Type Reconstruction in the Presence of Polymorphic Recursion and Recursive Types

Said Jahama *
Boston University
(jahama@cs.bu.edu)

A.J. Kfoury †
Boston University
(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,

*Partly supported by NSF grant CCR-9113196. Address: Department of Computer Science, Boston University, 111 Cummington St., Boston, MA 02215, USA.

†Partly supported by NSF grant CCR-9113196. Address: Department of Computer Science, Boston University, 111 Cummington St., Boston, MA 02215, USA.

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].[1]

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+**pos-μ, 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+**pos-μ (Section 3). We call the two system $\mathcal{S}$ and $\mathcal{S}_+$ 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

[1]This 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 λ-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+**μ and **ML+fix+**pos-μ, respectively (Sections 5 and 6).

Having established these equivalences, we conclude that the type-reconstruction problem for **ML+fix+**μ is undecidable and leave the problem open for **ML+fix+**pos-μ (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 $\mathcal{T}_\mu$ is defined as follows:*

1. $X \cup C \subseteq \mathcal{T}_\mu$.
2. *If* $\sigma, \tau \in \mathcal{T}_\mu$ *then* $\sigma \to \tau \in \mathcal{T}_\mu$.
3. *If* $\alpha \in X$, $\sigma \in \mathcal{T}_\mu$ *then* $\mu\alpha.\sigma \in \mathcal{T}_\mu$.

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 $\alpha_1, \cdots, \alpha_n \in X$, $n \geq 0$, and $\tau \in \mathcal{T}_\mu$. Let $\mathcal{T}_\mu^\forall$ be the set of all universal recursive types. The universal quantifier "$\forall$" and the operator μ bind type variables. We identify α-convertible types (types identical up to renaming of bound variables). A *substitution* is a function $S : X \to \mathcal{T}_\mu$. The notation $\sigma[\alpha := \tau]$ stands for the result of substituting in σ all free occurrences of α by τ (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 α is positive in a type σ iff every free occurrence of α is on the left hand side of an even number of $\to$'s. The set of *positive recursive types* $\mathcal{T}_{\mu,+}$ is defined as follows:

Definition 2

1. $X \cup C \subseteq \mathcal{T}_{\mu,+}$.

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

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

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

A type σ is *finite* if σ does not contain an occurrence of the μ operator. Let $\mathcal{T}_{fin}$ be the set of all finite types. Notice that $\mathcal{T}_{fin} \subseteq \mathcal{T}_{\mu,+} \subseteq \mathcal{T}_{\mu}$. 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 σ and all its descendants in σ. A (possibly infinite) type σ is *regular* if the set of its subtypes is finite. Let $\mathcal{T}_{reg}$ be the set of all regular types.

For a type σ of the form $\mu\alpha.\tau$ the *unfolding* of σ for one step results in the type $\tau[\alpha := \mu\alpha.\tau]$. Every recursive type σ represents an underlying regular type obtained by unfolding σ infinitely many times. More formally there is a map $(\ )^* : \mathcal{T}_{\mu} \to \mathcal{T}_{reg}$. We refer the reader to [3] for an exact definition of $(\ )^*$. It is also true that every type in $\mathcal{T}_{reg}$ 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 $\mathcal{T}_{reg}$ 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* $\tau \sim \tau' \Rightarrow \sigma \to \tau \sim \sigma' \to \tau'$.

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:

$$\mu\alpha.\alpha \to \alpha \approx \mu\alpha.((\alpha \to \alpha) \to \alpha)$$

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

3 Systems $\mathcal{S}$ and $\mathcal{S}_+$

In this thesis we consider a simple language consisting of λ-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 λ-terms considered here are defined by the grammar:

$$M ::= x \mid a \mid (M\ N) \mid (\lambda x\ M) \mid (\mathbf{fix}\ x\ M)$$

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

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

The type inference systems $\mathcal{S}$ and $\mathcal{S}_+$ 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 σ 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 τ a type. In such an assertion, the σ's (mentioned in A) are called the *environment types*, and τ the *assigned* or *derived* type. Derivability in $\mathcal{S}$ and $\mathcal{S}_+$ will be denoted be the symbols $\vdash_\mu$ and $\vdash_{\mu,+}$, respectively.

VAR	$A \vdash x : \sigma$	$A(x) = \sigma, \quad \sigma \in \mathcal{T}_\mu^\forall$
CONST	$A \vdash a : \sigma$	σ is a type constant , $\sigma \in \mathcal{T}_\mu$
INST	$\dfrac{A \vdash M : \forall\alpha.\sigma}{A \vdash M : \sigma[\alpha := \tau]}$	$\tau \in \mathcal{T}_\mu, \quad \sigma \in \mathcal{T}_\mu^\forall$
GEN	$\dfrac{A \vdash M : \sigma}{A \vdash M : \forall\alpha.\sigma}$	$\alpha \notin \mathrm{FV}(A), \quad \sigma \in \mathcal{T}_\mu^\forall$
APP	$\dfrac{A \vdash M : \sigma \to \tau, \qquad A \vdash N : \sigma}{A \vdash (M\ N) : \tau}$	$\tau, \sigma \in \mathcal{T}_\mu$
ABS	$\dfrac{A[x : \sigma] \vdash M : \tau}{A \vdash (\lambda x\ M) : \sigma \to \tau}$	$\tau, \sigma \in \mathcal{T}_\mu$
FIX	$\dfrac{A[x : \sigma] \vdash M : \sigma}{A \vdash (\mathbf{fix}\ x\ M) : \sigma}$	$\sigma \in \mathcal{T}_\mu^\forall$
$\approx$	$\dfrac{A \vdash M : \sigma, \qquad \sigma \approx \tau}{A \vdash M : \tau}$	$\tau, \sigma \in \mathcal{T}_\mu$

Figure 1. System $\mathcal{S}$: all environment types and derived types in $\mathcal{T}_\mu^\forall$.

VAR	$A \vdash x : \sigma$	$A(x) = \sigma, \quad \sigma \in \mathcal{T}^{\forall}_{\mu,+}$
CONST	$A \vdash a : \sigma$	σ is a type constant , $\sigma \in \mathcal{T}_{\mu,+}$
INST	$\dfrac{A \vdash M : \forall\alpha.\sigma}{A \vdash M : \sigma[\alpha := \tau]}$	$\tau \in \mathcal{T}_{\mu,+}, \quad \sigma \in \mathcal{T}^{\forall}_{\mu,+}$
GEN	$\dfrac{A \vdash M : \sigma}{A \vdash M : \forall\alpha.\sigma}$	$\alpha \notin \mathrm{FV}(A), \quad \sigma \in \mathcal{T}^{\forall}_{\mu,+}$
APP	$\dfrac{A \vdash M : \sigma \to \tau, \qquad A \vdash N : \sigma}{A \vdash (M\ N) : \tau}$	$\tau, \sigma \in \mathcal{T}_{\mu,+}$
ABS	$\dfrac{A[x : \sigma] \vdash M : \tau}{A \vdash (\lambda x\ M) : \sigma \to \tau}$	$\tau, \sigma \in \mathcal{T}_{\mu,+}$
FIX	$\dfrac{A[x : \sigma] \vdash M : \sigma}{A \vdash (\mathbf{fix}\ x\ M) : \sigma}$	$\sigma \in \mathcal{T}^{\forall}_{\mu,+}$
$\sim$	$\dfrac{A \vdash M : \sigma, \qquad \sigma \sim \tau}{A \vdash M : \tau}$	$\tau, \sigma \in \mathcal{T}_{\mu,+}$

Figure 2. System $\mathcal{S}_+$: all environment types and derived types in $\mathcal{T}^{\forall}_{\mu,+}$.

3.1 Syntax-oriented rules for $\mathcal{S}$ and $\mathcal{S}_+$

Both system $\mathcal{S}$ and $\mathcal{S}_+$ 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 $\mathcal{S}$ and $\mathcal{S}_+$. 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 $\sigma, \tau \in \mathcal{T}_\mu$ and $\vec{\alpha} = \alpha_1 \cdots \alpha_n$ for some $n \geq 0$. We write $\forall\vec{\alpha}.\sigma \preceq \tau$ to mean that τ is an *instantiation* of $\forall\vec{\alpha}.\sigma$

$$\tau \approx \sigma[\alpha_1 := \tau_1, \ldots, \alpha_n := \tau_n], \text{ for some } \tau_1, \ldots, \tau_n \in \mathcal{T}_\mu.$$

Similarly, for $\sigma, \tau \in \mathcal{T}_{\mu,+}$ and $\vec{\alpha} = \alpha_1 \cdots \alpha_n$. $\forall\vec{\alpha}.\sigma \preceq_+ \tau$ iff

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

Instantiation corresponds to a sequence of applications of rule INST and rule $\approx$ (rule $\sim$ in $\mathcal{S}'$) , 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 $\mathcal{S}'$ and $\mathcal{S}'_+$ of $\mathcal{S}$ and $\mathcal{S}_+$ 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 $\mathcal{S}$ and ($\sim$) in $\mathcal{S}_+$. Derivability in $\mathcal{S}'$ and $\mathcal{S}'_+$ will be denoted by the symbols $\vdash'_\mu$ 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.

VAR	$A \vdash x : \tau$	$A(x) = \sigma, \quad \sigma \in \mathcal{T}_\mu^\forall, \quad \tau \in \mathcal{T}_\mu, \quad \sigma \preceq \tau$
CONST	$A \vdash a : \sigma$	σ is a type constant $, \sigma \in \mathcal{T}_\mu$
APP	$\dfrac{A \vdash M : \sigma \rightarrow \tau, \qquad A \vdash N : \sigma}{A \vdash (M\ N) : \tau}$	$\tau, \sigma \in \mathcal{T}_\mu$
ABS	$\dfrac{A[x : \sigma] \vdash M : \tau}{A \vdash (\lambda x\ M) : \sigma \rightarrow \tau}$	$\tau, \sigma \in \mathcal{T}_\mu$
FIX	$\dfrac{A[x : \forall\vec{\alpha}.\sigma] \vdash M : \sigma}{A \vdash (\mathbf{fix}\ x\ M) : \tau}$	$\sigma, \tau \in \mathcal{T}_\mu, \quad \forall\vec{\alpha}.\sigma \preceq \tau, \quad \vec{\alpha} \notin \mathrm{FV}(A)$
$\approx$	$\dfrac{A \vdash M : \sigma, \qquad \sigma \approx \tau}{A \vdash M : \tau}$	$\tau, \sigma \in \mathcal{T}_\mu$

Figure 3. System $\mathcal{S}'$: all environment types in $\mathcal{T}_\mu^\forall$. All derived types in $\mathcal{T}_\mu$.

VAR	$A \vdash x : \tau$	$A(x) = \sigma, \quad \sigma \in \mathcal{T}^{\forall}_{\mu,+}, \quad \tau \in \mathcal{T}_{\mu,+}, \quad \sigma \preceq_{+} \tau$
CONST	$A \vdash a : \sigma$	σ is a type constant , $\sigma \in \mathcal{T}_{\mu,+}$
APP	$\dfrac{A \vdash M : \sigma \to \tau, \qquad A \vdash N : \sigma}{A \vdash (M\ N) : \tau}$	$\tau, \sigma \in \mathcal{T}_{\mu,+}$
ABS	$\dfrac{A[x : \sigma] \vdash M : \tau}{A \vdash (\lambda x\ M) : \sigma \to \tau}$	$\tau, \sigma \in \mathcal{T}_{\mu,+}$
FIX	$\dfrac{A[x : \forall\vec{\alpha}.\sigma] \vdash M : \sigma}{A \vdash (\mathbf{fix}\ x\ M) : \tau}$	$\sigma, \tau \in \mathcal{T}_{\mu,+}, \quad \forall\vec{\alpha}.\sigma \preceq_{+} \tau, \quad \vec{\alpha} \notin \mathrm{FV}(A)$
$\sim$	$\dfrac{A \vdash M : \sigma, \qquad \sigma \sim \tau}{A \vdash M : \tau}$	$\tau, \sigma \in \mathcal{T}_{\mu,+}$

Figure 4. System $\mathcal{S}'_{+}$: all environment types in $\mathcal{T}^{\forall}_{\mu,+}$. 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 $\mathcal{S}$ and $\mathcal{S}_+$. The cases where we have a single derivation are rules CONST and VAR. For the VAR rule, assume that $A(x) = \forall\vec{\alpha}.\sigma \in \mathcal{T}_\mu^\forall$. Applying the VAR rule in system $\mathcal{S}$ we have:

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

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

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

$$\frac{A[x : \forall\vec{\alpha}.\sigma] \vdash M : \forall\vec{\alpha}.\sigma}{A \vdash (\mathbf{fix}\ x\ M) : \forall\vec{\alpha}.\sigma}$$

By applying the FIX rule in $\mathcal{S}'$ 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 $\mathcal{S}'$ to get:

$$\frac{A[x : \forall\vec{\alpha}.\sigma] \vdash M : \sigma}{A \vdash (\mathbf{fix}\ x\ M) : \sigma}$$

Again, a similar argument is used for the FIX rule in $\mathcal{S}_+$. The inductive proof for the other rules in $\mathcal{S}$ and $\mathcal{S}_+$ is straightforward.

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

1. If $A \vdash'_\mu M : \sigma$ then $A \vdash_\mu M : \sigma$.
2. If $A \vdash'_{\mu,+} M : \sigma$ then $A \vdash_{\mu,+} M : \sigma$.

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

$$A \vdash \sigma$$

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

$$A \vdash \sigma.$$

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

$$\frac{A[x : \forall\vec{\alpha}.\tau] \vdash M : \tau}{A \vdash (\mathbf{fix}\ x\ M) : \sigma}$$

where $\forall\vec{\alpha}.\tau \preceq \sigma$. We get the same derivation in $\mathcal{S}$, by applying the GEN rule as needed to get

$$A[x : \forall\vec{\alpha}.\tau] \vdash M : \forall\vec{\alpha}.\tau$$

and then we apply the FIX rule in $\mathcal{S}$ to obtain

$$A \vdash (\mathbf{fix}\ x\ M) : \forall\vec{\alpha}.\tau$$

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

$$A \vdash (\mathbf{fix}\ x\ M) : \sigma$$

A similar argument applies for $\mathcal{S}'_+$. ∎

4 Positive Regular Semi-Unification

As a result of the equivalence of the sets $\mathcal{T}_{reg}$ and $\mathcal{T}_\mu$, we can look at a regular substitution as a substitution from $S : X \to \mathcal{T}_\mu$. In this section, we redefine regular semi-unification and define positive regular semi-unification. A regular (resp. positive regular) substitution S is a function $S : X \to \mathcal{T}_\mu$ (resp., $S : X \to \mathcal{T}_{\mu,+}$) . Every regular (resp. positive regular) substitution S can be extended in a natural way to a function $S : \mathcal{T}_\mu \to \mathcal{T}_\mu$ (resp., $S : \mathcal{T}_{\mu,+} \to \mathcal{T}_{\mu,+}$) [3].

An *instance* Γ of semi-unification is a finite set of inequalities:

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

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

$$S_1(S(\sigma_1)) \approx S(\tau_1),\ \ldots\ ,\ S_n(S(\sigma_n)) \approx S(\tau_n)$$

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

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

$$S_1(S(\sigma_1)) \sim S(\tau_1),\ \ldots\ ,\ S_n(S(\sigma_n)) \sim S(\tau_n)$$

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

5 From $\mathcal{S}$ to RSUP and from $\mathcal{S}_+$ to PRSUP

Given a term M we construct an instance Γ_M of semi-unification such that :

1. M is typable in S iff Γ_M has a regular solution.

2. M is typable in S' iff Γ_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

$$\Delta_M = \{\sigma_1 \doteq \tau_1, \ldots, \sigma_p \doteq \tau_p\}$$

where $\sigma_i, \tau_i \in \mathcal{T}_{fin}, i \in \{1, \ldots, p\}$. 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 λ-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 $(\mathbf{fix}\ v.M_i)$ 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 Δ_k for $k = 1, \ldots, n$:

Simultaneously with Δ_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, $\mathbf{fix}\ x_i$, 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 $\Delta_k = \Delta_i \cup \Delta_j \cup \{t_i \doteq t_j \to \delta\}$ and $t_k = \delta$, where δ is a fresh auxiliary variable.

5. If $M_k = (\lambda y_i.M_j)$ then set $\Delta_k = \Delta_j$ and $t_k = \gamma_i \to t_j$.

6. If $M_k = (\mathbf{fix}\ x_i.M_j)$ then set $\Delta_k = \Delta_j \cup \{\beta_i \doteq t_j\}$ and $t_k = \beta_i^{(\ell)}$, where $\ell \geq 0$ is the number of bound occurrences of x_i in M_j ($\beta_i^{(0)}, \ldots, \beta_i^{(\ell-1)}$ are already introduced in $\Delta_1, \ldots, \Delta_{k-1}$, corresponding to the bound occurrences of x_i).

Instead of Δ_n and t_n, we also write Δ_M and t_M.

The only difference between Δ_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 Δ_M as follows:

$$\begin{aligned} V_0 &= \{\beta_i^{(0)}, \ldots, \beta_i^{(\ell-1)} \mid \text{there are } \ell \geq 0 \text{ free or bound occurrences of } x_i \text{ in } M\} \\ &\quad \cup \{\gamma_i \mid y_i \text{ occurs in } M\} \\ &\quad \cup \{\delta_i \mid \delta_i \text{ occurs in } \Delta_M\} \\ V_1 &= \{\beta_i \mid x_i \text{ occurs in } M\} \end{aligned}$$

For $\sigma \in \mathcal{T}_\mu \cup \mathcal{T}_{\mu,+}$ and $\vec{\alpha}$ a finite sequence (possibly empty) of type variables, we define $\mathit{body}(\forall\vec{\alpha}.\sigma) = \sigma$.

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

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 ($\mathbf{fix}$) binding of x is in the scope of the λ-binding of y.

We now define what it means for S and S_+ to be a regular solution (positive regular solution, respectively) for Δ_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 Δ_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 Δ_M, $S(\beta_i) \preceq S(\beta_i^{(j)})$.

3. For all β_i occurring in Δ_M, the bound variables of $S(\beta_i)$ are precisely the set:
$\mathrm{FV}(\bar{S}(\beta_i)) \; - \; \bigcup\{\mathrm{FV}(\bar{S}(\gamma_j)) \mid x_i \leq_M y_j\}$

Definition 7 S_+ is a positive regular solution for Δ_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 Δ_M, $S_+(\beta_i) \preceq_+ S_+(\beta_i^{(j)})$.

3. For all β_i occurring in Δ_M, the bound variables of $S(\beta_i)$ are precisely the set:
$\mathrm{FV}(\bar{S}_+(\beta_i)) \; - \; \bigcup\{\mathrm{FV}(\bar{S}_+(\gamma_j)) \mid x_i \leq_M y_j\}$

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 Δ_M has a regular solution S such that* $\mathrm{body}(S(t_M)) \approx \tau$ *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 Δ_M has a positive regular solution S_+ such that* $\mathrm{body}(S_+(t_M)) \sim \tau$ *and $S_+(\beta_i) \sim A(x_i)$ for every i.*

3. *If S is a regular solution of Δ_M, then $A \vdash'_\mu M : \tau$ for some environment A and $\tau \in \mathcal{T}_\mu$ such that $\tau \approx$* $\mathrm{body}(S(t_M))$ *and $A(x_i) \approx S(\beta_i)$ for every i.*

4. *If S_+ is a positive regular solution of Δ_M, then $A \vdash'_{\mu,+} M : \tau$ for some environment A and $\tau \in \mathcal{T}_{\mu,+}$ such that $\tau \sim$* $\mathrm{body}(S(t_M))$ *and $A(x_i) \sim S(\beta_i)$ for every i.*

Proof: First we observe the following facts about derivations in systems $\mathcal{S}'$ and $\mathcal{S}'_+$:

- If $A \vdash N : \sigma$ is an assertion in a derivation and v is an x- or y-variable, then $A(v)$ is defined iff either v is free in M or v is bound in M and N is in the scope of the binding of v.
- If $A[x_i : \forall\vec{\alpha}.\sigma] \vdash N : \tau$ is the assertion immediately preceding an application of FIX that discharges the type assumption $(x_i : \forall\vec{\alpha}.\sigma)$, we can assume that the bound type variables $\vec{\alpha}$ are precisely:

$$\vec{\alpha} = \ \mathrm{FV}(\sigma) \ - \ \bigcup\{\mathrm{FV}(A(y_j)) \mid A(y_j) \text{ defined.}\}$$

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 $A \vdash'_\mu M_k : \tau$ for some environment A and a type $\tau \in \mathcal{T}_\mu$, then Δ_k has a solution S such that $\mathit{body}(S(t_k)) \approx \tau$, $S(\beta_i) \approx A(x_i)$ 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 Δ_1 such that $\mathit{body}(S(t_1)) \approx \tau$, $S(\beta_i) \approx A(x_i)$ 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 $M_k = (\mathbf{fix}\ x_i.M_j)$ and $M_k \vdash'_\mu \sigma$ which implies that $A(x_i) = \forall\vec{\alpha}.\sigma$ and $M_j \vdash \sigma$, by the FIX rule of system $\mathcal{S}$. By induction hypothesis, there is a solution S for Δ_j such that $\mathit{body}(S(t_j)) \approx \sigma$ 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 Δ_k, we can easily check that S is a solution for Δ_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 Δ_k, then there is an environment A and type $\tau \in \mathcal{T}_\mu$ such that: $A \vdash'_\mu M_k : \tau$, $\tau \approx$ *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 $M_k = (\mathbf{fix}\ x_i.M_j)$. Observe that if S is a solution for Δ_k then it is a solution for Δ_j. Assume that $S(t_j) \approx \forall\vec{\alpha}.\sigma$ and $S(t_k) \approx \tau$. From Step 6 of the construction, we can conclude that $S(t_j) \approx S(\beta_i)$ and $\forall\vec{\alpha}.\sigma \preceq \tau$. By induction hypothesis, there is an environment A such that $A \vdash'_\mu M_j : \sigma$ and $A(x_i) \approx \forall\vec{\alpha}.\sigma$. Let $B = A - [\forall\vec{\alpha}.\sigma]$. Using the FIX rule of system $\mathcal{S}'$ we can conclude that $B \vdash_\mu M_k : \tau$. ∎

We now define an instance Γ_M of semi-unification such that Γ_M has a solution in the sense of semi-unification iff Δ_M has a solution.

Definition of Γ_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 δ_q occurs in Δ_M.

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

$$\begin{aligned} T &= (\delta_{q+1} \rightarrow \delta_{q+1}) \rightarrow \cdots \rightarrow (\delta_{q+p} \rightarrow \delta_{q+p}) \\ U &= (\sigma_1 \rightarrow \tau_1) \rightarrow \cdots \rightarrow (\sigma_p \rightarrow \tau_p) \end{aligned}$$

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

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

$$\begin{aligned} T_{ij} &= \beta_i \rightarrow \gamma_{k_1} \rightarrow \cdots \rightarrow \gamma_{k_\ell} \\ U_{ij} &= \beta_i^{(j)} \rightarrow \gamma_{k_1} \rightarrow \cdots \rightarrow \gamma_{k_\ell} \end{aligned}$$

   where $\{\gamma_{k_1}, \ldots, \gamma_{k_\ell}\} = \{\gamma_m \mid x_i \leq_M y_m\}$.

3. Γ_M contains no other inequality.

Lemma 9 *If M is a term, Then:*

1. *For any $S : V \to \mathcal{T}^{\forall}_{\mu}$, $\bar{S}$ is a regular solution of Γ_M (in the sense of semi-unification) iff S is a regular solution of Δ_M (in the sense of definition 6).*

2. *For any $S_+ : V \to \mathcal{T}^{\forall}_{\mu,+}$, $\bar{S}_+$ is a positive regular solution of Γ_M (in the sense of semi-unification) iff S_+ is a positive regular solution of Δ_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 Γ_M. $\bar{S}$ is a regular (resp. positive regular) solution of Δ_M in the sense of definition 6 (resp. definition 7) iff $\bar{S}$ 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 Γ_M. It is readily checked that $S(\beta_i) \preceq S(\beta_i^{(j)})$ (resp. $S(\beta_i) \preceq_+ S(\beta_i^{(j)})$) and the bound variables of $S(\beta_i)$ are:

$$\mathrm{FV}(\bar{S}(\beta_i)) \; - \; \bigcup\{\mathrm{FV}(\bar{S}(\gamma_j)) \mid x_i \leq_M y_j\}$$

iff S is a regular (resp. positive regular) solution of $\{(T_{ij}, U_{ij})\}$ in the sense of semi-unification. ∎

6 From RSUP to $\mathcal{S}$ and from PRSUP to $\mathcal{S}_+$

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 τ 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 λ-term, denoted $\langle z : \tau \rangle$, by induction on finite types

1. if $\tau = c_i$ for $i \in \{1, \ldots, n\}$, then
   $\langle z : \tau \rangle = \lambda u_1.\lambda u_2.\; u_1(u_2 z)(u_2 a_i)$

2. if $\tau = \alpha_i$ for $i \in \omega$ then
   $\langle z : \tau \rangle = \lambda u_1.\lambda u_2.\; u_1(u_2 z)(u_2 v_i)$

3. if $\tau = \tau_1 \to \tau_2$ then
   $\langle z : \tau \rangle = \lambda z_0.\lambda z_1.\lambda z_2.\lambda u.\;\; z_0 \langle z_1 : \tau_1 \rangle \langle z_2 : \tau_2 \rangle (u(zz_1))(uz_2)$

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 $\tau \in \mathcal{T}_{fin}$ 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 $\tau' \sim \tau[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell]$.

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 τ. 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 $\tau = \tau_1 \to \tau_2$ 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 $\tau' \approx \tau_1' \to \tau_2'$. 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* Γ *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:*

$$\begin{aligned} M &\equiv \mathbf{fix}\ x.\lambda v_1 \ldots \lambda v_\ell.\lambda z_1 \ldots \lambda z_n.\ N, \quad \textit{where} \\ N &\equiv z_0\langle z_1 : \sigma_1\rangle \ldots \langle z_n : \sigma_n\rangle E_1 \ldots E_n, \quad \textit{where} \\ E_i &\equiv \lambda y_0.\lambda w_1 \ldots \lambda w_\ell.\lambda y_1 \ldots \lambda y_n.\ y_0(x w_1 \ldots w_\ell y_1 \ldots y_n)\langle y_i : \tau_i\rangle \end{aligned}$$

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

1. M *is typable in* $\mathcal{S}$ *iff* Γ *has a regular solution.*

2. M *is typable in* $\mathcal{S}_+$ *iff* Γ *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 $\mathcal{T}_\mu^\forall$) , these are all in $\mathcal{T}_\mu$. 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 ξ assigned to $\lambda v_1 \ldots \lambda v_\ell.\lambda z_1 \ldots \lambda z_n.\ N$ is of the form:

$$\xi = \rho_1 \to \ldots \to \rho_\ell \to \sigma_1' \to \ldots \to \sigma_n' \to \varphi$$

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

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

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

$$\xi_i = \pi_1^i \to \ldots \to \pi_\ell^i \to \theta_1^i \to \ldots \to \theta_{i-1}^i \to \tau_i' \to \theta_{i+1}^i \to \ldots \to \theta_n^i \to \psi^i$$

for some regular types θ^i_j, with $j \neq i$, π^i_j and ψ^i, and where

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

by Lemma 10. Each ξ_i is an instance of ξ — more precisely, there is a substitution $S_i : \mathcal{T}_\mu \rightarrow \mathcal{T}_\mu$ such that $S_i(\xi) \approx \xi_i$ and, in particular, $S_i(\sigma'_i) \approx \tau'_i$ for $i = 1, \ldots, n$. Hence, the substitution:

$$[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell]$$

is a regular solution of the instance Γ.

For the converse, suppose that Γ has a solution, i.e, there are regular types $\rho_1, \ldots, \rho_\ell$ and substitutions $S_1, \ldots, S_n : \mathcal{T}_\mu \rightarrow \mathcal{T}_\mu$ such that:

$$S_i(\sigma'_i) \approx \tau'_i \text{ for } i = 1, \ldots, n, \text{ where}$$

$$\begin{aligned} \sigma'_i &\approx \sigma_i[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell] \quad \text{and} \\ \tau'_i &\approx \tau_i[\alpha_1 := \rho_1, \ldots, \alpha_\ell := \rho_\ell]. \end{aligned}$$

We shall show that M is typable in $\mathcal{S}$. Let $A = \{(v_i : \rho_i) | i = 1, \ldots, \ell\}$ and define the environment:

$$A_i = A \cup \{(w_j : \pi_j) | j = 1, \ldots, \ell\} \cup \{(y_i : \tau'_i)\} \cup \{(y_j : \theta_j) | j \neq i\}$$

for $i = 1, \ldots, n$, where π_j and θ_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 ψ. 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 α and β 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 $B = A \cup \{(z_i : \sigma'_i) | i = 1, \ldots, n\}$. By Lemma 10, the term $\langle z_i : \sigma_i \rangle$ is typable in B. Define now $C = B \cup \{x : \vec{\forall}.\xi\}$ where

$$\vec{\forall}.\xi = \vec{\forall}.\rho_1 \rightarrow \ldots \rightarrow \rho_\ell \rightarrow \sigma'_1 \rightarrow \ldots \rightarrow \sigma'_n \rightarrow \alpha$$

where $\vec{\forall}$ stands for "quantify all variables in ξ except for α." Applying the substitution S_i to ξ, we obtain the type:

$$S_i(\rho_1) \to \ldots \to S_i(\rho_\ell) \to S_i(\sigma'_1) \to \ldots \to S_i(\sigma'_n) \to \alpha$$

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 φ,

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

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:

$$\{z_0 : \varphi, x : \vec{\forall}.\xi\} \vdash_\mu \lambda v_1 \ldots \lambda v_\ell . \lambda z_1 \ldots \lambda z_n . N : \rho_1 \to \ldots \to \rho_\ell \to \sigma'_1 \to \ldots \to \sigma'_n \to \alpha$$

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

$$\{z_0 : \varphi\} \vdash_\mu M : \vec{\forall}.\xi$$

which proves that M is typable in $\mathcal{S}$. ■

7 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:

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

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 $\mathcal{S}_+$ and the decidability of PRSUP.

References

[1] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 93.

[2] Courcelle B. Fundemental properties of infinite trees. *Theoretical Computer Science*, 25:48–80, 1983.

[3] F. Cardone and M. Coppo. Type inference with recursive types: Syntax and semantics. *Information and Computation*, 92:48–80, 1991.

[4] J. Dörre and W. Rounds. On subsumption and semiunification in feature algebras. In *Proceedings of IEEE 5th Annual Symposium on Logic in Computer Science*, pages 300 – 310, 1990.

[5] P. Giannini and S. Ronchi Della Rocca. Characterization of typings in polymorphic type discipline. In *Proceedings of IEEE 3rd Annual Symposium on Logic in Computer Science*, pages 61 – 71, 1988.

[6] F. Henglein. Type inference with polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):254–290, 93.

[7] S. Jahama and A.J. Kfoury. A general theory of semi-uniffication. Technical Report 93-018, Boston University, Department of Computer Science, December 1993.

[8] P. Kanellakis and J.C. Mitchell. Polymorphic unification and ml typing. In *Proceedings of* 16*th ACM Symposium on Principles of Programming Languages*, pages 105 – 115, 1989.

[9] D. Kapur, D. Musser, P. Narendran, and J. Stillman. Semi-unification. In Nori and Kumar, editors, *Proceedings of 8th Conference on Foundations of Software Technology and Theoretical Computer Science, LNCS 338*, pages 435 – 454. Springer Verlag, 1988.

[10] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. The hierarchy of finitely typed functional programs. In *Proceedings of IEEE 2nd Annual Symposium on Logic in Computer Science*, pages 225 – 235, 1987.

[11] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. On the computational power of universally polymorphic recursion. In *Proceedings of IEEE 3rd Annual Symposium on Logic in Computer Science*, pages 72 – 81, 1988.

[12] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ml typability. In Arnold, editor, *15th Colloquium on Trees in Algebra and Programming, CAAP 90, LNCS 431*, pages 206 – 220. Springer Verlag, 1990.

[13] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. The undecidability of the semi-unification problem. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore*, pages 468 – 476, 1990.

[14] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. Type-reconstruction in the presence of polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):290–311, 93.

[15] H. Leiss. Semi-unification and type inference for polymorphic recursion. Research Report INF 2-ASE-5-89, Siemens, München, 1989.

[16] H.G. Mairson. Deciding ml typability is complete for deterministic exponential time. In *Proceedings of* 17*th ACM Symposium on Principles of Programming Languages*, pages 382 – 401, 1990.

[17] N.P. Mendler. Inductive types and type constraints in the second-order lambda calculus. *Annals of Pure and Applied Logic*, 51:159–172, 1991.

[18] J.C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76(2/3):211 – 249, 1988.

[19] A. Mycroft. Polymorphic type schemes and recursive definitions, lncs 167. In Paul and Robinet, editors, *International Symposium on Programming*, pages 217 – 228. Springer Verlag, 1984.

[20] P. Pudlák. On a unification problem related to kreisel's conjecture. *Commentationes Mathematicae Universitatis Carolinae, Prague, Czechoslovakia*, 29(3):551 – 556, 1988.