1993-12-20

Type Reconstruction in the Presence of Polymorphic Recursion and the Recursive Types

Jahama, Said

Boston University Computer Science Department


https://hdl.handle.net/2144/1477

Boston University
Type Reconstruction in the Presence of Polymorphic Recursion and Recursive Types

Said Jahama *                          A.J. Kfoury †
Boston University                       Boston University
(jahama@cs.bu.edu)                      (kfoury@cs.bu.edu)

December 20, 1993
Technical Report: bu-cs #93-019

Abstract

We establish the equivalence of type reconstruction with polymorphic recursion and recursive types is equivalent to regular semi-unification which proves the undecidability of the corresponding type reconstruction problem. We also establish the equivalence of type reconstruction with polymorphic recursion and positive recursive types to a special case of regular semi-unification which we call positive regular semi-unification. The decidability of positive regular semi-unification is an open problem.

1 Introduction

Semi-unification has developed into a powerful tool in the study of polymorphic type systems in recent years. Various forms of the semi-unification problem, depending on the kind of terms allowed in the inequalities of an instance,
have been shown to be equivalent to the type-reconstruction problem for various polymorphically typed λ-calculi and functional programming languages. This equivalence generalizes the well-known relationship between standard (first-order) unification and typability in the simply-typed λ-calculus. For a sample of results in this area, the reader is referred to [6, 15, 12, 14].

In this report, we extend the theory of semi-unification to deal with polymorphic recursion and recursive types simultaneously. Polymorphic recursion is introduced by a fixpoint constructor, fix, at the object level; recursive types are introduced by a fixpoint constructor, μ, at the type level. Recursive types come in two varieties, with or without the restriction that μ only binds a type variable all of whose occurrences are positive. We obtain therefore two distinct polymorphic type systems, ML+fix+μ and ML+fix+posμ, the first extending the second and the second extending the ML type system.

The importance of polymorphic recursion in programming languages was first observed by Mycroft. Polymorphic recursion allows the definition of a function F to contain recursive calls to F at different types, all instances of the same generic type. Mycroft extended the ML type system with this feature, proved the principal-type property of the resulting system, but left open the corresponding type-reconstruction problem [19]. Subsequently, ML+fix was studied extensively by Henglein [6], Leiss [15], and Kfoury, Tiuryn and Urzyczyn [14], who finally proved the type-reconstruction problem to be undecidable [13]. The importance of recursive types and positive recursive types in programming language theory has been recognized for many years; a sample of recent results, restricted to aspects of type-checking and type-reconstruction, can be found in [1, 3, 17].

The report is organized as follows. We first give a precise definition of recursive and positive recursive types (Section 2) and introduce the systems ML+fix+μ and ML+fix+posμ (Section 3). We call the two system S and S+ for short. These two systems are in fact pared down versions which are sufficient for our purposes here; in particular, not only have we omitted the if-then-else and pairing constructors and other features without which interesting programs cannot be written, but we have also omitted the let-in constructor. The let-in constructor is the only source of polymorphism in

---

1This is not to diminish the importance of semi-unification for other parts of theoretical computer science. See for example [4, 9, 20] as well as the Introduction in [13] for a survey of other application areas. Nevertheless, the greatest successes of semi-unification theory are undoubtedly in the area of polymorphic type systems.
standard ML, and its addition to the simply-typed $\lambda$-calculus turns the type-reconstruction from PTIME-complete to DEXPTIME-complete [8, 16, 14]. However, as shown in [14]), if polymorphic recursion is also added (via the fix constructor), which turns type-reconstruction into an undecidable problem, then we can omit let-in.

We then define two forms of the semi-unification problem (Section 4), denoted RSUP (for regular SUP) and PRSUP (for positive-regular SUP). We prove that RSUP and PRSUP are equivalent to type-reconstruction for ML+fix+$\mu$ and ML+fix+$pos$-$\mu$, respectively (Sections 5 and 6).

Having established these equivalences, we conclude that the type-reconstruction problem for ML+fix+$\mu$ is undecidable and leave the problem open for ML+fix+$pos$-$\mu$ (Section 7).

2 Types

**Definition 1** Let $X$ and $C$ be a countably infinite set of type variables and type constants respectively. The set of recursive types $T_\mu$ is defined as follows:

1. $X \cup C \subseteq T_\mu$.
2. If $\sigma, \tau \in T_\mu$ then $\sigma \to \tau \in T_\mu$.
3. If $\alpha \in X, \sigma \in T_\mu$ then $\mu \alpha.\sigma \in T_\mu$.

We follow the standard convention that $\sigma \to \rho \to \tau$ stands for $(\sigma \to (\rho \to \tau))$. The universal recursive types are expressions of the form $\forall \alpha_1 \cdots \forall \alpha_n.\tau$ where $\alpha_1, \ldots, \alpha_n \in X, n \geq 0$, and $\tau \in T_\mu$. Let $T_\mu^\forall$ be the set of all universal recursive types. The universal quantifier "$\forall$" and the operator $\mu$ bind type variables. We identify $\alpha$-convertible types (types identical up to renaming of bound variables). A substitution is a function $S : X \to T_\mu$. The notation $\sigma[\alpha := \tau]$ stands for the result of substituting in $\sigma$ all free occurrences of $\alpha$ by $\tau$ (after an appropriate renaming of bound variables if necessary). We write $\tau = \sigma[\alpha_1 := \tau_1, \ldots, \alpha_n := \tau_n]$ for simultaneous substitution.

A variable $\alpha$ is positive in a type $\sigma$ iff every free occurrence of $\alpha$ is on the left hand side of an even number of $\to$'s. The set of positive recursive types $T_{\mu,+}$ is defined as follows:

**Definition 2**
1. If $\sigma, \tau \in \mathcal{T}_{\mu,+}$ then $\sigma \rightarrow \tau \in \mathcal{T}_{\mu,+}$.

2. If $\alpha \in \mathcal{T}_{\mu,+}$ then $\mu\alpha.\sigma \in \mathcal{T}_{\mu,+}$.

3. If $\alpha \in X$, $\sigma \in \mathcal{T}_{\mu,+}$ and $\alpha$ is positive in $\sigma$ then $\mu\alpha.\sigma \in \mathcal{T}_{\mu,+}$.

The set of all universal positive recursive types is $\mathcal{T}_{\mu,+} = \{ \forall \alpha_1 \cdots \forall \alpha_n. \tau \mid \tau \in \mathcal{T}_{\mu,+} \}$.

A type $\sigma$ is finite if $\sigma$ does not contain an occurrence of the $\mu$ operator. Let $\mathcal{T}_{\text{fin}}$ be the set of all finite types. Notice that $\mathcal{T}_{\text{fin}} \subseteq \mathcal{T}_{\mu,+} \subseteq \mathcal{T}_{\mu}$. Let $\mathcal{T}^*$ be the set of finite and infinite labeled binary trees with labels over $X \cup C \cup \rightarrow$.

A subtype of a type $\sigma \in \mathcal{T}^*$ consists of a node of $\sigma$ and all its descendants in $\sigma$. A (possibly infinite) type $\sigma$ is regular if the set of its subtypes is finite. Let $\mathcal{T}_{\text{reg}}$ be the set of all regular types.

For a type $\sigma$ of the form $\mu\alpha.\tau$ the unfolding of $\sigma$ for one step results in the type $\tau[\alpha := \mu\alpha.\tau]$. Every recursive type $\sigma$ represents an underlying regular type obtained by unfolding $\sigma$ infinitely many times. More formally there is a map $(\ )^* : \mathcal{T}_{\mu} \rightarrow \mathcal{T}_{\text{reg}}$. We refer the reader to [3] for an exact definition of $(\ )^*$. It is also true that every type in $\mathcal{T}_{\text{reg}}$ has a notation (not unique) in $\mathcal{T}_{\mu}$. We refer the reader to [2] for the proof of this fact, the reference also contains a detailed discussion of infinite and regular types.

This means that, whenever appropriate, we can use properties of $\mathcal{T}_{\text{reg}}$ to prove results for $\mathcal{T}_{\mu}$ and vice versa. In particular, we can view regular semi-unification as semi-unification on recursive terms. We use this fact to prove the undecidability of type reconstruction in system $\mathcal{S}$.

There are two standard notions of equivalence of recursive types, referred to as strong ($\approx$) and weak ($\sim$) equivalence. $\sigma \approx \tau$ iff $\sigma^* = \tau^*$, i.e. they represent the same regular type. For $\sim$ we use the definition given in [3]:

**Definition 3** Let $\sim \subseteq \mathcal{T}_{\mu} \times \mathcal{T}_{\mu}$ be the smallest equivalence relation satisfying

1. $\mu\alpha.\sigma \sim \sigma[\alpha := \mu\alpha.\sigma]$.

2. $\sigma \sim \sigma'$ and $\tau \sim \tau' \Rightarrow \sigma \rightarrow \tau \sim \sigma' \rightarrow \tau'$.

3. $\sigma \sim \sigma' \Rightarrow \mu\alpha.\sigma \sim \mu\alpha.\sigma'$.

Observe that $\sigma \sim \tau$ implies $\sigma \approx \tau$. However, the converse is false, for example:

$$\mu\alpha.\alpha \rightarrow \alpha \approx \mu\alpha.\left((\alpha \rightarrow \alpha) \rightarrow \alpha\right)$$
while it is not the case that $\mu\alpha.\alpha \rightarrow \alpha \sim \mu\alpha.((\alpha \rightarrow \alpha) \rightarrow \alpha)$. Observe also that the relations $\approx$ and $\sim$ are both decidable [3].

3 Systems $S$ and $S_+$

In this thesis we consider a simple language consisting of $\lambda$-terms augmented with a polymorphic fix constructor and a set of constants. Unless otherwise noted, we refer to object constants by $a, b, c, \cdots$ and object variables by $x, y, z, \cdots$. The augmented $\lambda$-terms considered here are defined by the grammar:

$$M ::= x \mid a \mid (MN) \mid (\lambda x \ M) \mid (\text{fix } x \ M)$$

As usual, the constructors $\lambda$ and fix are assumed to bind variables. We adopt the standard notion of $\alpha$-conversion, and we generally do not distinguish between $\alpha$-convertible terms.

We describe two type inference systems $S$ and $S_+$. The two systems differ on the types and the equivalence relation each uses. $S$ uses recursive types and the equivalence relation $\approx$, while $S_+$ uses positive recursive types and $\sim$.

The type inference systems $S$ and $S_+$ are shown in Figures 1 and 2 respectively. We follow standard notation and terminology. An environment $A$ is a finite set of type assumptions $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ associating at most one type $\sigma$ with each object variable $x$. By $FV(A)$ we denote the set of all type variables occurring free in $A$. Viewing $A$ as a partial function from object variables to types, we may write $A(x) = \sigma$ to mean that the assumption $x : \sigma$ is in $A$. An assertion is an expression of the form $A \vdash M : \tau$ where $A$ is an environment, $M$ a term and $\tau$ a type. In such an assertion, the $\sigma$'s (mentioned in $A$) are called the environment types, and $\tau$ the assigned or derived type. Derivability in $S$ and $S_+$ will be denoted be the symbols $\vdash_\mu$ and $\vdash_{\mu,+}$, respectively.
\[
\text{VAR} \quad A \vdash x : \sigma \quad A(x) = \sigma, \quad \sigma \in T^\gamma_
\]
\[
\text{CONST} \quad A \vdash a : \sigma \quad \sigma \text{ is a type constant}, \quad \sigma \in T^\gamma_
\]
\[
\text{INST} \quad \frac{A \vdash M : \forall \alpha.\sigma}{A \vdash M : \sigma[\alpha := \tau]} \quad \tau \in T^\mu, \quad \sigma \in T^\gamma_
\]
\[
\text{GEN} \quad \frac{A \vdash M : \sigma}{A \vdash M : \forall \alpha.\sigma} \quad \alpha \notin \text{FV}(A), \quad \sigma \in T^\gamma_
\]
\[
\text{APP} \quad \frac{A \vdash M : \sigma \rightarrow \tau, \quad A \vdash N : \sigma}{A \vdash (MN) : \tau} \quad \tau, \sigma \in T^\mu
\]
\[
\text{ABS} \quad \frac{A[x : \sigma] \vdash M : \tau}{A \vdash (\lambda x \ M) : \sigma \rightarrow \tau} \quad \tau, \sigma \in T^\mu
\]
\[
\text{FIX} \quad \frac{A[x : \sigma] \vdash M : \sigma}{A \vdash (\text{fix } x \ M) : \sigma} \quad \sigma \in T^\gamma
\]
\[
\approx \quad \frac{A \vdash M : \sigma, \quad \sigma \approx \tau}{\top \vdash M : \tau} \quad \tau, \sigma \in T^\mu
\]

**Figure 1.** System \(\mathcal{S}\): all environment types and derived types in \(T^\gamma\).
VAR  \[ A \vdash x : \sigma \quad A(x) = \sigma, \quad \sigma \in T_{\mu,+}^\forall \]

CONST  \[ A \vdash a : \sigma \quad \sigma \text{ is a type constant}, \quad \sigma \in T_{\mu,+}^\forall \]

INST  \[ \begin{array}{c} A \vdash M : \forall \alpha.\sigma \\
A \vdash M : \sigma[\alpha := \tau] \end{array} \quad \tau \in T_{\mu,+}^\forall, \quad \sigma \in T_{\mu,+}^\forall \]

GEN  \[ A \vdash M : \sigma \quad \alpha \notin \text{FV}(A), \quad \sigma \in T_{\mu,+}^\forall \]

APP  \[ \begin{array}{c} A \vdash M : \sigma \rightarrow \tau, \quad A \vdash N : \sigma \\
A \vdash (MN) : \tau \end{array} \quad \tau, \sigma \in T_{\mu,+}^\forall \]

ABS  \[ \begin{array}{c} A[x : \sigma] \vdash M : \tau \\
A \vdash (\lambda x\ M) : \sigma \rightarrow \tau \end{array} \quad \tau, \sigma \in T_{\mu,+}^\forall \]

FIX  \[ \begin{array}{c} A[x : \sigma] \vdash M : \sigma \\
A \vdash (\text{fix } x\ M) : \sigma \end{array} \quad \sigma \in T_{\mu,+}^\forall \]

\[ \sim \]

\[ \begin{array}{c} A \vdash M : \sigma, \quad \sigma \sim \tau \\
A \vdash M : \tau \end{array} \quad \tau, \sigma \in T_{\mu,+}^\forall \]

**Figure 2.** System \( S_+ \): all environment types and derived types in \( T_{\mu,+}^\forall \).

### 3.1 Syntax-oriented rules for \( S \) and \( S_+ \)

Both system \( S \) and \( S_+ \) are not syntax-oriented in the sense that there could be more than one derivation tree for a certain assertion. In this subsection, we give a syntax-oriented version of \( S \) and \( S_+ \). This simplifies the proofs in this report. This sort of simplification is a standard step in many papers dealing with polymorphic recursion; see [5, 6, 14, 18]. Let \( \sigma, \tau \in T_{\mu} \) and \( \tilde{\alpha} = \alpha_1 \cdots \alpha_n \) for some \( n \geq 0 \). We write \( \forall \tilde{\alpha}.\sigma \preceq \tau \) to mean that \( \tau \) is an instantiation of \( \forall \tilde{\alpha}.\sigma \)

\[ \tau \approx \sigma[\alpha_1 := \tau_1, \ldots, \alpha_n := \tau_n], \text{ for some } \tau_1, \ldots, \tau_n \in T_{\mu}. \]
Similarly, for $\sigma, \tau \in T_{\mu,+}$ and $\bar{\alpha} = \alpha_1 \cdots \alpha_n$, $\forall \bar{\alpha} \sigma \preceq_+ \tau$ iff

$$\tau \sim \sigma[\alpha_1 := \tau_1, \ldots, \alpha_n := \tau_n], \text{ for some } \tau_1, \ldots, \tau_n \in T_{\mu,+}.$$ 

Instantiation corresponds to a sequence of applications of rule INST and rule $\approx$ (rule $\sim$ in $S'$), which leads to the following lemma.

**Lemma 4**

1. If $A \vdash_{\mu} M : \sigma$ and $\sigma \preceq \tau$ then $A \vdash_{\mu} M : \tau$.

2. If $A \vdash_{\mu,+} M : \sigma$ and $\sigma \preceq_+ \tau$ then $A \vdash_{\mu,+} M : \tau$.

The modification $S'$ and $S'_+$ of $S$ and $S_+$ respectively, shown in Figures 3 and 4, consists in removing rules INST and GEN and modifying the VAR and FIX rules. The resulting systems are partially syntax-oriented in the sense that the derivation of an assertion is unique up to applications of rule $(\approx)$ in $S$ and $(\sim)$ in $S_+$. Derivability in $S'$ and $S'_+$ will be denoted by the symbols $\vdash'$ and $\vdash'_{\mu,+}$. To keep notation simple, when it is clear from the context which system we are considering, we will simply use the symbol $\vdash$ to denote derivability in that particular system.
<table>
<thead>
<tr>
<th><strong>VAR</strong></th>
<th>( A \vdash x : \tau )</th>
<th>( A(x) = \sigma, \ \sigma \in \mathcal{T}<em>\mu^\forall, \ \tau \in \mathcal{T}</em>\mu, \ \sigma \preceq \tau )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONST</strong></td>
<td>( A \vdash a : \sigma )</td>
<td>( \sigma ) is a type constant, ( \sigma \in \mathcal{T}_\mu )</td>
</tr>
<tr>
<td><strong>APP</strong></td>
<td>( A \vdash M : \sigma \rightarrow \tau, \ A \vdash N : \sigma )</td>
<td>( \frac{A \vdash (M \ N) : \tau}{\tau, \sigma \in \mathcal{T}_\mu} )</td>
</tr>
<tr>
<td><strong>ABS</strong></td>
<td>( A[x : \sigma] \vdash M : \tau )</td>
<td>( \frac{A \vdash (\lambda x \ M) : \sigma \rightarrow \tau}{\tau, \sigma \in \mathcal{T}_\mu} )</td>
</tr>
<tr>
<td><strong>FIX</strong></td>
<td>( A[x : \forall \bar{a} \sigma] \vdash M : \sigma )</td>
<td>( \frac{A \vdash (\text{fix } x \ M) : \tau}{\sigma, \tau \in \mathcal{T}_\mu, \ \forall \bar{a} \sigma \preceq \tau, \ \bar{a} \notin \text{FV}(A)} )</td>
</tr>
</tbody>
</table>

\[ \approx \]
| \( A \vdash M : \sigma, \ \sigma \approx \tau \) | \( \frac{A \vdash M : \tau}{\tau, \sigma \in \mathcal{T}_\mu} \) |

**Figure 3.** System \( S^\forall \): all environment types in \( \mathcal{T}_\mu^\forall \). All derived types in \( \mathcal{T}_\mu^* \).
\[
\begin{align*}
\text{VAR} & \quad A \vdash x : \tau & A(x) = \sigma, & \sigma \in \mathcal{T}_{\mu,+}^\gamma, & \tau \in \mathcal{T}_{\mu,+}, & \sigma \preceq_+ \tau \\
\text{CONST} & \quad A \vdash a : \sigma & \sigma \text{ is a type constant, } \sigma \in \mathcal{T}_{\mu,+} \\
\text{APP} & \quad \frac{A \vdash M : \sigma \to \tau, \quad A \vdash N : \sigma}{A \vdash (M \ N) : \tau} & \tau, \sigma \in \mathcal{T}_{\mu,+} \\
\text{ABS} & \quad \frac{A \vdash M : \tau}{A \vdash (\lambda x \ M) : \sigma \to \tau} & \tau, \sigma \in \mathcal{T}_{\mu,+} \\
\text{FIX} & \quad \frac{A \vdash \forall \alpha. \sigma \vdash M : \sigma}{A \vdash (\text{fix } x \ M) : \tau} & \sigma, \tau \in \mathcal{T}_{\mu,+}, & \forall \alpha. \sigma \preceq_+ \tau, & \alpha \notin \text{FV}(A) \\
\sim & \quad \frac{A \vdash M : \sigma, \quad \sigma \sim \tau}{A \vdash M : \tau} & \tau, \sigma \in \mathcal{T}_{\mu,+}
\end{align*}
\]

**Figure 4.** System \( S_\mu^+ \): all environment types in \( \mathcal{T}_{\mu,+}^\gamma \). All derived types in \( \mathcal{T}_{\mu,+} \).
The main result of this subsection is Lemma 5. It is similar to Lemma 5 in [14] and Lemma 5 in [6]. The proof of this lemma is adopted from the Proof of Lemma 5 in [6].

**Lemma 5** Let $M$ be a term, $A$ an environment, $\sigma \in \mathcal{T}_\mu$ (resp., $\sigma \in \mathcal{T}_{\mu,+}$) and $\vec{\alpha}$ a sequence of zero or more type variables where $\vec{\alpha} \notin FV(A)$:

$$A \vdash_{\mu} M : \forall \vec{\alpha}. \sigma \iff A \vdash'_{\mu} M : \sigma$$

(resp., $A \vdash_{\mu,+} M : \forall \vec{\alpha}. \sigma \iff A \vdash'_{\mu,+} M : \sigma$).

**Proof:** For the “only if” direction, we use structural induction on derivations in $S$ and $S_+$. The cases where we have a single derivation are rules CONST and VAR. For the VAR rule, assume that $A(x) = \forall \vec{\alpha}. \sigma \in \mathcal{T}_{\mu}'$. Applying the VAR rule in system $S$ we have:

$$A \vdash_{\mu} x : \forall \vec{\alpha}. \sigma$$

Using the VAR rule in $S'$ and by observing that $\forall \vec{\alpha}. \sigma \preceq \sigma$ (also if $\sigma \in \mathcal{T}_{\mu,+}$ then $\forall \vec{\alpha}. \sigma \preceq_+ \sigma$) we have:

$$A \vdash'_{\mu} x : \sigma$$

For the CONST rule, observe that if the CONST rule in $S$ is used to obtain $A \vdash_{\mu} a : \sigma$ then we can use the CONST rule in $S'$ to obtain $A \vdash'_{\mu} a : \sigma$. A similar argument can be used in the case of the VAR and CONST rules in $S_+$.

For the FIX rule in $S$, assume that $A \vdash_{\mu} (\text{fix } x M) : \forall \vec{\alpha}. \sigma$ is derivable using the FIX rule in $S$ i.e.

$$A_{[x : \forall \vec{\alpha}. \sigma]} \vdash M : \forall \vec{\alpha}. \sigma$$

By applying the FIX rule in $S'$ and using the induction hypothesis we get:

$$A_{[x : \forall \vec{\alpha}. \sigma]} \vdash M : \sigma$$

We also have $\forall \vec{\alpha}. \sigma \preceq \sigma$ and by assumption, $\forall \vec{\alpha} \notin FV(A)$. hence, we can apply the FIX rule in $S'$ to get:

$$A_{[x : \forall \vec{\alpha}. \sigma]} \vdash M : \sigma$$

$$A \vdash (\text{fix } x M) : \sigma$$
Again, a similar argument is used for the FIX rule in $S_+$. The inductive proof for the other rules in $S$ and $S_+$ is straightforward.

For the “if” direction, notice that, it is sufficient to show the following:

1. If $A \vdash'_μ M : σ$ then $A \vdash' μ M : σ$.
2. If $A \vdash'_μ + M : σ$ then $A \vdash' μ + M : σ$.

We prove this by producing for every rule in $S'$ and $S'_+$ a corresponding derivation in $S$ ($S_+$ respectively). The only non trivial cases are rules VAR and FIX. For the VAR rule in $S'$ ($S'_+$ respectively), assume that $A(x) = ∀\vec{α}.τ$ and $∀\vec{α}.τ \preceq σ$. Applying the VAR rule we get:

$$A \vdash σ$$

By using the VAR rule in $S$ (in $S_+$ respectively) and by Part 1 of Lemma 4 (Part 2 in the case of $S_+$), we derive the following in $S$ ($S_+$ respectively):

$$A \vdash σ.$$ 

Now, for the FIX rule, assume that the last rule we apply in a derivation in $S'$ is the FIX rule, i.e. :

$$A[x : ∀\vec{α}.τ] \vdash M : τ$$

$$\frac{A \vdash (\text{fix } x M) : σ}{A \vdash (\text{fix } x M) : σ}$$

where $∀\vec{α}.τ \preceq σ$. We get the same derivation in $S$, by applying the GEN rule as needed to get

$$A[x : ∀\vec{α}.τ] \vdash M : ∀\vec{α}.τ$$

and then we apply the FIX rule in $S$ to obtain

$$A \vdash (\text{fix } x M) : ∀\vec{α}.τ$$

Now, we use Part 1 of Lemma 4 to get:

$$A \vdash (\text{fix } x M) : σ$$

A similar argument applies for $S'_+$.
4 Positive Regular Semi-Unification

As a result of the equivalence of the sets $\mathcal{T}_{\text{reg}}$ and $\mathcal{T}_\mu$, we can look at a regular substitution as a substitution from $S : X \rightarrow \mathcal{T}_\mu$. In this section, we redefine regular semi-unification and define positive regular semi-unification.

A regular (resp. positive regular) substitution $S$ is a function $S : X \rightarrow \mathcal{T}_\mu$ (resp., $S : X \rightarrow \mathcal{T}_{\mu,+}$). Every regular (resp. positive regular) substitution $S$ can be extended in a natural way to a function $S : \mathcal{T}_\mu \rightarrow \mathcal{T}_\mu$ (resp., $S : \mathcal{T}_{\mu,+} \rightarrow \mathcal{T}_{\mu,+}$) [3].

An instance $\Gamma$ of semi-unification is a finite set of inequalities:

$$\Gamma = \{ \sigma_1 \leq \tau_1, \ldots, \sigma_n \leq \tau_n \}$$

where $t_i, u_i \in \mathcal{T}_{\text{fin}}$. A regular substitution $S$ is a regular solution of the instance $\Gamma$ iff there are substitutions $S_1, \ldots, S_n$ such that:

$$S_i(S(\sigma_i)) \approx S(\tau_i), \ldots, S_n(S(\sigma_n)) \approx S(\tau_n)$$

The Regular semi-unification Problem (RSUP) is the problem of deciding, for any such instance $\Gamma$, whether $\Gamma$ has a regular solution.

An instance $\Gamma$ has a positive regular solution if there is a positive regular substitution $S : \mathcal{T}_{\mu,+} \rightarrow \mathcal{T}_{\mu,+}$ and positive regular substitutions $S_1, \ldots, S_n : \mathcal{T}_{\mu,+} \rightarrow \mathcal{T}_{\mu,+}$ such that:

$$S_i(S(\sigma_i)) \sim S(\tau_i), \ldots, S_n(S(\sigma_n)) \sim S(\tau_n)$$

The Positive Regular semi-unification Problem (PRSUP) is the problem of deciding, for any such instance $\Gamma$, whether $\Gamma$ has a positive regular solution.

5 From $S$ to RSUP and from $S_+$ to PRSUP

Given a term $M$ we construct an instance $\Gamma_M$ of semi-unification such that:

1. $M$ is typable in $S$ iff $\Gamma_M$ has a regular solution.

2. $M$ is typable in $S'$ iff $\Gamma_M$ has a positive regular solution.
The construction given here is very similar to the construction given in Section 4.2 [14]. The proofs here differ slightly (but still the same style) because the syntax-oriented version given here does not have the GEN rule. Also, constants are added here. We view our construction as an extension of the construction given in [14] and we use most of the definitions related to it.

We begin by constructing a set of equalities

\[ \Delta_M = \{ \sigma_1 = \tau_1, \ldots, \sigma_p = \tau_p \} \]

where \( \sigma_i, \tau_i \in T_{\text{fin}}, i \in \{1, \ldots, p\} \). We follow the convention that any variable occurring in \( M \) is a member of one of the two lists \( x_0, x_1, \ldots \) and \( y_0, y_1, \ldots \). Furthermore, if a variable occurs free or fix-bound then it is a member of the list \( x_0, x_1, \ldots \). Otherwise, if a variable is \( \lambda \)-bound then it is a member of the list \( y_0, y_1, \ldots \). Any constant occurring in \( M \) is from the set \( a_0, a_1, \ldots \).

Let \( M_1, M_2, \ldots, M_n \) be an enumeration of all the subterms of \( M \) such that, for \( k = 1, \ldots, n \), if \( M_k \) is not an object variable, then \( M_k = (M_i M_j) \) or \( (\lambda v. M_i) \) or \((\text{fix } v. M_i)\) for some \( i \neq j \) and \( i, j \in \{1, 2, \ldots, k-1\} \). The set \( \{M_1, M_2, \ldots, M_n\} \) mentions all occurrences of the same subterm, i.e., we may have \( M_i = M_j \) for \( i \neq j \). Observe that \( M = M_n \).

**Definition of \( \Delta_k \) for \( k = 1, \ldots, n \):**

Simultaneously with \( \Delta_k \) we define a type expression \( t_k \) with variables in \( V \), by induction on \( k = 1, \ldots, n \):

1. If \( M_k \) is the \( j \)-th occurrence of \( x_i \) in \( M \), then set \( \Delta_k = \emptyset \) and \( t_k = \beta_i^{(j-1)} \).
   (We number the occurrences, free or bound, of \( x_i \) in \( M \) with \( 0, 1, 2, \ldots \), starting from the left end of \( M \). If \( x_i \) is bound in \( M \), the binding occurrence of \( x_i \), \( \text{fix } x_i \), is not counted.)

2. If \( M_k = y_i \), then set \( \Delta_k = \emptyset \) and \( t_k = \gamma_i \).

3. If \( M_k = a_i \), then set \( \Delta_k = \emptyset \) and \( t_k = c_i \).

4. If \( M_k = (M_i M_j) \) then set \( \Delta_k = \Delta_i \cup \Delta_j \cup \{t_i \rightarrow t_j \rightarrow \delta \} \) and \( t_k = \delta \), where \( \delta \) is a fresh auxiliary variable.

5. If \( M_k = (\lambda y_i. M_j) \) then set \( \Delta_k = \Delta_j \) and \( t_k = \gamma_i \rightarrow t_j \).
6. If \(M_k = (\text{fix } x_i, M_j)\) then set \(\Delta_k = \Delta_j \cup \{\beta_i \equiv t_j\}\) and \(t_k = \beta_i^{(l)}\), where \(l \geq 0\) is the number of bound occurrences of \(x_i\) in \(M_j\) \((\beta_i^{(0)}, \ldots, \beta_i^{(l-1)}\) are already introduced in \(\Delta_1, \ldots, \Delta_{k-1}\), corresponding to the bound occurrences of \(x_i\)).

Instead of \(\Delta_n\) and \(t_n\), we also write \(\Delta_M\) and \(t_M\).

The only difference between \(\Delta_M\) here and in [14] is that we add constants here and we do not allow polymorphic abstraction. We define subsets \(V_0, V_1\) of the variables occurring in \(\Delta_M\) as follows:

\[
V_0 = \{\beta_i^{(0)}, \ldots, \beta_i^{(l-1)} \mid \text{there are } l \geq 0 \text{ free or bound occurrences of } x_i \text{ in } M\} \\
\cup \{\gamma_i \mid y_i \text{ occurs in } M\} \\
\cup \{\delta_i \mid \delta_i \text{ occurs in } \Delta_M\} \\
V_1 = \{\beta_i \mid x_i \text{ occurs in } M\}
\]

For \(\sigma \in T_\mu \cup T_{\mu,+}\) and \(\bar{\alpha}\) a finite sequence (possibly empty) of type variables, we define \(\text{body}(\forall \bar{\alpha}.\sigma) = \sigma\).

In what follows \(S\) denotes a map from \(V\) to \(T_\mu\) and \(S_+\) denotes a map from \(V\) to \(T_{\mu,+}\) such that, for every \(\alpha\) not occurring in \(\Delta_M\), \(S(\alpha) = \alpha\) and \(S_+(\alpha) = \alpha\). We further restrict \(S\) and \(S_+\) so that, for every \(\beta_i \in V_1\), \(S(\beta_i) \in T_\mu\) and \(S_+(\beta_i) \in T_{\mu,+}\) and for every \(\alpha \in V_0\), \(S(\alpha) \in T_\mu\) and \(S_+(\alpha) \in T_{\mu,+}\). With every such \(S\) and \(S_+\) we associate the maps \(\bar{S}\) from \(V\) to \(T_\mu\) and \(\bar{S}_+\) from \(V\) to \(T_{\mu,+}\) respectively, satisfying the condition that, for every \(a \in V\), \(\bar{S}(a) = \text{body}(S(a))\) and \(\bar{S}_+(a) = \text{body}(S_+(a))\).

Given a term \(M\), the symbol \(\leq_M\) denotes a partial order on object variables relative to \(M\). For every \(x \in \{x_0, x_1, x_2, \ldots\}\) and every \(y \in \{y_0, y_1, y_2, \ldots\}\):

\(x \leq_M y\) iff both \(x\) and \(y\) are bound in \(M\) and the (fix) binding of \(x\) is in the scope of the \(\lambda\)-binding of \(y\).

We now define what it means for \(S\) and \(S_+\) to be a regular solution (positive regular solution, respectively) for \(\Delta_k\). Notice that such a solution is not the same as a solution for an instance of semi-unification.

**Definition 6** \(S\) is a regular solution for \(\Delta_k\) for \(k = 1, \ldots, n\) iff the following conditions hold:
1. For every equality $\sigma_i \doteq \tau_i \in \Delta_k$, $S(\sigma_i) \approx S(\tau_i)$.

2. For every $\beta_i, \beta_i^{(j)}$ occurring in $\Delta_M$, $S(\beta_i) \preceq S(\beta_i^{(j)})$.

3. For all $\beta_i$ occurring in $\Delta_M$, the bound variables of $S(\beta_i)$ are precisely the set:
   \[ \text{FV}(S(\beta_i)) = \bigcup \{ \text{FV}(S(\gamma_j)) \mid x_i \leq_M y_j \} \]

**Definition 7** $S_+$ is a positive regular solution for $\Delta_k$ for $k = 1, \ldots, n$ iff the following conditions hold:

1. For every equality $\sigma_i \doteq \tau_i \in \Delta_k$, $S_+(\sigma_i) \sim S_+(\tau_i)$.

2. For every $\beta_i, \beta_i^{(j)}$ occurring in $\Delta_M$, $S_+(\beta_i) \preceq_+ S_+(\beta_i^{(j)})$.

3. For all $\beta_i$ occurring in $\Delta_M$, the bound variables of $S(\beta_i)$ are precisely the set:
   \[ \text{FV}(S_+(\beta_i)) = \bigcup \{ \text{FV}(S_+(\gamma_j)) \mid x_i \leq_M y_j \} \]

The following lemma is an extension of Lemma 12 in [14].

**Lemma 8** Let $M$ be a term. Then:

1. If there is an environment $A$ and a type $\tau \in \mathcal{T}_\mu$ such that $A \vdash_\mu M : \tau$, then $\Delta_M$ has a regular solution $S$ such that $\text{body}(S(t_M)) \approx \tau$ and $S(\beta_i) \approx A(x_i)$ for every $i$.

2. If there is an environment $A$ and a type $\tau \in \mathcal{T}_{\mu,+}$ such that $A \vdash_{\mu,+} M : \tau$, then $\Delta_M$ has a positive regular solution $S_+$ such that $\text{body}(S_+(t_M)) \sim \tau$ and $S_+(\beta_i) \sim A(x_i)$ for every $i$.

3. If $S$ is a regular solution of $\Delta_M$, then $A \vdash_\mu M : \tau$ for some environment $A$ and $\tau \in \mathcal{T}_\mu$ such that $\tau \approx \text{body}(S(t_M))$ and $A(x_i) \approx S(\beta_i)$ for every $i$.

4. If $S_+$ is a positive regular solution of $\Delta_M$, then $A \vdash_{\mu,+} M : \tau$ for some environment $A$ and $\tau \in \mathcal{T}_{\mu,+}$ such that $\tau \sim \text{body}(S(t_M))$ and $A(x_i) \sim S(\beta_i)$ for every $i$. 

16
**Proof:** First we observe the following facts about derivations in systems $S'$ and $S'_0$:

- If $A \vdash N : \sigma$ is an assertion in a derivation and $v$ is an $x$- or $y$-variable, then $A(v)$ is defined if either $v$ is free in $M$ or $v$ is bound in $M$ and $N$ is in the scope of the binding of $v$.

- If $A[x_i : \forall \bar{a} \cdot \sigma] \vdash N : \tau$ is the assertion immediately preceding an application of FIX that discharges the type assumption $(x_i : \forall \bar{a} \cdot \sigma)$, we can assume that the bound type variables $\bar{a}$ are precisely:

$$\bar{a} = \text{FV}(\sigma) - \bigcup \{\text{FV}(A(y_j)) \mid A(y_j) \text{ defined.}\}$$

Let $M_1, M_2, \ldots, M_n$ be an enumeration of all the subterms of $M$. The proof of Parts 1 and 2 is by induction on $k = 1, \ldots, n$. The proofs of Parts 1 and 2 are very similar, and we show the inductive proof of Part 1 only. For Part 1 we need to show that for every $k = 1, \ldots, n$, if $A \vdash M_k : \tau$ for some environment $A$ and a type $\tau \in T_\mu$, then $\Delta_k$ has a solution $S$ such that $\text{body}(S(t_k)) \approx \tau$, $S(\beta_i) \approx A(x_i)$ for every $x_i$, and $S(\gamma_i) \approx A(y_i)$ for every $y_i$. For the basis step, we need to consider the following cases:

1. $M_1$ is the $j$-th occurrence of $x_i$ in $M$.
2. $M_1 = y_i$.
3. $M_1 = a_i$.

In the three cases above it is straightforward to see that there is a regular solution $S$ for $\Delta_j$ such that $\text{body}(S(t_j)) \approx \tau$, $S(\beta_i) \approx A(x_i)$ for every $x_i$, and $S(\gamma_i) \approx A(y_i)$ for every $y_i$. For the induction step, we just show one case as an example, the other cases are similar. Assume that $M_k = (\text{fix } x_i, M_j)$ and $M_k \vdash' \bar{\sigma}$ which implies that $A(x_i) = \forall \bar{a} \cdot \sigma$ and $M_j \vdash \bar{\sigma}$, by the FIX rule of system $S$. By induction hypothesis, there is a solution $S$ for $\Delta_j$ such that $\text{body}(S(t_j)) \approx \bar{\sigma}$ and $S(x_i) \approx A(\beta_i)$. We can easily adjust $S(t_j)$ to force $S(t_j) \approx A(\beta_i)$. Hence, from Step 6 of the construction of $\Delta_k$, we can easily check that $S$ is a solution for $\Delta_k$ satisfying all the conditions of Part 1.

The proofs of Parts 3 and 4 is also by induction on $k$. We show the proof of Part 3 and we omit the proof of Part 4 because it is very similar.
For Part 3, we need to show that if \( S \) is a regular solution of \( \Delta_k \), then there is an environment \( A \) and type \( \tau \in T_\mu \) such that: \( A \vdash^T \mu M_k : \tau, \tau \approx \text{body}(S(t_k)), A(x_i) \approx S(\beta_i) \) for every \( x_i \), and \( A(y_i) \approx S(\gamma_i) \) for every \( y_i \). For the basis step, again, we need to consider the three cases mentioned above. It is straightforward to see that the basis step is correct. For the induction step, again we only consider one case as an example. Assume that \( M_k = (\text{fix } x_i, M_j) \). Observe that if \( S \) is a solution for \( \Delta_k \) then it is a solution for \( \Delta_j \). Assume that \( S(t_j) \approx \forall \alpha \cdot \sigma \) and \( S(t_k) \approx \tau \). From Step 6 of the construction, we can conclude that \( S(t_j) \approx S(\beta_i) \) and \( \forall \alpha \cdot \sigma \preceq \tau \). By induction hypothesis, there is an environment \( A \) such that \( A \vdash^T \mu M_j : \sigma \) and \( A(x_i) \approx \forall \alpha \cdot \sigma \). Let \( B = A - [\forall \alpha \cdot \sigma] \). Using the FIX rule of system \( S' \) we can conclude that \( B \vdash^T \mu M_k : \tau \). 

We now define an instance \( \Gamma_M \) of semi-unification such that \( \Gamma_M \) has a solution in the sense of semi-unification if \( \Delta_M \) has a solution.

**Definition of \( \Gamma_M \):**

Let \( M \) be a term and let \( \Delta_M = \{ \sigma_1 \doteq \tau_1, \ldots, \sigma_p \doteq \tau_p \} \) be the set of equalities obtained as described above. Let \( q \) be the largest index such that \( \delta_q \) occurs in \( \Delta_M \).

1. \( \Gamma_M \) contains the inequality \( (T, U) \) where:

   \[
   T = (\delta_{q+1} \rightarrow \delta_{q+1}) \rightarrow \cdots \rightarrow (\delta_{q+p} \rightarrow \delta_{q+p})
   \]

   \[
   U = (\sigma_1 \rightarrow \tau_1) \rightarrow \cdots \rightarrow (\sigma_p \rightarrow \tau_p)
   \]

   where \( \delta_{q+1}, \ldots, \delta_{q+p} \) are fresh auxiliary variables.

2. For every \( \beta_i, \beta_i^{(j)} \) where \( \beta_i \in V_1 \), \( \Gamma_M \) contains the inequality \( (T_{ij}, U_{ij}) \) where:

   \[
   T_{ij} = \beta_i \rightarrow \gamma_{k_1} \rightarrow \cdots \rightarrow \gamma_{k_t}
   \]

   \[
   U_{ij} = \beta_i^{(j)} \rightarrow \gamma_{k_1} \rightarrow \cdots \rightarrow \gamma_{k_t}
   \]

   where \( \{ \gamma_{k_1}, \ldots, \gamma_{k_t} \} = \{ \gamma_m \mid x_i \leq_M y_m \} \).

3. \( \Gamma_M \) contains no other inequality.

**Lemma 9** If \( M \) is a term, Then:
1. For any \( S : V \rightarrow T^\forall \), \( \bar{S} \) is a regular solution of \( \Gamma_M \) (in the sense of semi-unification) iff \( S \) is a regular solution of \( \Delta_M \) (in the sense of definition 6).

2. For any \( S_+ : V \rightarrow T^\forall_+ \), \( \bar{S}_+ \) is a positive regular solution of \( \Gamma_M \) (in the sense of semi-unification) iff \( S_+ \) is a positive regular solution of \( \Delta_M \) in the sense definition 7).

**Proof:** This reproduces the proof of Lemma 13 in [14] with the necessary terminological changes. Consider the inequality \((T, U)\) introduced in part 1 of the definition of \( \Gamma_M \). \( \bar{S} \) is a regular (resp. positive regular) solution of \( \Delta_M \) in the sense of definition 6 (resp. definition 7) iff \( S \) is a regular (resp. positive regular) solution of \( \{(T, U)\} \) in the sense of semi-unification. Consider an inequality \((T_{ij}, U_{ij})\) introduced in part 2 of definition 6 (resp. definition 7) of \( \Gamma_M \). It is readily checked that \( S(\beta_i) \preceq S(\beta^{(j)}_i) \) (resp. \( S(\beta_i) \preceq_+ S(\beta^{(j)}_i) \)) and the bound variables of \( S(\beta_i) \) are:
\[
\text{FV}(\bar{S}(\beta_i)) = \bigcup \{\text{FV}(\bar{S}(\gamma_{ij})) \mid x_i \leq_M y_j\}
\]
iff \( S \) is a regular (resp. positive regular) solution of \( \{(T_{ij}, U_{ij})\} \) in the sense of semi-unification.

6 From RSUP to \( S \) and from PRSUP to \( S_+ \)

In this section, we use the same construction given in Section 4.3 of [14] and we reproduce most of the text of Section 4.3 with the necessary modifications. We begin with a technical trick which is used to force an object variable to be assigned a particular finite type (or a substitution instance of it). Let \( z \) be an object variable and \( \tau \) a finite type. Type variables are named \( \alpha_0, \alpha_1, \alpha_2, \ldots \), corresponding to which we introduce object variables \( v_0, v_1, v_2, \ldots \). Type constants are named \( c_0, c_1, c_2, \ldots \), corresponding to which we introduce object constants \( a_0, a_1, a_2, \ldots \). We define a \( \lambda \)-term, denoted \((z : \tau)\), by induction on finite types

1. if \( \tau = c_i \) for \( i \in \{1, \ldots, n\} \), then \( (z : \tau) = \lambda u_1. \lambda u_2. u_1(u_2 z)(u_2 a_i) \)
2. if \( \tau = \alpha_i \) for \( i \in \omega \) then \( (z : \tau) = \lambda u_1. \lambda u_2. u_1(u_2 z)(u_2 v_i) \)
3. if \( \tau = \tau_1 \rightarrow \tau_2 \) then
\[
\langle z : \tau \rangle = \lambda z_0. \lambda z_1. \lambda z_2. \lambda u. \ z_0(\langle z_1 : \tau_1 \rangle \langle z_2 : \tau_2 \rangle)(u(\langle z_1 \rangle))(u(\langle z_2 \rangle))
\]

It is clear from the induction above that: \( FV(\langle z : \tau \rangle) = \{ z \} \cup \{ v_i | \alpha_i \in FV(\tau) \} \). The following lemma, which is an extension of Lemma 14 in [14], explains the crucial property of the term \( \langle z : \tau \rangle \).

**Lemma 10** Let \( \tau \in \mathcal{T}_{fn} \) be an arbitrary finite type such that
\[
FV(\tau) \subseteq \{ \alpha_1, \ldots, \alpha_\ell \}.
\]

1. Let \( \tau', \rho_1, \ldots, \rho_\ell \) be arbitrary recursive types. The term \( \langle z : \tau \rangle \) is typable in \( \mathcal{S} \) in the environment \( A \):
\[
A = \{ z : \tau', v_1 : \rho_1, \ldots, v_\ell : \rho_\ell \}
\]
i.e., \( A \vdash_{\mu} \langle z : \tau \rangle : \tau'' \) for some \( \tau'' \in \mathcal{T}_\mu \), iff \( \tau' \approx \tau[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell] \).

2. Let \( \tau', \rho_1, \ldots, \rho_\ell \) be arbitrary positive recursive types. The term \( \langle z : \tau \rangle \) is typable in \( \mathcal{S}_+ \) in the environment \( A \):
\[
A = \{ z : \tau', v_1 : \rho_1, \ldots, v_\ell : \rho_\ell \}
\]
i.e., \( A \vdash_{\mu,+} \langle z : \tau \rangle : \tau'' \) for some \( \tau'' \in \mathcal{T}_{\mu,+} \), iff \( \tau' \approx \tau[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell] \).

**Proof:** We give the proof of Part 1 of the lemma and leave Part 2 for the reader since the proofs are very similar. The proof is by induction on \( \tau \). For the basis step, \( \tau = \alpha_i \) or \( \tau = c_i \) where \( i \in \{ 1, \ldots, \ell \} \). It is easily checked that \( \langle z : \tau \rangle \) is typable in \( A \) iff \( \tau' \approx \tau[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell] \), i.e., iff \( \tau' \approx \rho_i \).

For the induction step, assume that \( \langle z_1 : \tau_1 \rangle \) and \( \langle z_2 : \tau_2 \rangle \) are typable in \( \mathcal{S} \) in the environment
\[
A = \{ z_1 : \tau_1', z_2 : \tau_2', v_1 : \rho_1, \ldots, v_\ell : \rho_\ell \}
\]
iff \( \tau'_j \approx \tau_j[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell] \) for \( j = 1, 2 \). It is now readily checked that if \( \tau = \tau_1 \rightarrow \tau_2 \) then the term \( \langle z : \tau \rangle \) is typable in \( \mathcal{S} \) in the environment
\[
B = \{ z : \tau', v_1 : \rho_1, \ldots, v_\ell : \rho_\ell \}
\]
iff \( \tau' \approx \tau'_1 \rightarrow \tau'_2 \). Hence, \( \langle z : \tau \rangle \) is typable in \( B \) iff \( \tau' \approx \tau[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell] \), by the induction hypothesis. ■
Lemma 11 Consider an instance $\Gamma$ of semi-unification of the form

$$\Gamma = \{(\sigma_1, \tau_1), \ldots, (\sigma_n, \tau_n)\}$$

which mentions only type variables $\alpha_1, \ldots, \alpha_\ell$. Define the term $M$ as:

\[
M \equiv \text{fix } x. \lambda v_1 \ldots \lambda v_\ell. \lambda z_1 \ldots \lambda z_n. \ N, \text{ where } \\
N \equiv z_0 \langle z_1 : \sigma_1 \rangle \ldots \langle z_n : \sigma_n \rangle E_1 \ldots E_n, \text{ where } \\
E_i \equiv \lambda y_0. \lambda w_1 \ldots \lambda w_\ell. \lambda y_1 \ldots \lambda y_n. \ y_0(\langle x w_1 \ldots w_\ell y_1 \ldots y_n \rangle \langle y_i : \tau_i \rangle)
\]

for $i = 1, \ldots, n$.

1. $M$ is typable in $S$ iff $\Gamma$ has a regular solution.

2. $M$ is typable in $S_+$ iff $\Gamma$ has a positive regular solution.

Proof: Here we just show the proof of Part 1 of the lemma. The proof is just a reproduction of the proof of Lemma 13 in [14] with the necessary modifications. For the left to right implication, suppose that $M$ is typable. This means that $N$ is typable in an environment $A$ assigning types to $x, v_1, \ldots, v_\ell, z_0, z_1, \ldots, z_n$. Except for the type of $x$ (which is in $T^\nu$), these are all in $T^\mu$. Assume that the types assigned to $v_1, \ldots, v_\ell$ are $\rho_1, \ldots, \rho_\ell$, respectively. Because $\langle z_i : \sigma_i \rangle$ is typable in $A$, for $i = 1, \ldots, n$, the environment $A$ must contain the type assumption $z_i : \sigma_i$ where

$$\sigma'_i \approx \sigma_i[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell]$$

by Lemma 10. Hence, the type $\xi$ assigned to $\lambda v_1 \ldots \lambda v_\ell. \lambda z_1 \ldots \lambda z_n. \ N$ is of the form:

$$\xi = \rho_1 \rightarrow \ldots \rightarrow \rho_\ell \rightarrow \sigma'_1 \rightarrow \ldots \rightarrow \sigma'_n \rightarrow \varphi$$

where $\varphi$ depends on the type of $z_0$. Moreover, for each $i = 1, \ldots, n$, the term:

$$y_0(\langle x w_1 \ldots w_\ell y_1 \ldots y_n \rangle \langle y_i : \tau_i \rangle)$$

is also typable, in an appropriately extended environment. It follows that the type $\xi_i$ assigned to the $i$-th occurrence of $x$ is of the form:

$$\xi_i = \pi_1^i \rightarrow \ldots \rightarrow \pi_\ell^i \rightarrow \theta_1^i \rightarrow \ldots \rightarrow \theta_{i-1}^i \rightarrow \tau_i^i \rightarrow \theta_{i+1}^i \rightarrow \ldots \rightarrow \theta_n^i \rightarrow \psi^i$$
for some regular types $\theta_j^i$, with $j \neq i$, $\pi_j^i$ and $\psi^i$, and where
\[
\tau_i' \approx \tau_i[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell]
\]
by Lemma 10. Each $\xi_i$ is an instance of $\xi$ — more precisely, there is a substitution $S_i : T_\mu \rightarrow T_\mu$ such that $S_i(\xi) \approx \xi_i$; and, in particular, $S_i(\sigma_i^i) \approx \tau_i'$ for $i = 1, \ldots, n$. Hence, the substitution:
\[
[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell]
\]
is a regular solution of the instance $\Gamma$.

For the converse, suppose that $\Gamma$ has a solution, i.e., there are regular types $\rho_1, \ldots, \rho_\ell$ and substitutions $S_1, \ldots, S_n : T_\mu \rightarrow T_\mu$ such that:
\[
S_i(\sigma_i^i) \approx \tau_i' \quad \text{for} \quad i = 1, \ldots, n,
\]
where
\[
\sigma_i^i \approx \sigma_i[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell]
\] and
\[
\tau_i' \approx \tau_i[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell].
\]
We shall show that $M$ is typable in $\mathcal{S}$. Let $A = \{ (v_i : \rho_i) | i = 1, \ldots, \ell \}$ and define the environment:
\[
A_i = A \cup \{(w_j : \pi_j) | j = 1, \ldots, \ell \} \cup \{(y_j : \tau_j') \} \cup \{(y_j : \theta_j) | j \neq i \}
\]
for $i = 1, \ldots, n$, where $\pi_j$ and $\theta_j$ are arbitrary regular types. By Lemma 10, it must be the case that $A_i \vdash_{\mu} \langle y_i : \tau_i \rangle : \psi$ for some open type $\psi$. It then follows that:
\[
A_i \cup \{(x : \xi_i), (y_0 : \alpha \rightarrow \psi \rightarrow \beta) \} \vdash_{\mu} y_0(xw_1 \ldots w_\ell y_1 \ldots y_n)\langle y_i : \tau_i \rangle : \beta
\]
where $\alpha$ and $\beta$ are new type variables and:
\[
\xi_i = \pi_1 \rightarrow \ldots \rightarrow \pi_\ell \rightarrow \theta_1 \rightarrow \ldots \rightarrow \theta_{i-1} \rightarrow \tau_i' \rightarrow \theta_{i+1} \rightarrow \ldots \rightarrow \theta_n \rightarrow \alpha
\]
Hence $E_i$ is typable in $A \cup \{ x : \xi_i \}$. Let $B = A \cup \{ (z_i : \sigma_i^i) | i = 1, \ldots, n \}$. By Lemma 10, the term $\langle z_i : \sigma_i \rangle$ is typable in $B$. Define now $C = B \cup \{ x : \nu \xi \}$ where
\[
\nu \xi = \nu. \rho_1 \rightarrow \ldots \rightarrow \rho_\ell \rightarrow \sigma_1' \rightarrow \ldots \rightarrow \sigma_n' \rightarrow \alpha
\]
where $\bar{\forall}$ stands for “quantify all variables in $\xi$ except for $\alpha$.” Applying the substitution $S_i$ to $\xi$, we obtain the type:

$$S_i(\rho_1) \rightarrow \ldots \rightarrow S_i(\rho_i) \rightarrow S_i(\sigma'_1) \rightarrow \ldots \rightarrow S_i(\sigma'_{\alpha_i}) \rightarrow \alpha$$

Now take $\pi_j = \rho_j S_i$ and, for $j \neq i$, $\theta_j = \sigma'_j S_i$ in the environment $A_i$ above and we see that $C \vdash \mu x : \xi_i$. Hence, for an appropriate $\varphi$,

$$C \cup \{ z_0 : \varphi \} \vdash \mu N : \alpha$$

because every $\langle z_i : \sigma_i \rangle$ is typable in $C$, and so is every $E_i$, for $i = 1, \ldots, n$. After repeated abstractions:

$$\{ z_0 : \varphi, x : \bar{\forall} \xi \} \vdash \mu \lambda v_1 \ldots \lambda v_{\ell_n} \lambda z_1 \ldots \lambda z_n. N : \rho_1 \rightarrow \ldots \rightarrow \rho_\ell \rightarrow \sigma'_1 \rightarrow \ldots \rightarrow \sigma'_{\alpha_i} \rightarrow \alpha$$

and, finally, by application of the GEN rule repeatedly, followed by the FIX rule once:

$$\{ z_0 : \varphi \} \vdash \mu M : \bar{\forall} \xi$$

which proves that $M$ is typable in $S$.

\section{Decidability Results}

Regular semi-unification on arbitrary trees is undecidable. The proof of this result is in [4]. This result is further restricted to semi-unification on binary trees in [7] which leads to the following:

\textbf{Theorem 12} Type Reconstruction in system $S$ is undecidable.

\textbf{Proof:} The proof is directly obtained by the undecidability of regular semi-unification [4] and the equivalence of regular semi-unification to regular semi-unification on binary trees [7].

We have to leave open the decidability of Type Reconstruction in system $S_+$ and the decidability of PRSUP.

\section*{References}


