Half}(T_\#(A)) \) and \( \text{Half}(T_\varepsilon(A)) \) in \( \mathbb{R}^{m+n} \):

\[
(5) \quad \text{Half}(T_\#(A)) = \{ r \in \mathbb{R}^{m+n} \mid r \text{ satisfies } T_\#(A) \}
\]

\[
\text{Half}(T_\varepsilon(A)) = \{ r \in \mathbb{R}^{m+n} \mid r \text{ satisfies } T_\varepsilon(A) \}
\]

We can therefore define \( \text{Poly}(T) \) formally as follows:

\[
\text{Poly}(T) = \bigcap \{ \text{Half}(T_\#(A)) \cap \text{Half}(T_\varepsilon(A)) \mid \emptyset \neq A \subseteq A_{\text{in,out}} \text{ and } T(A) \text{ is defined} \}
\]

Generally, many of the inequalities induced by the typing \( T \) will be redundant, and the induced \( \text{Poly}(T) \) will be defined by far fewer than \( 2 \cdot (2^{m+n} - 1) \) halfspaces. For later reference, we give the name \( \text{Constraints}(T) \) to the set of all inequalities/constraints that define \( \text{Poly}(T) \):

\[
(6) \quad \text{Constraints}(T) = \{ T_\#(A) \mid \emptyset \neq A \subseteq A_{\text{in,out}} \text{ and } T(A) \text{ is defined} \}
\]

\[
\cup \{ T_\varepsilon(A) \mid \emptyset \neq A \subseteq A_{\text{in,out}} \text{ and } T(A) \text{ is defined} \}
\]
Restriction 16. We agree that, in order for $T : \mathcal{P}(A_{in,out}) \rightarrow \mathcal{I}(\mathbb{R})$ to be a typing, three requirements must be satisfied:

1. $T(\emptyset) = T(A_{in,out}) = [0,0] = \{0\}$. Informally, this corresponds to global flow conservation: The total amount entering a network must equal the total amount exiting it – after turning all producer/consumer nodes into constant input/output arcs, which is always possible to assume by the discussion in Section 2.1.

2. Poly$(T)$ must be a bounded subspace of $\mathbb{R}^{m+n}$ and therefore a convex polytope, rather than just a convex polyhedron. This means that for every $1 \leq i \leq m + n$, there is an bounded interval $[s, s']$, such that for every $(r_1, \ldots, r_i, \ldots, r_{m+n}) \in \text{Poly}(T)$, it must be that $s \leq r_i \leq s'$. This is a mild restriction, obviating the need to deal separately with cases of unboundedly large flows.

3. Poly$(T)$ is entirely contained within the first orthant of the hyperspace $\mathbb{R}^{m+n}$, i.e., the subspace $(\mathbb{R}^+)^{m+n}$.

Even assuming that the three preceding requirements are satisfied, not all typings are “inhabited”, i.e., some are not typings of any networks. We want to characterize the typings $T$ that are inhabited; specifically, we want to formulate necessary and sufficient conditions (preferably algebraic) that precisely select, among all typings, those that are tight and principal typings (of networks) – which we define in the two next subsections.

Definition 17 (Input-Output Functions). Let $A = A_{\#} \uplus A_{in} \uplus A_{out}$ be the set of arcs in a network $\mathcal{N}$, where $A_{in} = \{a_1, \ldots, a_m\}$ and $A_{out} = \{a_{m+1}, \ldots, a_{m+n}\}$. We call a function $f : A_{in,out} \rightarrow \mathbb{R}^+$ an input-output function or, more briefly, an IO function for $\mathcal{N}$, where $A_{in,out} = A_{in} \uplus A_{out}$ as before.

If $f' : A \rightarrow \mathbb{R}^+$ is a flow in the network $\mathcal{N}$, then the restriction of $f'$ to $A_{in,out}$, denoted $[f']_{A_{in,out}}$, is an IO function. We say that an IO function $f : A_{in,out} \rightarrow \mathbb{R}^+$ is feasible if there is a feasible flow $f' : A \rightarrow \mathbb{R}^+$ such that $f = [f']_{A_{in,out}}$.

A typing $T : \mathcal{P}(A_{in,out}) \rightarrow \mathcal{I}(\mathbb{R})$ for $\mathcal{N}$ is defined independently of the internal arcs $A_{\#}$. Hence, the notion of satisfaction of $T$ by a flow $f'$ as in Definition 4 directly applies, with no change, to an IO function $f$. □

Proposition 18 (Typing Satisfaction for Input-Output Functions). Let $T : \mathcal{P}(A_{in,out}) \rightarrow \mathcal{I}(\mathbb{R})$ be a typing and let $f : A_{in,out} \rightarrow \mathbb{R}^+$ be an IO function. Then $f$ satisfies $T$ iff

$$\{f(a_1), \ldots, f(a_{m+n})\} \in \text{Poly}(T)$$

i.e., the point determined by $f$ in the $(m+n)$-dimensional hyperspace is inside Poly$(T)$. By a slight abuse of notation, we may write “$f \in \text{Poly}(T)$” to indicate that $f$ satisfies this condition.

Proof. This readily follows from Definition 4 and the notions introduced earlier in this section. □

4.1 Uniqueness and Redundancy in Typings

We can view a typing $T$ as a syntactic expression, with its semantics Poly$(T)$ being a polytope in Euclidean hyperspace. As in other situations connecting syntax and semantics, there are generally distinct typings $T$ and $T'$ such that Poly$(T) = \text{Poly}(T')$. This is an obvious consequence of the fact that the same polytope can be defined by many different equivalent sets of linear inequalities, which is the source of some complications when we combine two typings to produce a new one.

If $T$ and $U$ are typings over $A_{in,out}$, we write $T \equiv U$ whenever Poly$(T) = \text{Poly}(U)$, in which case we say that $T$ and $U$ are equivalent.
**Definition 19 (Tight Typings).** Let $T$ be a typing over $A_{in,out}$. $T$ is tight if, for every $A \in \mathcal{P}(A_{in,out})$ for which $T(A)$ is defined and for every $r \in T(A)$, there is an IO function $f \in \text{Poly}(T)$ such that

$$r = \sum f(A \cap A_{in}) - \sum f(A \cap A_{out}).$$

Informally, $T$ is tight if none of the intervals/types assigned by $T$ to members of $\mathcal{P}(A_{in,out})$ contains redundant information.

Let $T$ be a typing over $A_{in,out}$. If $T(A)$ is defined for $A \subseteq A_{in,out}$, with $T(A) = [r_1, r_2]$ for some $r_1 \leq r_2$, we write $T_{\text{min}}(A)$ and $T_{\text{max}}(A)$ to denote the endpoints of $T(A)$:

$$T_{\text{min}}(A) = r_1 \quad \text{and} \quad T_{\text{max}}(A) = r_2.$$

The following is an easier-to-use characterization of tight typings.

**Proposition 20 (Equivalent Definition of Tight Typings).** Let $T$ be a typing over $A_{in,out}$. $T$ is tight iff, for every $A \subseteq A_{in,out}$ for which $T(A)$ is defined, there are $f_1, f_2 \in \text{Poly}(T)$ such that:

$$T_{\text{min}}(A) = \sum f_1(A \cap A_{in}) - \sum f_1(A \cap A_{out}),$$

$$T_{\text{max}}(A) = \sum f_2(A \cap A_{in}) - \sum f_2(A \cap A_{out}).$$

**Proof.** The left-to-right implication follows immediately from Definition 19. The right-to-left implication is a straightforward consequence of the linearity of the constraints that define $T$. 

**Proposition 21 (Every Typing Is Equivalent to a Tight Typing).** There is an algorithm $\text{Tight}(\cdot)$ which, given a typing $T$ as input, always terminates and returns an equivalent tight (and total) typing $\text{Tight}(T)$.

**Proof.** Starting from the given typing $T : \mathcal{P}(A_{in,out}) \rightarrow \mathcal{I}(\mathbb{R})$, we first determine the set of linear inequalities Constraints($T$) that defines Poly($T$), as given in (6) above. We compute a total and tight typing $T' : \mathcal{P}(A_{in,out}) \rightarrow \mathcal{I}(\mathbb{R})$ by assigning an appropriate interval/type $T'(A)$ to every $A \in \mathcal{P}(A_{in,out})$ as follows. For such a set $A$ of input/output arcs, let $\theta_A$ be the objective function: $\theta_A = \sum A \cap A_{in} - \sum A \cap A_{out}$. Relative to Constraints($T$), using standard procedures of linear programming, we minimize and maximize $\theta_A$ to obtain two values $r_1$ and $r_2$, respectively. The desired type $T'(A) = [r_1, r_2]$ and the desired $\text{Tight}(T)$ is $T'$.

### 4.2 Valid Typings and Principal Typings

Let $\mathcal{N}$ be a network with input arcs $A_{in}$ and output arcs $A_{out}$. A feasible flow $f : A \rightarrow \mathbb{R}^+$ in $\mathcal{N}$ is non-trivial if it is not identically zero on all arcs $a \in A$. (If $L(a) \neq 0$ for some input/output arc $a \in A_{in,out}$, then a feasible flow in $\mathcal{N}$ is automatically non-trivial.)

Let $T : \mathcal{P}(A_{in,out}) \rightarrow \mathcal{I}(\mathbb{R})$ be a typing over $A_{in,out}$ and $\mathcal{N}$ a network. We say $T$ is a valid typing for $\mathcal{N}$, sometimes denoted $(\mathcal{N} : T)$, if it is sound in the following sense:

**(soundness)\text{$_0$}** Every IO function $f : A_{in,out} \rightarrow \mathbb{R}^+$ satisfying $T$ can be extended to a feasible flow $g : A \rightarrow \mathbb{R}^+$.

This definition does not disqualify the following case: If $T$ is not satisfied by any $f : A_{in,out} \rightarrow \mathbb{R}^+$, then $T$ is automatically valid for $\mathcal{N}$ according to (soundness)\text{$_0$}). For some of the later results, it is more appropriate to use the following more restrictive definition of soundness:

**(soundness)\text{$_1$}** There is a non-trivial feasible flow $g : A \rightarrow \mathbb{R}^+$ whose IO restriction $f = [g]_{A_{in,out}}$ satisfies $T$. 

---

13
The definition according to (soundness) is that of a strongly valid typing. Nonetheless, for the sake of brevity in this report, we will understand validity of \( T \) according to (soundness\(_0\)).\(^4\) For a network \( \mathcal{N} \), we say the typing \((\mathcal{N} : T)\) is a principal typing if it is both sound and complete:

**(completeness)** Every feasible flow \( g : A \to \mathbb{R}^+ \) satisfies \( T \).

A useful notion in type theories is subtyping. Let \( T \) and \( U \) be valid typings over the same set of input/output arcs \( A_{\text{in}, \text{out}} \), i.e., \( T, U : \mathcal{P}(A_{\text{in}, \text{out}}) \to \mathcal{I}(\mathbb{R}) \). If \( T \) is a subtyping of \( U \), in symbols \( T <: U \), then any object for which \( T \) is a valid typing can be safely used in a context where an object with typing \( U \) is expected:

**(subtyping)** \( T <: U \) iff \( \text{Poly}(U) \subseteq \text{Poly}(T) \).

Note we require that both \( T \) and \( U \) be valid in order that “<:” work as expected, i.e., if \( \text{Poly}(U) \subseteq \text{Poly}(T) \) and \( T \) or \( U \) is not valid, then it is not necessarily that \( T <: U \).

Our subtyping relation is contravariant w.r.t. the subset relation, i.e., the supertyping \( U \) is more restrictive than the subtyping \( T \). From the definition of equivalence between typings in Section 4.1, if both \( T <: U \) and \( U <: T \), then \( T \equiv U \), naturally enough.

**Remark 22.** The notion of subtyping is fundamental in typing theories for strongly-typed programming languages. It defines formal conditions for the safe substitution of a component for another, i.e., without harming the behavior of the larger assembly in which components are inserted.

When components behave non-deterministically, the same substitution may or may not be safe, dependent on whether non-determinism is angelic or demonic. When values of some of the outer arcs are left unspecified, a network \( \mathcal{N} \)'s behavior is generally non-deterministic. For example, an assignment of values to \( A \)'s behavior is generally non-deterministic. For example, an assignment of values to the input, angelic non-determinism tries to make “good” choices in order to preserve safety, whereas demonic non-determinism is adversarial and tries to make “bad” choices in order to disrupt safety.

Our notion of subtyping, and with it the notion of safe substitution, works as expected if non-determinism is angelic. In Section 11.3, we re-visit the distinction between angelic and demonic, with an example illustrating issues that have to be further examined for our typing theory to support both kinds of non-determinism. \( \square \)

**Proposition 23 (Principal Typings Are Subtypings of Valid Typings).** If \((\mathcal{N} : T)\) is a principal typing, and \((\mathcal{N} : U)\) a valid typing for the same \( \mathcal{N} \), then \( T <: U \).

**Proof.** Given an arbitrary \( f : A_{\text{in}, \text{out}} \to \mathbb{R}^+ \), it suffices to show that if \( f \) satisfies \( T_2 \), then \( f \) satisfies \( T \), i.e., any point in \( \text{Poly}(U) \) is also in \( \text{Poly}(T) \). If \( f \) satisfies \( U \), then \( f \) can be extended to a feasible flow \( g \). Because \( T \) is principal, \( g \) satisfies \( T \). This implies that the restriction of \( g \) to \( A_{\text{in}, \text{out}} \), which is exactly \( f \), satisfies \( T \). \( \square \)

Any two principal typings \( T \) and \( U \) for the same network are not necessarily identical, but they always denote the same polytope, as formally stated in the next proposition. First, a lemma of more general interest.

**Lemma 24.** Let \((\mathcal{N} : T)\) and \((\mathcal{N} : T')\) be typings for the same \( \mathcal{N} \). If \( T \) and \( T' \) are total, tight, and \( \text{Poly}(T) = \text{Poly}(T') \), then \( T = T' \).

**Proof.** This follows from the construction in the proof of Proposition 21, where \( \text{Tight}(T) \) returns a typing which is both total and tight (and equivalent to \( T \)). \( \square \)

\(^4\)If \( T \) is valid according to (soundness\(_0\)) then it is valid according to (soundness\(_1\)), but not the other way around. To see this, write (soundness\(_0\)) equivalently as the disjunction:

\[
(\forall f : A_{\text{in}, \text{out}} \to \mathbb{R}^+) \left[ (f \text{ does not satisfy } T) \lor (\exists g : A \to \mathbb{R}^+) \left[ g \text{ feasible and } f \equiv [g]_{A_{\text{in}, \text{out}}} \right] \right]
\]
**Proposition 25** (Principal Typings Are Equivalent). If \((N : T)\) and \((N : U)\) are two principal typings for the same network \(N\), then \(T \equiv U\). Moreover, if \(T\) and \(U\) are tight and total, then \(T = U\).

**Proof.** Both \((N : T)\) and \((N : U)\) are valid. Hence, by Proposition 23, both \(T \prec U\) and \(U \prec T\). This implies that \(T \equiv U\). When \(T\) and \(U\) are uniformly tight, then the equality \(T = U\) follows from Lemma 24.

The next example illustrates several notions introduced earlier in this section, as well as points to issues we will examine carefully in later sections.

**Example 26.** The network \(N_1\) in Example 12 is simple enough that we can directly determine a tight, total, and principal typing \(T_1\) for it. By easy inspection, in addition to the type assignments:

\[
T_1(\varnothing) = T_1(\{a_1, a_2, a_3\}) = [0, 0],
\]

\(T_1\) makes six further assignments, shown below, one for every \(\varnothing \subsetneq A \subseteq \{a_1, a_2, a_3\}\). Note our conventions for writing these assignments, which we follow in later examples: input variables/arc names are listed positively, output variables/arc names are listed negatively. When \(r = s = t = 0\), a principal typing \(T_1\) for \(N_1\) makes the following type assignments:

\[
\begin{align*}
    a_1 & : [0, 15] & a_2 & : [0, 35] & -a_3 & : [-35, 0] \\
    a_1 + a_2 & : [0, 35] & a_1 - a_3 & : [-35, 0] & a_2 - a_3 & : [-15, 0]
\end{align*}
\]

When \(r = t = 0\) and \(s = 10\), a principal typing \(T_1\) for \(N_1\) makes the type assignments:

\[
\begin{align*}
    a_1 & : [0, 15] & a_2 & : [0, 35] & -a_3 & : [-35, -10] \\
    a_1 + a_2 & : [10, 35] & a_1 - a_3 & : [-35, 0] & a_2 - a_3 & : [-15, 0]
\end{align*}
\]

When \(s = t = 0\) and \(r = 5\), a principal typing \(T_1\) for \(N_1\) makes the type assignments:

\[
\begin{align*}
    a_1 & : [0, 15] & a_2 & : [5, 35] & -a_3 & : [-35, -5] \\
    a_1 + a_2 & : [5, 35] & a_1 - a_3 & : [-35, -5] & a_2 - a_3 & : [-15, 0]
\end{align*}
\]

The underlined assignments are those affected by the change of \(s\) from 0 to 10, or the change of \(r\) from 0 to 5. Figure 5 shows \(\text{Poly}(T_1)\) in all three cases.

There is considerable redundancy in \(T_1\) in all three cases, in that several of the type assignments can be omitted without changing \(\text{Poly}(T_1)\). For example, when \(r = s = t = 0\), a partial typing \(T'_1\) that makes only three assignments, instead of 8 by \(T_1\), is the following:

\[
\begin{align*}
    a_1 & : [0, 15] & -a_3 & : [-35, 0] & a_1 + a_2 - a_3 & : [0, 0]
\end{align*}
\]

which is equivalent to \(T_1\), i.e., \(\text{Poly}(T_1) = \text{Poly}(T'_1)\). To see this, consider the diagram on the left in Figure 5: The light-shaded area on the left is the same bounded convex surface defined by both \(T_1\) and \(T'_1\).

And there are other partial typings besides \(T'_1\) which are equivalent to \(T_1\) and make only three assignments. For example, \(T''_1\) given by:

\[
\begin{align*}
    a_1 & : [0, 15] & a_1 + a_2 & : [0, 35] & a_1 + a_2 - a_3 & : [0, 0]
\end{align*}
\]

is also equivalent to \(T_1\).

Figure 5 also shows the line \(a_1 = a_3\) in the two-dimensional \((a_1, a_3)\)-plane. This is the intersection of a vertical plane, call it \(P\) (not shown), containing the \(a_2\) axis with the horizontal \((a_1, a_3)\)-plane. \(P\) geometrically defines the requirement that the amount carried by arc \(a_1\) is equal to that carried by \(a_3\). This is a requirement we impose if we want to re-direct the flow out of arc \(a_3\) and back into arc \(a_1\). Figure 5 shows that this requirement can be satisfied when \(r = s = t = 0\), or when \(r = t = 0\) and \(s = 10\), but not when \(s = t = 0\) and \(r = 5\); in the first two cases, \(P\) intersects \(\text{Poly}(T_1)\), but in the third, \(P\) does not intersect \(\text{Poly}(T_1)\). \(\square\)
5 Flows Versus Path Assignments

In several places in later sections, we shift from a view of flows \( f : A \rightarrow \mathbb{R}^+ \) in a network \( N \) to an equivalent view of path assignments \( h : \Gamma \rightarrow \mathbb{R}^+ \), where \( \Gamma \) is the set of all full acyclic paths in \( N \).

**Definition 27** (Ordering the Arcs of a Network). Let the input arcs, output arcs, and internal arcs of a network \( N \) be indexed as follows:

\[
A_{\text{in}} = \{a_1, \ldots, a_m\}, \\
A_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\}, \\
A_{\#} = \{a_{m+n+1}, \ldots, a_{m+n+p}\}.
\]

As before, \( A = A_{\text{in}} \cup A_{\text{out}} \cup A_{\#} \) and we assume \( A_{\text{in}} \neq \emptyset \neq A_{\text{out}} \), though we do not exclude the possibility that \( A_{\#} = \emptyset \). We assume that the members of \( A \) are totally ordered, using \( "\prec" \) as the ordering relation, according to their indices from 1 to \( m + n + p \), i.e.,

\[
a_1 < \cdots < a_{m+1} < \cdots < a_{m+n+1} < \cdots < a_{m+n+p}.
\]

where we list all input arcs first, then all output arcs, and all internal arcs last.

**Definition 28** (Ordering the Good Paths). Let \( \Pi = \Gamma \cup \Delta \) be the set of good paths in a network \( N \) according to Definition 10. Relative to the ordering of arcs chosen in Definition 27, we totally order the members of \( \Pi \) lexicographically. Specifically, for all \( \pi_1, \pi_2 \in \Pi \) such that \( \pi_1 \neq \pi_2 \), we write \( \pi_1 \prec \pi_2 \) ("\( \pi_1 \) precedes \( \pi_2 \)"") if there are paths \( \pi_0, \pi'_1, \pi'_2 \) such that two conditions hold:

1. \( \pi_1 = \pi_0 \cdot \pi'_1 \) and \( \pi_2 = \pi_0 \cdot \pi'_2 \),
2. \( \text{first}(\pi'_1) \prec \text{first}(\pi'_2) \).

For this definition to make sense, we need to agree on a uniform way of listing the arcs of simple cycles. Let \( \delta = a_{i_1} a_{i_2} \cdots a_{i_k} \) be a simple cycle. Then every "shift" \( a_{i_j} \cdots a_{i_k} a_{i_1} \cdots a_{i_{j-1}} \) refers to the same simple cycle \( \delta \).

We agree that we always list the smallest-index arc first, i.e., \( i_1 < \min\{i_2, \ldots, i_k\} \).

Note that we use "\( \prec \)" to order only the good paths, not all paths. We impose this restriction because for an arbitrary path \( \pi \), if we decompose it as \( \pi = \gamma \oplus \delta_1 \oplus \cdots \oplus \delta_\ell \), the left-to-right listing of the arcs of a simple cycle \( \delta_i \), with \( 1 \leq i \leq \ell \), does not generally respect the ordering "\( \prec \)".

Note also that "\( \prec \)" places all full acyclic paths ahead of all simple cycles, i.e., all the members of \( \Gamma \) precede all the members of \( \Delta \).

**Lemma 29.** The good paths of a network \( N = (N, A) \) cover all its arcs, i.e., \( A = \cup \{\bar{\pi} | \pi \in \Pi\} \).
Proof. This easily follows from the definitions. Details omitted.

Example 30. We illustrate the notions introduced in Definitions 10, 27, and 28. Consider the network $N_2$ shown in Figure 2 with all its arcs named. As proposed in Definition 27, we choose to order the arcs according to their indices, i.e., $a_1 < a_2 < \cdots < a_{18}$, which in turn induces a total ordering on the full acyclic paths by Definition 28. We list the full acyclic paths in the ordering “$<$”, from left to right and from top to bottom, for a total of 20:

$$
\begin{align*}
\gamma_1 &= a_1 a_6 a_{12} a_4 & \gamma_2 &= a_1 a_6 a_{12} a_{16} a_5 & \gamma_3 &= a_1 a_7 a_{13} a_4 \\
\gamma_4 &= a_1 a_7 a_{13} a_{16} a_5 & \gamma_5 &= a_1 a_7 a_{14} a_4 & \gamma_6 &= a_1 a_7 a_{14} a_{17} a_{16} a_5 \\
\gamma_7 &= a_1 a_7 a_{14} a_{18} a_{11} a_5 & \gamma_8 &= a_2 a_8 a_{12} a_4 & \gamma_9 &= a_2 a_8 a_{12} a_{16} a_5 & \gamma_{10} &= a_2 a_9 a_{13} a_4 \\
\gamma_{11} &= a_2 a_9 a_{13} a_{16} a_5 & \gamma_{12} &= a_2 a_9 a_{14} a_4 & \gamma_{13} &= a_2 a_9 a_{14} a_{17} a_{16} a_5 \\
\gamma_{14} &= a_2 a_9 a_{14} a_{18} a_{11} a_5 & \gamma_{15} &= a_3 a_{10} a_{13} a_4 & \gamma_{16} &= a_3 a_{10} a_{13} a_{16} a_5 & \gamma_{17} &= a_3 a_{10} a_{14} a_{17} a_4 \\
\gamma_{18} &= a_3 a_{10} a_{14} a_{17} a_{16} a_5 & \gamma_{19} &= a_3 a_{11} a_5 & \gamma_{20} &= a_3 a_{11} a_{15} a_{17} a_4 \\
\end{align*}
$$

We have organized the indexing of the full acyclic paths so that $\gamma_1 < \gamma_2 < \cdots < \gamma_{20}$. These full acyclic paths are shown in Figure 6. In addition to the full acyclic paths, there are also the simple cycles:

$$
\begin{align*}
\delta_1 &= a_{10} a_{13} a_{16} a_{15} a_{18} & \delta_2 &= a_{10} a_{14} a_{18} & \delta_3 &= a_{11} a_{15} a_{18} & \delta_4 &= a_{15} a_{17} a_{16} \\
\end{align*}
$$

which are shown in Figure 7. In each of the 4 simple cycles, we start with the smallest-index arc. Using the ordering on the arcs, namely, $a_1 < a_2 < \cdots < a_{18}$, the ordering on the 4 cycles is:

$$
\delta_1 < \delta_2 < \delta_3 < \delta_4
$$

Any path $\pi$ in $N_2$ from input to output, which does not visit the same simple cycle more than once, can be obtained from a single $\gamma_i \in \{\gamma_1, \gamma_2, \ldots, \gamma_{20}\}$ together with at most one occurrence of each simple cycle in $\{\delta_1, \ldots, \delta_4\}$. For example, we can obtain such a path $\pi$ by combining $\gamma_1$ and $\delta_4$:

$$
\gamma_1 \oplus \delta_4 = a_1 a_6 a_{12} \underbrace{a_{16} a_{15} a_{17}}_{\delta_4} a_4
$$

We obtain another path from input to output without repeating occurrences of the same simple cycle by combining $\gamma_2$ and $\delta_4$:

$$
\gamma_2 \oplus \delta_4 = a_1 a_6 a_{12} \underbrace{a_{16} a_{15} a_{17}}_{\delta_4} a_{16} a_4
$$

And another one by combining $\gamma_7$ together with $\delta_1$ and $\delta_3$:

$$
\gamma_7 \oplus \delta_1 \oplus \delta_3 = a_1 a_7 a_{14} a_{18} \underbrace{a_{11} a_{15} a_{18}}_{\delta_1} \underbrace{a_{10} a_{13} a_{16} a_5}_{\delta_3}
$$

There are many other such combinations of a single full acyclic path together with several simple cycles to obtain a path from input to output without repeated occurrences of the same simple cycle. □
**Figure 6:** The full acyclic paths in the network $N_2$ of Figure 2.

**Definition 31 (Path Assignments).** Let $\Gamma$ be the set of full acyclic paths in $N$ according to Definitions 10 and 28. We use the letter $h$ (possibly decorated) to range over functions of the form $h: \Gamma \rightarrow \mathbb{R}^+$, in order to distinguish them from IO functions $f: A_{\text{in,out}} \rightarrow \mathbb{R}^+$ and from flows $g: A \rightarrow \mathbb{R}^+$. We reserve the letters $f$ and $g$ (possibly decorated) for the latter kinds of functions.

We call the function $h: \Gamma \rightarrow \mathbb{R}^+$ an *assignment of values* to paths, or simply a *path assignment*. A path assignment $h: \Gamma \rightarrow \mathbb{R}^+$ is *feasible* if for every $a \in A$:

$$\text{LC}(a) \leq \sum \{ h(\gamma) \mid a \in \gamma \} \leq \text{UC}(a)$$

These constraints enforce the condition that flow at every arc $a$ remains between its lower and upper bounds. We refer by $\mathcal{C}^*$ to the set of all such constraints, each consisting of two inequalities for each $a \in A$. The set $\mathcal{C}^*$ is implied or induced by the original $\mathcal{E} \cup \mathcal{C}$, from Definitions 1 and 2, and is the starting point of an alternative
Remark 32. According to Property (†) in Section 2.3 and Proposition 11, all feasible flows in $N$ can be restricted to flow along full acyclic paths. This justifies our decision in Definition 31 to restrict a path assignment $h$ to the set $\Gamma$ of full acyclic paths and not extend it to the set $\Delta$ of simple cycles. Equivalently, we can assume that $h$ is extended to $\Gamma \cup \Delta$ with $h(\delta) = 0$ for every $\delta \in \Delta$.

Definition 33 (From Flows to Path Assignments, Uniquely). Let $\Gamma = \{\gamma_1, \ldots, \gamma_u\}$, the set of full acyclic paths in $N$, which are indexed according to the ordering “$<$” in Definition 28, i.e., $\gamma_1 < \gamma_2 < \cdots < \gamma_u$.

Given a flow $g : A \to \mathbb{R}^+$ we uniquely define a path assignment $h : \Gamma \to \mathbb{R}^+$ in $u$ stages. We first define the value of $h(\gamma_1)$, and then proceed inductively to define the values of $h(\gamma_2), \ldots, h(\gamma_u)$:

$$h(\gamma_1) = \min \{ g(a) \mid a \in \gamma_1 \}$$

In words, $h(\gamma_1)$ is the maximum value that can be pushed along path $\gamma_1$ without exceeding $g(a)$ on any arc $a \in \gamma_1$. Proceeding inductively, we assume that $h(\gamma_1), h(\gamma_2), \ldots, h(\gamma_{k-1})$ are already defined for some $2 \leq k \leq u$, and we next define $h(\gamma_k)$ as follows:

$$h(\gamma_k) = \min \left\{ g(a) - \sum_{1 \leq i < k} h(\gamma_i) \mid 1 \leq i < k \text{ and } a \in \pi_i \right\}$$

In words, $h(\gamma_k)$ is the maximum value that can be pushed through, from arc $\text{first}(\gamma_k)$ to arc $\text{last}(\gamma_k)$, taking into account the values already pushed along paths $\gamma_1, \gamma_2, \ldots, \gamma_{k-1}$ by $h$.

Lemma 34. If the flow $g : A \to \mathbb{R}^+$ is feasible, then so is the path assignment $h : \Gamma \to \mathbb{R}^+$ induced by $g$ according to Definition 33.

Proof. This is straightforward from the definition. It is worth pointing out that, for every $1 \leq k \leq u$ in the construction of Definition 33, none of the upper bounds $\{ UC(a) \mid a \in \gamma_k \}$ is violated by $h(\gamma_k)$, since $g$ is feasible; however, before assigning values to the remaining paths $\gamma_{k+1}, \ldots, \gamma_u$, it may be that $h(\gamma_k)$ violates one of the lower bounds $\{ LC(a) \mid a \in \gamma_k \}$.
**Example 35.** We illustrate the inductive construction in Definition 33. Consider the following flow $g : A \to \mathbb{R}^+$ in $N_2$ of Examples 13 and 30:

| $g(a_1)$ | $g(a_2)$ | $g(a_3)$ | input arcs |
| 2        | 2        | 4        |
| $g(a_4)$ | $g(a_5)$ |           | output arcs |
| 3        | 5        |           |
| $g(a_6)$ | $g(a_7)$ | $g(a_8)$ |           | internal arcs |
| 2        | 0        | 0        |
| $g(a_{10})$ | $g(a_{11})$ | $g(a_{12})$ | $g(a_{13})$ | 3 |
| 1        | 3        | 2        | 3 |
| $g(a_{14})$ | $g(a_{15})$ | $g(a_{16})$ | $g(a_{17})$ | 0 |
| 0        | 0        | 2        | 0 |
| $g(a_{18})$ |            |           | 0 |

The path assignment $h$ induced by $g$ according to Definition 33 is:

$$h(\gamma_1) = 2 \quad h(\gamma_{10}) = 1 \quad h(\gamma_{11}) = 1 \quad h(\gamma_{16}) = 1 \quad h(\gamma_{19}) = 3$$

and $h(\gamma) = 0$ for all $\gamma \notin \{\gamma_1, \gamma_{10}, \gamma_{11}, \gamma_{16}, \gamma_{19}\}$. It is readily checked that both $g$ and $h$ are feasible. The full acyclic paths $\gamma_1, \gamma_{10}, \gamma_{11}, \gamma_{16}, \gamma_{19}$ are shown in Figure 6. 

**Definition 36 (From Path Assignments to Flows, Uniquely).** Let $\Gamma$ be the set of full acyclic paths in $N$ and $h : \Gamma \to \mathbb{R}^+$ a path assignment. There is a flow $g : A \to \mathbb{R}^+$ uniquely induced by $h$, namely:

$$g(a) = \sum \{ h(\gamma) \mid \gamma \in \Gamma \text{ and } a \in \gamma \}$$

for every $a \in A$. 

**Lemma 37.** If the path assignment $h : \Gamma \to \mathbb{R}^+$ is feasible, then so is the flow $g : A \to \mathbb{R}^+$ induced by $h$ according to Definition 36.

**Proof.** Immediate from the construction in Definition 36. 

**Notation 38.** In what follows we may name a function (e.g., a typing $T$ for flows, a typing $\mathcal{T}$ for path assignments, a set $\mathcal{C}$ of constraints, etc.) together with another function induced by the first (e.g., $T^*, \mathcal{T}^*, \mathcal{C}^*$, etc., respectively). Here, we use the superscript “*”, not as part of the name of the induced function, but as an operator from the original function to the induced function.

To make explicit the use of “*” as an operator, we can write $(T)^*, (\mathcal{T})^*, (\mathcal{C})^*$, etc., but we do not, for economy of notation. If we apply “*” twice, we can write $((T)^*)^*, ((\mathcal{T})^*)^*, ((\mathcal{C})^*)^*$, etc., to make “*” explicit as an operator, but again we prefer to write the more succinct $T^{**}, \mathcal{T}^{**}, \mathcal{C}^{**}$, etc.

**Example 39.** This continues the running example we started in Example 13, now shifting to a view of the full acyclic paths in the network $N_2$. In this view there is no counterpart to the set $\mathcal{E}_2$ of equations in Example 13, only a set $\mathcal{C}_2^*$ of inequalities induced by the original $\mathcal{C}_2$ (together with the original $\mathcal{E}_2$). From Example 30, there are 20 full acyclic paths in $N_2$, namely, $\Gamma = \{\gamma_1, \ldots, \gamma_{20}\}$, which we use as variables ranging over the coordinates of the 20-dimensional hyperspace $(\mathbb{R}^+)^{20}$. The set $\mathcal{C}_2^*$ consists of 18 inequalities, one for each of
the arcs in $\mathcal{N}_2$, together with the requirement that every path is assigned a non-negative value:

$$\mathcal{C}_2^* = \left\{ \begin{array}{l}
2 \leq \gamma_1 + \gamma_2 + \gamma_3 + \gamma_4 + \gamma_5 + \gamma_6 + \gamma_7 \leq 15, \\
0 \leq \gamma_8 + \gamma_9 + \gamma_{10} + \gamma_{11} + \gamma_{12} + \gamma_{13} + \gamma_{14} \leq 20, \\
4 \leq \gamma_{15} + \gamma_{16} + \gamma_{17} + \gamma_{18} + \gamma_{19} + \gamma_{20} \leq 25, \\
3 \leq \gamma_1 + \gamma_3 + \gamma_5 + \gamma_{10} + \gamma_{12} + \gamma_{15} + \gamma_{17} + \gamma_{20} \leq 8, \\
4 \leq \gamma_2 + \gamma_4 + \gamma_6 + \gamma_7 + \gamma_{11} + \gamma_{13} + \gamma_{14} + \gamma_{16} + \gamma_{18} + \gamma_{19} \leq 15, \\
0 \leq \gamma_1 + \gamma_2 \leq 5, \\
0 \leq \gamma_8 + \gamma_9 \leq 2, \\
0 \leq \gamma_{15} + \gamma_{16} + \gamma_{17} + \gamma_{18} \leq 10, \\
0 \leq \gamma_1 + \gamma_2 + \gamma_8 + \gamma_9 \leq 5, \\
0 \leq \gamma_5 + \gamma_6 + \gamma_7 + \gamma_{12} + \gamma_{13} + \gamma_{14} + \gamma_{17} + \gamma_{18} \leq 2, \\
0 \leq \gamma_2 + \gamma_4 + \gamma_6 + \gamma_9 + \gamma_{11} + \gamma_{13} + \gamma_{16} + \gamma_{18} \leq 10, \\
0 \leq \gamma_5 + \gamma_6 + \gamma_{12} + \gamma_{13} + \gamma_{17} + \gamma_{18} + \gamma_{20} \leq 7, \\
0 \leq \gamma_7 + \gamma_{14} \leq 6 \end{array} \right\}$$

at $a_1$

$$\bigcup \left\{ \gamma_i \geq 0 \mid 1 \leq i \leq 20 \right\}$$

A path assignment $h : \Gamma \to \mathbb{R}^+$ is feasible iff $h$ satisfies the inequalities in $\mathcal{C}_2^*$. Put differently, $h : \Gamma \to \mathbb{R}^+$ is feasible iff the “point” in the 20-dimensional hyperspace $(\mathbb{R}^+)^{20}$:

$$\{h(\gamma_1), \ldots, h(\gamma_{20})\}$$

is inside the polytope defined by the inequalities in $\mathcal{C}_2^*$.

5.1 Typings for Path-Assignments and Their Satisfaction

Let $\Gamma = \{\gamma_1, \ldots, \gamma_u\}$, the set of all full acyclic paths in the network $\mathcal{N}$. We use the $\gamma$’s as variables with values in $\mathbb{R}^+$. A typing $\mathcal{T}$ of path assignments is a partial function of the form:

$$\mathcal{T} : \mathcal{P}(\Gamma) \to \mathcal{I}(\mathbb{R}^+)$$

where $\mathcal{I}(\mathbb{R}^+)$ is the set of bounded closed intervals of non-negative reals.

**Definition 40 (Type Satisfaction, Again).** We extend type-satisfaction, as given in Definition 4 for flows and in Definition 17 for IO assignments, to path assignments. A path assignment $h : \Gamma \to \mathbb{R}^+$ satisfies the typing $\mathcal{T}$ if, for every $X \in \mathcal{P}(\Gamma)$ for which $\mathcal{T}(X)$ is defined and $\mathcal{T}(X)$ is the interval/type $[r_1, r_2]$, it holds that:

$$r_1 \leq \sum h(X) \leq r_2$$

Compare with the inequalities (3) in Definition 4, used again in Definition 17.

The notions of “valid” and “principal” typings for path assignments are obvious generalizations of the definitions at the beginning of Section 4.2. Specifically, we say a typing $\mathcal{T}$ for path assignments in $\mathcal{N}$ is valid if it is sound in the following sense:
Every feasible path assignment \( h : \Gamma \rightarrow \mathbb{R}^+ \) satisfying \( \mathcal{T} \) is feasible.

We follow \((\text{soundness})_0\) rather than the more restrictive \((\text{soundness})_1\) in Section 4.2.\(^5\) We say the typing \( \mathcal{T} \) is \emph{principal} if it is both sound and complete:

\textbf{(completeness)} Every feasible path assignment \( h : \Gamma \rightarrow \mathbb{R}^+ \) satisfies \( \mathcal{T} \).

In several places later we make use of the following lemma.

\textbf{Lemma 41.} Let \( \mathcal{N} \) be a network and \( \Gamma \) its set of full acyclic paths. For every \( X \in \mathcal{P}(\Gamma) \) we can effectively compute the quantity:

\[
\min \{ \sum X h(X) \mid h : \Gamma \rightarrow \mathbb{R}^+ \text{ is feasible} \}
\]

which we denote \( \min(X) \) for simplicity. By convention, if \( X = \emptyset \), we set \( \sum X h(X) = 0 = \min(X) \).

\textbf{Proof.} We identify the set of first arcs and last arcs of the paths in \( X \):

\[
A_1 = \{ \text{first}(\gamma) \mid \gamma \in X \} \quad \text{and} \quad A_2 = \{ \text{last}(\gamma) \mid \gamma \in X \}.
\]

We have \( A_1 \subseteq A_{\text{in}} \) and \( A_2 \subseteq A_{\text{out}} \). By Property (†) in Section 2.3, all lower bounds of internal arcs are zero. We must therefore have:

\[
\min(X) \leq \max \{ \sum LC(A_1), \sum LC(A_2) \}
\]

But the equality in the preceding line does not necessarily hold, because there may be some \( \gamma \in (\Gamma - X) \), with \( \text{first}(\gamma) \in A_1 \) or \( \text{last}(\gamma) \in A_2 \), such that \( h(\gamma) \neq 0 \) for some feasible path assignment \( h \), allowing \( \min(X) \) to be strictly less than \( \max\{ \sum LC(A_1), \sum LC(A_2) \} \).

Instead, the precise way to compute \( \min(X) \) proceeds differently, according to two different approaches.

The first approach uses linear programming, with the set \( \mathcal{C}^* \) of linear constraints in Definition 31 and the objective function \( \theta_X = \sum X \). The desired value \( \min(X) \) is obtained by minimizing \( \theta_X \) relative to the set \( \mathcal{C}^* \) of linear constraints.\(^6\)

The second approach works directly on the network \( \mathcal{N} \) and proceeds in stages, starting with the zero-value path-assignment \( h_0 \), i.e., \( \sum h_0(\Gamma) = 0 \). If \( h_0 \) is already feasible, then \( \min(X) = 0 \). Otherwise, if \( h_0 \) is not feasible, we proceed by identifying a succession of minimum-value “augmenting paths” in \((\Gamma - X)\) to define successive path-assignments \( h_1, h_2, \ldots \) until we find one \( h_k \) which is feasible, in which case \( \min(X) = 0 \) again. If no such feasible \( h_k \) can be found by using minimum-value augmenting paths in \((\Gamma - X)\), we continue the process using minimum-value augmenting paths in \( X \). If there are feasible flows in \( \mathcal{N} \), and therefore feasible path-assignments in \( \mathcal{N} \), this process is bound to terminate and return a feasible path-assignment \( h_\ell \) at which point we set \( \min(X) = h_\ell(X) \).

\( \square \)

### 5.2 From Typings for Flows to Typings for Path Assignments – and Back

In Lemmas 42 and 43, we map sets of input/output arcs to sets of full acyclic paths. For an arbitrary set \( A \subseteq A_{\text{in/out}} \), with \( A_1 = A \cap A_{\text{in}} \) and \( A_2 = A \cap A_{\text{out}} \), we define the set \( \text{Paths}(A_1, A_2) \) of full acyclic paths from input arcs \( A_1 \) to output arcs \( A_2 \):

\[
\text{Paths}(A_1, A_2) = \{ \gamma \in \Gamma \mid \text{first}(\gamma) \in A_1 \text{ and } \text{last}(\gamma) \in A_2 \}
\]

\(^5\)Strong validity of a typing \( \mathcal{F} \) for path assignments is according to \((\text{soundness})_1\) which is: There is a non-trivial feasible path assignment \( h : \Gamma \rightarrow \mathbb{R}^+ \) which satisfies \( \mathcal{F} \).

\(^6\)This is a case of the \textit{network linear programming} problem, which is still more restricted here because all coefficients are +1, for which there are faster implementations. But we do not need this fact for the proof.
In the two lemmas below, we use this definition to distinguish two subsets of $\Gamma$:

$$X_A^+ = \text{Paths}(A_1, A_{\text{out}} - A_2)$$
$$X_A^- = \text{Paths}(A_{\text{in}} - A_1, A_2)$$

By this definition, the two sets are pairwise disjoint:

$$X_A^+ \cap X_A^- = \emptyset$$

One special case is $X_A^+ = X_A^- = \emptyset$, which happens when $A = A_{\text{in, out}}$ for example. Another special case is $X_A^+ = \emptyset$ and $X_A^- \neq \emptyset$, or $X_A^+ \neq \emptyset$ and $X_A^- = \emptyset$, which happens when $A \cap A_{\text{in}} = \emptyset$ or $A \cap A_{\text{out}} = \emptyset$, respectively.

**Lemma 42.** Let $\mathcal{N}$ be a network, $\Gamma$ its set of full acyclic paths, and $T : \mathcal{P}(A_{\text{in, out}}) \rightarrow I(\mathbb{R})$ a typing for the flows in $\mathcal{N}$ (which may or may not be tight and/or total and/or principal). The typing $T$ for flows induces a typing $T^* : \mathcal{P}(\Gamma) \rightarrow I(\mathbb{R}^+)$ for path assignments as follows. For every $A \in \mathcal{P}(A_{\text{in, out}})$, if $T(A)$ is defined and $T(A) = [r_1, r_2]$, we define the linear constraint $C_A$ on paths/variables:

$$C_A : \quad r_1 \leq \sum X_A^+ - \sum X_A^- \leq r_2$$

For every $Y \in \mathcal{P}(\Gamma)$ we define the objective function $\theta_Y$ by:

$$\theta_Y = \sum Y.$$ 

We define the set Constraints$(T)$ of linear constraints over the set $\Gamma$ of paths/variables by:

$$\text{Constraints}(T) = \{ C_A \mid A \in \mathcal{P}(A_{\text{in, out}}) \}.$$ 

Relative to Constraints$(T)$, for every $Y \in \mathcal{P}(\Gamma)$, let the minimum and the maximum of the objective function $\theta_Y$ be $s_{Y,1}$ and $s_{Y,2}$, respectively. For every $Y \in \mathcal{P}(\Gamma)$, let $T^*(Y)$ be the type:

$$T^*(Y) = [s_{Y,1}, s_{Y,2}]$$

**Conclusion:** The typing $T^*$ for path assignments thus induced by $T$ satisfies the following properties:

1. $T^*$ is tight and total.
2. $\text{Poly}(T^*) = \text{Poly}(\text{Constraints}(T))$.
3. $T^*$ is principal for path assignments iff $T$ is principal for flows.

**Proof.** Part 1, that $T^*$ is tight and total, is immediate from the construction. Part 2 is straightforward from the definitions earlier in this section. Part 3 is a consequence of part 2. All details omitted. \(\square\)

**Lemma 43.** Let $\mathcal{N}$ be a network, $\Gamma$ its set of full acyclic paths, and $\mathcal{T} : \mathcal{P}(\Gamma) \rightarrow I(\mathbb{R}^+)$ a typing for its path assignments which is principal. The typing $\mathcal{T}$ for path assignments induces a typing $\mathcal{T}^* : \mathcal{P}(A_{\text{in, out}}) \rightarrow I(\mathbb{R})$ for flows as follows. For every $Y \in \mathcal{P}(\Gamma)$, if $\mathcal{T}(Y)$ is defined and $\mathcal{T}(Y) = [r_1, r_2]$, we define the linear constraint $C_Y$ on paths-cum-variables:

$$C_Y : \quad r_1 \leq \sum Y \leq r_2$$

For every $A \in \mathcal{P}(A_{\text{in, out}})$ we define the objective function $\theta_A$ by:

$$\theta_A = \sum \{ \gamma \mid \gamma \in X_A^+ \} - \sum \{ \gamma \mid \gamma \in X_A^- \}.$$ 

\(^7\)Note the difference with the hypothesis of Lemma 42.
We define the set Constraints(\mathcal{I}) of linear constraints over the paths/variables \Gamma by:

\text{Constraints}(\mathcal{I}) = \{ C_Y \mid Y \in \mathcal{P}(\Gamma) \}.

Relative to Constraints(\mathcal{I}), for every \( A \in \mathcal{P}(A_{\text{in, out}}) \), let the minimum and maximum of the objective function \( \theta_A \) be \( s_{A,1} \) and \( s_{A,2} \), respectively. For every \( A \in \mathcal{P}(A_{\text{in, out}}) \), let \( \mathcal{T}^*(A) \) be the type:

\[ \mathcal{T}^*(A) = [s_{A,1}, s_{A,2}] \]

**Conclusion:** The typing \( \mathcal{T}^* \) for flows thus induced by \( \mathcal{I} \) is tight, total, and principal.

**Proof.** As in the proof of Lemma 42, that \( \mathcal{T}^* \) is tight and total is immediate from the construction. The remaining part of the conclusion is also straightforward from the definitions earlier in this section. All details omitted.

**Theorem 44** (From Flows to Path Assignments and Back). Let \( \mathcal{N} \) be a network, \( \Gamma \) its set of full acyclic paths, \( T : \mathcal{P}(A_{\text{in, out}}) \rightarrow I(\mathbb{R}^+) \) a tight, total, and principal typing for its flows, and \( \mathcal{I} : \mathcal{P}(\Gamma) \rightarrow I(\mathbb{R}^+) \) a tight, total, and principal typing for its path assignments.

Let \( \mathcal{I}' : \mathcal{P}(\Gamma) \rightarrow I(\mathbb{R}^+) \) be the typing for path assignments induced by \( T \) according to Lemma 42, and let \( \mathcal{T}^* : \mathcal{P}(A_{\text{in, out}}) \rightarrow I(\mathbb{R}) \) be the typing for flows induced by \( \mathcal{I} \) according to Lemma 43. We then have:

1. \( \mathcal{T}^* = \mathcal{I} \).
2. \( \mathcal{T}^* = \mathcal{T} \).

**Proof.** Both parts in the conclusion follow from Lemmas 42 and 43, and from the uniqueness of principal typings when they are tight and total, whether for flows or for path assignments. 

### 6 Inferring Typings that Are Total, Tight, and Principal

Let \( A = A_{\#} \cup A_{\text{in}} \cup A_{\text{out}} \) be the set of arcs in a network \( \mathcal{N} \). As in Section 4, we use the arc names in \( A \) as variables to which we assign values in \( \mathbb{R}^+ \).

**Procedure 45** (How To Compute Principal Typings). Let \( \mathcal{E} \) be the collection of all equations enforcing flow conservation, and \( \mathcal{C} \) the collection of all inequalities enforcing capacity constraints, in \( \mathcal{N} \).

We define the total typing \( T : \mathcal{P}(A_{\text{in, out}}) \rightarrow I(\mathbb{R}) \) as follows. For every \( A \in \mathcal{P}(A_{\text{in, out}}) \), relative to the equations and inequalities in \( \mathcal{E} \cup \mathcal{C} \), we use linear programming to minimize and maximize the same objective function:

\[
\theta_A = \sum \{ a \mid a \in A \cap A_{\text{in}} \} - \sum \{ a \mid a \in A \cap A_{\text{out}} \}
\]

Relative to \( \mathcal{E} \cup \mathcal{C} \), the determination of the type/interval assigned to \( T(A) \) is in three steps:

1. Compute the minimum possible value \( r_1 \in \mathbb{R} \) for the objective \( \theta_A \).
2. Compute the maximum possible value \( r_2 \in \mathbb{R} \) for the objective \( \theta_A \).
3. Assign to \( T(A) \) the interval \([r_1, r_2]\).

Theorem 54 confirms that the total typing \( T \) as just defined is also tight and principal for \( \mathcal{N} \).
Example 46. We compute a total typing $T_2$ for the network $\mathcal{N}_2$ shown in Figure 2 according to Procedure 45. We can either use linear programming to compute every interval/type $T_2(A)$ – as proposed in Procedure 45 – or, because $\mathcal{N}_2$ is fairly small, compute $T_2(A)$ by brute-force inspection, though very tediously. Following the conventions in Section 4, we use arc names as variables. Hence, for every $A \in \mathcal{P}(A_{\text{in, out}})$, we write:

$$\sum (A \cap A_{\text{in}}) - \sum (A \cap A_{\text{out}}): [r_1, r_2]$$

instead of $T_2(A) = [r_1, r_2]$. The assignment of types by $T_2$ are $T_2(\emptyset) = [0, 0]$ and for every $A \neq \emptyset$:

$$a_1: [2, 10] \quad a_2: [0, 7] \quad a_3: [4, 9] \quad a_4: [-8, -3]$$

$$-a_5: [-11, -4] \quad a_1 + a_2: [2, 10] \quad a_1 + a_3: [6, 14] \quad a_1 - a_4: [-6, 7]$$

$$a_1 - a_5: [-8, 4] \quad a_2 + a_3: [4, 11] \quad a_2 - a_4: [-8, 4] \quad a_2 - a_5: [-11, 2]$$

$$a_3 - a_4: [-4, 6] \quad a_3 - a_5: [-7, 5] \quad -a_4 - a_5: [-14, -7] \quad a_1 + a_2 + a_3: [7, 14]$$

$$a_1 + a_2 - a_4: [-5, 7] \quad a_1 + a_2 - a_5: [-6, 4] \quad a_1 + a_3 - a_4: [-2, 11] \quad a_1 + a_3 - a_5: [-4, 8]$$

$$a_1 - a_4 - a_5: [-11, -4] \quad a_2 + a_3 - a_4: [-4, 8] \quad a_2 + a_5 - a_4: [-7, 6] \quad a_2 - a_4 - a_5: [-14, -6]$$

$$a_3 - a_4 - a_5: [-10, -2] \quad a_1 + a_2 + a_3 - a_4: [4, 11] \quad a_1 + a_2 + a_3 - a_5: [3, 8] \quad a_1 + a_2 - a_4 - a_5: [-9, -4]$$

$$a_1 + a_3 - a_4 - a_5: [-7, 0] \quad a_2 + a_3 - a_4 - a_5: [-10, -2] \quad a_1 + a_2 + a_3 - a_4 - a_5: [0, 0]$$

As expected, the types assigned to $A_{\text{in}} = \{a_1, a_2, a_3\}$ and $A_{\text{out}} = \{a_4, a_5\}$ are the negations of each other, namely $[7, 14]$ and $[-14, -7]$, and demarcate the interval of feasible flows from a minimum of 7 to a maximum of 14. According to Theorem 54, the total typing $T_2$ as just defined is tight and principal for $\mathcal{N}_2$. □

Example 47. We use Procedure 45 to infer a tight, total, and principal typing $T_3$ for the network $\mathcal{N}_3$ in Example 14. In addition to $T_3(\emptyset) = [0, 0]$, $T_3$ makes the following assignments:

$$a_1: [0, 15] \quad a_2: [0, 25] \quad a_3: [-15, 0] \quad a_4: [-25, 0]$$

$$a_1 + a_2: [0, 30] \quad a_1 - a_3: [-10, 10] \quad a_1 - a_4: [-25, 15]$$

$$a_2 - a_3: [-15, 25] \quad a_2 - a_4: [-10, 10] \quad -a_3 - a_4: [-30, 0]$$

$$a_1 + a_2 - a_3: [0, 25] \quad a_1 + a_2 - a_4: [0, 15] \quad a_1 - a_3 - a_4: [-25, 0] \quad a_2 - a_3 - a_4: [-15, 0]$$

$$a_1 + a_2 - a_3 - a_4: [0, 0]$$

The boxed type assignments and the underlined type assignments are for purposes of comparison with the typing $T_4$ in Example 48. □

Example 48. We use Procedure 45 to infer a tight, total, and principal typing $T_4$ for the network $\mathcal{N}_4$ in Example 15. In addition to $T_4(\emptyset) = [0, 0]$, $T_4$ makes the following assignments:

$$a_1: [0, 15] \quad a_2: [0, 25] \quad a_3: [-15, 0] \quad a_4: [-25, 0]$$

$$a_1 + a_2: [0, 30] \quad a_1 - a_3: [-10, 12] \quad a_1 - a_4: [-23, 15]$$

$$a_2 - a_3: [-15, 23] \quad a_2 - a_4: [-12, 10] \quad -a_3 - a_4: [-30, 0]$$

$$a_1 + a_2 - a_3: [0, 25] \quad a_1 + a_2 - a_4: [0, 15] \quad a_1 - a_3 - a_4: [-25, 0] \quad a_2 - a_3 - a_4: [-15, 0]$$

$$a_1 + a_2 - a_3 - a_4: [0, 0]$$

In this example and the preceding one, the type assignments in rectangular boxes are for subsets of the input arcs $\{a_1, a_2\}$, and for subsets of the output arcs $\{a_3, a_4\}$, but not for subsets mixing input arcs and output arcs. Note that these are the same for the typing $T_3$ in Example 47 and the typing $T_4$ in this Example 48.
The underlined type assignments are among those that mix input and output arcs. We underline those in Example 47 that are different from the corresponding ones in this Example 48. This difference implies there are IO functions \( f : \{a_1, a_2, a_3, a_4\} \rightarrow \mathbb{R} \) which can be extended to feasible flows in \( \mathcal{N}_3 \) (resp. in \( \mathcal{N}_4 \)) but not in \( \mathcal{N}_3 \) (resp. in \( \mathcal{N}_4 \)). This is perhaps counter-intuitive, since \( T_3 \) and \( T_4 \) make exactly the same type assignments to input arcs and, separately, output arcs (the boxed assignments). For example, the IO function \( f \) defined by:

\[
\begin{align*}
    f(a_1) &= 15 \\
    f(a_2) &= 0 \\
    f(a_3) &= 3 \\
    f(a_4) &= 12
\end{align*}
\]

is readily checked to be extendible to a feasible flow in \( \mathcal{N}_4 \) but not in \( \mathcal{N}_3 \). The reason is that \( f(a_1) - f(a_3) = 12 \) violates (i.e., is outside) the type \( T_3(\{a_1, a_3\}) = [-10, 10] \). Similarly, the IO function \( f \) defined by:

\[
\begin{align*}
    f(a_0) &= 0 \\
    f(a_2) &= 25 \\
    f(a_3) &= 0 \\
    f(a_4) &= 25
\end{align*}
\]

can be extended to a feasible flow in \( \mathcal{N}_3 \) but not in \( \mathcal{N}_4 \), the reason being that \( f(a_2) - f(a_3) = 25 \) violates the type \( T_4(\{a_2, a_3\}) = [-15, 23] \).

For all the forementioned reasons, neither \( T_3 \) nor \( T_4 \) is a subtyping of the other, in the sense explained in Section 4.2. However, the “meet” (to be formally defined in Section 10) of \( T_3 \) and \( T_4 \), denoted \( T_3 \land T_4 \), obtained by intersecting types assigned to the same subsets of \( \{a_1, a_2, a_3, a_4\} \), will turn out to be a valid (necessarily not principal) typing for both \( \mathcal{N}_3 \) and \( \mathcal{N}_4 \). What is more, by the results of Section 7 and later, \( T_3 \land T_4 \) is a tight, total, and principal, typing for some other network (necessarily distinct from \( \mathcal{N}_3 \) and \( \mathcal{N}_4 \)).

**Procedure 49 (How To Compute Principal Typings, Again).** An alternative procedure is to use the inequalities \( \mathcal{C}^* \) that enforce capacity constraints on path assignments, as in Definition 31.

All coefficients in \( \mathcal{C}^* \) are +1 and there are no equations enforcing flow-conservation when we deal with path assignments. Hence, in contrast to the situation in Procedure 45, we can directly extract from \( \mathcal{C}^* \) a typing \( \mathcal{T}_0 \) for path assignments, with no need for further computation. Specifically, if \( \mathcal{C}^* \) contains the two inequalities:

\[
s_1 \leq \sum \{ \gamma \mid \gamma \in X \} \leq s_2
\]

for some \( \emptyset \neq X \subseteq \Gamma \) and \( 0 \leq s_1 < s_2 \), then we set \( \mathcal{T}_0(X) = [s_1, s_2] \). However, the resulting typing \( \mathcal{T}_0 \) is generally not total and not tight (but it can be shown that \( \mathcal{T}_0 \) is principal).

To define a typing \( \mathcal{T} : \mathcal{P}(\Gamma) \rightarrow \mathcal{I}(\mathbb{R}) \) which is total, tight, and principal, we proceed as follows. We first pose \( \mathcal{T}(\emptyset) = [0, 0] \). And for every \( \emptyset \neq X \subseteq \Gamma \), relative to the inequalities in \( \mathcal{C}^* \), we use linear programming to minimimize and maximize the same objective function:

\[
\theta_X = \sum \{ \gamma \mid \gamma \in X \}
\]

Relative to \( \mathcal{C}^* \), the determination of the type/interval assigned to \( \mathcal{T}(X) \) is in three steps:

1. Compute the minimum possible value \( r_1 \in \mathbb{R} \) for the objective \( \theta_X \).
2. Compute the maximum possible value \( r_2 \in \mathbb{R} \) for the objective \( \theta_X \).
3. Assign to \( \mathcal{T}(X) \) the interval \([r_1, r_2] \).

In practice, we are interested in inferring a typing for flows. So, we first compute the typing \( \mathcal{T} \) for path assignments as outlined above, and then return the typing \( \mathcal{T}^* \) for flows induced by \( \mathcal{T} \), the latter according to the procedure in the statement of Lemma 43. Theorem 54 confirms that the total typing \( \mathcal{T} \) as just defined is also tight and principal for \( \mathcal{N} \).

**Lemma 50.** Let \( T \) be the typing for flows returned by Procedure 45 and \( \mathcal{T} \) the typing for path assignments returned by Procedure 49. If \( T^* \) is the typing for path assignments induced by \( T \) according to Lemma 42, then \( T^* = \mathcal{T} \).

26
Proof. All of the typings under consideration \( (T, T^*, \text{ and } \mathcal{T}) \) are tight and total by construction. That \( T^* = \mathcal{T} \) is a straightforward consequence of how the two typings are produced: \( T \) by Procedure 45 using \( \mathcal{E} \cup \mathcal{C} \), and \( \mathcal{T} \) by Procedure 49 using the inequalities in \( \mathcal{C}^* \) induced by \( \mathcal{E} \cup \mathcal{C} \). All details omitted.

Example 51. We illustrate the computation according to Procedure 49 for the network \( N_2 \) shown in Figure 2.

We compute \( \mathcal{T}(Y) \) for a few \( Y \subseteq \Gamma \):

\[
\mathcal{T}(\{\gamma_5\}) = \mathcal{T}(\{\gamma_6\}) = \mathcal{T}(\{\gamma_7\}) = \mathcal{T}(\{\gamma_8\}) = \mathcal{T}(\{\gamma_9\}) = \mathcal{T}(\{\gamma_{12}\}) = \mathcal{T}(\{\gamma_{13}\}) = [0, 2] \\
\mathcal{T}(\{\gamma_{14}\}) = \mathcal{T}(\{\gamma_{17}\}) = \mathcal{T}(\{\gamma_{18}\}) = [0, 2] \\
\mathcal{T}(\{\gamma_3\}) = \mathcal{T}(\{\gamma_4\}) = \mathcal{T}(\{\gamma_{10}\}) = \mathcal{T}(\{\gamma_{11}\}) = \mathcal{T}(\{\gamma_{15}\}) = \mathcal{T}(\{\gamma_{16}\}) = \mathcal{T}(\{\gamma_{20}\}) = [0, 3] \\
\mathcal{T}(\{\gamma_{19}\}) = [0, 4] \\
\mathcal{T}(\{\gamma_{1}\}) = \mathcal{T}(\{\gamma_{2}\}) = [0, 5]
\]

The sets \( X^+_A, X^-_A \subseteq \Gamma \) are defined at the beginning of Section 5.2. If \([r_1, r_2]\) is an interval/type, we write \( -[r_2, r_1] \) for \([-r_2, -r_1]\). When \( Y = X^+_A \) and \( X^-_A = \emptyset \), or \( X^+_A = \emptyset \) and \( Y = X^-_A \), for \( A \in \mathcal{P}(\mathcal{A}_{in,out}) \), we can directly compare \( \mathcal{T}(Y) \) and \( T(A) \) in Example 46 (they are equal!):

\[
\mathcal{T}(\{\gamma \in \Gamma \mid a_1 = first(\gamma)\}) = [2, 10] \quad \text{same as } T(\{a_1\}) \\
\mathcal{T}(\{\gamma \in \Gamma \mid a_2 = first(\gamma)\}) = [0, 7] \quad \text{same as } T(\{a_2\}) \\
\mathcal{T}(\{\gamma \in \Gamma \mid a_3 = first(\gamma)\}) = [4, 9] \quad \text{same as } T(\{a_3\}) \\
\mathcal{T}(\{\gamma \in \Gamma \mid a_4 = last(\gamma)\}) = [3, 8] \quad \text{same as } -T(\{a_4\}) \\
\mathcal{T}(\{\gamma \in \Gamma \mid a_4 = last(\gamma)\}) = [4, 11] \quad \text{same as } -T(\{a_5\}) \\
\mathcal{T}(\Gamma) = [7, 14] \quad \text{same as } T(\mathcal{A}_{in}) = -T(\mathcal{A}_{out})
\]

Although we can directly extract a typing \( \mathcal{T}_0 \) from \( \mathcal{C}^* \) as pointed in Procedure 49, \( \mathcal{T} \)'s types are tighter than those of \( \mathcal{T}_0 \). For example, if \( Y = \{\gamma \in \Gamma \mid a_1 = first(\gamma)\} \), then \( \mathcal{T}(Y) = [2, 10] \nsubseteq [2, 15] = \mathcal{T}_0(Y) \).

For a non-trivial \( Y \subseteq \Gamma \), corresponding to some \( A \in \mathcal{P}(\mathcal{A}_{in,out}) \) such that \( X^+_A = \emptyset \) and \( X^-_A \neq \emptyset \), consider \( A = \{a_2, a_4, a_5\} \). For such \( A \), we have \( A \cap \mathcal{A}_{in} = \{a_2\} \) and \( A \cap \mathcal{A}_{out} = \{a_4, a_5\} \), so that:

\[
X^+_A = \emptyset \quad \text{and} \quad X^-_A = \{\gamma_1, \ldots, \gamma_7\} \cup \{\gamma_{15}, \ldots, \gamma_{20}\}.
\]

In this case, we obtain:

\[
\mathcal{T}(\{\gamma_1, \ldots, \gamma_7\} \cup \{\gamma_{15}, \ldots, \gamma_{20}\}) = [6, 14] \quad \text{same as } -T(\{a_2, a_4, a_5\})
\]

The comparison is less trivial for \( A \) such that both \( X^+_A \neq \emptyset \neq X^-_A \). For example, if we take \( A = \{a_1, a_3, a_5\} \), so that \( A \cap \mathcal{A}_{in} = \{a_1, a_3\} \) and \( A \cap \mathcal{A}_{out} = \{a_5\} \), we obtain:

\[
X^+_A = \{\gamma_1, \gamma_3, \gamma_5, \gamma_{15}, \gamma_{17}, \gamma_{20}\} \quad \text{and} \quad X^-_A = \{\gamma_{91}, \gamma_{11}, \gamma_{13}, \gamma_{14}\}.
\]

The determination of \( \mathcal{T}^*(A) \) according to Lemma 43 produces \( \mathcal{T}^*(A) = [-4, 8] = T(A) \), as expected.

In Procedure 49, we use linear programming to compute a typing \( \mathcal{T} : \mathcal{P}(\Gamma) \rightarrow \mathcal{I}(\mathbb{R}^+) \) for path assignments which is total, tight, and principal. In some situations, we can compute such a typing \( \mathcal{T} \) more
directly, without invoking linear programming. As explained in Procedure 52 below, this happens when we know \( \min(X) \) for every \( X \) such that \( \sum X \) appears as a proper subsum in one of the constraints in \( C^* \).

We can compute \( \min(X) \) for arbitrary \( X \in P(\Gamma) \) using Lemma 41. But such a computation is unnecessary if, for example, \( LC(a) = 0 \) for every \( a \in A \), in which case \( \min(X) = 0 \) for every \( X \in P(\Gamma) \). Or, if there is a fixed value \( r \in \mathbb{R}^+ \) such that \( \min(\{\gamma\}) = r \) for every \( \gamma \in \Gamma \), in which case \( \min(X) = |X| \cdot r \) for every \( X \in P(\Gamma) \).

Procedure 52 (How To Compute Principal Typings, Again and Faster). Starting from the set \( C^* \) of constraints in Definition 31, let \( \mathcal{Y} \subseteq P(\Gamma) \) be the collection of all non-empty \( Y \in P(\Gamma) \) such that \( \sum Y \) appears as a proper subsum in one of the constraints; that is, we include \( Y \) in \( \mathcal{Y} \) if there is a constraint in \( C^* \) of the form:

\[
s \leq \sum \{ \gamma \mid \gamma \in X \} \leq s'
\]

for some \( \emptyset \neq X \subseteq \Gamma \) and \( 0 \leq s < s' \) such that \( Y \) is a proper subset of \( X \). The procedure below can be used if we already know the value of \( \min(Y) \) for every \( Y \in \mathcal{Y} \), which we now assume.

We compute a typing \( T : P(\Gamma) \to I(\mathbb{R}^+) \) which is total and tight – and later shown principal – in two phases. Phase one is to compute a partial typing \( T_0 : P(\Gamma) \to I(\mathbb{R}^+) \) as follows. Given an arbitrary \( \emptyset \neq X \subseteq \Gamma \), we list all constraints in \( C^* \) where \( \sum X \) appears as a proper sum:

\[
s_1 \leq \sum X + \sum Y_1 \leq s'_1 \\
\vdots \\
s_k \leq \sum X + \sum Y_k \leq s'_k
\]

where \( 0 \leq s_i < s'_i \) and \( Y_i \in \mathcal{Y} \) for every \( 1 \leq i \leq k \). If \( k = 0 \) and no such constraint in \( C^* \) exists, we leave \( T_0(X) \) undefined. Otherwise, we define the new constraints:

\[
t_1 = s_1 - r_1 \leq \sum X \leq s'_1 - r_1 = t'_1 \\
\vdots \\
t_k = s_k - r_k \leq \sum X \leq s'_k - r_k = t'_k
\]

where \( r_1 = \min(Y_1), \ldots, r_k = \min(Y_k) \) and the “monus” operation is defined as:

\[
s - r = \begin{cases} 
  s - r & \text{if } s \geq r, \\
  0 & \text{if } s < r.
\end{cases}
\]

Finally, we define the type/interval \( T_0(X) \):

\[
T_0(X) = [\max\{t_1, \ldots, t_k\}, \min\{t'_1, \ldots, t'_k\}]
\]

Phase two consists in computing a total typing \( T_1 : P(\Gamma) \to I(\mathbb{R}^+) \) that extends \( T_0 \).

Lemma 53. Let \( T \) be the typing for path assignments returned by Procedure 49 and \( T_1 \) the typing for path assignments returned by Procedure 52. Then \( T = T_1 \).

Proof. Straightforward from the constructions in Procedure 49 and Procedure 52. The two procedures are minor variations of each other, with the former making explicit its use of linear programming.

Theorem 54 (Inferring Total, Tight, and Principal Typings).

1. The typing \( T : P(A_{\text{in,out}}) \to I(\mathbb{R}) \) returned by Procedure 45 is total, tight and principal for flows in network \( \mathcal{N} \).
2. The typing $\mathcal{T} : \mathcal{P}(\Gamma) \to \mathcal{I}(\mathbb{R}^+)$ returned by Procedure 49 is total, tight and principal for path assignments in network $\mathcal{N}$.

3. The typing $\mathcal{T}_1 : \mathcal{P}(\Gamma) \to \mathcal{I}(\mathbb{R}^+)$ returned by Procedure 52 is total, tight and principal for path assignments in network $\mathcal{N}$.

Proof. By Lemmas 42, 50, and 53, it suffices to prove part 2 in the theorem statement above. By the construction in Procedure 49, we already know that $\mathcal{T}$ is total and tight. It remains to show that $\mathcal{T}$ is principal.

We first show that if $h : \Gamma \to \mathbb{R}^+$ is a feasible path assignment, then $h$ satisfies $\mathcal{T}$. This is what we call the “completeness” of a typing in Section 4.2 and Section 5.1. Consider the set of linear inequalities $\mathcal{C}^*$ used by Procedure 49. Because $h$ is feasible, the values assigned by $h$ to the variables/paths in $\Gamma$ satisfy $\mathcal{C}^*$. We can represent $h$ by a “point” in the Euclidean hyperspace of dimension $u = |\Gamma|$, namely, by the vector:

$$h = \{h(\gamma_1), \ldots, h(\gamma_u)\}$$

As a point, $h$ is inside the bounded convex set defined by $\mathcal{C}^*$. Hence, $h$ is inside $\text{Poly}(\mathcal{T})$, as the types/intervals assigned by $\mathcal{T}$ are produced according to Procedure 49, which in turn implies that $h$ satisfies $\mathcal{T}$.

We next prove the converse, namely, that if $h : \Gamma \to \mathbb{R}^+$ satisfies $\mathcal{T}$, then $h$ is a feasible path assignment, which is what we call the “soundness” of $\mathcal{T}$.

We show the converse: If $h$ is not feasible, then $h$ does not satisfy $\mathcal{T}$. If $h$ is not feasible, then $h$ violates a constraint in $\mathcal{C}^*$ of the form $r_1 \leq \sum Y \leq r_2$ for some $Y \in \Gamma$, i.e., $\sum h(Y) \notin [r_1, r_2]$. Because $\mathcal{T}$ is tight by the construction in Procedure 49, we must have $\mathcal{T}(Y) = [s_1, s_2]$ such that $[s_1, s_2] \notin [r_1, r_2]$, which in turn implies that $h$ does not satisfy $\mathcal{T}$.

One payoff of establishing the equivalence of the two views – “feasible flows” versus “feasible path assignments” – is the proof of Theorem 54. We did not present a direct proof of part 1 which, without the benefit of its equivalence with part 2, is subtle and difficult. By comparison, the direct proof of part 2 above is short and transparent.

7 Necessary and Sufficient Conditions

Let $A_{\text{in}} = \{a_1, \ldots, a_m\}$ and $A_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\}$ be fixed, where $m, n \geq 1$. Let $T : \mathcal{P}(A_{\text{in},\text{out}}) \to \mathcal{I}(\mathbb{R})$ where $A_{\text{in},\text{out}} = A_{\text{in}} \cup A_{\text{out}}$. We formulate conditions on $T$ which are satisfied if (necessity), and only if (sufficiency), $T$ is a valid typing, i.e., $T$ is a valid typing for some network $\mathcal{N}$ (Theorem 57). If $[r, s]$ is an interval of real numbers for some $r \leq s$, we write $-[r, s]$ to denote the interval $[-s, -r]$, specifically:

$$-[r, s] = \{ t \in \mathbb{R} \mid -s \leq t \leq -r \}.$$

If $T(A) = [r, s]$ for some $A \subseteq A_{\text{in},\text{out}}$, we define $T^\text{min}(A) = r$ and $T^\text{max}(A) = s$.

Lemma 55. Let $T : \mathcal{P}(A_{\text{in},\text{out}}) \to \mathcal{I}(\mathbb{R})$ be a tight and total typing such that:

$T(\emptyset) = T(A_{\text{in},\text{out}}) = [0, 0] = \{0\}$.

Conclusion: For all $\emptyset \neq A, B \subseteq A_{\text{in},\text{out}}$ such that $A \cup B = A_{\text{in},\text{out}}$, it holds that $T(A) = -T(B)$.

Proof. From Section 4, $\text{Poly}(T)$ is the polytope defined by $T$ and $\text{Constraints}(T)$ is the set of linear inequalities induced by $T$, the latter defined by equation (6). For every $(m + n)$-dimensional point $f \in \text{Poly}(T)$, we have:

$$\sum f(A_{\text{in}}) - \sum f(A_{\text{out}}) = 0$$
because \( T(A_{\text{in, out}}) = \{0\} \) and therefore:

\[
0 \leq \sum \{ a \mid a \in A_{\text{in}} \} - \sum \{ a \mid a \in A_{\text{out}} \} \leq 0
\]

are among the inequalities in \( \text{Constraints(T)} \). Consider arbitrary \( \emptyset \neq A, B \subseteq A_{\text{in, out}} \) such that \( A \bowtie B = A_{\text{in, out}} \). We can therefore write the equation:

\[
\sum f(A \cap A_{\text{in}}) + \sum f(B \cap A_{\text{in}}) - \sum f(A \cap A_{\text{out}}) - \sum f(B \cap A_{\text{out}}) = 0
\]

Or, equivalently:

\[
\left( \frac{\ddagger}{\ddagger} \right) \quad \sum f(A \cap A_{\text{in}}) - \sum f(A \cap A_{\text{out}}) = - \sum f(B \cap A_{\text{in}}) + \sum f(B \cap A_{\text{out}})
\]

for every \( f \in \text{Poly}(T) \). Hence, relative to \( \text{Constraints(T)} \), \( f \) maximizes (resp. minimizes) the left-hand side of equation \((\ddagger)\) iff \( f \) maximizes (resp. minimizes) the right-hand side of \((\ddagger)\). Negating the right-hand side of \((\ddagger)\), we also have:

\[
\text{if and only if}
\]

\[
f \text{ maximizes (resp. minimizes)} \sum f(A \cap A_{\text{in}}) - \sum f(A \cap A_{\text{out}}) \quad \text{and the two quantities are equal.}
\]

Because \( T \) is tight, by Proposition 20, every point \( f \in \text{Poly}(T) \) which maximizes (resp. minimizes) the objective function:

\[
\sum \{ a \mid a \in A \cap A_{\text{in}} \} - \sum \{ a \mid a \in A \cap A_{\text{out}} \}
\]

must be such that:

\[
T_{\text{max}}(A) = \sum f(A \cap A_{\text{in}}) - \sum f(A \cap A_{\text{out}})
\]

(\text{resp. } T_{\text{min}}(A) = \sum f(A \cap A_{\text{in}}) - \sum f(A \cap A_{\text{out}}))

We can repeat the same reasoning for \( B \). Hence, if \( f \in \text{Poly}(T) \) maximizes both sides of \((\ddagger)\):

\[
T_{\text{max}}(A) = + \sum f(A \cap A_{\text{in}}) - \sum f(A \cap A_{\text{out}})
\]

\[
= - \sum f(B \cap A_{\text{in}}) + \sum f(B \cap A_{\text{out}})
\]

\[
= - T_{\text{min}}(B)
\]

and, respectively, if \( f \in \text{Poly}(T) \) minimizes both sides of \((\ddagger)\):

\[
T_{\text{min}}(A) = + \sum f(A \cap A_{\text{in}}) - \sum f(A \cap A_{\text{out}})
\]

\[
= - \sum f(B \cap A_{\text{in}}) + \sum f(B \cap A_{\text{out}})
\]

\[
= - T_{\text{max}}(B)
\]

The preceding implies \( T(A) = -T(B) \) and concludes the proof.

In the preceding proof we do not need to assume that \( T \) is valid, \textit{i.e.,} \( T \) is valid for some network. The proof of the next lemma is far more elaborate and we delay it to Section 8 and Section 9.

\textbf{Lemma 56.} Let \( T : \mathcal{P}(A_{\text{in, out}}) \rightarrow \mathcal{I}(\mathbb{R}) \) be a tight and total typing such that:

\[
T(\emptyset) = T(A_{\text{in, out}}) = \{0, 0\} = \{0\},
\]

and for all \( \emptyset \neq A, B \subseteq A_{\text{in, out}} \) such that \( A \bowtie B = A_{\text{in, out}} \), it holds that \( T(A) = -T(B) \).

\textbf{Conclusion:} \( T \) is principal (and thus valid).
**Theorem 57** (Necessary and Sufficient Conditions for Validity). Let \( T : \mathcal{P}(A_{\text{in, out}}) \to \mathcal{T}(\mathbb{R}) \) be a tight and total typing. Then \( T \) is valid (for some network) iff two conditions are satisfied:

1. \( T(\emptyset) = T(A_{\text{in, out}}) = [0, 0] \).
2. For all \( \emptyset \neq A, B \subseteq A_{\text{in, out}} \) such that \( A \uplus B = A_{\text{in, out}} \), it holds that \( T(A) = -T(B) \).

**Proof.** Immediate consequence of Lemmas 55 and 56. \( \square \)

The necessary and sufficient conditions for validity in Theorem 57 hold as well for principality.

**Corollary 58** (Necessary and Sufficient Conditions for Principality). Let \( T : \mathcal{P}(A_{\text{in, out}}) \to \mathcal{T}(\mathbb{R}) \) be a tight and total typing. Then \( T \) is principal (for some network) iff two conditions are satisfied:

1. \( T(\emptyset) = T(A_{\text{in, out}}) = [0, 0] \).
2. For all \( \emptyset \neq A, B \subseteq A_{\text{in, out}} \) such that \( A \uplus B = A_{\text{in, out}} \), it holds that \( T(A) = -T(B) \).

**Proof.** This follows from Lemma 56 which establishes sufficiency for principality (not only validity), Theorem 57, and the fact that every principal typing is also valid. \( \square \)

From Theorem 57 and Corollary 58, for every typing \( T : \mathcal{P}(A_{\text{in, out}}) \to \mathcal{T}(\mathbb{R}) \) which is tight and total, it holds that \( T \) is valid iff \( T \) is principal. Note carefully how this assertion should be read:

\( T \) is valid for some network \( \mathcal{N} \) iff \( T \) is principal for some network \( \mathcal{N}' \).

The right to left implication is obvious, with \( \mathcal{N} = \mathcal{N}' \), because principality implies validity. However, for the left to right implication, we cannot generally expect that \( \mathcal{N} = \mathcal{N}' \).

As already established in preceding sections, for the same \( \mathcal{N} \), a principal typing \( T \) for \( \mathcal{N} \) (that is also tight and total) is uniquely defined, while there are infinitely many valid typings for the same \( \mathcal{N} \).

### 8 Special Flow Networks

We define a particular class of flow networks which we call “special”. Their topology is particularly simple.

Given \( A_{\text{in}} = \{a_1, \ldots, a_m\} \) and \( A_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\} \), where \( m, n \geq 1 \), the underlying directed graph of the special network over \( A_{\text{in, out}} = A_{\text{in}} \uplus A_{\text{out}} \), denoted \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \), is uniquely defined. We call \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) the special graph over \( A_{\text{in, out}} \), with input arcs \( A_{\text{in}} \) and output arcs \( A_{\text{out}} \).

There is one node in \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) for every non-empty subset \( A \subseteq A_{\text{in}} \), denoted \( \nu_A \), and again one node for every non-empty subset \( B \subseteq A_{\text{out}} \), denoted \( \nu_B \):

\[
\text{nodes}(\text{Graph}(A_{\text{in}}, A_{\text{out}})) = N_{\text{in},#} \cup N_{\text{out},#} \quad \text{where}
\]

\[
N_{\text{in},#} = \{ \nu_A \mid \emptyset \neq A \in \mathcal{P}(A_{\text{in}}) \},
\]

\[
N_{\text{out},#} = \{ \nu_B \mid \emptyset \neq B \in \mathcal{P}(A_{\text{out}}) \}.
\]

If \( A \subseteq A_{\text{in}} \) or \( B \subseteq A_{\text{out}} \) is a singleton set \( \{a\} \), we may write \( \nu_a \) instead of \( \nu_{\{a\}} \). The total number of nodes in \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) is therefore \( 2^m + 2^n - 2 \).

We next define the input arcs, the output arcs, and the internal arcs of \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \). By our conventions, the tail of an input arc and the head of an output arc are not defined.

**Input arcs.** The set of input arcs is:

\[
\text{in}(\text{Graph}(A_{\text{in}}, A_{\text{out}})) = \{a_1, \ldots, a_m\}
\]

where for every \( a_i \), we have \( \text{head}(a_i) = \nu_{a_i} \).
Output arcs. The set of output arcs is:

$$\text{out}(\text{Graph}(A_{\text{in}}, A_{\text{out}})) = \{a_{m+1}, \ldots, a_{m+n}\}$$

where for every $a_j$, we have $\text{tail}(a_j) = \nu_{a_j}$.

Internal arcs. Each internal arc is a pair $\langle\nu_A, \nu_B\rangle$ with $\text{tail}(\langle\nu_A, \nu_B\rangle) = \nu_A$ and $\text{head}(\langle\nu_A, \nu_B\rangle) = \nu_B$. The set of internal arcs is:

$$\#(\text{Graph}(A_{\text{in}}, A_{\text{out}})) = A_{\text{in}, \#} \cup A_{\#, \text{out}} \cup A_{\#, \#}$$

where

- $A_{\text{in}, \#} = \{\langle\nu_A, \nu_A\rangle | a \in A_{\text{in}} \land \emptyset \neq A\}$,
- $A_{\#, \text{out}} = \{\langle\nu_B, \nu_B\rangle | b \in A_{\text{out}} \land \emptyset \neq B\}$,
- $A_{\#, \#} = N_{\text{in}, \#} \times N_{\#, \text{out}} = \{\langle\nu_A, \nu_B\rangle | \emptyset \neq A \in A_{\text{in}} \land \emptyset \neq B \in A_{\text{out}}\}$.

By a little computation using standard formulas of binomial coefficients, the total number of internal arcs is, when both $m \geq 2$ and $n \geq 2$:

$$\sum_{i=2}^{m} i \cdot \binom{m}{i} + \sum_{j=2}^{n} j \cdot \binom{n}{j} + (2^n - 1) \cdot (2^n - 1) =$$

$$2^{m+n} + (m-2) \cdot 2^{n-1} + (n-2) \cdot 2^{m-1} - m - n + 1$$

This is a relatively large number, but the construction in Section 9 will typically assign zero upper-bound capacities to many of them, making them unusable by feasible flows. When $m = 1$, we have $\binom{1}{i} = 0$ for every $i \geq 2$, and similarly when $n = 1$, we have $\binom{1}{j} = 0$ for every $j \geq 2$, and the preceding formula can be simplified a little (omitted here). In particular, when both $m = n = 1$, the total number of internal arcs is exactly 1.

As defined so far, $\text{Graph}(A_{\text{in}}, A_{\text{out}})$ is a directed acyclic graph. In the next section, we may want to reverse the direction of some arcs $\langle\nu_A, \nu_B\rangle$ in $A_{\#, \#}$, for which we write:

$$\langle\nu_B, \nu_A\rangle = \text{reverse}(\langle\nu_A, \nu_B\rangle)$$

so that

$$\text{tail}(\langle\nu_A, \nu_B\rangle) = \text{head}(\langle\nu_B, \nu_A\rangle) = \nu_A \land \text{head}(\langle\nu_A, \nu_B\rangle) = \text{tail}(\langle\nu_B, \nu_A\rangle) = \nu_B.$$

Reversing the direction of some arcs in $A_{\#, \#}$ may introduce cycles in the graph. In the next section, we turn $\text{Graph}(A_{\text{in}}, A_{\text{out}})$ into a flow network in four stages:

(I) Designate the nodes in $N_{\text{in}, \#}$ as consumers, by defining a consumer assignment $\kappa_{\text{in}}: N_{\text{in}, \#} \rightarrow \mathbb{R}^-$. 

(II) Designate the nodes in $N_{\#, \text{out}}$ as producers, by defining a producer assignment $\kappa_{\text{out}}: N_{\#, \text{out}} \rightarrow \mathbb{R}^+$. 

(III) Assign an upper-bound capacity to every arc in $A_{\#, \#}$. 

(IV) Assign to every arc outside $A_{\#, \#}$ the trivial lower-bound capacity 0, and the trivial upper-bound capacity $K$ which denotes a “very large number”.

The definition of a consumer/producer assignment $\kappa$ is in Section 2.1; here, we break up $\kappa$ into two parts, $\kappa_{\text{in}}$ and $\kappa_{\text{out}}$. Stages (I), (II), and (IV), are relatively straightforward; stage (III) requires more work. Details for all four stages are given in Section 9.

Figure 8 is a graphic representation of $\text{Graph}(A_{\text{in}}, A_{\text{out}})$ for the particular case when $A_{\text{in}} = \{a_1, a_2, a_3\}$ and $A_{\text{out}} = \{a_4, a_5\}$. For each node, we only show its label (subscript), i.e., we write the label “$\text{A}$” not the full name “$\nu_A$” where $\emptyset \neq A \subseteq A_{\text{in}}$ or $\emptyset \neq A \subseteq A_{\text{out}}$.
9 Proof of Sufficiency

$A_{in} = \{a_1, \ldots, a_m\}$ and $A_{out} = \{a_{m+1}, \ldots, a_{m+n}\}$, where $m, n \geq 1$, are fixed throughout this section. Let $T$ be a tight and total typing over $A_{in,out}$ which also satisfies the hypothesis of Lemma 56.

9.1 Assigning Lower-Bound Capacities

We first carry out stages (I) and (II), as defined at the end of Section 8. For these two stages, we use $T_{\min}(A)$ for every $\emptyset \neq A \subseteq A_{in}$ and $T_{\max}(B)$ for every $\emptyset \neq B \subseteq A_{out}$, as follows:

$$
\begin{align*}
\kappa_{\text{in}}(\nu_A) &= \begin{cases} 
-T_{\min}(A) & \text{if } A = \{a\} \text{ and } a \in A_{in}, \\
-T_{\min}(A) + \sum \{-\kappa_{\text{in}}(\nu_{A'}) \mid \emptyset \neq A' \subseteq A\} & \text{if } |A| \geq 2 \text{ and } A \subseteq A_{in}.
\end{cases} \\
\kappa_{\text{out}}(\nu_B) &= \begin{cases} 
-T_{\max}(B) & \text{if } B = \{b\} \text{ and } b \in A_{out}, \\
-T_{\max}(B) + \sum \{-\kappa_{\text{out}}(\nu_{B'}) \mid \emptyset \neq B' \subseteq B\} & \text{if } |B| \geq 2 \text{ and } B \subseteq A_{out}.
\end{cases}
\end{align*}
$$

The definition of $\kappa_{\text{in}}$ and $\kappa_{\text{out}}$ completes stages (I) and (II).

We collect a few facts about $\kappa_{\text{in}}$ and $\kappa_{\text{out}}$. Because $\text{Poly}(T)$ is inside the first orthant of $\mathbb{R}^{m+n}$, by our standing assumption (point 3 in Restriction 16), and because $T$ is tight, it is straightforward to check that:

1. For every $\emptyset \neq A \in \mathcal{P}(A_{in})$, we have:
   $$T_{\min}(A) = -\sum \{\kappa_{\text{in}}(\nu_{A'}) \mid \emptyset \neq A' \subseteq A\} \geq 0$$

2. For every $\emptyset \neq B \in \mathcal{P}(A_{out})$, we have:
   $$T_{\max}(B) = -\sum \{\kappa_{\text{out}}(\nu_{B'}) \mid \emptyset \neq B' \subseteq B\} \leq 0$$
Because $T^{\min}(A) \geq 0$ for every $\emptyset \neq A \in \mathcal{P}(\text{in})$, and $T^{\max}(B) \leq 0$ for every $\emptyset \neq B \in \mathcal{P}(\text{out})$, the preceding implies $\kappa^{\text{in}}(\nu_A) \leq 0$ and $\kappa^{\text{out}}(\nu_B) \geq 0$, respectively. Hence, also:

\[
T^{\min}(\text{in}) = - \sum \{ \kappa^{\text{in}}(\nu_A) | \emptyset \neq A \subseteq \text{in} \} \geq 0,
\]

\[
T^{\max}(\text{out}) = - \sum \{ \kappa^{\text{out}}(\nu_B) | \emptyset \neq B \subseteq \text{out} \} \leq 0,
\]

and, because $T$ satisfies the hypothesis of Lemma 56, $T^{\min}(\text{in}) + T^{\max}(\text{out}) = 0$. In words, this last assertion says that "the minimum possible flow entering the network" must equal the "the minimum possible flow exiting the network". Recall that exiting flow is a negative number, which means that $T^{\max}(\text{out})$ is the smallest possible amount at the output arcs.

**Lemma 59.** Let $T$ be a tight and total typing over $\text{in,out} = \{a_1, \ldots, a_{m+n}\}$. Then:

1. $\kappa^{\text{in}}(\nu_A) \leq 0$ for every $\emptyset \neq A \subseteq \text{in}$, implying $\nu_A$ is a consumer node.
2. $\kappa^{\text{out}}(\nu_B) \geq 0$ for every $\emptyset \neq B \subseteq \text{out}$, implying $\nu_B$ is a producer node.

**Proof.** Immediate consequence of the observations preceding the lemma.

**Remark 60.** There is an asymmetry in the way we turn the special Graph($\text{in}$, $\text{out}$) into a flow network. Indeed, we start with an assignment of lower-bound capacities, implemented by turning nodes in $\text{in}$ into consumers and nodes in $\text{out}$ into producers, in stages (I) and (II). Can’t we start with an assignment of upper-bound capacities instead, and then handle the lower-bound capacities in a subsequent stage? We do not know if this is possible, nor do we know how to do it.

Though lower-bound capacities and upper-bound capacities behave symmetrically in many respects, they do not always. This is illustrated by Example 61.

**Example 61.** Let $T_2$ be the total and tight typing computed in Example 46. Using the definition of $\kappa^{\text{in}}$ and $\kappa^{\text{out}}$ in stages (I) and (II) above, we turn nodes in $\text{in, out}$ into consumers and producers, respectively, in Graph($\text{in}$, $\text{out}$) of Figure 8. The result is shown in Figure 9.

We thus enforce a lower-bound of 2 on input flow at arc $a_1$ by making node $\nu_{\{a_1\}}$ a consumer of 2 units, a lower-bound of 4 on input flow at arc $a_3$ by making node $\nu_{\{a_3\}}$ a consumer of 4 units, and a lower-bound of 7 on input flow at arcs $\{a_1, a_2, a_3\}$ by making node $\nu_{\{a_1, a_2, a_3\}}$ a consumer of 1 unit. With one pass – starting with singleton subsets of $\text{in}$, then going to two-element subsets, then to three-element subsets, etc. – we can thus enforce lower bounds on input flows.

We proceed similarly to enforce lower bounds on output flows. In this example, it requires turning $\nu_{\{a_4\}}$ into a producer of 3 units and $\nu_{\{a_5\}}$ into a producer of 4 units.

This consumer/producer assignment, $\kappa^{\text{in}} \cup \kappa^{\text{out}}$, will not change as a result of computing upper-bound capacities in stage (III) below.

Note however that, had we wanted to start with enforcing upper-bounds on input flow and output flow before handling the lower-bounds, it is not clear how we could have done it by turning nodes into consumers and producers.
9.2 Assigning Upper-Bound Capacities

Stage (III) is the most involved of the four stages to turn \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) into a flow network. The plan is to set up a system of linear equations, written in matrix form as \( Cx = d \), where \( C \) is a square \( \ell \times \ell \) coefficient matrix where \( \ell = (2^m - 1) \cdot (2^n - 1) \), the column vector \( x \) is an ordering of a set \( X \) of \( \ell \) variables:

\[
X = \{ x(A, B) \mid \emptyset \neq A \subseteq A_{\text{in}} \text{ and } \emptyset \neq B \subseteq A_{\text{out}} \},
\]

and the column vector \( d \) consists of \( \ell \) constants which we collect from the given typing \( T \).

We define \( C \) as a non-singular matrix of 0’s and 1’s. The variable \( x(A, B) \) corresponds to the upper bound capacity we want to assign to the arc \( \langle \nu_A, \nu_B \rangle \) in the set \( A_{\#}. \)

Remark 62. It is worth putting a few things in perspective before proceeding further. All the information we have are the facts provided by the given typing \( T \) – tight, total, and satisfying the hypothesis of Lemma 56. These consist of the two endpoints (the lower end and the upper end) of each of the \( 2^{m+n} \) types specified by \( T \). Discounting the two extremal types for \( \emptyset \) and \( A_{\text{in,out}} \), because \( T(\emptyset) = T(A_{\text{in,out}}) = [0, 0] = \{0\} \), we have a total of \( 2 \cdot (2^{m+n} - 2) \) endpoints to work with. Because \( T \) satisfies the hypothesis of Lemma 56, i.e., \( T(A \cup B) = -T((A_{\text{in}} - A) \cup (A_{\text{out}} - B)) \) for all \( A \subseteq A_{\text{in}} \) and \( B \subseteq A_{\text{out}} \), we only have \( (2^{m+n} - 2) \) endpoints to work with.

We have already used \( (2^m - 1) \) and \( (2^n - 1) \) endpoints in the definition of \( \kappa_{\text{in}} \) and \( \kappa_{\text{out}} \) above, namely, the endpoint \( T_{\min}(A) \) for every \( \emptyset \neq A \subseteq A_{\text{in}} \) and the endpoint \( T_{\max}(B) \) for every \( \emptyset \neq B \subseteq A_{\text{out}} \). In fact, because \( T(A_{\text{in}}) = -T(A_{\text{out}}) \) implies \( T_{\min}(A_{\text{in}}) = -T_{\max}(A_{\text{out}}) \), we have used \( (2^m - 1) + (2^n - 1) - 1 = 2^m + 2^n - 3 \) already of the endpoints provided by \( T \). Based on the preceding, the number of remaining endpoints we can still use to solve the matrix equation \( Cx = d \) is:

\[
(2^{m+n} - 2) - (2^m + 2^n - 3) = 2^{m+n} - 2^m - 2^n + 1 = (2^m - 1) \cdot (2^n - 1).
\]

Note that \((2^m - 1) \cdot (2^n - 1)\) is exactly the number of arcs in the set \( A_{\#} \) and, therefore, the number of variables in the set \( X \).

We need to order the set \( X \). We choose to represent each subset \( A \) of \( A_{\text{in}} = \{a_1, \ldots, a_m\} \) by its characteristic function \( p : \{1, \ldots, m\} \rightarrow \{0, 1\} \) with \( a_i \in A \) iff \( p(i) = 1 \). We choose to do the same for the subsets \( B \) of \( A_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\} \). Using characteristic functions as indeces, we can then write:

\[
\mathcal{P}(A_{\text{in}}) = \left\{ A_p \mid p : \{1, \ldots, m\} \rightarrow \{0, 1\} \right\} \quad \text{and} \quad \mathcal{P}(A_{\text{out}}) = \left\{ B_q \mid q : \{1, \ldots, n\} \rightarrow \{0, 1\} \right\}.
\]
We can view \( p : \{1, \ldots, m\} \to \{0, 1\} \) as a \( m \)-vector of 0’s and 1’s, and \( q : \{1, \ldots, n\} \to \{0, 1\} \) as a \( n \)-vector of 0’s and 1’s. Viewing \( p \) as a \( m \)-vector, we write \( p = (p(1), p(2), \ldots, p(m)) \). Similarly, viewing \( q \) as a vector, we write \( q = (q(1), q(2), \ldots, q(n)) \).

**Definition 63 (Ordering the Characteristic Functions).** Let \( p_1, p_2 : \{1, \ldots, m\} \to \{0, 1\} \). Suppose \( p_1 \neq p_2 \) and let \( (p_1(i), \ldots, p_1(m)) = (p_2(i), \ldots, p_2(m)) \) be the longest common suffix of \( p_1 \) and \( p_2 \), for some \( 2 \leq i \leq m+1 \). (If \( i = m+1 \), then the longest common suffix is the empty vector \( \langle \rangle \).) We define:

\[
p_1 < p_2 \quad \text{iff} \quad p_1(i-1) = 0 \quad \text{and} \quad p_2(i-1) = 1.
\]

The order \( \prec \) is therefore the lexicographic order on the characteristic functions when read from right to left. Hence, read as binary numbers from right to left, the characteristic functions of the subsets of \( \mathbb{A}_{in} \) as a \(-\)-vector of 0’s and 1’s, and \( \mathbb{A}_{out} \) as a \(-\)-vector.

We extend the order \( \prec \) on characteristic functions to pairs of characteristic functions. Consider arbitrary \( p_1, p_2 : \{1, \ldots, m\} \to \{0, 1\} \) and \( q_1, q_2 : \{1, \ldots, n\} \to \{0, 1\} \). We define:

\[
(p_1, q_1) < (p_2, q_2) \quad \text{iff} \quad \text{either} \ (q_1 < q_2) \ \text{or} \ (p_1 < p_2 \ \text{and} \ q_1 = q_2).
\]

The order \( \prec \) on characteristic functions and pairs of characteristic functions induces an ordering on the power set \( \mathcal{P}(\mathbb{A}_{in, out}) \) and on the set \( X \) of variables. This is partly illustrated in Example 64.

**Example 64.** Let \( \mathbb{A}_{in} = \{a_1, a_2, a_3\} \) and \( \mathbb{A}_{out} = \{a_4, a_5\} \). The characteristic functions for subsets of \( \mathbb{A}_{in} \) and for subsets of \( \mathbb{A}_{out} \) are maps \( p : \{1, 2, 3\} \to \{0, 1\} \) and \( q : \{1, 2\} \to \{0, 1\} \), respectively. We use the names of characteristic functions as subscripts to identify the sets they define, i.e., \( A_p \subseteq \mathbb{A}_{in} \) and \( B_q \subseteq \mathbb{A}_{out} \). The ordering \( \prec \) on the set of pairs of characteristic functions:

\[
\left\{ (p, q) \mid p : \{1, 2, 3\} \to \{0, 1\} \text{ and } q : \{1, 2\} \to \{0, 1\} \right\}
\]

induces a total order on the subsets \( A_p \cup B_q \subseteq \mathbb{A}_{in, out} \) where \( A_p \subseteq \mathbb{A}_{in} \) and \( B_q \subseteq \mathbb{A}_{out} \), as well as a total order on the set \( X \) of variables, each written as \( x(A_p, B_q) \). Figure 10 gives the details:

1. The second column lists all the pairs \( (p, q) \) of characteristic functions, from top to bottom, according to the ordering \( \prec \) (the first column identifies their positions in this ordering with consecutive natural numbers).
2. The third column lists all the subsets of \( \mathbb{A}_{in, out} \), where every input arc is included positively and every output arc is included negatively.
3. A total and tight typing \( T \) maps every set \( A_p \cup B_q \) to a type/interval \( T(A_p \cup B_q) \). The fourth column shows whether endpoint \( T^\text{min}(A_p \cup B_q) \), indicated by \( \text{“m”} \), or endpoint \( T^\text{max}(A_p \cup B_q) \), indicated by \( \text{“M”} \), is used to compute \( \kappa_{in} \) and \( \kappa_{out} \).
4. The fifth column shows which of the endpoints \( T^\text{max}(A_p \cup B_q) \), indicated by \( \text{“M”} \), are used to solve the equations \( C : x = d \).
5. The sixth column lists all the variables \( x(A_p, B_q) \), from top to bottom, in the order induced by \( \prec \). These variables are defined only for \( A_p \neq \emptyset \) and \( B_q \neq \emptyset \).

---

9Most significant digit to the right, least significant digit to the left.

9An alternative and equivalent definition is to introduce concatenation of two characteristic functions \( p \) and \( q \) by writing \( p \cdot q \), and then define: \( (p_1, q_1) \prec (p_2, q_2) \iff p_1 \cdot q_1 \prec p_2 \cdot q_2 \).
It is instructive to note that, even though Figure 10 does not show it, all the endpoints provided by \( T \) are used in computing \( \kappa^\text{in} \) and \( \kappa^\text{out} \) and solving the equations \( Cx = d \). This is so because \( T \) is assumed to satisfy the hypothesis of Lemma 56. Indeed, looking down the list in the third column, we notice that the \( k \)-th set from the top is the complement of the \( k \)-th set from the bottom, for every \( 0 \leq k \leq 15 \). For example, the 10-th set from the top \( \{ a_1, a_4 \} \) (written as \( a_1 - a_4 \) on line 9) is the complement of the 10-th set from the bottom \( \{ a_2, a_3, a_5 \} \) (written as \( a_2 + a_3 - a_5 \) on line 22). The hypothesis of Lemma 56 says:

\[
T(A_p \cup B_q) = -T(A_{p'} \cup B_{q'}) \quad \text{whenever } A_{p'} = A_{\text{in}} - A_p \text{ and } B_{q'} = A_{\text{out}} - B_q,
\]

which implies \( T^\text{min}(A_p \cup B_q) = -T^\text{max}(A_{p'} \cup B_{q'}) \) and \( T^\text{max}(A_p \cup B_q) = -T^\text{min}(A_{p'} \cup B_{q'}) \). Discounting the endpoints of \( T(\emptyset) \) and \( T(A_{\text{in}, \text{out}}) \), in lines 0 and 31, which are always 0 by Lemma 56, this shows that all the endpoints provided by \( T \) are used and each is used exactly once. \( \square \)

We set up \( \ell = (2^m - 1) \cdot (2^n - 1) = 2^{m+n} - 2^m - 2^n + 1 \) linear equations, based on which we then define the matrix equation \( Cx = d \). For a justification of these numbers, see Remark 62 and Example 64 above. For every \( \emptyset \subseteq A_p \subseteq A_{\text{in}} \) and every \( \emptyset \subseteq B_q \subseteq A_{\text{out}} \), we define an equation, denoted \( \text{Eq}(\max, p, q) \), in terms of the higher endpoint of the given \( T(A_p \cup B_q) \):

\[
\text{Eq}(\max, p, q) : \quad T^\text{max}(A_p \cup B_q) = + \sum \left\{ x(A, B) \mid A \cap A_p \neq \emptyset \text{ and } B \cap (A_{\text{out}} - B_q) \neq \emptyset \right\}
\]

\[
- \sum \left\{ \kappa^\text{in}(\nu_A) \mid A \cap A_p \neq \emptyset \right\}
\]

\[
- \sum \left\{ \kappa^\text{out}(\nu_B) \mid \emptyset \neq B \subseteq B_q \right\}.
\]

It is possible that the three summations on the right-hand side are over empty sets, in which case their value is 0. Recalling from Procedure 45 that \( T^\text{max}(A_p \cup B_q) \) measures:

(maximum possible flow into \( A_p \)) – (minimum possible flow out of \( B_q \)),

we have the following interpretation of the preceding equation:

1. The first summation of variables \( x(A, B) \) expresses the sum of all flows from input arcs in \( A_p \) to output arcs not in \( B_q \). The variable \( x(A, B) \) denotes flow on arc \( (\nu_A, \nu_B) \) where \( A \) intersects (not just a subset of) \( A_p \) and \( B \) intersects (not just a subset of) the complement \( (A_{\text{out}} - B_q) \).

2. The second summation of constants \( \kappa^\text{in}(\nu_A) \) expresses a quantity consumed at nodes where all of the arcs included in the first summation start from. Keep in mind that every \( \kappa^\text{in}(\nu_A) \) is non-positive, so that

\( -\sum \{ \kappa^\text{in}(\nu_A) \mid A \cap A_p \neq \emptyset \} \geq 0 \).

3. The third summation of constants \( \kappa^\text{out}(\nu_B) \) expresses a quantity produced at nodes where all the arcs excluded in the first summation end at. Every \( \kappa^\text{out}(\nu_B) \geq 0 \), so that

\( -\sum \{ \kappa^\text{out}(\nu_B) \mid \emptyset \neq B \subseteq B_q \} \leq 0 \).

For every \( \emptyset \subseteq A_p \subseteq A_{\text{in}} \) and every \( \emptyset \subseteq B_q \subseteq A_{\text{out}} \), we also define an equation in terms of the lower endpoint of the given \( T(A_p \cup B_q) \). The justification and interpretation for \( \text{Eq}(\max, p, q) \), with obvious modification, apply again to \( \text{Eq}(\min, p, q) \):

\[
\text{Eq}(\min, p, q) : \quad T^\text{min}(A_p \cup B_q) = - \sum \left\{ x(A, B) \mid A \cap (A_{\text{in}} - A_p) \neq \emptyset \text{ and } B \cap B_q \neq \emptyset \right\}
\]

\[
- \sum \left\{ \kappa^\text{in}(\nu_A) \mid \emptyset \neq A \subseteq A_p \right\}
\]

\[
- \sum \left\{ \kappa^\text{out}(\nu_B) \mid B \cap B_q \neq \emptyset \right\}.
\]
<table>
<thead>
<tr>
<th>ordering of ((p, q))</th>
<th>(A_p \cup B_q) where (A_p \subseteq A_{\text{in}}) and (A_q \subseteq A_{\text{out}}) with signs inserted</th>
<th>endpoints used to compute (\kappa_{\text{in}}) and (\kappa_{\text{out}})</th>
<th>endpoints used to solve (C x = d)</th>
<th>(x(A_p, B_q)) where (\partial \neq A_p \subseteq A_{\text{in}}) and (\partial \neq B_q \subseteq A_{\text{out}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (((0, 0), (0, 0)))</td>
<td>(\emptyset)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 (((1, 0), (0, 0)))</td>
<td>(a_1)</td>
<td>(m)</td>
<td>(M)</td>
<td></td>
</tr>
<tr>
<td>2 (((0, 1), (0, 0)))</td>
<td>(a_2)</td>
<td>(m)</td>
<td>(M)</td>
<td></td>
</tr>
<tr>
<td>3 (((1, 1), (0, 0)))</td>
<td>(a_1 + a_2)</td>
<td>(m)</td>
<td>(M)</td>
<td></td>
</tr>
<tr>
<td>4 (((0, 0), (1, 0)))</td>
<td>(a_3)</td>
<td>(m)</td>
<td>(M)</td>
<td></td>
</tr>
<tr>
<td>5 (((1, 0), (1, 0)))</td>
<td>(a_1 + a_3)</td>
<td>(m)</td>
<td>(M)</td>
<td></td>
</tr>
<tr>
<td>6 (((0, 1), (1, 0)))</td>
<td>(a_2 + a_3)</td>
<td>(m)</td>
<td>(M)</td>
<td></td>
</tr>
<tr>
<td>7 (((1, 1), (1, 0)))</td>
<td>(a_1 + a_2 + a_3)</td>
<td>(m)</td>
<td>(M)</td>
<td></td>
</tr>
<tr>
<td>8 (((0, 0), (0, 1)))</td>
<td>(-a_4)</td>
<td>(M)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 (((1, 0), (0, 1)))</td>
<td>(a_1 - a_4)</td>
<td>(M)</td>
<td>(x({a_1}, {a_4}))</td>
<td></td>
</tr>
<tr>
<td>10 (((0, 1), (0, 1)))</td>
<td>(a_2 - a_4)</td>
<td>(M)</td>
<td>(x({a_2}, {a_4}))</td>
<td></td>
</tr>
<tr>
<td>11 (((1, 0), (0, 1)))</td>
<td>(a_1 + a_2 - a_4)</td>
<td>(M)</td>
<td>(x({a_1, a_2}, {a_4}))</td>
<td></td>
</tr>
<tr>
<td>12 (((0, 0), (0, 1)))</td>
<td>(a_3 - a_4)</td>
<td>(M)</td>
<td>(x({a_3}, {a_4}))</td>
<td></td>
</tr>
<tr>
<td>13 (((1, 0), (1, 1)))</td>
<td>(a_1 + a_3 - a_4)</td>
<td>(M)</td>
<td>(x({a_1, a_3}, {a_4}))</td>
<td></td>
</tr>
<tr>
<td>14 (((0, 1), (1, 0)))</td>
<td>(a_2 + a_3 - a_4)</td>
<td>(M)</td>
<td>(x({a_2, a_3}, {a_4}))</td>
<td></td>
</tr>
<tr>
<td>15 (((1, 1), (1, 0)))</td>
<td>(a_1 + a_2 + a_3 - a_4)</td>
<td>(M)</td>
<td>(x({a_1, a_2, a_3}, {a_4}))</td>
<td></td>
</tr>
<tr>
<td>16 (((0, 0), (1, 1)))</td>
<td>(-a_5)</td>
<td>(M)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 (((1, 0), (0, 1)))</td>
<td>(a_1 - a_5)</td>
<td>(M)</td>
<td>(x({a_1}, {a_5}))</td>
<td></td>
</tr>
<tr>
<td>18 (((0, 1), (0, 1)))</td>
<td>(a_2 - a_5)</td>
<td>(M)</td>
<td>(x({a_2}, {a_5}))</td>
<td></td>
</tr>
<tr>
<td>19 (((1, 1), (0, 1)))</td>
<td>(a_1 + a_2 - a_5)</td>
<td>(M)</td>
<td>(x({a_1, a_2}, {a_5}))</td>
<td></td>
</tr>
<tr>
<td>20 (((0, 0), (1, 0)))</td>
<td>(a_3 - a_5)</td>
<td>(M)</td>
<td>(x({a_3}, {a_5}))</td>
<td></td>
</tr>
<tr>
<td>21 (((1, 0), (1, 1)))</td>
<td>(a_1 + a_3 - a_5)</td>
<td>(M)</td>
<td>(x({a_1, a_3}, {a_5}))</td>
<td></td>
</tr>
<tr>
<td>22 (((0, 1), (1, 1)))</td>
<td>(a_2 + a_3 - a_5)</td>
<td>(M)</td>
<td>(x({a_2, a_3}, {a_5}))</td>
<td></td>
</tr>
<tr>
<td>23 (((1, 1), (1, 1)))</td>
<td>(a_1 + a_2 + a_3 - a_5)</td>
<td>(M)</td>
<td>(x({a_1, a_2, a_3}, {a_5}))</td>
<td></td>
</tr>
<tr>
<td>24 (((0, 0), (1, 1)))</td>
<td>(-a_4 - a_5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 (((1, 0), (1, 1)))</td>
<td>(a_1 - a_4 - a_5)</td>
<td></td>
<td>(x({a_1}, {a_4, a_5}))</td>
<td></td>
</tr>
<tr>
<td>26 (((0, 1), (1, 1)))</td>
<td>(a_2 - a_4 - a_5)</td>
<td></td>
<td>(x({a_2}, {a_4, a_5}))</td>
<td></td>
</tr>
<tr>
<td>27 (((1, 1), (1, 0)))</td>
<td>(a_1 + a_2 - a_4 - a_5)</td>
<td></td>
<td>(x({a_1, a_2}, {a_4, a_5}))</td>
<td></td>
</tr>
<tr>
<td>28 (((0, 1), (1, 1)))</td>
<td>(a_3 - a_4 - a_5)</td>
<td></td>
<td>(x({a_3}, {a_4, a_5}))</td>
<td></td>
</tr>
<tr>
<td>29 (((1, 0), (1, 1)))</td>
<td>(a_1 + a_3 - a_4 - a_5)</td>
<td></td>
<td>(x({a_1, a_3}, {a_4, a_5}))</td>
<td></td>
</tr>
<tr>
<td>30 (((0, 1), (1, 1)))</td>
<td>(a_2 + a_3 - a_4 - a_5)</td>
<td></td>
<td>(x({a_2, a_3}, {a_4, a_5}))</td>
<td></td>
</tr>
<tr>
<td>31 (((1, 1), (1, 1)))</td>
<td>(a_1 + a_2 + a_3 - a_4 - a_5)</td>
<td></td>
<td>(x({a_1, a_2, a_3}, {a_4, a_5}))</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 10:** For Example 64 and Definition 67: Ordering “\(<\)” on pairs \((p, q)\), and its induced orderings on \(\mathcal{P}(A_{\text{in, out}})\) and variables \(x(A_p, B_q)\), when \(A_{\text{in}} = \{a_1, a_2, a_3\}\) and \(A_{\text{out}} = \{a_4, a_5\}\). “\(<\)” (resp. “\(\geq\)” \(\partial\) refers to lower (resp. higher) endpoint of interval \(T(A_p \cup B_q)\).
We re-arrange Eq(\(\max(p, q)\)) and Eq(\(\min(p, q)\)) as linear equations to solve for \(\mathcal{X}\), with variables on the left and constants on the right:

\[
\text{Eq(\(\max(p, q)\)) :} \\
\sum \left\{ x(A, B) \mid A \cap A_p \neq \emptyset \text{ and } B \cap (A_{\text{out}} - B_q) \neq \emptyset \right\} = \\
T^{\max}(A_p \cup B_q) + \sum \left\{ \kappa^{\text{in}}(\nu_A) \mid A \cap A_p \neq \emptyset \right\} + \sum \left\{ \kappa^{\text{out}}(\nu_B) \mid \emptyset \neq B \subseteq B_q \right\}.
\]

\[
\text{Eq(\(\min(p, q)\)) :} \\
\sum \left\{ x(A, B) \mid A \cap (A_{\text{in}} - A_p) \neq \emptyset \text{ and } B \cap B_{q'} \neq \emptyset \right\} = \\
- T^{\min}(A_p \cup B_q) - \sum \left\{ \kappa^{\text{in}}(\nu_A) \mid \emptyset \neq A \subseteq A_p \right\} - \sum \left\{ \kappa^{\text{out}}(\nu_B) \mid B \cap B_q \neq \emptyset \right\}.
\]

**Lemma 65.** Let \(T\) satisfy the hypothesis of Lemma 56. Let \(\emptyset \subseteq A, A_{p'} \subseteq A_{\text{in}}\) and \(\emptyset \subseteq B_q, B_{q'} \subseteq A_{\text{out}}\) such that \(A_{p'} = A_{\text{in}} - A_p\) and \(B_{q'} = A_{\text{out}} - B_q\). Then equations Eq(\(\max(p, q)\)) and Eq(\(\min(p', q')\)) are identical.

**Proof.** We first show that the left-hand side of Eq(\(\min(p', q')\)) is the same as the left-hand side of Eq(\(\max(p, q)\)):

\[
\sum \left\{ x(A, B) \mid A \cap (A_{\text{in}} - A_p) \neq \emptyset \text{ and } B \cap B_{q'} \neq \emptyset \right\} = \\
\sum \left\{ x(A, B) \mid A \cap A_p \neq \emptyset \text{ and } B \cap (A_{\text{out}} - B_q) \neq \emptyset \right\},
\]

because \(A_p \cup A_{p'} = A_{\text{in}}\), and therefore \((A_{\text{in}} - A_{p'}) = A_p\), and \(B_{q'} = (A_{\text{out}} - B_q)\).

We next show that the right-hand side of Eq(\(\min(p', q')\)) is the same as the right-hand side of Eq(\(\max(p, q)\)).

Observe that every non-empty \(A \subseteq A_{\text{in}}\) satisfies one of two conditions:

\[
A \cap A_p \neq \emptyset \quad \text{or} \quad \emptyset \neq A \subseteq (A_{\text{in}} - A_p) = A_{p'}.
\]

It then follows that:

\[
\sum \left\{ \kappa^{\text{in}}(\nu_A) \mid A \cap A_p \neq \emptyset \right\} + \sum \left\{ \kappa^{\text{in}}(\nu_A) \mid \emptyset \neq A \subseteq A_{p'} \right\} = \sum \left\{ \kappa^{\text{in}}(\nu_A) \mid \emptyset \neq A \subseteq A_{\text{in}} \right\}
\]

By a similar reasoning, we also have:

\[
\sum \left\{ \kappa^{\text{out}}(\nu_B) \mid \emptyset \neq B \subseteq B_q \right\} + \sum \left\{ \kappa^{\text{out}}(\nu_B) \mid B \cap B_{q'} \neq \emptyset \right\} = \sum \left\{ \kappa^{\text{out}}(\nu_B) \mid \emptyset \neq B \subseteq A_{\text{out}} \right\}
\]

By the definition of \(\kappa^{\text{in}}\) and \(\kappa^{\text{out}}\) at the beginning of this section, and the fact that \(T\) satisfies the hypothesis of Lemma 56, we have:

\[
- \sum \left\{ \kappa^{\text{in}}(\nu_A) \mid \emptyset \neq A \subseteq A_{\text{in}} \right\} = \sum \left\{ \kappa^{\text{out}}(\nu_B) \mid \emptyset \neq B \subseteq A_{\text{out}} \right\}
\]

which in turn implies:

\[
+ \sum \left\{ \kappa^{\text{in}}(\nu_A) \mid A \cap A_p \neq \emptyset \right\} + \sum \left\{ \kappa^{\text{in}}(\nu_A) \mid \emptyset \neq A \subseteq A_{p'} \right\} = \\
- \sum \left\{ \kappa^{\text{out}}(\nu_B) \mid \emptyset \neq B \subseteq B_q \right\} - \sum \left\{ \kappa^{\text{out}}(\nu_B) \mid B \cap B_{q'} \neq \emptyset \right\}
\]
Because $T$ satisfies the hypothesis of Lemma 56, we also have $T^\text{max}(A_p \cup B_q) = -T^\text{min}(A_{p'} \cup B_{q'})$, so that:

$$T^\text{max}(A_p \cup B_q) + \sum \left\{ \kappa^\text{in}(\nu_A) \mid A \cap A_p \neq \emptyset \right\} + \sum \left\{ \kappa^\text{out}(\nu_A) \mid \emptyset \neq A \subseteq A_{p'} \right\} =$$

$$-T^\text{min}(A_{p'} \cup B_{q'}) - \sum \left\{ \kappa^\text{out}(\nu_B) \mid \emptyset \neq B \subseteq B_q \right\} - \sum \left\{ \kappa^\text{out}(\nu_B) \mid B \cap B_{q'} \neq \emptyset \right\}$$

Switching summations in the preceding equality, we get:

$$T^\text{max}(A_p \cup B_q) + \sum \left\{ \kappa^\text{in}(\nu_A) \mid A \cap A_p \neq \emptyset \right\} + \sum \left\{ \kappa^\text{out}(\nu_B) \mid \emptyset \neq B \subseteq B_q \right\} =$$

$$-T^\text{min}(A_{p'} \cup B_{q'}) - \sum \left\{ \kappa^\text{in}(\nu_A) \mid \emptyset \neq A \subseteq A_{p'} \right\} - \sum \left\{ \kappa^\text{out}(\nu_B) \mid B \cap B_{q'} \neq \emptyset \right\}$$

Hence, the right-hand sides of $\text{Eq}(\text{max}, p, q)$ and $\text{Eq}(\text{min}, p', q')$ are the same, as desired. \qed

**Definition 66 (Complements and Conjunctions of Characteristic Functions).** Let $p, p' : \{1, \ldots, m\} \to \{0, 1\}$.

The complement $\bar{p}$ of $p$ is the characteristic function given by:

$$\bar{p}(i) = \begin{cases} 0 & \text{if } p(i) = 1, \\ 1 & \text{if } p(i) = 0, \end{cases}$$

for all $1 \leq i \leq m$. Thus, if $p$ is the characteristic function of the set $A \subseteq \mathbf{A}_\text{in}$, then $\bar{p}$ is the characteristic function of $(\mathbf{A}_\text{in} - A)$. Let $0$ denote the $m$-vector of all $0$'s and $1$ denote the $m$-vector of all $1$'s. We also define the conjunction $p \land p'$ of $p$ and $p'$:

$$(p \land p')(i) = \begin{cases} 1 & \text{if } p(i) = 1 = p'(i), \\ 0 & \text{otherwise}. \end{cases}$$

Thus, $p \land p' \neq 0$ iff $A_p \cap A_{p'} \neq \emptyset$. We use complement and conjunction of characteristic functions in the same way for subsets of $\mathbf{A}_\text{out}$. \qed

**Definition 67 (The Coefficient Matrix $C$).** We label the rows (and the columns) of the coefficient matrix $C$, from top to bottom (and from left to right), with pairs $(p, q)$ where $p$ and $q$ are characteristic functions of subsets of $\mathbf{A}_\text{in}$ and $\mathbf{A}_\text{out}$, respectively. We list these pairs in order according to “$<$” as given in Definition 63. Keep in mind this ordering is just the “reverse lexicographic order” on binary numbers, i.e., binary numbers read from right to left (see Figures 10 and 11).

For the rows, we use all pairs $(p, q)$ such that $p \neq 0$ and $q \neq 1$. For the columns, we use all pairs $(p, q)$ such that both $p \neq 0$ and $q \neq 0$. The result is the same number $\ell$ of rows and columns, where $\ell = (2^m - 1) \cdot (2^n - 1)$.

For a concrete example of our numbering of the rows and columns, see Figure 10: The rows are identified by the pairs $(p, q)$ on lines 1, 2, \ldots, 23 skipping lines 8 and 16, while the columns are identified by the pairs $(p, q)$ on lines 9, 10, \ldots, 31 skipping lines 16 and 24, and then see how they appear in Figure 11.

We denote the entry in row $(p, q)$ and column $(p', q')$ of matrix $C$ by $C[(p, q), (p', q')]$. Every entry $C[(p, q), (p', q')]$ is 0 or 1, determined according to $\text{Eq}(\text{max}, p, q)$ or $\text{Eq}(\text{min}, p, q)$ preceding Lemma 65. Specifically, if according to equation $\text{Eq}(\text{max}, p, q)$, we have:

$$C[(p, q), (p', q')] = \begin{cases} 1 & \text{if } p \land p' \neq 0 \text{ and } q \land q' \neq 0, \\ 0 & \text{otherwise}. \end{cases}$$

We can also define $C[(p, q), (p', q')]$ according to $\text{Eq}(\text{min}, p, q)$, but we do not need this alternative definition, because of Lemma 65. $C$ is a square matrix of $0$'s and $1$'s, which we next prove to be non-singular. \qed
Lemma 68. Let $P = (p_{i,j})_{1 \leq i,j \leq t}$ and $Q = (q_{i,j})_{1 \leq i,j \leq u}$ be square matrices of dimensions $t \times t$ and $u \times u$, respectively, for some integers $t, u \geq 1$. We construct a new square matrix of dimension $(t \cdot u) \times (t \cdot u)$, denoted $P[Q]$, by replacing $p_{i,j}$ in $P$ by a copy of $Q$ multiplied by the scalar $p_{i,j}$:

$$P[Q] = (p_{i,j} Q)_{1 \leq i,j \leq t}$$

i.e., $P[Q]$ consists of $t^2$ submatrices each of size $u \times u$, with the submatrix in position $(i, j)$ being a copy of $Q$.
of $Q$ multiplied by the scalar $p_{i,j}$. **Conclusion:** If $P$ and $Q$ are invertible, then so is $P[Q]$ invertible, with $(P[Q])^{-1} = P^{-1}[Q^{-1}]$.

**Proof.** This is a straightforward exercise in matrix algebra. Let $\text{eye}(u)$ be the $u \times u$ identity matrix (1’s along the diagonal, 0’s everywhere else) and let $\text{zeros}(u, u)$ be the $u \times u$ matrix of all 0’s. We need to prove that the matrix multiplication $(P[Q])(P^{-1}[Q^{-1}])$ is equal to the identity matrix of size $(t \cdot u) \times (t \cdot u)$. Let $P^{-1} = (p'_{i,j})_{1 \leq i,j \leq t}$. Hence, the submatrix in position $(i,j)$ in $(P[Q])(P^{-1}[Q^{-1}])$ is:

$$
\begin{align*}
p_{i,1}p'_{1,j}QQ^{-1} + p_{i,2}p'_{2,j}QQ^{-1} + \cdots + p_{i,t}p'_{t,j}QQ^{-1} &= (p_{i,1}p'_{1,j} + p_{i,2}p'_{2,j} + \cdots + p_{i,t}p'_{t,j})QQ^{-1} \\
&= \begin{cases} 
1 \cdot \text{eye}(u) = \text{eye}(u) & \text{if } i = j, \\
0 \cdot \text{eye}(u) = \text{zeros}(u, u) & \text{if } i \neq j,
\end{cases}
\end{align*}
$$

which implies the desired conclusion.\(^{10}\)

**Lemma 69.** Let $X = \{x_1, \ldots, x_n\}$ be a non-empty set of elements. The intersection matrix of $X$, denoted $R_X$, is a $0, 1$-matrix of size $(2^n - 1) \times (2^n - 1)$ whose rows and columns are labelled with all the non-empty subsets of $X$. For every $\emptyset \neq A, B \subseteq X$,

$$R_X[A, B] = \begin{cases} 
1 & \text{if } A \cap B \neq \emptyset, \\
0 & \text{if } A \cap B = \emptyset.
\end{cases}
$$

where $R[A, B]$ denotes the entry in row labelled $A$ and column labelled $B$. **Conclusion:** $R_X$ is invertible.

**Proof.** The order of the rows and columns of $R_X$ does not matter. For convenience, we choose the order induced by the characteristic functions $p : X \to \{0, 1\}$ according to “$<$” in Definition 63, which is “reverse lexicographic order” on binary numbers. The proof proceeds by induction on $|X| = n \geq 1$. For $n = 1$ there is nothing to prove. For $n = 2$, the order on characteristic functions is $(1, 0) < (0, 1) < (1, 1)$, so that $R_X = R_2$ is:

$$R_2 = \begin{bmatrix} 
1 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
$$

By inspection, the inverse of $R_2$ is:

$$(R_2)^{-1} = \begin{bmatrix} 
1 & 1 & 0 \\
0 & -1 & 1 \\
-1 & 0 & 1 \\
1 & 1 & -1 \\
\end{bmatrix}
$$

Proceeding inductively, assume we have shown that the intersection matrix $R_n$ for $n \geq 2$ is invertible, and we next show that $R_{n+1}$ is also invertible. The matrix $R_n$ is of size $(2^n - 1) \times (2^n - 1)$. To push the induction through, we need to add three invariants, for every $n \geq 1$:

1. Both $R_n$ and $R_n^{-1}$ are symmetric square matrices.
2. The entries of each of the first $(2^n - 2)$ rows, and each of the first $(2^n - 2)$ columns, in $R_n^{-1}$ add up to 0.
3. The entries of the last row, and separately the entries of the last column, in $R_n^{-1}$ add up to 1.

\(^{10}\)We borrow from Matlab the names of the special matrices $\text{eye}(t)$, $\text{zeros}(t, u)$, and $\text{ones}(t, u)$ – the identity $t \times t$ matrix, the $t \times u$ matrix of all 0’s, and the $t \times u$ matrix of all 1’s, respectively – for all integers $t, u \geq 1$. 

42
Clearly, these three invariants are true for $R_2$ and $R_2^{-1}$ above.

Relative to the “reverse lexicographic order” we have chosen on characteristic functions, we can define $R_{n+1}$ in terms of $R_n$ as follows, where $(\cdot)^T$ stands for the transpose operator on matrices:

$$R_{n+1} = \begin{bmatrix} R_n & \text{zeros}(2^n - 1, 1) & R_n \\ \text{zeros}(1, 2^n - 1) & 1 & \text{ones}(1, 2^n - 1) \\ R_n & \text{ones}(2^n - 1, 1) & \text{ones}(2^n - 1, 2^n - 1) \end{bmatrix}$$

We define the inverse of $R_{n+1}$ as follows:

$$(R_{n+1})^{-1} = \begin{bmatrix} \text{zeros}(2^n - 1, 2^n - 1) & \left[ \text{zeros}(1, 2^n - 2) - 1 \right]^T & R_n^{-1} \\ \text{zeros}(1, 2^n - 2) & -1 & 0 \\ R_n^{-1} & \left[ \text{zeros}(1, 2^n - 2) - 1 \right]^T & -R_n^{-1} \end{bmatrix}$$

The matrices $\left[ \text{zeros}(1, 2^n - 2) - 1 \right]$ and $\left[ \text{zeros}(1, 2^n - 2) - 1 \right]^T$ are of size $1 \times (2^n - 1)$, consisting of all 0’s except for the last entry in each which is 1 or $-1$. It is now easy to check that $R_{n+1}$ and $(R_{n+1})^{-1}$ are matrices of size $(2^{n+1} - 1) \times (2^{n+1} - 1)$ satisfying the the invariants (1), (2), and (3).

It remains to confirm that $R_{n+1}$ and $(R_{n+1})^{-1}$ are inverses of each other. If we carry out the matrix multiplication $R_{n+1} (R_{n+1})^{-1}$, also using invariants (1), (2), and (3) for $R_n$ and $R_n^{-1}$, we obtain:

$$R_{n+1} (R_{n+1})^{-1} = \begin{bmatrix} R_n R_n^{-1} & \text{zeros}(2^n - 1, 1) & R_n R_n^{-1} - R_n R_n^{-1} \\ \text{zeros}(1, 2^n - 1) & 1 & \text{zeros}(1, 2^n - 1) \\ \text{zeros}(2^n - 1, 2^n - 1) & \text{zeros}(2^n - 1, 1) & R_n R_n^{-1} \end{bmatrix}$$

$$= \begin{bmatrix} \text{eye}(2^n - 1) & \text{zeros}(2^n - 1, 1) & \text{zeros}(2^n - 1, 2^n - 1) \\ \text{zeros}(1, 2^n - 1) & 1 & \text{zeros}(1, 2^n - 1) \\ \text{zeros}(2^n - 1, 2^n - 1) & \text{zeros}(2^n - 1, 1) & \text{eye}(2^n - 1) \end{bmatrix}$$

which concludes the inductive step and the proof of the lemma.

**Lemma 70.** The coefficient matrix $C$ is invertible.

**Proof.** As assumed throughout, $A_{\text{in}} = \{a_1, \ldots, a_m\}$ and $A_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\}$, for some $m, n \geq 1$. In the notation of Lemma 69, let $R_{A_{\text{in}}} = R_m$ and $R_{A_{\text{out}}} = R_n$ be the intersection matrices induced by $A_{\text{in}}$ and $A_{\text{out}}$, respectively. Both $R_m$ and $R_n$ are invertible, as well as any matrix obtained from these by permuting rows and columns.

Let $S_n$ be the $(0,1)$-matrix whose columns are labelled with characteristic functions $q$ of subsets of an $n$-element set such that $q \neq \emptyset$ and ordered according to “$<$”. We also label the rows of $S_n$ with characteristic functions $p$ of subsets of an $n$-element set, but such that $p \neq 1$. Specifically, the labelling of the rows of $S_n$,
from top to bottom, is done according to:

| label of row 1: | \( (1, 1, 1, \ldots, 1) \) | complement of \( p_0 \) = \( (0, 0, 0, \ldots, 0) \) |
| label of row 2: | \( (0, 1, 1, \ldots, 1) \) | complement of \( p_1 \) = \( (1, 0, 0, \ldots, 0) \) |
| label of row 3: | \( (1, 0, 1, \ldots, 1) \) | complement of \( p_2 \) = \( (0, 1, 0, \ldots, 0) \) |
| label of row 4: | \( (0, 0, 1, \ldots, 1) \) | complement of \( p_3 \) = \( (1, 1, 0, \ldots, 0) \) |
| ... | ... | ... |
| label of row \((2^n - 1)\): | \( (1, 0, 0, \ldots, 0) \) | complement of \( p_{2^n - 1} \) = \( (0, 1, 1, \ldots, 1) \) |

where the sequence in the rightmost column is ordered by \( < \), i.e., \( p_0 < p_1 < p_2 < p_3 < \cdots < p_{2^n - 1} \). It is easy to see that \( S_n \) is the intersection matrix \( R_n \) after reflecting (or rotating) \( R_n \) around row \( 2^{n-1} \), with \((2^{n-1} - 1)\) rows above it and \((2^{n-1} - 1)\) rows below it.

The coefficient matrix \( C \) is just \( S_n[R_m] \), following the construction in the statement of Lemma 68. In Definition 67, keep in mind that characteristic functions \( p \) of subsets of \( A_{in} \) are all \( \neq 0 \), both horizontally and vertically, while characteristic functions \( q \) of subsets of \( A_{out} \) are all \( \neq 1 \) vertically and all \( \neq 0 \) horizontally. (For a concrete example, consider \( C \) in Figure 11, where \( C = S_2[R_3] \).) This implies the desired conclusion. \( \square \)

**Remark 71.** An alternative proof of Lemma 70 is to show that the determinant of \( C \) is not zero. In fact, \( \det(C) = \det(C^{-1}) = -1 \), for all \( n \geq 1 \). Although somewhat shorter, this alternative proof is less direct because it invokes properties from the theory of determinants which require their own proofs. We give a sketch below, where we use two facts:

1. Repeated subtractions of columns from other columns, and rows from other rows, do not change the value of the determinant.
2. Repeated exchanges of columns do not change the absolute value of the determinant, only its sign (the sign is switched from \(+ \) to \(- \), or from \(- \) to \(+ \), in every exchange).

Starting with intersection matrix \( R_n \), we subtract column \( 2^{n-1} \) from all the columns to its right, i.e., columns in positions \( 2^{n-1} + 1, 2^{n-1} + 2, \ldots, 2^{n-1} + 2^{n-1} - 1 = 2^n - 1 \). We next subtract row \( 2^{n-1} \) from all the rows below it. Finally, using the corresponding permutations, we exchange column 1 with column \( 2^{n-1} + 1 \), column 2 with column \( 2^{n-1} + 2, \ldots, \), column \( 2^{n-1} - 1 \) with column \( 2^{n-1} + 2^{n-1} - 1 = 2^n - 1 \). The resulting matrix \( M \) is:

\[
M = \begin{bmatrix} R_{n-1} & N \\ P & Q \end{bmatrix}
\]

where

\[
N = \begin{bmatrix} \text{zeros}(2^{n-1} - 1, 1) & R_{n-1} \end{bmatrix},
\]

\[
P = \text{zeros}(2^{n-1}, 2^{n-1} - 1),
\]

\[
Q = \begin{bmatrix} 1 & \text{zeros}(1, 2^{n-1} - 1) \\ \text{ones}(2^{n-1} - 1, 1) & R_{n-1} \end{bmatrix}.
\]

We thus have \( \det(M) = \det(R_{n-1}) \cdot 1 \cdot \det(R_{n-1}) \). If \( \det(R_{n-1}) = -1 \), then \( \det(M) = 1 \). But \( M \) is obtained by an odd number of column-exchanges starting from the initial \( R_n \). Hence, \( \det(R_n) = -1 \). \( \square \)

**Definition 72 (The Vector of Constants \( d \)).** The vector of constants \( d \) has \( \ell = (2^n - 1) \cdot (2^n - 1) \) entries, which are ordered according to the ordering “\( < \)” on pairs \( (p, q) \) such that \( p \neq 0 \) and \( q \neq 1 \). Each \( p \) is a characteristic function of a subset of \( A_{in} \), and each \( q \) is a characteristic function of a subset of \( A_{out} \). We denote the entry of \( d \) in position \( (p, q) \) by \( d(\{p, q\}) \), which is determined according to equation Eq(max, \( p, q \)) preceding Lemma 65:

\[
d(\{p, q\}) = T^{max}(A_p \cup B_q) + \sum \{ \kappa^{in}(\nu_A) \mid A \cap A_p \neq \emptyset \} + \sum \{ \kappa^{out}(\nu_B) \mid \emptyset \neq B \subseteq B_q \},
\]
with \( p \) and \( q \) the characteristic functions of \( A_p \in \mathcal{P}(A_{\text{in}}) - \{\emptyset\} \) and \( B_q \in \mathcal{P}(A_{\text{out}}) - \{A_{\text{out}}\} \), respectively. \( \square \)

The matrix equation \( Cx = d \) has a unique solution for the vector \( x \), because \( C \) is invertible. Specifically, \( x = C^{-1}d \). We label rows and columns of the inverse matrix \( C^{-1} \) in the same way as \( C \). We denote row \((p, q)\) of the inverse \( C^{-1} \), where \( p \neq 0 \) and \( q \neq 1 \), by writing \( C^{-1}[(p, q), \star] \). We can therefore write for the solution of the matrix equation \( x = C^{-1}d \) the following \( \ell = (2^m - 1) \cdot (2^n - 1) \) assignments – we write \( \bar{x}(A, B) \) for the value assigned to variable \( x(A, B) \):

\[
\bar{x}(A_{p_1}, B_{q_1}) = C^{-1}[(p_1', q_1'), \star] \cdot d \\
\bar{x}(A_{p_2}, B_{q_2}) = C^{-1}[(p_2', q_2'), \star] \cdot d \\
\vdots \\
\bar{x}(A_{p_{\ell}}, B_{q_{\ell}}) = C^{-1}[(p_{\ell}', q_{\ell}'), \star] \cdot d
\]

where “\( \cdot \)” is the dot product of vectors of the same dimension \( \ell \) and:

\[
(p_1, q_1) < (p_2, q_2) < \cdots < (p_{\ell}, q_{\ell}) \quad \text{with } p_i \neq 0 \text{ and } q_i \neq 0 \text{ for every } 1 \leq i \leq \ell,
\]

\[
(p_1', q_1') < (p_2', q_2') < \cdots < (p_{\ell}', q_{\ell}') \quad \text{with } p_i' \neq 0 \text{ and } q_i' \neq 1 \text{ for every } 1 \leq i \leq \ell.
\]

We are now ready to assign capacities to the arcs in \( A_{\#, \#} \). For every \( \emptyset \neq A_p \subset A_{\text{in}} \) and every \( \emptyset \neq B_q \subset A_{\text{out}} \):

- If \( \bar{x}(A_p, B_q) \geq 0 \), then
  \[
  \text{LC}(\nu_{A_p}, \nu_{B_q}) = 0 \quad \text{and} \quad \text{UC}(\nu_{A_p}, \nu_{B_q}) = \bar{x}(A_p, B_q).
  \]

- If \( \bar{x}(A_p, B_q) < 0 \), then
  \[
  \text{LC}(\text{reverse}(\nu_{A_p}, \nu_{B_q})) = \text{UC}(\text{reverse}(\nu_{A_p}, \nu_{B_q})) = -\bar{x}(A_p, B_q).
  \]

Note the asymmetry: If \( \bar{x}(A_p, B_q) \geq 0 \), then the lower-bound capacity is 0, but if \( \bar{x}(A_p, B_q) < 0 \), then the lower-bound and upper-bound capacities are equal. Hence, in the former case, a feasible flow may use arc \( \nu_{A_p}, \nu_{B_q} \) with a quantity ranging from 0 up to its full upper-bound capacity \( \bar{x}(A_p, B_q) \); in the latter case, all feasible flows must use arc \( \text{reverse}(\nu_{A_p}, \nu_{B_q}) = \nu_{B_q}, \nu_{A_p} \) with the same quantity \(-\bar{x}(A_p, B_q)\), introducing a cycle in \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) that must be used by every feasible flow.

### 9.3 Assigning Remaining Capacities

Using the notation of Section 8, for every arc \( a \in A_{\text{in}} \cup A_{\text{out}} \cup A_{\text{in}, \#} \cup A_{\#, \text{out}} \), we set:

\[
\text{LC}(a) = 0 \quad \text{and} \quad \text{UC}(a) = K.
\]

where \( K \) is a fixed, but otherwise arbitrary, “very large number”. This completes stage (IV) and turning \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) into a flow network. Example 73 illustrates the construction.

**Proof of Lemma 56.** Let \( T \) be a tight and total typing satisfying the hypothesis of Lemma 56. Consider the special graph \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) defined in Section 8, which is turned into a flow network using the given \( T \), as done so far in this section. To conclude the proof, we show that the principal typing \( \bar{T} \) for \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) is none other than the given \( T \).
For arbitrary $A_p \subseteq A_{in}$ and $B_q \subseteq A_{out}$, we show that $T(A_p \cup B_q) = T(A_p \cup B_q)$. If $A_p \cup B_q = \emptyset$ or $A_p \cup B_q = A_{in/out}$, this is immediate, because $T(A_p \cup B_q) = T(A_p \cup B_q) = 0$. Consider therefore the case when $\emptyset \neq A_p \cup B_q \neq A_{in/out}$. Define the objective function $\theta_{p,q}$ as follows:

$$\theta_{p,q} = \sum_{a} a | a \in A_p - \sum_{b} b | b \in B_q$$

where, as in earlier sections, we use arc names as variables of an objective function. The maximum possible value of $\theta_{p,q}$ is $\Gamma^{\max}(A_p \cup B_q)$, and the minimum possible value of $\theta_{p,q}$ is $\Gamma^{\min}(A_p \cup B_q)$.

We can determine the maximum and minimum of the objective $\theta_{p,q}$ by standard methods of linear programming, to maximize and minimize $\theta_{p,q}$ relative to a set $E$ of equations enforcing flow conservation at nodes and a set $C$ of inequalities enforcing capacity constraints on the arcs.

However, an easier and more perspicuous approach is to directly consider $\text{Graph}(A_{in}, A_{out})$, which we have follow. By our construction of $\text{Graph}(A_{in}, A_{out})$, it is readily checked that:

$$\Gamma^{\max}(A_p \cup B_q) = + \sum \left\{ \text{UC}(\nu_A, \nu_B) \mid A \cap A_p \neq \emptyset \text{ and } B \subseteq A_{out} - B_q \right\}$$

$$- \sum \left\{ \text{LC}(\nu_B, \nu_A) \mid A \cap B \neq \emptyset \text{ and } B \subseteq A_{out} - B_q \right\}$$

$$- \sum \left\{ \kappa_{in}(\nu_A) \mid A \cap A_p \neq \emptyset \right\}$$

$$- \sum \left\{ \kappa_{out}(\nu_B) \mid B \subseteq B_q \right\}$$

$$\Gamma^{\min}(A_p \cup B_q) = - \sum \left\{ \text{UC}(\nu_A, \nu_B) \mid A \cap (A_{in} - A_p) \neq \emptyset \text{ and } B \cap B_q \neq \emptyset \right\}$$

$$+ \sum \left\{ \text{LC}(\nu_B, \nu_A) \mid A \cap (A_{in} - A_p) \neq \emptyset \text{ and } B \cap B_q \neq \emptyset \right\}$$

$$- \sum \left\{ \kappa_{in}(\nu_A) \mid \emptyset \neq A \subseteq A_p \right\}$$

$$- \sum \left\{ \kappa_{out}(\nu_B) \mid B \cap B_q \neq \emptyset \right\}.$$
From our definition of the equations \( \text{Eq}(\max, p, q) \) and \( \text{Eq}(\min, p, q) \), this implies:

\[
\overline{T}^\max(A_p \cup B_q) = T^\max(A_p \cup B_q) \quad \text{and} \quad \overline{T}^\min(A_p \cup B_q) = T^\min(A_p \cup B_q)
\]

which is the desired conclusion. This completes the proof of Lemma 56.

We conclude with two examples to illustrate the methodology of this section: Given a tight, total, and valid, typing \( T \), i.e., a typing satisfying the conditions of Theorem 57, we construct a network for which \( T \) is principal.

The two examples exhibit different aspects of our methodology. The first, Example 73, completes a construction started in Examples 61 and 64.

**Example 73.** In addition to Examples 61 and 64, this should be read in conjunction with Examples 13 and 46, where \( \mathcal{A}_{\text{in}} = \{a_1, a_2, a_3\} \) and \( \mathcal{A}_{\text{out}} = \{a_4, a_5\} \). For this case, the coefficient matrix \( C \) is shown in Figure 11. To compute the vector of constants \( d \), we use the consumer/producer assignments \( \kappa^{\text{in}} \) and \( \kappa^{\text{out}} \), already determined in Example 61. Together with \( T_2^{\max} \), based on the formula in Definition 72. \( T_2 \) here is the particular typing computed in Example 46 and used in Example 61. The resulting entries of \( d \) are shown in Figure 13.

With the vector of constants \( d \) in hand, we compute the capacities \( \text{LC}(\nu_A, \nu_B) \) and \( \text{UC}(\nu_A, \nu_B) \) for every arc \( \nu_{\#} \) in \( \mathcal{A}_{\#, \#} \). We use Matlab (or a similar package) to compute \( \text{C}^{-1}d \) and solve for the vector of variables \( x \). The resulting solution assigns 0 to every variable \( x(A, B) \) where \( \emptyset \neq A \subseteq \mathcal{A}_{\text{in}} \) and \( \emptyset \neq B \subseteq \mathcal{A}_{\text{out}} \), except for the following four:

\[
\bar{x}(\{a_1\}, \{a_5\}) = 1, \quad \bar{x}(\{a_1, a_2\}, \{a_5\}) = 1,
\]

\[
\bar{x}(\{a_1, a_2\}, \{a_4, a_5\}) = 1, \quad \bar{x}(\{a_1, a_2, a_3\}, \{a_4, a_5\}) = 4.
\]

The finalized network is shown in Figure 12, which we call \( N_2' \) for later reference. In this example, no variable \( x(A, B) \) is assigned a negative value, implying that no cycle is introduced in \( N_2' = \text{Graph}(\mathcal{A}_{\text{in}}, \mathcal{A}_{\text{out}}) \).

We can set up a set of equations \( \mathcal{E} \) enforcing flow conservation at the nodes, and a set of inequalities \( \mathcal{C} \) enforcing lower-bound and upper-bound constraints on the arcs, of \( N_2' \). By brute-force inspection (very tedious!) or by using a standard linear-programming package, we can compute a principal typing \( \overline{T} \) relative to \( \mathcal{E} \cup \mathcal{C} \), according to Procedure 45. As expected, the resulting \( \overline{T} \) is the same as the initial typing \( T_2 \), which we used to define lower-bound and upper-bound capacities for \( N_2' = \text{Graph}(\mathcal{A}_{\text{in}}, \mathcal{A}_{\text{out}}) \). We have thus obtained another network \( N_2' \), shown in Figure 12, which is equivalent to the network \( N_2 \) in Figure 2.

![Figure 12](image-url)

**Figure 12:** The special graph \( N_2' = \text{Graph}(\mathcal{A}_{\text{in}}, \mathcal{A}_{\text{out}}) \) when \( \mathcal{A}_{\text{in}} = \{a_1, a_2, a_3\} \) and \( \mathcal{A}_{\text{out}} = \{a_4, a_5\} \) in Example 73, with capacities induced by typing \( T_2 \) in Example 46. Nodes and arcs in dotted lines from Figures 8 and 9 are now dead. Missing lower-bound capacities are 0, missing upper-bound capacities are the “very large number” \( K \).
\[ \mathbf{d}(1,0,0),(0,0) = T_{\text{max}}(\{a_1\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) = 10 - 2 - 1 = 7 \]

\[ \mathbf{d}(0,1,0),(0,0) = T_{\text{max}}(\{a_2\}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) = 7 - 1 = 6 \]

\[ \mathbf{d}(1,1,0),(0,0) = T_{\text{max}}(\{a_1,a_2\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) = 10 - 2 - 1 = 7 \]

\[ \mathbf{d}(0,0,1),(0,0) = T_{\text{max}}(\{a_3\}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) = 9 - 4 - 1 = 4 \]

\[ \mathbf{d}(1,0,1),(0,0) = T_{\text{max}}(\{a_1,a_3\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) = 14 - 2 - 4 - 1 = 7 \]

\[ \mathbf{d}(0,1,1),(0,0) = T_{\text{max}}(\{a_2,a_3\}) + \kappa_{\text{in}}(\nu_{\{a_2\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) = 11 - 4 - 1 = 6 \]

\[ \mathbf{d}(1,1,1),(0,0) = T_{\text{max}}(\{a_1,a_2,a_3\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) = 14 - 2 - 4 - 1 = 7 \]

\[ \mathbf{d}(1,0,0),(1,0) = T_{\text{max}}(\{a_1,a_4\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_4\}}) = 7 - 2 - 1 + 3 = 7 \]

\[ \mathbf{d}(0,1,0),(1,0) = T_{\text{max}}(\{a_2,a_4\}) + \kappa_{\text{in}}(\nu_{\{a_2\}}) + \kappa_{\text{out}}(\nu_{\{a_4\}}) = 4 - 1 + 3 = 6 \]

\[ \mathbf{d}(1,1,0),(1,0) = T_{\text{max}}(\{a_1,a_2,a_4\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_2,a_3,a_4\}}) + \kappa_{\text{out}}(\nu_{\{a_4\}}) = 7 - 2 - 1 + 3 = 7 \]

\[ \mathbf{d}(0,0,1),(1,0) = T_{\text{max}}(\{a_3,a_4\}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_4\}}) = 6 - 4 - 1 + 3 = 4 \]

\[ \mathbf{d}(1,0,1),(1,0) = T_{\text{max}}(\{a_1,a_3,a_4\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_4\}}) = 11 - 2 - 4 - 1 + 3 = 7 \]

\[ \mathbf{d}(0,1,1),(1,0) = T_{\text{max}}(\{a_2,a_3,a_4\}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_4\}}) = 8 - 4 - 1 + 3 = 6 \]

\[ \mathbf{d}(1,1,1),(1,0) = T_{\text{max}}(\{a_1,a_2,a_3,a_4\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_4\}}) = 11 - 2 - 4 - 1 + 3 = 7 \]

\[ \mathbf{d}(1,0,0),(0,1) = T_{\text{max}}(\{a_1,a_5\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_5\}}) = 4 - 2 - 1 + 4 = 5 \]

\[ \mathbf{d}(0,1,0),(0,1) = T_{\text{max}}(\{a_2,a_5\}) + \kappa_{\text{in}}(\nu_{\{a_2\}}) + \kappa_{\text{out}}(\nu_{\{a_5\}}) = 2 - 1 + 4 = 5 \]

\[ \mathbf{d}(1,1,0),(0,1) = T_{\text{max}}(\{a_1,a_2,a_5\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_5\}}) = 4 - 2 - 1 + 4 = 5 \]

\[ \mathbf{d}(0,0,1),(0,1) = T_{\text{max}}(\{a_3,a_5\}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_5\}}) = 5 - 4 - 1 + 4 = 4 \]

\[ \mathbf{d}(1,0,1),(0,1) = T_{\text{max}}(\{a_1,a_3,a_5\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_5\}}) = 8 - 2 - 4 - 1 + 4 = 5 \]

\[ \mathbf{d}(0,1,1),(0,1) = T_{\text{max}}(\{a_2,a_3,a_5\}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_5\}}) = 6 - 4 - 1 + 4 = 5 \]

\[ \mathbf{d}(1,1,1),(0,1) = T_{\text{max}}(\{a_1,a_2,a_3,a_5\}) + \kappa_{\text{in}}(\nu_{\{a_1\}}) + \kappa_{\text{in}}(\nu_{\{a_3\}}) + \kappa_{\text{in}}(\nu_{\{a_1,a_2,a_3\}}) + \kappa_{\text{out}}(\nu_{\{a_5\}}) = 8 - 2 - 4 - 1 + 4 = 5 \]

Figure 13: The vector of constants \( \mathbf{d} \) in Example 73. \( T_{\text{max}} \) here is the particular \( T_{2\text{max}} \) in Example 73.
Example 74. This continues Examples 15 and 48, where $A_{in} = \{a_1, a_2\}$ and $A_{out} = \{a_3, a_4\}$. We use the typing $T_4$ computed in Example 48 to construct a special network $\text{Graph}(A_{in}, A_{out})$, which we call $\mathcal{N}'_4$ for later reference. Recall that $T_4$ is a tight, total, and principal, typing for network $\mathcal{N}_4$ in Example 15. We omit most of the intermediate details, which are very much the same as in Example 73, and only show the final result.

This is the network on the left in Figure 14.

In Figure 14, as in Figure 8, we only show the label (subscript) of each node for succinctness, i.e., we write the label “$A$” not the full name “$\nu_A$” where $\emptyset \neq A \subseteq \{a_1, a_2\}$ or $\emptyset \neq A \subseteq \{a_3, a_4\}$.

Because all lower-bounds in network $\mathcal{N}_4$ in Example 15 are zero, there is no assignment of lower-bound capacities for the corresponding special $\text{Graph}(A_{in}, A_{out})$. Hence, in contrast to Example 73, we can skip stages (I) and (II) in Section 9.1 and go directly to stage (III) in Section 9.2.

But, again in contrast to Example 73, one of the variables $x(A, B)$ is assigned a negative value, namely, $\bar{x}(\{a_2\}, \{a_3, a_4\}) = -2$. By our construction in stage (III), this introduces a backward arc $\langle \nu_{\{a_3, a_4\}}, \nu_{\{a_2\}} \rangle$, such that:

$$\text{LC}(\nu_{\{a_3, a_4\}}, \nu_{\{a_2\}}) = UC(\nu_{\{a_3, a_4\}}, \nu_{\{a_2\}}) = 2.$$

The arc $\langle \nu_{\{a_3, a_4\}}, \nu_{\{a_2\}} \rangle$ is shown as a dashed arrow in Figure 14 on the left. If we wish, we can get rid of the backward arc with a non-zero lower bound, using the procedure of Proposition 7. This makes node $\nu_{\{a_2\}}$ a producer of two units, and node $\nu_{\{a_3, a_4\}}$ a consumer of two units. The result is shown on the right in Figure 14.

Using Procedure 45 in Section 6, or else by brute-force inspection, a tight, total, and principal typing for the two networks in Figure 14 is the same and turns out to be equal to $T_4$ – just as expected.

A significant aspect of our methodology is that, while it produces a special network $\mathcal{N}'_4 = \text{Graph}(A_{in}, A_{out})$ equivalent to the original $\mathcal{N}_4$, the resulting $\mathcal{N}'_4$ contains an active cycle (which, if so wished, can be eliminated by introducing producer/consumer nodes), in contrast to the original $\mathcal{N}_4$ which contains no cycles and no producer/consumer nodes. This raises important open questions we discuss in Section 11.2.

Figure 14: The special $\mathcal{N}'_4 = \text{Graph}(A_{in}, A_{out})$ when $A_{in} = \{a_1, a_2\}$ and $A_{out} = \{a_3, a_4\}$ in Example 74, with capacities induced by typing $T_4$ in Example 48, on the left, and after elimination of the (dashed) backward arc on the right. Missing lower bounds are 0, missing upper bounds are the “very large number” $K$.

10 Operating on Network Typings

There are two basic ways in which we can assemble and connect networks together:

1. Let $\mathcal{N}$ and $\mathcal{N}'$ be two networks, with outer arcs $A_{in,out} = A_{in} \cup A_{out}$ and $A'_{in,out} = A'_{in} \cup A'_{out}$, respectively.

The parallel addition of $\mathcal{N}$ and $\mathcal{N}'$, denoted $\mathcal{N} \parallel \mathcal{N}'$, simply places $\mathcal{N}$ and $\mathcal{N}'$ next to each other without
connecting any of their outer arcs. The input and output arcs of \( N \parallel N' \) are \( A_{\text{in}} \cup A'_{\text{in}} \) and \( A_{\text{out}} \cup A'_{\text{out}} \), respectively.

2. Let \( N' \) be a network with outer arcs \( A_{\text{in, out}} = A_{\text{in}} \cup A_{\text{out}} \), and let \( a \in A_{\text{in}} \) and \( b \in A_{\text{out}} \). The binding of output arc \( b \) to input arc \( a \), denoted \( \text{bind}_{(a, b)}(N') \), means to connect head \( (b) \) to tail \( (a) \) and thus set \( \text{tail}(a) = \text{head}(b) \). The input and output arcs of \( \text{bind}_{(a, b)}(N') \) are \( A'_{\text{in}} = A_{\text{in}} - \{a\} \) and \( A'_{\text{out}} = A_{\text{out}} - \{b\} \), respectively.

Many natural assemblies of finitely many networks can be obtained by applying the two preceding operations repeatedly. For example, if we want to build a new network from previously defined networks \( N \) and \( N' \), by connecting the output arcs \( b, b' \) and \( b'' \) in \( N' \) to input arcs \( a, a' \) and \( a'' \) in \( N' \), then we can write:

\[
\text{bind}_{(a'', b'')}\left(\text{bind}_{(a', b')}(\text{bind}_{(a, b)}(N \parallel N'))\right)
\]

For another example, if we want to merge two output arcs \( b \) and \( b' \) in an already defined \( N' \), we introduce a new single-node network \( N'' = (\{\nu\}, \{a, a', a''\}) \) such that:

\[
\text{head}(a) = \text{head}(a') = \nu \quad \text{and} \quad \text{tail}(a'') = \nu
\]

i.e., \( a \) and \( a' \) are the two input arcs of \( N' \) and \( a'' \) is its sole output arc. Merging \( b \) and \( b' \) can be written as:

\[
\text{bind}_{(a', b')}(\text{bind}_{(a, b)}(N \parallel N''))
\]

In the resulting network, \( A_{\text{in}} \) is unchanged but the new set of output arcs is \( (A_{\text{out}} - \{b, b'\}) \cup \{a''\} \).

### 10.1 The Lattice of Tight, Total, and Valid Typings

Given a set of input/output arcs \( A_{\text{in, out}} \), we specify the pieces we need in order to organize all tight, total, and valid typings over \( A_{\text{in, out}} \) as a distributive lattice. We omit many of the straightforward details in the proofs. This will in turn facilitate the process of inferring a valid typing for a network from valid typings of its components.

**Definition 75 (Unrestricted Networks).** Let \( A_{\text{in}} = \{a_1, \ldots, a_m\} \) and \( A_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\} \) with \( m, n \geq 1 \). The unrestricted (or universal or free) network over \( A_{\text{in, out}} = A_{\text{in}} \cup A_{\text{out}} \), denoted \( \mathcal{U}(A_{\text{in, out}}) \), has no internal arcs and only one node \( \nu \) such that:

1. The head of every input arc \( a \in A_{\text{in}} \) is \( \nu \), i.e., \( \text{head}(a) = \nu \).
2. The tail of every output arc \( a \in A_{\text{out}} \) is \( \nu \), i.e., \( \text{tail}(a) = \nu \).
3. The lower-bound capacity of every outer arc \( a \in A_{\text{in, out}} \) is 0.
4. The upper-bound capacity of every outer arc \( a \in A_{\text{in, out}} \) is the “very large number” \( K \).

We call \( \mathcal{U}(A_{\text{in, out}}) \) “unrestricted” because every typing satisfying the hypothesis of Lemma 56 is a valid typing of \( \mathcal{U}(A_{\text{in, out}}) \), though not necessarily principal. (For particular unrestricted networks, each over a different \( A_{\text{in, out}} \), see Examples 92 and 97 below.)

**Convention 76.** “\( \mathcal{U}(A_{\text{in, out}}) \)” is an ambiguous denotation of the unrestricted network over \( A_{\text{in, out}} \) because it does not distinguish between input and output arcs. To disambiguate it, we have to write “\( \mathcal{U}(A_{\text{in}}, A_{\text{out}}) \)”, which we nonetheless avoid for economy of notation. The context will always make clear which members of \( A_{\text{in, out}} \) are input arcs and which are output arcs.

---

11There are other ways of assembling flow networks, which we briefly discuss in relation to future work in Section 11.3.
**Lemma 77.** Consider the unrestricted network $\mathcal{U}(A_{in, out})$ over $A_{in, out} = A_{in} \uplus A_{out}$ where $|A_{in}| = m \geq 1$ and $|A_{out}| = n \geq 1$. Define the following typing $T : \mathcal{P}(A_{in, out}) \to \mathcal{I}(\mathbb{R})$:

1. $T(\emptyset) = T(A_{in, out}) = [0, 0]$.
2. For every $\emptyset \neq A \subseteq A_{in, out}$ with $|A \cap A_{in}| = p \geq 0$ and $|A \cap A_{out}| = q \geq 0$, let $T(A) = [r_1, r_2]$ where:
   
   \[ r_1 = - \min \{ (m-p) \cdot K, q \cdot K \} = - K \cdot \left( \min \{ (m-p), q \} \right) \]
   
   \[ r_2 = + \min \{ p \cdot K, (n-q) \cdot K \} = + K \cdot \left( \min \{ p, (n-q) \} \right) \]

**Conclusion:** $T$ is a tight, total, and principal typing for $\mathcal{U}(A_{in, out})$.

For later reference, we call the typing $T$ for $\mathcal{U}(A_{in, out})$ as just defined the unrestricted typing over $A_{in, out}$ and denote it $\mathcal{T}[A_{in, out}]$.

**Proof.** $T$ is total because it is defined for every $A \in \mathcal{P}(A_{in, out})$. It is also readily seen to be tight – or we can trivially apply algorithm Tight() of Proposition 21 to find that Tight($T$) = $T$. Finally, it is principal because Procedure 45 infers a typing for $\mathcal{U}(A_{in, out})$ which is equal to $T$ – or we can obtain it directly by inspection.

At the opposite end of the unrestricted typing $\mathcal{T}[A_{in, out}]$ defined in Lemma 77 are the “smallest” or most restrictive typings over $A_{in, out}$, whose types are intervals consisting of a single real.

**Definition 78 (Non-Total Atomic Typings).** A typing $T : \mathcal{P}(A_{in, out}) \to \mathcal{I}(\mathbb{R})$ over $A_{in, out}$ is atomic if:

- For every $a \in A_{in}$ there is a number $r \in \mathbb{R}^+$ such that $T(\{a\}) = [r, r] = \{r\}$.
- For every $a \in A_{out}$ there is a number $r \in \mathbb{R}^+$ such that $T(\{a\}) = [-r, -r] = \{-r\}$.

Hence, in particular, $T^{\min}(\{a\}) = T^{\max}(\{a\})$ for every $a \in A_{in, out}$. As defined so far, an atomic typing $T$ is not total as it assigns types to singleton subsets of $A_{in, out}$ only; but, by the next lemma, $\text{Poly}(T)$ is fully specified by its singleton assignments and there is no need to explicitly assign types to non-singleton subsets.

**Lemma 79.** Let $T : \mathcal{P}(A_{in, out}) \to \mathcal{I}(\mathbb{R})$ be a non-total atomic typing such that:

\[ \sum \{ T^{\max}(\{a\}) \mid a \in A_{in, out} \} = 0. \]

Define a new typing $\overline{T} : \mathcal{P}(A_{in, out}) \to \mathcal{I}(\mathbb{R})$ by setting $\overline{T}(\emptyset) = [0, 0]$ and for every $\emptyset \neq A \subseteq A_{in, out}$:

\[ \overline{T}^{\min}(A) = \overline{T}^{\max}(A) = \sum \{ T^{\max}(\{a\}) \mid a \in A \}. \]

**Conclusion:** It holds that

1. $\overline{T}$ is tight and total.
2. $\text{Poly}(T) = \text{Poly}(\overline{T})$.
3. There is a network $\mathcal{N}$ with input arcs $A_{in}$ and output arcs $A_{out}$ for which $\overline{T}$ is the principal typing.

We call $\overline{T}$ the total atomic typing that extends the non-total atomic typing $T$.

**Proof.** Parts 1 and 2 are straightforward, all details omitted. For part 3, we construct the desired network $\mathcal{N}$ with input arcs $A_{in}$, output arcs $A_{out}$, and a single node $\nu$. All arcs in $A_{in}$ are directed into $\nu$, and all arcs in $A_{out}$ are directed out of $\nu$. For every $a \in A_{in}$, we set $\text{LC}(a) = \text{UC}(a) = T^{\max}(\{a\})$, and for every $a \in A_{out}$, we set $\text{LC}(a) = \text{UC}(a) = - T^{\max}(\{a\})$. It is readily checked that $\overline{T}$ is the principal typing for $\mathcal{N}$.

---

\(^{12}\) The notation $\overline{T}[A_{in, out}]$ is ambiguous for the same reason that “$\mathcal{U}(A_{in, out})$” is. Convention 76 applies to both.
Lemma 80. Let $T, U : \mathcal{P}(A_{\text{in, out}}) \to \mathbb{T}(\mathbb{R})$ be two typings over $A_{\text{in, out}}$ which are tight, total, and valid. The meet of $T$ and $U$, denoted $(T \wedge U)$, is defined by setting:

$$(T \wedge U)(A) = T(A) \cap U(A)$$

for every $A \in \mathcal{P}(A_{\text{in, out}})$. Conclusion: $(T \wedge U)$ is tight, total, and valid.

One special case occurs when $T(A) \cap U(A) = \emptyset$ for every $A \in \mathcal{P}(A_{\text{in, out}})$. We denote the resulting typing $\bot[A_{\text{in, out}}]$ and call it the inconsistent typing over $A_{\text{in, out}}$.

Note that $\bot[A_{\text{in, out}}]$ is different from the typing that assigns the type/interval $[0, 0]$ to every $A \in \mathcal{P}(A_{\text{in, out}})$. Call the latter $\bot_{\text{in}}[A_{\text{in, out}}]$. The distinction between $\bot[A_{\text{in, out}}]$ and $\bot_{\text{in}}[A_{\text{in, out}}]$ is required by the algebra, though in practice they can be identified: Both $\bot[A_{\text{in, out}}]$ and $\bot_{\text{in}}[A_{\text{in, out}}]$ are typings for a network for which there are no non-trivial feasible flows.

Proof. Because $T$ and $U$ are valid, they satisfy the necessary and sufficient conditions in Theorem 57. Hence, it suffices to show that $(T \wedge U)$ satisfies the same necessary and sufficient conditions.

Because $T(\emptyset) = U(\emptyset) = T(A_{\text{in, out}}) = U(A_{\text{in, out}}) = [0, 0]$, it immediately follows that $(T \wedge U)(\emptyset) = (T \wedge U)(A_{\text{in, out}}) = [0, 0]$, as desired.

Consider arbitrary $\emptyset \notin A, B \notin A_{\text{in, out}}$ such that $A \uplus B = A_{\text{in, out}}$. Let $T(A) = -T(B) = [r_1, s_1]$ and $U(A) = -U(B) = [r_2, s_2]$ for some $r_1 \leq s_1$ and $r_2 \leq s_2$. It follows that:

$$T(A) \cap U(A) = -T(B) \cap -U(B) =$$

$$\begin{cases} 
\emptyset & \text{if } \max\{r_1, r_2\} \leq \min\{s_1, s_2\}, \\
[\max\{r_1, r_2\}, \min\{s_1, s_2\}] & \text{otherwise}.
\end{cases}$$

This implies $(T \wedge U)(A) = -(T \wedge U)(B)$, as desired.

Lemma 81. Let $T, U : \mathcal{P}(A_{\text{in, out}}) \to \mathbb{T}(\mathbb{R})$ be two typings over $A_{\text{in, out}}$ which are tight, total, and valid. The join of $T$ and $U$, denoted $(T \vee U)$, is defined by setting:

$$(T \vee U)(A) = \left[ \min \{T^{\text{min}}(A), U^{\text{min}}(A)\}, \max \{T^{\text{max}}(A), U^{\text{max}}(A)\} \right]$$

for every $A \in \mathcal{P}(A_{\text{in, out}})$. Conclusion: $(T \vee U)$ is tight, total, and valid.

Proof. The proof here is very similar to the proof of Lemma 80. Starting from the fact that $T$ and $U$ satisfy the necessary and sufficient conditions in Theorem 57, we show that $(T \vee U)$ satisfies the same conditions.

When $A = \emptyset$ and $B = A_{\text{in, out}}$, we have $T(A) = T(B) = U(A) = U(B) = [0, 0]$. This in turn implies $(T \vee U)(A) = [0, 0] = -(T \vee U)(B)$, as desired.

Consider arbitrary $\emptyset \notin A, B \notin A_{\text{in, out}}$ such that $A \uplus B = A_{\text{in, out}}$. Let $T(A) = -T(B) = [r_1, s_1]$ and $U(A) = -U(B) = [r_2, s_2]$ for some $r_1 \leq s_1$ and $r_2 \leq s_2$. This implies the following:

$$T^{\text{min}}(A) = r_1 = -T^{\text{max}}(B),$$

$$T^{\text{max}}(A) = s_1 = -T^{\text{min}}(B),$$

$$U^{\text{min}}(A) = r_2 = -U^{\text{max}}(B),$$

$$U^{\text{max}}(A) = s_2 = -U^{\text{min}}(B).$$

This in turn implies that:

$$\left[ \min \{T^{\text{min}}(A), U^{\text{min}}(A)\}, \max \{T^{\text{max}}(A), U^{\text{max}}(A)\} \right]$$

$$= +\left[ \min \{-T^{\text{max}}(B), -U^{\text{max}}(B)\}, \max \{-T^{\text{min}}(B), -U^{\text{min}}(B)\} \right]$$

$$= +\left[ -\max \{T^{\text{max}}(B), U^{\text{max}}(B)\}, -\min \{T^{\text{min}}(B), U^{\text{min}}(B)\} \right]$$

$$= -\left[ \min \{T^{\text{min}}(B), U^{\text{min}}(B)\}, \max \{T^{\text{max}}(B), U^{\text{max}}(B)\} \right]$$

Hence, $(T \vee U)(A) = -(T \vee U)(B)$, as desired.
Lemma 82. If \( S, T, U : \mathcal{P}(\mathbf{A}_{\text{in,out}}) \rightarrow \mathcal{I}(\mathbb{R}) \) are tight, total, and valid, typings over \( \mathbf{A}_{\text{in,out}} \), then:

1. \( S \land (T \lor U) = (S \land T) \lor (S \land U) \).
2. \( S \lor (T \land U) = (S \lor T) \land (S \lor U) \).

Proof. This is a straightforward chasing of the endpoints of each of the given typings \( S, T \) and \( U \), following the style of Lemmas 80 and 81. All details omitted. \( \square \)

Lemma 83. If \( T : \mathcal{P}(\mathbf{A}_{\text{in,out}}) \rightarrow \mathcal{I}(\mathbb{R}) \) is a tight, total, and valid, typing over \( \mathbf{A}_{\text{in,out}} \), then:

1. \( T \land \top[\mathbf{A}_{\text{in,out}}] = \top[\mathbf{A}_{\text{in,out}}] \land T = T \quad \text{and} \quad T \lor \top[\mathbf{A}_{\text{in,out}}] = \top[\mathbf{A}_{\text{in,out}}] \lor T = \top[\mathbf{A}_{\text{in,out}}] \).
2. \( T \land \bot[\mathbf{A}_{\text{in,out}}] = \bot[\mathbf{A}_{\text{in,out}}] \land T = \bot[\mathbf{A}_{\text{in,out}}] \quad \text{and} \quad T \lor \bot[\mathbf{A}_{\text{in,out}}] = \bot[\mathbf{A}_{\text{in,out}}] \lor T = T \).

\( \top[\mathbf{A}_{\text{in,out}}] \) is the typing defined in Lemma 77, \( \bot[\mathbf{A}_{\text{in,out}}] \) is the typing defined in Lemma 80.

Proof. Immediate from the preceding definitions and lemmas. All details omitted. \( \square \)

The definitions and lemmas, from 75 to 83, provide the necessary elements to organize all tight, total, and valid, typings over the set \( \mathbf{A}_{\text{in,out}} \) of outer arcs as a distributive atomic lattice.

Definition 84 (Lattice of Tight, Total, and Valid, Typings). For every set of input/output arcs \( \mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \cup \mathbf{A}_{\text{out}} \), we define the lattice of all tight, total, and valid, typings over \( \mathbf{A}_{\text{in,out}} \), denoted \( \text{Valid}(\mathbf{A}_{\text{in,out}}) \), as follows. The underlying set of \( \text{Valid}(\mathbf{A}_{\text{in,out}}) \) consists of all the typings over \( \mathbf{A}_{\text{in,out}} \) satisfying the necessary and sufficient conditions of Theorem 57 (or, equivalently, Corollary 58) augmented with the inconsistent typing \( \bot[\mathbf{A}_{\text{in,out}}] \).

For uniform statements, we include \( \bot[\mathbf{A}_{\text{in,out}}] \) among the valid typings. The elements of \( \text{Valid}(\mathbf{A}_{\text{in,out}}) \) are:

1. The top (or maximum) element is the unrestricted typing \( \top[\mathbf{A}_{\text{in,out}}] \).
2. The bottom (or minimum) element is the inconsistent typing \( \bot[\mathbf{A}_{\text{in,out}}] \).
3. The ordering is the \textit{subtyping} relation, denoted “\( <:\)” in Section 4.2, directed downward:
   (a) For all valid \( T \neq \bot[\mathbf{A}_{\text{in,out}}] \) and \( U \neq \bot[\mathbf{A}_{\text{in,out}}] \), if \( T < U \) then \( T \) is “higher” than \( U \) in the lattice. Expressed differently, the ordering is specified contravariantly by \( \text{Poly}(T) \supset \text{Poly}(U) \).
   (b) For every valid \( T \), we set \( T < \bot[\mathbf{A}_{\text{in,out}}] \).
4. For all valid \( T \) and \( U \), their least upper bound is \( (T \lor U) \) and their greatest lower bound is \( (T \land U) \).
5. For all valid \( S, T, \) and \( U \), the distributive laws hold:
   \[ S \land (T \lor U) = (S \land T) \lor (S \land U) \quad \text{and} \quad S \lor (T \land U) = (S \lor T) \land (S \lor U) . \]

The “lowest” elements in the lattice \( \text{Valid}(\mathbf{A}_{\text{in,out}}) \) that are right above \( \bot[\mathbf{A}_{\text{in,out}}] \) are the total atomic typings, as specified in Definition 78, of which there are infinitely many. \( \text{Valid}(\mathbf{A}_{\text{in,out}}) \) is an \textit{atomic lattice} in that any valid typing \( T \) is the join of all the total atomic typings below it. \( \square \)

The lattice \( \text{Valid}(\mathbf{A}_{\text{in,out}}) \) can be studied in its own right, with interesting implications for how to operate on valid typings (not done in this report).
10.2 Compositionality of Valid Typings

We limit the rest of this section to simple results that demonstrate the relevance of our algebraic characterization of valid typings for a compositional analysis of flow networks. As described in Section 1, we take compositionality as an enhanced form of modularity, in the following sense: Not only is the analysis of a network obtained by (easily) combining the analyses of its separate components, with no need to re-examine the internals of the components, but the latter analyses can be also carried out independently of each other and in any order. We do this for the two operations for assembling networks defined in the opening paragraph of Section 10; the results are summed up in Proposition 86 and Proposition 91, for which we provide enough of the background material to make them plausible, but without supplying all the details in their respective proofs.

Lemma 85. Let $T$ and $U$ be tight, total, and valid, typings over $A_{in,out} = A_{in} \cup A_{out}$ and $B_{in,out} = B_{in} \cup B_{out}$, respectively. We define the parallel addition $(T \parallel U)$ of $T$ and $U$ on every $A \subseteq A_{in,out} \cup B_{in,out}$:

$$(T \parallel U)(A) = \begin{cases} 
[0,0] & \text{if } A = \emptyset \text{ or } A = A_{in,out} \cup B_{in,out}, \\
T(A) & \text{if } A \subseteq A_{in,out}, \\
U(A) & \text{if } A \subseteq B_{in,out}, \\
(T(A \cap A_{in,out}) + U(A \cap B_{in,out})) & \text{if } A \cap A_{in,out} \neq \emptyset \text{ and } A \cap B_{in,out} \neq \emptyset,
\end{cases}$$

where, for the 4th case, addition of two intervals/types $[r_1, r_2]$ and $[s_1, s_2]$ is defined by:

$$[r_1, r_2] + [s_1, s_2] = [r_1 + s_1, r_2 + s_2]$$

Conclusion: $(T \parallel U)$ is a tight, total, and valid typing over $A_{in,out} \cup B_{in,out}$.

Proof. That $T$ and $U$ are tight and total immediately implies the same for $(T \parallel U)$. To prove that $(T \parallel U)$ is valid, it suffices to show that it satisfies the necessary and sufficient conditions of Theorem 57. This easily follows from the definition of $(T \parallel U)$ and the fact $T$ and $U$ satisfy the same conditions. All details omitted.

Proposition 86 (Typings for Parallel Additions). Let $M$ and $N$ be networks with outer arcs $A_{in,out} = A_{in} \cup A_{out}$ and $B_{in,out} = B_{in} \cup B_{out}$, respectively. Let $T$ and $U$ be tight and total typings which are principal for $M$ and $N$, respectively. Conclusion: $(T \parallel U)$ is a tight and total typing which is principal for the network $(M \parallel N)$.

Proof. Lemma 85 implies that $(T \parallel U)$ is a tight, total, and valid, typing — and, therefore, valid for some network, which is easily verified to be the same as $(M \parallel N)$. We omit the proof, also straightforward (and tedious!), that $(T \parallel U)$ is not only valid but also principal for $(M \parallel N)$.

Lemma 87. Let $T$ be a tight, total, and valid, typing over the set of input/output arcs $A_{in,out} = A_{in} \cup A_{out}$. Let $A_1 \cup A_2 = A_{in,out}$ be a two-part partition of $A_{in,out}$ such that $A_1 \neq \emptyset \neq A_2$ and $T(A_1) = T(A_2) = [0,0]$. Define two new typings $T': \mathcal{P}(A_{in,out}) \to \mathcal{I}(\mathbb{R})$ and $T'': \mathcal{P}(A_{in,out}) \to \mathcal{I}(\mathbb{R})$ by setting:

$$T'(B) = T(B \cap A_1) \quad \text{and} \quad T''(B) = T(B \cap A_2)$$

for every $B \in \mathcal{P}(A_{in,out})$. Conclusion: $T'$ and $T''$ are tight, total, and valid, typings over $A_{in,out}$.

Proof. The tightness and totality of $T'$ and $T''$ are immediate. We show that $T'$ and $T''$ are valid, i.e., they satisfy the necessary and sufficient conditions of Theorem 57. Consider $T'$ only, the proof for $T''$ is similar. If $B = \emptyset$ or $B = A_{in,out}$, then $T'(B) = [0,0]$, as desired. Suppose $\emptyset \neq B \neq A_{in,out}$.

In the rest of the proof we use the addition operator “+” on intervals/types, as defined in the statement of Lemma 85. We have:

$$T'(B) = T(B \cap A_1) \subseteq T(B) + T(A_1) = T(B)$$

54
where the first equality follows from the definition of $T'$, the inclusion $T(X \cap Y) \subseteq T(X) + T(Y)$ holds for every valid typing $T$, and the second equality follows from the fact that $T(A_1) = [0, 0]$. Similarly, we have:

$$T'(A_{in,out} - B) = T( (A_{in,out} - B) \cap A_1 ) \subseteq T(A_{in,out} - B) + T(A_1) = T(A_{in,out} - B)$$

But $T(B) + T(A_{in,out} - B) = [0, 0]$ because $T$ satisfies the necessary and sufficient conditions of Theorem 57. Hence $T'(B) + T'(A_{in,out} - B) \subseteq [0, 0]$, which implies $T'(B) + T'(A_{in,out} - B) = [0, 0]$, thus completing the proof that $T'$ is valid.

**Definition 88 (Splitting).** Let $T$ be a tight, total, and valid, typing over the input/output arcs $A_{in,out} = A_{in} \uplus A_{out}$ as in Lemma 87, with:

$$T(A_1) = T(A_2) = [0, 0]$$

for a two-part partition $A_1 \uplus A_2 = A_{in,out}$ such that $A_1 \neq \emptyset \neq A_2$. By the definition of the typings $T'$ and $T''$ in Lemma 87:

$$T'(B_1) = [0, 0] \text{ for every } B_1 \subseteq A_2,$n
$$T''(B_2) = [0, 0] \text{ for every } B_2 \subseteq A_1.$$n

Thus, ignoring the trivial mappings by $T'$ (resp. $T''$) of subsets of $A_2$ (resp. $A_1$) to $[0, 0]$, we can define the total and valid typings $S'$ (resp. $S''$) by restricting $T'$ (resp. $T''$) to $\mathcal{P}(A_1)$ (resp. $\mathcal{P}(A_2)$):

$$S' : \mathcal{P}(A_1) \to \mathcal{I}(\mathbb{R}) \quad \text{with } S'(B_1) = T'(B_1) \text{ for every } B_1 \subseteq A_1,$n
$$S'' : \mathcal{P}(A_2) \to \mathcal{I}(\mathbb{R}) \quad \text{with } S''(B_2) = T''(B_2) \text{ for every } B_2 \subseteq A_2.$$n

While $T'$ and $T''$ are tight, $S'$ and $S''$ may or may not be, in a subtle and perhaps unexpected twist; hence, in order to preserve tightness, we define:

$$U' = \text{Tight}(S') \quad \text{and} \quad U'' = \text{Tight}(S''),$$

where $\text{Tight}(\cdot)$ is the algorithm defined in Proposition 21. We call $\{U', U''\}$ the **splitting** of $T$ induced by $T(A_1) = T(A_2) = [0, 0]$. We write $\text{split}[A_1, A_2](T)$ to denote the set $\{U', U''\}$.

The next lemma confirms that splitting of typings works as expected: Adding two typings in parallel, then splitting them, restores the original two.

**Lemma 89.** If $T$ and $U$ are tight, total, and valid, typings over $A_{in,out} = A_{in} \uplus A_{out}$ and $B_{in,out} = B_{in} \uplus B_{out}$, respectively, as in Lemma 85, then:

$$\text{split}[A_{in,out}, B_{in,out}](T \parallel U) = \{T, U\}.$$n

**Proof.** Immediate from Lemmas 85 and 87, and Definition 88.

**Definition 90 (Binding).** Let $T : \mathcal{P}(A_{in,out}) \to \mathcal{I}(\mathbb{R})$ be a tight, total, and valid, typing over the set of input/output arcs $A_{in,out} = A_{in} \uplus A_{out}$. Let $a \in A_{in}$ and $b \in A_{in}$. Compute the typing $U : \mathcal{P}(A_{in,out}) \to \mathcal{I}(\mathbb{R})$ and its splitting into $\{U_1, U_2\}$ as follows:

$$U = \left( T[A_1] \parallel T[A_2] \right) \land T,$n
$$= \left( (T[A_1] \land T) \parallel (T[A_2] \land T) \right),$$n
and $\{U_1, U_2\} = \text{split}[A_1, A_2](U)$.

---

13This fact, among others, is illustrated by Example 92.
so that \( U_1 : \mathcal{P}(A_1) \to \mathcal{T}(\mathbb{R}) \) and \( U_2 : \mathcal{P}(A_2) \to \mathcal{T}(\mathbb{R}) \). The typings \( T[A_1] \) and \( T[A_2] \) are the unrestricted typings over \( A_1 \subseteq A_{\text{in out}} \) and \( A_2 \subseteq A_{\text{in out}} \) with \( A_1 \sqcup A_2 = A_{\text{in out}} \), defined in Lemma 77.

The splitting of \( U \) into \( \{U_1, U_2\} \) is well defined because \( U(A_1) = U(A_2) = [0,0] \). We write \( \text{bind}_{(a,b)}(T) \) to denote the typing \( U_1 \), i.e., \( \text{bind}_{(a,b)}(T) = U_1 \).

By Lemmas 87 and 89, \( \text{bind}_{(a,b)}(T) \) is a tight, total, and valid, typing over \( A_{\text{in out}} - \{a,b\} \).

**Proposition 91 (Typings for Single Bindings).** Let \( \mathcal{N} \) be a network with outer arcs \( A_{\text{in out}} = A_{\text{in}} \sqcup A_{\text{out}} \), and let \( a \in A_{\text{in}} \) and \( b \in A_{\text{out}} \). Let \( T \) be a tight and total typing, which is principal for \( \mathcal{N} \). **Conclusion:** \( \text{bind}_{(a,b)}(T) \) is a tight and total typing, which is principal for the network \( \text{bind}_{(a,b)}(\mathcal{N}) \).

**Proof.** That \( \text{bind}_{(a,b)}(T) \) is a tight, total, and valid, typing follows from the preceding lemmas and definitions. We omit the straightforward proof that it is also principal for the network \( \text{bind}_{(a,b)}(\mathcal{N}) \). Note that \( \text{bind}_{(a,b)}(T) \) and \( \text{bind}_{(a,b)}(\mathcal{N}) \) are over the same set of outer arcs \( A_{\text{in out}} - \{a,b\} \).

The lemmas and definitions leading to \( \text{bind}_{(a,b)}(T) \) in Proposition 91 are somewhat involved, but the actual computation of \( \text{bind}_{(a,b)}(T) \) from a given \( T \) is quite simple, as illustrated by the next example.

**Example 92.** We consider network \( \mathcal{N}_2 \) from Example 13 where we bind output arc \( a_4 \) to input arc \( a_1 \). This is the network \( \text{bind}_{(a_1,a_4)}(\mathcal{N}_2) \) in the notation of this section, which we now call \( \mathcal{N}_5 \) (shown on the right in Figure 15). We want to compute a tight and total typing which is principal for \( \mathcal{N}_5 \).

There are two approaches. **Approach 1** directly works on \( \mathcal{N}_2 \), by setting up appropriate flow-conservation equations \( \mathcal{E}_5 \) and constraint inequalities \( \mathcal{C}_5 \), and then applying Procedure 45 (or one of the other procedures in Section 6) to \( \mathcal{E}_5 \cup \mathcal{C}_5 \). **Approach 2**, faster and simpler, uses Proposition 91, provided we already know a tight and total typing which is principal for the original \( \mathcal{N}_2 \). For the network \( \mathcal{N}_2 \) in particular, we already made the effort of calculating a typing \( T_2 \) for it in Example 46.

**Approach 1:** It suffices to identify the two arcs \( a_1 \) and \( a_4 \), by changing the name “\( a_4 \)” to “\( a_1 \)” (or the name “\( a_1 \)” to “\( a_4 \)” in the flow-conservation equations \( \mathcal{E}_2 \) and constraint inequalities \( \mathcal{C}_2 \) of \( \mathcal{N}_2 \). We therefore define:

\[
\mathcal{E}_5 = \mathcal{E}_2[ a_4 := a_1 ] \quad \text{and} \quad \mathcal{C}_5 = \mathcal{C}_2[ a_4 := a_1 ],
\]

where “[\( a_4 := a_1 \)]” refers to the substitution of \( a_1 \) for \( a_4 \). Applying Procedure 45 to \( \mathcal{E}_5 \cup \mathcal{C}_5 \), using a standard linear programming package (such as Matlab), we obtain the following typing, call it \( T_5 \), which makes the assignment \( T_5(\emptyset) = [0,0] \) in addition to:

\[
\begin{align*}
a_2 &: [0,7] & a_3 &: [4,9] & a_5 &: [-11,-4] \\
a_2 + a_3 &: [4,11] & a_2 - a_5 &: [-9,-4] & a_3 - a_5 &: [-7,0] & a_2 + a_3 - a_5 &: [0,0]
\end{align*}
\]

**Approach 2:** We only need to compute \( \text{bind}_{(a_1,a_4)}(T_2) \). For this, we use the intermediary principal typing \( T[\{a_2, a_3, a_5\}] \) of the unrestricted network \( \mathcal{U}(\{a_2, a_3, a_5\}) \). Though unnecessary for the computation of \( \text{bind}_{(a_1,a_4)}(T_2) \), we also show the typing \( T[\{a_1, a_4\}] \) of \( \mathcal{U}(\{a_1, a_4\}) \) for illustrative purposes. Both \( \mathcal{U}(\{a_1, a_4\}) \) and \( \mathcal{U}(\{a_2, a_3, a_5\}) \) are on the left in Figure 15. By Lemma 77, \( T[\{a_2, a_3, a_5\}] \) and \( T[\{a_1, a_4\}] \) are:

\[
\begin{align*}
T[\{a_2, a_3, a_5\}] & \quad a_2 &: [0,K] & a_3 &: [0,K] & a_5 &: [-K,0] \\
a_2 + a_3 &: [0,K] & a_2 - a_5 &: [-K,0] & a_3 - a_5 &: [-K,0] & a_2 + a_3 - a_5 &: [0,0]
\end{align*}
\]

\[
\begin{align*}
T[\{a_1, a_4\}] & \quad a_1 &: [0,K] & a_4 &: [-K,0] & a_1 - a_4 &: [0,0]
\end{align*}
\]
The desired typing is \( \text{bind}_{\{a_1, a_4\}}(T_2) = \text{Tight}(T[\{a_2, a_3, a_5\}] \land T_2) \). The computation of \( T[\{a_2, a_3, a_5\}] \land T_2 \) is immediate: It simply collects the type/interval assignments by \( T_2 \) to the subsets of \( \{a_2, a_3, a_5\} \) and makes them narrower based on \( T[\{a_2, a_3, a_5\}] \):

\[
T[\{a_2, a_3, a_5\}] \land T_2
\]

\[
\begin{align*}
  a_2 &: [0, 7] \quad a_3 &: [4, 9] \quad -a_5 &: [-11, -4] \\
  a_2 + a_3 &: [4, 11] \quad a_2 - a_5 &: [-11, 0] \quad a_3 - a_5 &: [-7, 0] \quad a_2 + a_3 - a_5 &: [0, 0]
\end{align*}
\]

As it stands now, the preceding typing is not tight, nor does it satisfy the second of the necessary and sufficient conditions of Theorem 57. Indeed, the type of \( \{a_3\} \) is not the negation of the type of \( \{a_2, a_5\} \), namely, \( [4, 9] \neq [-11, 0] \). However, applying \( \text{Tight}() \), we obtain:

\[
\text{Tight}(T[\{a_2, a_3, a_5\}] \land T_2)
\]

\[
\begin{align*}
  a_2 &: [0, 7] \quad a_3 &: [4, 9] \quad -a_5 &: [-11, -4] \\
  a_2 + a_3 &: [4, 11] \quad a_2 - a_5 &: [-9, -4] \quad a_3 - a_5 &: [-7, 0] \quad a_2 + a_3 - a_5 &: [0, 0]
\end{align*}
\]

which is the same as the typing \( T_5 \) obtained in Approach 1. In fact, the example is simple enough that we do not need to apply \( \text{Tight}() \) and can compute the tight typing by inspection, which is typically the case whenever the set of outer arcs (here \( \{a_2, a_3, a_5\} \)) is “small”.

\[\square\]

![Figure 15: Networks for Example 92. The unrestricted networks \( \mathcal{U}(\{a_1, a_4\}) \) and \( \mathcal{U}(\{a_2, a_3, a_5\}) \) are on the left, and the network \( N_5 \) is on the right, obtained from \( N_2 \) by binding output arc \( a_4 \) to input arc \( a_1 \).](image-url)

**Example 93.** In Example 47 we computed the typing \( T_3 \) which is principal for network \( N_3 \). In Example 48 we computed the typing \( T_3 \) which is principal for network \( N_4 \). We noted that neither of the two typings is a subtyping of the other. By Lemma 80, the meet \( (T_3 \land T_4) \) of the two typings makes the type assignments \( (T_3 \land T_4)(\emptyset) = (T_3 \land T_4)(\{a_1, a_2, a_3, a_4\}) = [0, 0] \) in addition to:

\[
\begin{align*}
  a_1 &: [0, 15] \quad a_2 &: [0, 25] \quad -a_3 &: [-15, 0] \quad -a_4 &: [-25, 0] \\
  a_1 + a_2 &: [0, 30] \quad a_1 - a_3 &: [-10, 10] \quad a_1 - a_4 &: [-23, 15] \\
  a_2 - a_3 &: [-15, 23] \quad a_2 - a_4 &: [-10, 10] \quad -a_3 - a_4 &: [-30, 0] \\
  a_1 + a_2 - a_3 &: [0, 25] \quad a_1 + a_2 - a_4 &: [0, 15] \quad a_1 - a_3 - a_4 &: [-25, 0] \quad a_2 - a_3 - a_4 &: [-15, 0]
\end{align*}
\]

\( (T_3 \land T_4) \) is valid and a subtyping of both \( T_3 \) and \( T_4 \). The underlined type assignments here are those that differ from the corresponding type assignments made by \( T_3 \) and \( T_4 \). The other type assignments are the same for the three typings: \( T_3, T_4, \) and \( (T_3 \land T_4) \). Denote \( (T_3 \land T_4) \) by the shorthand \( T_{3 \land 4} \).
Because \( T_{3 \land 4} \) is valid, we can use it to assign capacities to \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) where \( A_{\text{in}} = \{a_1, a_2\} \) and \( A_{\text{out}} = \{a_3, a_4\} \), according to the methodology of Section 9. Call the resulting network \( N_{3 \land 4} \). The typing \( T_{3 \land 4} \) is only valid for \( N_3 \) and \( N_4 \), but is principal (and therefore valid) for \( N_{3 \land 4} \), shown in Figure 16. We can safely substitute \( N_{3 \land 4} \) for any occurrence of \( N_3 \) or \( N_4 \), but not the other way around.

![Figure 16: Network \( N_{3 \land 4} \) in Example 93, obtained by assigning capacities induced by \( T_{3 \land 4} \) to \( \text{Graph}(A_{\text{in}}, A_{\text{out}}) \) when \( A_{\text{in}} = \{a_1, a_2\} \) and \( A_{\text{out}} = \{a_3, a_4\} \). Missing lower-bounds/upper-bounds are 0/“very large” \( K \).](image)

### 11 Open Problems and Future Work

There are several important questions we have not settled in this report. Although several algorithms have been introduced, we have not carefully examined the time complexity of any of them. All the algorithms that appeal to some linear programming optimization can be executed in low-degree polynomial time, but this does not determine a precise estimate of the cost of running them, together or in sequence, within the same larger algorithm. There are also questions of alternative, or improved, algebraic definitions of network typings.

#### 11.1 More Efficient Algorithms

Most important perhaps, the source of the most significant complexity cost is not related to linear programming, but to something else: We have focused almost exclusively on total typings \( T : \mathcal{P}(A_{\text{in, out}}) \to I(\mathbb{R}) \), which specify a type/interval for every subset of \( A_{\text{in, out}} = A_{\text{in}} \cup A_{\text{out}} \). For even moderately small \( m + n = |A_{\text{in}}| + |A_{\text{out}}| \), all of the algorithms in Section 6 will be exponential in \( m + n \) and therefore expensive.

Although total typings offer uniform properties, for purposes of comparison and operating on them (as in Section 10), they typically include redundant information, as we pointed out in Example 26. We formalize this redundancy next.

**Definition 94 (Minimal Typings).** Let \( T : \mathcal{P}(A_{\text{in, out}}) \to I(\mathbb{R}) \) be a typing, not necessarily total. Denote by \( ||T|| \) the number of assignments made by \( T \):

\[
||T|| = \left| \{ A \subseteq A_{\text{in, out}} \mid T(A) \text{ is defined} \} \right|
\]

We say \( T \) is **minimal** iff two conditions:

- \( T \) is tight, and

- for every typing \( T' : \mathcal{P}(A_{\text{in, out}}) \to I(\mathbb{R}) \), if \( \text{Poly}(T') = \text{Poly}(T) \) then \( ||T'|| \geq ||T|| \).
The second bullet point says: If $T'$ is equivalent to $T$, then $T'$ makes at least as many type assignments as $T$.

Informally, not only are the types assigned by $T$ to members of $\mathcal{P}(A_{\text{in, out}})$ free of redundant values (tightness), but none can be omitted without changing $\text{Poly}(T)$ and are therefore necessary (minimality).

**Proposition 95** (Every Typing Is Equivalent to a Minimal Typing). *There is an algorithm Minimal() which, given a typing $T$ as input, always terminates and returns an equivalent minimal typing Minimal($T$).*

**Proof.** Let \( \{A_1, \ldots, A_k\} = \mathcal{P}(A_{\text{in, out}}) - \{\emptyset\} \), where \( k = 2^{m+n} - 1 \), the set of non-empty subsets of $A_{\text{in, out}}$ listed in some fixed order. By Proposition 21, we may assume the typing $T : \mathcal{P}(A_{\text{in, out}}) \to \mathcal{I}(\mathbb{R})$ is total and tight. Consider the types assigned by $T$ in the specified order, starting from the first one:

\[
T(A_1) = [r_1, r'_1] \quad T(A_2) = [r_2, r'_2] \quad \ldots \quad T(A_k) = [r_k, r'_k]
\]

Using standard procedures of linear algebra, we test whether the first type $[r_1, r'_1]$ is implied by the other types; that is, given the objective function:

\[
\theta_{A_1} = \sum A_{\text{in}} \cap A_1 - \sum A_{\text{out}} \cap A_1
\]

and relative to the $2 \cdot (k-1)$ inequalities corresponding to the other types, we check whether the minimum $s_1$ and maximum $s'_1$ of $\theta_{A_1}$ match $r_1$ and $r'_1$, respectively. In general, $[r_1, r'_1] \subseteq [s_1, s'_1]$. If they do match, we omit the type $T(A_1)$ from the list, otherwise we keep it in. We continue in a similar way through the entire list of types assigned by $T$, in total of $k$ stages. After the $k$-th stage, we are left with a list of types none of which is implied by the others. \( \square \)

**Definition 96** (Minimality Index). Let $T : \mathcal{P}(A_{\text{in, out}}) \to \mathcal{I}(\mathbb{R})$ be a typing over $A_{\text{in, out}} = A_{\text{in}} \cup A_{\text{out}}$, not necessarily total, with $|A_{\text{in}}| = m \geq 1$ and $|A_{\text{out}}| = n \geq 1$. Define the *minimality index* of $T$ as follows:

\[
\text{index}(T) = \min \left\{ \|U\| \mid U : \mathcal{P}(A_{\text{in, out}}) \to \mathcal{I}(\mathbb{R}) \text{ such that } \text{Poly}(T) = \text{Poly}(U) \right\},
\]

i.e., $\text{index}(T)$ is the smallest possible number of type assignments made by a typing $U$ equivalent to $T$. In particular, if $T$ is minimal, then $\text{index}(T)$ is the number of type assignments made by $T$ itself.

Let now $T : \mathcal{P}(A_{\text{in, out}}) \to \mathcal{I}(\mathbb{R})$ be a tight, total, and valid typing over $A_{\text{in, out}}$. Every minimal typing $U : \mathcal{P}(A_{\text{in, out}}) \to \mathcal{I}(\mathbb{R})$, which is equivalent to $T$ and tight, selects a subset of all the type assignments made by $T$, i.e., for every $A \subseteq A_{\text{in, out}}$, if $U(A)$ is defined, then $U(A) = T(A)$. For such a typing $U$, we have $m + n \leq \|U\| \leq 2^{m+n}$. Hence, $m + n \leq \text{index}(T) \leq 2^{m+n}$. In fact, the latter number can be halved, down to $2^{m+n-1}$, because $T(A) = -T(A_{\text{in, out}} - A)$ for every $A \subseteq A_{\text{in, out}}$ by Theorem 57. \( \square \)

**Example 97.** Consider the unrestricted network $\mathcal{U}(A_{\text{in, out}})$ over $A_{\text{in, out}} = A_{\text{in}} \cup A_{\text{out}}$, as given in Definition 75, with $A_{\text{in}} = \{a_1, a_2\}$ and $A_{\text{out}} = \{a_3, a_4\}$. According to Lemma 77, a tight and total typing, which is principal for $\mathcal{U}(A_{\text{in, out}})$, is:

\[
\begin{align*}
 a_1 &: [0, +K] \\
 a_1 + a_2 &: [0, +2 \cdot K] \\
 a_2 - a_4 &: [-K, +K] \\
 a_1 + a_2 - a_4 &: [0, +K] \\
 a_1 - a_3 - a_4 &: [-K, 0] \\
 a_2 - a_3 - a_4 &: [K, 0] \\
 a_3 - a_4 &: [-K, 0] \\
 a_4 &: [-K, 0]
\end{align*}
\]

We called this typing $T[A_{\text{in, out}}]$, which is the least restrictive over $A_{\text{in, out}}$. It turns out that $T[A_{\text{in, out}}]$ is not minimal. Indeed, an equivalent tight and principal, but not total, typing for $\mathcal{U}(A_{\text{in, out}})$ is the following $T$:

\[
\begin{align*}
 a_1 &: [0, +K] \\
 a_2 &: [0, +K] \\
 a_3 &: [-K, 0] \\
 a_4 &: [-K, 0]
\end{align*}
\]
It is easy to check that \( \text{Poly}(T[A_{\text{in, out}}]) = \text{Poly}(T) \). And it is also easy to see there are many other minimal typings besides \( T \), which are equivalent to \( T[A_{\text{in, out}}] \) and tight, but not total. For example, we can replace the type assignment \( \langle -a_1 : [-K, 0] \rangle \) by \( \langle -a_3 - a_4 : [-2K, 0] \rangle \) to obtain another minimal typing \( T' \), equivalent to \( T[A_{\text{in, out}}] \) and tight.

For this example, it is easy to see that \( \text{index}(T[A_{\text{in, out}}]) = 4 \) and no typing which makes fewer than 4 type assignments is equivalent to \( T[A_{\text{in, out}}] \).

**Open Problem 98.** Let \( T : \mathcal{P}(A_{\text{in, out}}) \rightarrow \mathcal{I}(\mathbb{R}) \) be a tight, total, and valid, typing over \( A_{\text{in, out}} = A_{\text{in}} \uplus A_{\text{out}} \). The following questions will have a bearing on the complexity of our algorithms invoking linear optimization:

1. Specify conditions on \( T \) with the smallest possible \( \text{index}(T) \geq m + n \).

2. Specify conditions on \( T \) with the largest possible \( \text{index}(T) \leq 2^{m+n-1} \).

3. Develop a methodology to uniquely select a minimal typing \( U \), equivalent to \( T \) and tight.

In (1), \( m + n \) is the smallest possible number of required type assignments, which is that of a minimal typing equivalent to the unrestricted typing \( T[A_{\text{in, out}}] \) defined in Lemma 77. See Example 26 or Example 97, for specific cases. Are there other typings \( T \) such that \( \text{index}(T) = m + n \)? And what are their properties?

In (2), \( 2^{m+n-1} \) is an upper bound on the number of required type assignments. None of the typings \( T \) presented earlier in this report reaches this upper bound \( \text{index}(T) = 2^{m+n-1} \). What is the tightest upper bound on \( \text{index}(T) \), if \( 2^{m+n-1} \) is not?

In the case of at least two algorithms, \( \text{Tight}() \) in Proposition 21 and \( \text{Minimal}() \) in Proposition 95, invoking external linear-optimization packages appears to be an excessive overkill. Both of these algorithms are called to work on a very particular case of linear inequalities and both are likely candidates for improvements, without any need to invoke an external linear optimizer.

Of the latter two, \( \text{Tight}() \) is now more important. \( \text{Tight}() \) is the only algorithm invoked in Section 10, where we operate on the lattice of tight, total, and valid, typings. If such typings are already given, operating on them is simple and very efficient, with the single possible exception of the splitting operation in Definitions 88 and 90 which makes use of the algorithm \( \text{Tight}() \). If an efficient version of \( \text{Tight}() \) is available, then starting from a small collection of tight, total, and valid, typings – each being principal for a relatively small network component – we can efficiently derive tight, total, and principal typings of unboundedly large networks built up from this small collection.

**Open Problem 99.** Our implementation of algorithm \( \text{Tight}() \) in Proposition 21 is the crudest. For example, if the typing \( T : \mathcal{P}(A_{\text{in, out}}) \rightarrow \mathcal{I}(\mathbb{R}) \) is given as input argument, \( \text{Tight}() \) does not take advantage of the fact that all the coefficients are \(+1, 0, \text{or} -1\), in the linear inequalities in the set \( \text{Constraints}(T) \).

Nor does it take advantage of the fact that, if \( T \) is given as total and valid, \( \text{Tight}(T) \) should satisfy the necessary and sufficient conditions of Theorem 57, e.g., if \( T(A) \) is found to be tight because both of its endpoints are on the boundary of \( \text{Poly}(T) \), then both \( \text{Tight}(T)(A) = T(A) \) and \( \text{Tight}(T)(A_{\text{in, out}} - A) = -T(A) \). This will have a bearing on the efficiency of the splitting operation in Definitions 88 and 90, and therefore on the efficiency of computing \( \text{bind}_{(a,b)}(T) \) for \( a \in A_{\text{in}} \) and \( b \in A_{\text{out}} \).

**Open Problem 100.** Though not used anywhere in earlier sections in this report, similar considerations apply to algorithm \( \text{Minimal}() \) in Proposition 95. But there are reasons to pursue it, because a more efficient implementation of \( \text{Minimal}() \) will have a bearing on any examination of Open Problem 98 above.

### 11.2 Alternative or Improved Algebraic Formulations

A common particular case of flow networks assigns only upper-bound capacities to the edges, contains no producer/consumer nodes, and makes all lower-bound capacities \( = 0 \). This is the case first encountered in introductory studies of graph algorithms. The standard \( \text{min-cut/max-flow} \) theorem considers this case and corresponds to the following typings.
Definition 101 (Typings That Include the Null Flow). Let \( T : \mathcal{P}(\mathbf{A}_{\text{in,out}}) \rightarrow \mathcal{I}(\mathbb{R}) \) be a valid typing over the input/output set \( \mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} \). The null IO function assigns 0 to every outer arc \( a \), which we thus denote by (boldface) 0, i.e., 0 : \( \mathbf{A}_{\text{in,out}} \rightarrow \mathbb{R}^+ \) and 0(\( a \)) = 0 for every \( a \in \mathbf{A}_{\text{in,out}} \). We say that 0 satisfies \( T \), and that \( T \) includes 0, whenever 0 \( \in \) Poly(\( T \)). If \( T \) is tight, the preceding is equivalent to saying that:

- For every \( \emptyset \neq A \subseteq \mathbf{A}_{\text{in}} \), if \( T(A) \) is defined, then \( T^{\min}(A) = 0 \).
- For every \( \emptyset \neq A \subseteq \mathbf{A}_{\text{out}} \), if \( T(A) \) is defined, then \( T^{\max}(A) = 0 \).

(The functions \( T^{\min} \) and \( T^{\max} \) induced by \( T \) are defined in Section 4.1.) Put differently still, \( T \) includes the null IO function 0 if the polytope Poly(\( T \)) contains the origin in the hyperspace \( \mathbb{R}^{m+n} \).

Examples of typings that include 0 are \( T_1 \) in Example 26 when \( r = s = t = 0 \), \( T_3 \) in Example 47, \( T_4 \) in Example 48, \( T_{3,3,4} \) in Example 93, and the unrestricted typings \( T[\mathbf{A}_{\text{in,out}}] \) in Lemma 77.

Our results from Sections 7, 8, and 9, show that the set of valid typings and principal typings coincide: Every principal typing \( T \) is valid (trivially), and every valid typing \( T \) is principal for some network \( \mathcal{N} \).

But note the following asymmetry: Even if the valid typing \( T \) includes the IO function 0, the network \( \mathcal{N} \) which we build according to the methodology of Section 9, and for which \( T \) is a principal typing, may contain (internal and backward) arcs with non-zero lower bounds.

This is illustrated by typing \( T_4 \) in Example 48, which includes 0 but which induces non-zero lower bounds in the network \( \mathcal{N}_{T_4} \) in Example 74. We can get rid of the non-zero lower bounds (and internal backward arcs) by appropriately turning some node pairs into producer/consumer pairs or, equivalently, constant input/output arcs, but this re-introduces (now outer) arcs with non-zero lower-bounds.

The following is therefore a natural question: If \( T \) is a valid typing that includes the null IO function 0, is there an implementation \( \mathcal{N} \) of \( T \) which contains no non-zero lower bounds and no constant input/output arcs? We conjecture a negative answer to this question, with a specific counter-example for it, next.

Conjecture 102. There is a valid typing \( T \) which includes the null IO function 0 and which is principal only for networks necessarily containing non-zero lower bounds (on internal backward arcs) and/or constant input/output arcs.

We conjecture that an example of such a typing, call it \( T_6 \), over \( \mathbf{A}_{\text{in,out}} = \mathbf{A}_{\text{in}} \uplus \mathbf{A}_{\text{out}} \) with \( \mathbf{A}_{\text{in}} = \{ a_1, a_2 \} \) and \( \mathbf{A}_{\text{out}} = \{ a_3, a_4 \} \), makes the following type assignments, in addition to \( T_6(\emptyset) = T_6(\{ a_1, a_2, a_3, a_4 \}) = [0, 0] \):

\[
\begin{align*}
  a_1 &: [0, 15] & a_2 &: [0, 25] & - a_3 &: [-15, 0] & - a_4 &: [-25, 0] \\
  a_1 + a_2 &: [0, 30] & a_1 - a_3 &: [-5, 5] & a_1 - a_4 &: [-20, 15] \\
  a_2 - a_3 &: [-15, 20] & a_2 - a_4 &: [-5, 5] & - a_3 - a_4 &: [-30, 0] \\
  a_1 + a_2 - a_3 &: [0, 25] & a_1 + a_2 - a_4 &: [0, 15] & a_1 - a_3 - a_4 &: [-25, 0] & a_2 - a_3 - a_4 &: [-15, 0]
\end{align*}
\]

Typing \( T_6 \) is tight, total, and satisfies the necessary and sufficient conditions of Theorem 57. Hence, in particular, \( T_6 \) is valid. \( T_6 \) also includes the null IO function 0. We constructed \( T_6 \) by trying different valid subtypings of \( T_{3,3,4} \) in Example 93. The underlined assignments are the only different from the corresponding ones in \( T_{3,3,4} \).

We can use \( T_6 \) to assign capacities to Graph(\( \mathbf{A}_{\text{in}}, \mathbf{A}_{\text{out}} \)) when \( \mathbf{A}_{\text{in}} = \{ a_1, a_2 \} \) and \( \mathbf{A}_{\text{out}} = \{ a_3, a_4 \} \), according to the methodology of Section 9. The resulting network \( \mathcal{N}_6 \) is shown in Figure 17, for which \( T_6 \) is a principal typing. It is easy to see that a maximum-throughput (resp. minimum-throughput) feasible flow in \( \mathcal{N}_6 \) carries 30 units (resp. 0 units), just as it is for \( \mathcal{N}_3, \mathcal{N}_4, \) and \( \mathcal{N}_{3,3,4} \).

There seems to be no network, without non-zero lower bounds on internal backward arcs and without constant input/output arcs, for which \( T_6 \) is principal. \( \square \)
Open Problem 103. Define necessary and sufficient conditions, expressed algebraically in the style of Theorem 57, such that a typing \( T : \mathcal{P}(A_{\text{in}}, A_{\text{out}}) \rightarrow \mathcal{I}(\mathbb{R}) \) satisfies these conditions iff \( T \) is principal for a network \( N \) with only zero lower-bound capacities and no constant input/output arcs.

If Conjecture 102 turns out to be true, then clearly such necessary and sufficient conditions cannot consist of only the two in Theorem 57 in addition to a third requiring that \( T \) include the null IO function \( 0 \). A third condition will have to be more stringent than simply requiring that \( T \) include \( 0 \).

Using such a characterization, develop a methodology which, given a tight, total, and valid typing \( T \), implements \( T \) in the form of a network \( N \) such that:

- \( T \) is principal for \( N \),
- If \( T \) includes the null IO function \( 0 \) and \( N \) contains non-zero lower bounds and/or constant input/output arcs, then every network \( N' \) for which \( T \) is principal also contains non-zero lower bounds and/or constant input/output arcs.

In other words, the sought-for methodology will introduce non-zero lower bounds and/or constant input/output arcs in the implementation \( N \) only if absolutely necessary for making \( T \) principal for \( N \). The sought-for methodology will likely be a refinement of our methodology in Section 9.

As suggested by the preceding remarks and open problem, something is lost in going from the principal typing \( T_4 \) of network \( N_4 \), in Example 48, to the equivalent network \( N_4' \) in Example 74, because there are non-zero lower-bounds in the latter but not in the former. But something is also gained: \( N_4' \) is a smaller implementation of the same typing \( T_4 \), i.e., if for any network \( N = (N, A) \) we set:

\[
|N| = \text{number of nodes in } N + \text{number of nodes in } A,
\]

then \( |N_4'| = 6 + 14 = 20 \) while \( |N_4| = 8 + 16 = 24 \). Is the price of a smaller implementation \( N_4' \) for the same specification/typing \( T_4 \) the introduction of non-zero lower bounds? The answer is no, by the next example.

Example 104. The network \( N_4'' \) in Figure 18 was obtained by brute-force trial-and-error. It is equivalent to \( N_4 \) in Example 48 and \( N_4' \) in Example 74, and qualifies as a better implementation of \( T_4 \): Not only is \( N_4'' \) smaller in size, because \( |N_4''| = 6 + 12 = 18 \), it does not use non-zero lower-bounds, in harmony with the fact that \( T_4 \) includes the null IO function \( 0 \).
Figure 18: For Example 104: Network \( N'_4 \) equivalent to \( N_4 \) in Example 48 and \( N'_4 \) in Example 74. Missing lower bounds are 0, missing upper bounds are the “very large number” \( K \).

Hence, and naturally enough, related to Open Problem 103 but different is the following.

**Open Problem 105.** Let \( T : \mathcal{P}(A_{\text{in, out}}) \to I(\mathbb{R}) \) be a tight, total, and valid, typing over \( A_{\text{in, out}} = A_{\text{in}} \uplus A_{\text{out}} \).

Develop a methodology to construct a network \( N \) for which \( T \) is principal such that:

\[
\forall N', \text{ if } T \text{ is principal for } N', \text{ then } \left| N \right| \leq \left| N' \right|
\]

Such a smallest-size network \( N \) can be viewed as the “best” implementation of the given \( T \). Our methodology in Section 9 does not satisfy \( \forall \), as shown by Example 104.

11.3 Beyond Efficiency and Alternative Algebraic Formulations

Studying and resolving the preceding open problems will fine-tune our typing theory for flow networks – within the limits set out in this report. Beyond these limits, however, there are broader research directions in which our framework can be expanded. The following are perhaps among the most relevant.

A domain-specific language for flow-network design. If in the course of assembling larger networks from smaller networks we leave some empty “holes”, i.e., places where missing components are yet to be inserted, then we can specify this process rigorously by means of a domain-specific language (DSL) especially adapted for the task. This is a DSL to write formal network specifications, built up from a finite supply of small network modules in addition to holes using, at a minimum, the two operations of parallel addition and binding introduced in the opening paragraph of Section 10.

The idea of introducing holes in formal specifications has several beneficial aspects: (A) the ability to pursue network design and analysis without having to wait for missing (or broken) components to be inserted (or replaced), (B) the ability to abstract away details through the retention of only the salient variables and constraints at network interfaces as we transition from smaller to larger networks, and (C) the ability to leverage diverse, unrelated theories to derive properties of smaller network components, as long as such components share a common language at their interfaces – namely, the language of network typings. In this sense, typings are the “glue” holding together network components and holes in a consistent way, enforcing the preservation of safety properties across the whole assembly.

**Open Problem 106.** Initial work on a strongly-typed DSL for flow-network design is reported in [3, 7, 8]. These reports present a slimmed down version of the ultimately needed DSL, which would call for additional assembling operations. Already included in our DSL is a constructor of the form “let \( X = M \) in \( N \)”\(^\text{63}\), which informally says “network \( M \) may be safely placed in the occurrences of hole \( X \) in network \( N \)”\(^\text{63}\). Relatively easy variations of this constructor are:

1. \( \textbf{let } X \in \{ M_1, \ldots, M_n \} \textbf{ in } N \)
2. \( \textbf{try } X \in \{ M_1, \ldots, M_n \} \textbf{ in } N \)
3. \( \textbf{mix } X \in \{ M_1, \ldots, M_n \} \textbf{ in } N \)
The specification in (1) informally says “every $M_i$ may be safely placed in all the occurrences of $X$ in $N$”, the one in (2) says “at least one $M_i$ may be safely placed in all the occurrences of $X$ in $N$”, and the one in (3) says “every mix of several $M_i$’s may be selected and safely placed in the occurrences of $X$ in $N$, generally placing different $M_i$’s from that mix in different occurrences”.  

More challenging will be the addition of a constructor to specify recursively defined components, with (unbounded) repeated patterns. In its simplest form, it can be written as:

```
letrec X = M[X] in N[X]
```

where we write $M[X]$ to indicate that hole $X$ occurs free in $M$, and again in $N$. Informally, this specification corresponds to placing an open-ended network of the form $M[M[M[...]]]$ in the occurrences of $X$ in $N$.

In all of these constructors involving holes, an obvious well-formedness condition is that the number (and order) of input arcs and output arcs of network $M$ match those of hole $X$. Safe placement of $M$ in $X$ will be the result of respecting types, to be formulated in an extension of our typing theory, which enforce the preservation of flow feasibility at interfaces.

Adding objective functions and predicates. Let $A_{\text{in,out}} = A_{\text{in}} \cup A_{\text{out}}$ be a set of outer arcs, used as variable names, with $A_{\text{in}} = \{a_1, \ldots, a_m\}$ and $A_{\text{out}} = \{a_{m+1}, \ldots, a_{m+n}\}$ and $m, n \geq 1$. An objective is a function or predicate which we introduce to qualify a typing $T : \mathcal{P}(A_{\text{in,out}}) \rightarrow \mathcal{I}(\mathbb{R})$. We write the combination of typing $T$ and objective $\Phi$ as a pair $(T, \Phi)$ and call it a qualified typing or, if the context is clear, just a typing.

The purpose of $\Phi$ is to carve out a subset of $\text{Poly}(P)$ satisfying desirable properties, in addition to feasibility in the sense of Section 2.2 and as used till now in this report. There are different ways of setting this up, some more suitable than others depending on the objectives. With some objectives $\Phi$, it is possible to combine the constraints induced by $T$ with those of $\Phi$, but then we lose the clean algebraic characterization of valid typings $T$ in Theorem 57.

We choose here a particular formulation for illustrative purposes. Assume $T : \mathcal{P}(A_{\text{in,out}}) \rightarrow \mathcal{I}(\mathbb{R})$ is valid. This implies $T$ is principal for some network $N$, though such $N$ is not uniquely defined. We take the objective $\Phi$ as a predicate on $\text{Poly}(T)$: If $f \in \text{Poly}(T)$ makes $\Phi$ true, we write $f \in \Phi$. We define $\text{Poly}^*(T, \Phi)$ as follows:

\[
\text{Poly}^*(T, \Phi) = \{ f \in (\mathbb{R}^+)^{m+n} \mid f \in \text{Poly}(T) \text{ and } f \in \Phi \}.
\]

The objective $\Phi$ acts on $f \in \text{Poly}(T)$ possibly depending on other arguments, such as $T$, or a particular network $N$, or some value $k$. In such cases, we may write $\Phi[T]$, or $\Phi[N]$, or $\Phi[k]$, to make the dependence explicit.\(^{14}\)

The notions of validity and principality in Section 4.2 extend in the obvious way. The typing $(T, \Phi)$ is valid for a network $N = (N, A)$ where $A = A_{#} \cup A_{\text{in,out}}$ iff:

**(soundness)** Every IO function $f \in \text{Poly}^*(T, \Phi)$ can be extended to a feasible flow $g : A \rightarrow \mathbb{R}^+$.  

The typing $(T, \Phi)$ is principal for the network $N$ iff it is valid for $N$ and:

**(completeness)** Every feasible flow $g : A \rightarrow \mathbb{R}^+$ in $N$ is such that $[g]_{A_{\text{in,out}}} \in \text{Poly}^*(T, \Phi)$.

We give a simple example, to make these notions more concrete, for which we need a few preliminary definitions. Let $N = (N, A)$ be a network, where $A = A_{#} \cup A_{\text{in,out}}$. Given a flow $g : A \rightarrow \mathbb{R}^+$ in $N$, the hop routing of $g$ in $N$ is defined by:

\[
hops(g, N) = |\{a \mid a \in A \text{ and } g(a) \neq 0\}|.
\]

\(^{14}(T, \Phi[N]) \) may be called a “dependent typing”, because it depends on the “value” $N$, in analogy with “dependent types” in type systems for intuitionistic logic and programming languages.
i.e., the number of arcs used by \( g \) in \( N \). For an IO function \( f : A_{\text{in,out}} \rightarrow \mathbb{R}^+ \), the hop routing of \( f \) in \( N \) is:

\[
\text{hops}(f,N) = \min \{ \text{hops}(g,N) \mid g : A \rightarrow \mathbb{R}^+ \text{ extends } f, \text{ i.e. } [g]_{A_{\text{in,out}}} = f \},
\]

which is the smallest possible number of arcs used by a flow \( g \) extending \( f \). Finally, if \( T : \mathcal{P}(A_{\text{in,out}}) \rightarrow \mathcal{T}(\mathbb{R}) \) is a valid typing, the hop routing of \( f \) relative to \( T \) is:

\[
\text{hops}(f,T) = \min \{ \text{hops}(f,N) \mid N \text{ is a network for which } T \text{ is a principal typing} \}.
\]

**Example 107.** Consider networks \( \mathcal{N}_4, \mathcal{N}_4', \) and \( \mathcal{N}_4'' \), in Examples 15, 74, and 104, respectively. The typing \( T_4 \) computed in Example 48 is principal for these three networks (as well as for infinitely many others).

Here \( A_{\text{in}} = \{ a_1, a_2 \} \) and \( A_{\text{out}} = \{ a_3, a_4 \} \), so that \( A_{\text{in,out}} = \{ a_1, a_2, a_3, a_4 \} \) and \( \text{Poly}(T_4) \subseteq (\mathbb{R}^+)^4 \). Let the objective \( \Phi[T_4, k] \) be the following predicate on arbitrary \( f \in \text{Poly}(T_4) \):

\[
f \in \Phi[T_4, k] \iff \begin{cases} \sum f(A_{\text{in}}) \geq k & \text{for every } f' \in \text{Poly}(T_4) \\ \text{if } \sum f(A_{\text{in}}) = \sum f'(A_{\text{in}}) & \text{hops}(f, T_4) \leq \text{hops}(f', T_4) \end{cases}
\]

Informally, this objective expresses the requirement that an IO function \( f \) whose throughput is \( k \) or greater can be extended to a flow \( g : A \rightarrow \mathbb{R}^+ \) which uses the smallest possible number of arcs (“hop routing”), and this smallest number of arcs is computed by considering all possible implementations \( N \) of \( T_4 \).

Because \( \text{Poly}^*(T_4, \Phi) \) is a subset of \( \text{Poly}(T_4) \), and because \( T_4 \) is valid for the three networks \( \mathcal{N}_4, \mathcal{N}_4', \) and \( \mathcal{N}_4'' \), it follows that \( (T_4, \Phi) \) is also valid for the same three networks. While \( T_4 \) is principal for \( \mathcal{N}_4, \mathcal{N}_4' \), and \( \mathcal{N}_4'' \), the qualified typing \( (T_4, \Phi) \) is not principal for any of the three, as we argue next. Consider the case when \( k = 30 \), which is the value of a maximum throughput in all three networks. By brute-force inspection, any feasible flow \( g \) in \( \mathcal{N}_4 \) (resp. \( \mathcal{N}_4', \text{ resp. } \mathcal{N}_4'' \)) extending a maximum-throughput IO function \( f \) must use at least 14 arcs (resp. 13 arcs, resp. 10 arcs). Moreover, the lower-bound of 10 arcs is achieved in \( \mathcal{N}_4'' \) for exactly two maximum-throughput IO functions, call them \( f_1 \) and \( f_2 \), namely:

\[
\begin{align*}
\langle f_1(a_1), f_1(a_2), f_1(a_3), f_1(a_4) \rangle &= \{5, 25, 7, 23\}, \\
\langle f_2(a_1), f_2(a_2), f_2(a_3), f_2(a_4) \rangle &= \{5, 25, 5, 25\}.
\end{align*}
\]

Hence, \( \text{Poly}^*(T_4, \Phi) \) includes only \( f_1 \) and \( f_2 \) among all maximum-throughput IO functions. But there are other maximum-throughput IO functions \( f \) extendable to feasible flows \( g \) in \( \mathcal{N}_4, \mathcal{N}_4', \) and \( \mathcal{N}_4'' \), e.g., consider one for which \( f(a_1) > 5 \). Hence, \( (T_4, \Phi) \) is not principal for any of the three networks.

Consider a slightly adjusted objective \( \Phi' \) which uses an unspecified network \( \mathcal{N} \) as a parameter, in addition to \( T_4 \) and \( k \). On an arbitrary \( f \in \text{Poly}(T_4) \), we define:

\[
f \in \Phi'[\mathcal{N}, T_4, k] \iff \begin{cases} \sum f(A_{\text{in}}) \geq k & \text{for every } f' \in \text{Poly}(T_4) \\ \text{if } \sum f(A_{\text{in}}) = \sum f'(A_{\text{in}}) & 14 \leq \text{hops}(f, \mathcal{N}) \leq \text{hops}(f', \mathcal{N}) + 2 \end{cases}
\]

Informally, this adjusted objective requires that an IO function \( f \) whose throughput is \( k \) or greater can be extended to a flow \( g : A \rightarrow \mathbb{R}^+ \) which uses no less than 14 arcs and no more than \( 2 + \text{the number of arcs used in a particular implementation } \mathcal{N} \) of \( T_4 \).

\[\text{A geometric interpretation of } \Phi[T_4, k] \text{ is also useful. Let } S \text{ be the half space of } (\mathbb{R}^+)^m \text{ defined by the inequality } a_1 + a_2 < k. \text{ If } k > \text{“max throughput allowed by } T_4' \text{” = 30, then } \Phi[T_4, k] \text{ imposes no restriction on any } f \in \text{Poly}(T_4) \cap S. \text{ If } k \geq \text{max throughput allowed by } T_4' \text{, then } \Phi[T_4, k] \text{ imposes no restriction on any } f \in \text{Poly}(T_4) \text{ because } \text{Poly}(T_4) \subseteq S. \text{ Only when } k = 30 \text{ and } f \in (\text{Poly}(T_4) - S) \neq \emptyset \text{ does } \Phi[T_4, k] \text{ put restrictions on flows extending } f.\]
(\(T_4, \Phi'\)) may or may not be principal for the networks \(\mathcal{N}_4, \mathcal{N}_4', \) and \(\mathcal{N}_4''\), depending on the parameters \(\mathcal{N}\) and \(k\). Consider the case \(k = 30\) again. As pointed out above, a feasible flow \(g\) extending a maximum-throughput IO function \(f\) must use between 14 and 16 arcs in \(\mathcal{N}_4\), 13 arcs in \(\mathcal{N}_4'\), and between 10 and 12 arcs in \(\mathcal{N}_4''\). Hence, if \(\mathcal{N} = \mathcal{N}_4\), then \(\Phi'[\mathcal{N}, T_4, k]\) puts no restriction on \(f \in \text{Poly}(T_4)\), so that \((T_4, \Phi') = (T_4, \Phi'[\mathcal{N}, T_4, k])\) is equivalent to \(T_4\) and thus principal for all three networks. But this is not very interesting, since \(\Phi'[\mathcal{N}, T_4, k]\) does not discriminate between feasible flows.

If \(\mathcal{N} = \mathcal{N}_4''\), there are no maximum-throughput \(f \in \text{Poly}(T_4)\) satisfying \(\Phi'[\mathcal{N}, T_4, k]\), thus implying \((T_4, \Phi') = (T_4, \Phi'[\mathcal{N}, T_4, k])\) is not principal for any of the three networks. When \(k < 30\), things are a little more complicated.

The preceding example uses an objective based on \textit{minimization of hop routing}. There are other objectives commonly considered in the area of “traffic engineering” (see, e.g., [2] and references therein). Another which can be dealt with in a similar way is \textit{minimization of arc utilization}, the “utilization of arc \(a\)” being the ratio of the flow value at \(a\) over the upper-bound allowed at \(a\), and there are others still.

**Open Problem 108.** Partial work on adding objectives, and how they can be examined in our type-theoretic framework, is reported in [3, 7], where our formulation of \(\text{Poly}^+(T, \Phi)\) is a little more complicated than the one in the preceding example, but also more flexible.

The challenge when we introduce an objective \(\Phi\) is to preserve the \textit{compositionality} of the typing system, as first defined in Section 1 and for the reasons amplified in Section 10.2. Conditions have to be imposed on \(\Phi\) in order to preserve compositionality (see [3, 7]), and it appears that further conditions are necessary so that \(\text{Poly}^+(T, \Phi)\) can be principal (and not just valid) for some network \(\mathcal{N}\), i.e., there is some \(\mathcal{N}\) implementing \(\text{Poly}^+(T, \Phi)\): the set of all feasible flows in \(\mathcal{N}\) coincide with the set of all flows satisfying \(T\) and \(\Phi\).

A further challenge is when different objectives, say \(\Phi\) and \(\Phi'\), are used in different components of a larger assembly of networks. Are there conditions to be imposed on \(\Phi\) and \(\Phi'\) so that \(\text{Poly}^+(T, \Phi)\) and \(\text{Poly}^+(T, \Phi')\) – or the networks \(\mathcal{N}\) and \(\mathcal{N}'\) implementing them – can communicate in some consistent way at their interfaces, \(i.e.,\) satisfaction of \(\Phi\) in one component does not contradict satisfaction of \(\Phi'\) in the other component?\]

---

**Angelic versus Demonic Non-Determinism.** Suppose \(\mathcal{A}\) is a larger assembly of networks containing network \(\mathcal{N}\) as a component. Under what conditions can we safely substitute another network \(\mathcal{N}'\) for \(\mathcal{N}\)?

Beyond the match between input/output arcs in \(\mathcal{N}\) and input/output arcs in \(\mathcal{N}'\), if we are given principal typings \(T\) and \(T'\) for \(\mathcal{N}\) and \(\mathcal{N}'\), respectively, we should have enough information to decide whether the substitution is safe. To simplify, let \(T\) and \(T'\) be tight and total.

To be a little more specific, let the input and output arcs of \(\mathcal{N}\) and \(\mathcal{N}'\) be \(A_{\text{in}} = \{a_1, a_2\}\) and \(A_{\text{out}} = \{a_3, a_4\}\). If the substitution of \(\mathcal{N}'\) for \(\mathcal{N}\) is safe, then \(\mathcal{N}'\) should be able to consume every input flow that \(\mathcal{N}\) is able to consume, \(i.e.,\) if an input assignment \(f_{\text{in}} : \{a_1, a_2\} \to \mathbb{R}^+\) satisfies \([T]\{a_1, a_2\}\) then it must also satisfy \([T']\{a_1, a_2\}\). Hence, we must have the following inclusions:

\[
(\blacklozenge) \quad T(a_1) \subseteq T'(a_1), \quad T(a_2) \subseteq T'(a_2), \quad \text{and} \quad T'(a_1, a_2) \subseteq T'(a_1, a_2).
\]

Symmetrically, for a safe substitution, every output flow produced by \(\mathcal{N}'\) should not exceed the limits of an output flow produced by \(\mathcal{N}\), \(i.e.,\) if an input assignment \(f_{\text{out}} : \{a_3, a_4\} \to \mathbb{R}^+\) satisfies \([T']\{a_3, a_4\}\) then it must also satisfy \([T]\{a_3, a_4\}\). Hence, we must also have the following reversed inclusions:

\[
(\blacklozenge) \quad T'(a_3) \supseteq T'(a_3), \quad T'(a_4) \supseteq T'(a_4), \quad \text{and} \quad T'(a_3, a_4) \supseteq T'(a_3, a_4).
\]

If \(T'\) satisfies both (\(\blacklozenge\)) and (\(\blacklozenge\)), is the substitution of \(\mathcal{N}'\) for \(\mathcal{N}\) in \(\mathcal{A}\) safe? It depends. The next example elaborates some of the issues. Something peculiar about (\(\blacklozenge\)) and (\(\blacklozenge\)) is that they ignore possible relationships between \(f_{\text{in}}\) and \(f_{\text{out}}\), whenever the first does not uniquely determine the second or the second uniquely the first, which is generally the case for networks with multiple input ports and/or multiple output ports.

---

66
Example 109. In the larger assembly $\mathcal{A}$ described above, let $\mathcal{N} = \mathcal{N}_{3,4}$ from Example 93 and $\mathcal{N}' = \mathcal{N}_3$ from Example 14. Tight, total, and principal typings for $\mathcal{N}_{3,4}$ and $\mathcal{N}_3$ are $T_{3,4}$ and $T_3$, computed in Example 93 and Example 47, respectively. We have the following relationship $T_3 < T_{3,4}$, where “$<$” is the subtyping relation defined in Section 4.2. More, in fact, $T = T_{3,4}$ and $T' = T_3$ satisfy both (●) and (●).

As we explain below, if $\mathcal{N}_3$ operates in a way to preserve the feasibility of flows in $\mathcal{A}$, i.e., if it operates angelically and tries to keep $\mathcal{A}$ in good working order, then replacing $\mathcal{N}_{3,4}$ by $\mathcal{N}_3$ is safe. However, if $\mathcal{N}_3$ makes choices that disrupt $\mathcal{A}$’s good working order, maliciously or unintentionally, i.e., if it operates demonically and violates the feasibility of flows in $\mathcal{A}$, then the substitution is unsafe. This can happen because for the same assignment $f_{\text{in}}$ to the input arcs (resp., the same assignment $f_{\text{out}}$ to the output arcs), corresponds several possible output assignments $f_{\text{out}}$ (resp., output assignments $f_{\text{in}}$), without violating any of $\mathcal{N}_3$’s internal constraints.

Suppose $\mathcal{N}_{3,4}$ in $\mathcal{A}$ is prompted to consume some flow entering at input arcs $a_1$ and $a_2$. (A similar and symmetric argument can be made when $\mathcal{N}_{3,4}$ is asked to produce some flow at output arcs $a_3$ and $a_4$.) Suppose the incoming flow is given by the assignment $f_{\text{in}}(a_1) = 0$ and $f_{\text{in}}(a_2) = 25$. Flow is then pushed along the internal arcs of $\mathcal{N}_{3,4}$, respecting capacity constraints and flow conservation at nodes. There are many different ways in which flow can be pushed through. By direct inspection, relative to the given $f_{\text{in}}$, the largest possible quantity exiting at output arc $a_4$ is 23. So, relative to the given $f_{\text{in}}$, the output assignment which is most skewed in favor of $a_4$ is $f_{\text{out}}(a_3) = 2$ and $f_{\text{out}}(a_4) = 23$. Under the assumption that $\mathcal{A}$ works safely with $\mathcal{N}_{3,4}$ inserted, we take this conclusion to mean that any output quantity exceeding 23 at arc $a_4$, when $f_{\text{in}}(a_1) = 0$ and $f_{\text{in}}(a_2) = 25$, disrupts $\mathcal{A}$’s overall operation.

Next, suppose we substitute $\mathcal{N}_3$ for $\mathcal{N}_{3,4}$ and examine $\mathcal{N}_3$’s behavior with the same $f_{\text{in}}(a_1) = 0$ and $f_{\text{in}}(a_2) = 25$. By inspection, the flow that is most skewed in favor of $a_4$ gives rise to the output assignment $f_{\text{out}}(a_3) = 0$ and $f_{\text{out}}(a_4) = 25$. In this case, the output quantity at $a_4$ exceeds 23, which, as argued above, we take to be disruptive of $\mathcal{A}$’s overall operation. Note the presumed disruption occurs in the enclosing context that is part of $\mathcal{A}$, not inside $\mathcal{N}_3$ itself, where flow is still directed by respecting flow conservation at $\mathcal{N}_3$’s nodes and lower-bound/upper-bound capacities at $\mathcal{N}_3$’s arcs. Thus, $\mathcal{N}_3$’s harmful behavior is not the result of violating its own internal constraints, but of its malicious or (unintended) faulty interaction with the enclosing context.

Consider now a slight adjustment of $\mathcal{N}_{3,4}$, call it $\mathcal{N}_8$, where we make a single change from $\mathcal{N}_{3,4}$, namely, in the upper-bound capacity of input arc $a_2$: Decrease UC($a_2$) from $K$ (“very large number”) to 23. The typing $T_{3,4}$ is no longer valid for $\mathcal{N}_8$, let alone principal for it. We compute a new principal typing $T_8$ for $\mathcal{N}_8$ which, in addition to the type assignments $T_8(\emptyset) = T_8(\{a_1, a_2, a_3, a_4\}) = [0, 0]$, makes the following assignments:

\[
\begin{align*}
    a_1 &: [0, 15] & a_2 &: [0, 23] & -a_3 &: [-15, 0] & -a_4 &: [-25, 0] \\
    a_1 + a_2 &: [0, 30] & a_1 - a_3 &: [-10, 10] & a_1 - a_4 &: [-23, 15] \\
    a_2 - a_3 &: [-15, 23] & a_2 - a_4 &: [-10, 10] & -a_3 - a_4 &: [-30, 0] \\
    a_1 + a_2 - a_3 &: [0, 25] & a_1 + a_2 - a_4 &: [0, 15] & a_1 - a_3 - a_4 &: [-23, 0] & a_2 - a_3 - a_4 &: [-15, 0]
\end{align*}
\]

The underlined type assignments here are those that differ from the corresponding type assignments made by $T_{3,4}$. It is easy to check that, however demonically $\mathcal{N}_3$ chooses to push flow through its internal arcs, the substitution of $\mathcal{N}_3$ for $\mathcal{N}_8$ is “input safe”; specifically, for every input assignment $f_{\text{in}} : \{a_1, a_2\} \rightarrow \mathbb{R}^+$ satisfying $\{T_8\}_{\{a_1, a_2\}}$, and every extension $g : \{a_1, a_2, a_3, a_4\} \rightarrow \mathbb{R}^+$ of $f_{\text{in}}$, the IO function $g$ satisfies $T_3$ iff $g$ satisfies $T_8$.

Similarly, we can adjust $\mathcal{N}_{3,4}$ to define another network $\mathcal{N}_9$, for which the substitution of $\mathcal{N}_3$ is “output safe”. $\mathcal{N}_9$ is obtained by making a single change: Decrease UC($a_4$) from $K$ (“very large number”) to 23. The
principal typing \( T_3 \) for \( N_3 \) makes the assignments \( T_3(\emptyset) = T_3(\{a_1, a_2, a_3, a_4\}) = [0, 0] \) in addition to:

- \( a_1 : [0, 15] \)
- \( a_2 : [0, 25] \)
- \( a_3 : [-15, 0] \)
- \( a_4 : [-23, 0] \)

Finally, if we take the meet \((T_8 \land T_9)\), and build a network \( N_{8 \land 9} \) for which \((T_8 \land T_9)\) is a principal typing, then the substitution of \( N_3 \) (or any other network whose principal typing is \( T_3 \)) for \( N_{8 \land 9} \) is both “input safe” and “output safe”. The typing \((T_8 \land T_9)\) is simple enough that we can build \( N_{8 \land 9} \) by inspection, or by using the methodology of Section 9; the two versions of network \( N_{8 \land 9} \) are shown in Figure 19.

It is worth pointing out that substitution of network \( N_4 \) of Example 15 for \( N_{8 \land 9} \), and substitution of network \( N_3 \land 4 \) for \( N_{8 \land 9} \), are also safe when non-determinism is demonic (relatively easy inspection omitted). \( \square \)

**Figure 19:** For Example 109: Two versions of network \( N_{8 \land 9} \) for which \((T_8 \land T_9)\) is a principal typing, built by inspection from \( N_3 \land 4 \) in Figure 16 (on the left) and by using the methodology of Section 9 (on the right).

The preceding example discusses conditions under which a network \( N_3 \land 4 \) with principal typing \( T_3 \land 4 \), in a larger assembly \( \mathcal{A} \), can be safely replaced by another network. The discussion can be repeated with a hole \( X \) instead of network \( N_3 \land 4 \), where \( X \) is assigned typing \( T_3 \land 4 \) dictated by the rest of \( \mathcal{A} \). In this case \( X \) is a hole with two input ports and two output ports. It does not make sense to say that “\( T_3 \land 4 \) is a principal typing for hole \( X \)”; instead, we say that \( T_3 \land 4 \) is “the most general” (or “least restrictive”) typing for hole \( X \) as allowed by the safe operation of \( \mathcal{A} \). Again here, depending on whether \( N_3 \)’s non-determinism is angelic or demonic, inserting \( N_3 \) in hole \( X \) is safe or unsafe.

Based on the discussion in Example 109, in the presence of demonic non-determinism, we need a notion of subtyping more restrictive than “\( <: \)”, which we call “strong subtyping” and denote by “\( <\!\!\!\!\!: \)”.

**Definition 110 (Strong Subtyping).** Let \( T, U : \mathcal{P}(A_{in,out}) \to \mathcal{P}(\mathbb{R}) \) be two valid typings over the same set \( A_{in,out} \) of input and output arcs. We say \( T \) is input-safe for \( U \) iff:

- For every \( f_{in} : A_{in} \to \mathbb{R}^+ \) satisfying \( [U]_{A_{in}} \), and for every \( g : A_{in,out} \to \mathbb{R}^+ \) extending \( f_{in} \), it holds that: \( g \) satisfies \( T \iff g \) satisfies \( U \).

We say \( T \) is output-safe for \( U \) iff:

- For every \( f_{out} : A_{out} \to \mathbb{R}^+ \) satisfying \( [U]_{A_{out}} \), and for every \( g : A_{in,out} \to \mathbb{R}^+ \) extending \( f_{out} \), it holds that: \( g \) satisfies \( T \iff g \) satisfies \( U \).

68
We say $T$ is safe for $U$, or say $T$ is a strong subtyping of $U$ and write $T \ll U$, iff $T$ is both input-safe and output-safe for $U$.

We state without proof some simple properties of “$\ll$” and conclude with one more open problem.

**Fact 111.** Let $S, T, U : \mathcal{P}(A_{\text{in,out}}) \rightarrow \mathcal{I}(\mathbb{R})$ be tight, total, and valid typings over the input/output set $A_{\text{in,out}}$.

1. $T \ll T$, i.e., “$\ll$” is reflexive.
2. If $T \ll U$ and $U \ll T$, then $T = U$, i.e., “$\ll$” is antisymmetric.
3. If $T \ll S$ and $S \ll T$, then $S \ll U$, i.e., “$\ll$” is transitive.
4. If $T \ll U$ then $T < U$, but not the other way around. (A counter-example for the converse is in Example 109, where $T_3 \ll T_3 \land T_4$ but $T_3 < T_3 \land T_4$.)

Points 1-3 say that “$\ll$” is a partial order, and point 4 says that this partial order can be embedded in the partial order of “$<$”.

**Open Problem 112.** Extend our typing theory to account for strong subtyping “$\ll$”. The set $\text{Valid}(A_{\text{in,out}})$ of all tight, total, and valid typings over $A_{\text{in,out}}$, has the structure of a distributive lattice, with “$<$” as a partial order (directed downward), with a top element $T[A_{\text{in,out}}]$, and a bottom element $T[A_{\text{in,out}}]$, as shown in Section 10. Examine the way in which the partial order “$\ll$” is embedded in this lattice. In particular:

- In analogy with the operators $\lor$ and $\land$ in Section 10, define a least upper bound operator $\forall$, and a greatest lower bound operator $\exists$, that will produce a sublattice $\text{Valid}(A_{\text{in,out}})$ under the partial order “$\ll$”.
- Design efficient algorithms for such operators $\forall$ and $\exists$.
- Design an efficient algorithm to test, given arbitrary $T, U \in \text{Valid}(A_{\text{in,out}})$, whether $T \ll U$.

Observe that the counterparts of these algorithms relative to “$<$” are simple and efficient. For example, to decide whether $T \ll U$ is just a test for interval inclusion, given that $T$ and $U$ are tight, total, and valid typings: $T \ll U$ iff $T(A) \supseteq U(A)$ for every $A \subseteq A_{\text{in,out}}$.

**References**


