Type Inference for a Polynomial Lambda Calculus*

Marco Gaboardi and Simona Ronchi Della Rocca

Dipartimento di Informatica
Università degli Studi di Torino
Corso Svizzera 185, 10149 Torino, Italy
{gaboardi,ronchi}@di.unito.it

Abstract. We study the type inference problem for the Soft Type Assignment system (STA) for λ-calculus introduced in [1], which is correct and complete for polynomial time computations. In particular we design an algorithm which, given in input a λ-term, provides all the constraints that need to be satisfied in order to type it. For the propositional fragment of STA, the satisfiability of the constraints is decidable. We conjecture that, for the whole system, the type inference is undecidable, but our algorithm can be used for checking the typability of some particular terms.

1 Introduction

In [1], we have introduced a type assignment system for λ-calculus, named STA (Soft Type Assignment), inspired by the Soft Linear Logic of Lafont [2], which characterizes the polynomial time computations, in the sense that a well typed term can be reduced to normal form in a number of β-reduction steps which is polynomial in its size, and moreover all polynomial time functions can be represented by well typed terms, through an appropriate coding. In this paper we approach the problem of type inference in STA. In the simple types setting, type inference is decidable, and it corresponds to the property of having a principal typing, i.e., a typing for a term from which all (and only) the types derivable for the term itself can be built, through a substitution. STA has both modal and second order types, so the type inference is more difficult to be studied in this setting. We approach the problem in two steps, first for the propositional fragment and then for the full system. In both cases we need the notion of *type scheme*, which is an abstract representation of a set of types. Namely types can be obtained from type schemes through an operation of substitution. A notion of type scheme, for reasoning about type inference, was introduced first in [3] in the setting of intersection types, and it has been used, in different forms, for second order type inference [4], and for modal type inference [5]. We prove that, in propositional case, the type inference for STA is decidable. We introduce an algorithm which, given a term $\mathtt{M}$, generates a triple $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{H} \rangle$, where U is a

* Paper partially supported by MIUR-Cofin'07 CONCERTO Project.

S. Berardi, F. Damiani, and U. de'Liguoro (Eds.): TYPES 2008, LNCS 5497, pp. 136–152, 2009.

type scheme, Ψ is a context assigning type schemes to the free variables of M, and $\mathcal{H} = \langle \mathcal{P}, \mathcal{C} \rangle$ is a pair of constraint sets. The constraint set $\mathcal{P}$ is a unification set of type schemes while $\mathcal{C}$ is a set of (in)equalities between exponentials. Informally $\mathcal{P}$ represents the conditions on the terms functionality, while $\mathcal{C}$ represents the conditions on the modalities. A pair of constraint sets is *satisfied* if the unification in $\mathcal{P}$ succeeds and moreover there is a substitution replacing exponentials by natural numbers in $\mathcal{C}$, in such a way that the (in)equalities become true. The algorithm is correct and complete, in the sense that M can be typed only in case the sets of constraints can be satisfied, and moreover all the typings for M can be built from Ψ and U through substitutions satisfying them. Since the satisfiability of the constraints is decidable in polynomial time, the type inference is decidable in polynomial time too.

Then we extend our study to second order types. We define an algorithm showing all the conditions that must be satisfied in order to type a term in the system. Namely, when applied to a term M, the algorithm produces as output a type scheme, a type scheme context, and five sets of constraints $\mathcal{G}, \mathcal{F}, \mathcal{Q}, \mathcal{P}$ and $\mathcal{C}$, where, $\mathcal{P}, \mathcal{C}$ are as in the propositional case, $\mathcal{G}$ is a semi-unification set of type schemes, and $\mathcal{F}$ and $\mathcal{Q}$ represent the conditions on the quantified abstracted variables. Also in this case the algorithm is correct and complete, but we conjecture that the satisfiability of the second order constraints is undecidable. We think the proof of Wells of undecidability of typability in System F adapts also in this case [6]. In any case, the algorithm is quite useful for checking the typability in particular cases, and in fact we use it for building two terms, the first one typable in System F but untypable in STA, and the second one typable in STA and not typable in DLAL [7], which is an alternative polynomial type assignment inspired by Light Affine Logic [8].

The paper is organized as follows. In Section 2 we introduce the type assignment system STA, and we recall its properties. In Section 3 we present the type inference algorithm for the propositional fragment and we prove it correct and complete. Moreover in Section 4 we discuss its complexity. In Section 5 we extend the analysis to second order types. Finally Section 6 contains a short conclusion.

2 The System STA

In this section we introduce the type assignment system STA, and we show its properties. STA is presented in a version which is slightly different from the presentation given in [1]. The difference is only in the management of contexts, in [1] contexts were sets of type assignments, here instead they are multisets of type assignments. This version is clearly equivalent to the original one [9], preserving the complexity properties, but it makes easier the design of the type inference algorithm.

Definition 1

i) The set $\mathbf{T}$ *of* soft types *is defined as follows:*

$$A, B, C ::= \alpha \mid \sigma \multimap A \mid \forall \alpha.A \text{ (Linear Types)} \qquad \sigma, \tau ::= A \mid !\sigma$$

Table 1. STA in the multiset version

$$\frac{}{\mathtt{x} : A \vdash \mathtt{x} : A}\ (Ax) \qquad \frac{\mathfrak{A} \vdash \mathtt{M} : \sigma}{\mathfrak{A}, \mathtt{x} : A \vdash \mathtt{M} : \sigma}\ (w) \qquad \frac{\mathfrak{A}, (\mathtt{x} : \tau)^{(r)} \vdash \mathtt{M} : \sigma}{\mathfrak{A}, \mathtt{x} :! \tau, \vdash \mathtt{M} : \sigma}\ (m)$$

$$\frac{\mathfrak{A} \vdash \mathtt{M} : \sigma \multimap A \quad \mathfrak{B} \vdash \mathtt{N} : \sigma \quad \mathfrak{A} \approx \mathfrak{B}}{\mathfrak{A}, \mathfrak{B} \vdash \mathtt{MN} : A}\ (\multimap E) \qquad \frac{\mathfrak{A} \vdash \mathtt{M} : \forall \alpha . A}{\mathfrak{A} \vdash \mathtt{M} : A[B/\alpha]}\ (\forall E)$$

$$\frac{\mathfrak{A}, \mathtt{x} : \sigma \vdash \mathtt{M} : A \quad \mathtt{x} \notin \mathrm{dom}(\mathfrak{A})}{\mathfrak{A} \vdash \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap A}\ (\multimap I) \qquad \frac{\mathfrak{A} \vdash \mathtt{M} : A \quad \alpha \notin \mathrm{FTV}(\mathfrak{A})}{\mathfrak{A} \vdash \mathtt{M} : \forall \alpha . A}\ (\forall I) \qquad \frac{\mathfrak{A} \vdash \mathtt{M} : \sigma}{!\mathfrak{A} \vdash \mathtt{M} : !\sigma}\ (sp)$$

where α, β range over a countable set of type variables. $\equiv$ denotes the syntactical identity between types.

ii) A context $\mathfrak{A}$ is a finite multiset of type assignments of the shape $\mathtt{x} : \sigma$, such that if $\mathtt{x} : \sigma_1 \in \mathfrak{A}$ and $\mathtt{x} : \sigma_2 \in \mathfrak{A}$ then there exists $A \in \mathbf{T}$ and $n, m \in \mathbb{N}$ such that $\sigma_1 \equiv \underbrace{!...!}_{n} A$ and $\sigma_2 \equiv \underbrace{!...!}_{m} A$. Contexts are ranged over by $\mathfrak{A}$,$\mathfrak{B}$,$\mathfrak{C}$. When a context is a set we denote it by Γ, Δ.

iii) STA *proves statements of the shape $\mathfrak{A} \vdash \mathtt{M} : \sigma$ where $\mathfrak{A}$ is a context, $\mathtt{M}$ is a term of λ-calculus, and σ is a type. The rules of the system are given in Table 1. The term $\mathtt{M}$ is* typable *in* STA *if there is a context $\mathfrak{A}$ and a type σ such that $\mathfrak{A} \vdash \mathtt{M} : \sigma$.*

As usual $\multimap$ associates to the right and has precedence on $\forall$, while ! has precedence on everything else. $\mathrm{FTV}(\sigma)$ denotes the set of free type variables of the type σ. $B[A/\alpha]$ denotes the capture free substitution of all occurrences of the type variable α by the linear type A: note that this kind of substitution preserves the correct syntax of types. $\forall \boldsymbol{\alpha}.A$ shortens $\forall \alpha_1 ... \alpha_n.A$ for $n \geq 0$. Two contexts $\mathfrak{A}$ and $\mathfrak{B}$ are *coherent*, denoted $\mathfrak{A} \approx \mathfrak{B}$, if and only if their multiset union $\mathfrak{A}, \mathfrak{B}$ is a context. Let $\mathfrak{A} = \{\mathtt{x}_1 : \sigma_1, \ldots, \mathtt{x}_n : \sigma_n\}$ then $\mathrm{dom}(\mathfrak{A}) = \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$, $!\mathfrak{A} = \{\mathtt{x}_1 :! \sigma_1, \ldots, \mathtt{x}_n :! \sigma_n\}$ and $\mathrm{FTV}(\mathfrak{A}) = \{\alpha \in \mathrm{FTV}(\sigma) \mid \mathtt{x} : \sigma \in \mathfrak{A}\}$. $\Sigma \rhd \mathfrak{A} \vdash \mathtt{M} : \sigma$ denotes that there is a derivation Σ proving $\mathfrak{A} \vdash \mathtt{M} : \sigma$. $|\mathtt{M}|$ denotes the number of sybols in $\mathtt{M}$.

Hygiene condition. We assume that free and bound type variables have different names, and also type variables bounded by different quantifiers are named differently.

Theorem 2 (Complexity of STA **[1])**

i) (Soundness) *If $\Pi \rhd \Gamma \vdash \mathtt{M} : \sigma$, then $\mathtt{M}$ can be evaluated to normal form in a number of β-reduction steps $O(|\mathtt{M}|^{(\mathtt{d}(\Pi)+1)})$, where $\mathtt{d}(\Pi)$ is the maximum nesting of rules (sp) in Π.*

ii) (Completeness) *Every polynomial time function can be encoded by a term typable in* STA.

3 Type Inference for the Propositional Fragment

As we said in the introduction, if we restrict ourselves to consider just the propositional fragment, the type inference is decidable. In this section we will show the type inference algorithm, which is based on the notion of type scheme and a unification procedure for type schemes.

Table 2. Unification Algorithm

$$\frac{}{\mathtt{U}(a,a)=\langle\emptyset,\emptyset\rangle}\ (\mathtt{U}_1) \qquad \frac{a\notin \mathrm{FV}(U)}{\mathtt{U}(a,U)=\langle\{a=U\},\emptyset\rangle}\ (\mathtt{U}_2) \qquad \frac{\mathtt{U}(a,V)=\langle\mathcal{P},\mathcal{C}\rangle}{\mathtt{U}(a,!^pV)=\langle\mathcal{P},\mathcal{C}\cup\{p=0\}\rangle}\ (\mathtt{U}_3)$$

$$\frac{\mathtt{U}(\phi,\psi)=\langle\mathcal{P}_1,\mathcal{C}_1\rangle \quad \mathtt{U}(U,V)=\langle\mathcal{P}_2,\mathcal{C}_2\rangle}{\mathtt{U}(\phi\multimap U,\psi\multimap V)=\langle\mathcal{P}_1\cup\mathcal{P}_2,\mathcal{C}_1\cup\mathcal{C}_2\rangle}\ (\mathtt{U}_4) \qquad \frac{\mathtt{U}(\phi\multimap U,V)=\langle\mathcal{P},\mathcal{C}\rangle}{\mathtt{U}(\phi\multimap U,!^pV)=\langle\mathcal{P},\mathcal{C}\cup\{p=0\}\rangle}\ (\mathtt{U}_5)$$

$$\frac{\mathtt{U}(\psi,\phi)=\langle\mathcal{P},\mathcal{C}\rangle}{\mathtt{U}(\phi,\psi)=\langle\mathcal{P},\mathcal{C}\rangle}\ (\mathtt{U}_6) \qquad \frac{\mathtt{U}(U,V)=\langle\mathcal{P},\mathcal{C}\rangle}{\mathtt{U}(!^pU,!^qV)=\langle\mathcal{P},\mathcal{C}\cup\{p=q\}\rangle}\ (\mathtt{U}_7)$$

3.1 Type Schemes, Substitutions and Constraints

Definition 3. Linear type schemes *and* type schemes *are respectively defined by the grammars*

$$U,V,Z ::= a \mid \phi\multimap U \qquad\qquad \phi,\psi,\xi ::= U \mid !^pU$$

where the exponential p,q,r *belong to a countable set,* a,b,c,d *belong to a countable set of* linear scheme variables. $\mathcal{T}$ *denotes the set of type schemes.*

$\mathrm{FV}(\phi)$ is the set of all linear scheme variables and exponentials occurring in ϕ. Two type schemes ϕ,ψ are *disjoint* if $\mathrm{FV}(\phi)\cap\mathrm{FV}(\psi)=\emptyset$.

A *scheme substitution* s is a total function mapping linear scheme variables to linear types and exponentials to natural numbers. So a scheme substitution maps type schemes to types. The application of s to a type scheme is defined as

$$s(a)=A \text{ if } [a\mapsto A]\in s \qquad s(\phi\multimap U)=s(\phi)\multimap s(U)$$

$$s(!^pU)=\underbrace{!\ldots!}_{n}s(U) \text{ if } [p\mapsto n]\in s$$

In what follows, $s[a_1\mapsto\tau_1,\ldots,a_n\mapsto\tau_n]$ denotes the scheme substitution defined as s except on variables $a_1,\ldots,a_n$ to which it assigns $\tau_1,\ldots\tau_n$.

A type scheme can be seen as an abstract representation of the set of types that can be obtained from it through a scheme substitution. For example, a represents the set of all linear types, while $!^p(a\multimap b)$ represents the set of types $\{\underbrace{!\ldots!}_{n}(A\multimap B)\mid A,B \text{ are linear types and } n\geq 0\}$.

Two type schemes ϕ and ψ can be *unified* if there is a scheme substitution s such that $s(\phi)\equiv s(\psi)$.

A *type scheme context* is a multiset of *variable type scheme assignments* of the shape $\mathtt{x} : \phi$ where $\mathtt{x}$ is a variable and ϕ is a type scheme. Type scheme contexts are ranged over by Ψ, Φ. $\mathrm{dom}(\Psi)$ denotes the set of variables $\{\mathtt{x} \mid \mathtt{x} : \phi \in \Psi\}$. Multiset union of type scheme contexts is denoted by $\sqcup$. The expression $\Phi = \Phi' \odot \Psi$ denotes the fact that $\Phi = \Phi' \sqcup \Psi$ and $\mathrm{dom}(\Phi') \cap \mathrm{dom}(\Psi) = \emptyset$. Scheme substitutions are easily extended to type scheme contexts, *i.e.* if $\Psi = \mathtt{x}_1 : \phi_1, \ldots, \mathtt{x}_n : \phi_n$ then $s(\Psi) = \mathtt{x}_1 : s(\phi_1), \ldots, \mathtt{x}_n : s(\phi_n)$.

A *constraints sequence* $\mathcal{H}$ is a couple $\langle \mathcal{P}, \mathcal{C} \rangle$ of constraints sets. The set of *scheme variable constraints* $\mathcal{P}$ is a set of constraints of the shape $a = U$ where a is a linear scheme variable and U is a linear type scheme such that $a \notin \mathrm{FV}(U)$. The set of *exponentials constraints* $\mathcal{C}$ is a set of linear (in)equations of the shape $p = q, p \geq q, p \geq q_1 + q_2, p > q$ or $p = 0$. $\mathcal{H}_1 \uplus \mathcal{H}_2$ denotes the component-wise union of the constraints sequences $\mathcal{H}_1$ and $\mathcal{H}_2$. Sometimes we omit the empty set of a constraints sequence, i.e. $\mathcal{H} \uplus \{p = q\}$ denotes $\mathcal{H} \uplus \langle \emptyset, \{p = q\} \rangle$.

A scheme substitution s satisfies $\mathcal{H}$ if and only if $s(a) \equiv s(U)$, for every equation $a = U$ in $\mathcal{P}$, and $s(p)$ $\mathtt{op}$ $s(q)$, for every p $\mathtt{op}$ q in $\mathcal{C}$.

3.2 Unification Algorithm

In Table 2, we introduce the algorithm $\mathtt{U}$, which allows to unify type schemes under some assumptions. $\mathtt{U}$ proves judgments of the shape

$$\mathtt{U}(\phi, \psi) = \mathcal{H}$$

where ϕ and ψ are two type schemes and $\mathcal{H} = \langle \mathcal{P}, \mathcal{C} \rangle$ is a constraint sequence representing the constraints under which ϕ and ψ can be unified. Namely $\mathcal{P}$ represents the constraints on the structure of the type schemes, while $\mathcal{C}$ the constraints on the number of modalities.

Note that rule $(\mathtt{U}_6)$ keeps down the number of rules, nevertheless it can be cause of non termination (infinite derivations). It is easy (but boring) to give a different definition of the algorithm without the rule $(\mathtt{U}_6)$, by making explicit a symmetric version of all the rules. In what follows we assume to use such an extended version of the algorithm. Note that some inputs does not admit a derivation, by rule $(\mathtt{U}_2)$, in such a cases the unification fails. The following easy theorem assures a weak form of successful termination which will be useful in the sequel.

Theorem 4 ($\mathtt{U}$ Termination). *Let $\phi, \psi \in \mathcal{T}$ be disjoint. Then, there exists $\mathcal{H}$ such that $\mathtt{U}(\phi, \psi) = \mathcal{H}$.*

The algorithm $\mathtt{U}$ is correct and complete, as shown in the following.

Theorem 5 ($\mathtt{U}$ Correctness). *Let $\phi, \psi \in \mathcal{T}$. If $\mathtt{U}(\phi, \psi) = \mathcal{H}$ then for every scheme substitution s satisfying $\mathcal{H}$*

$$s(\phi) \equiv s(\psi)$$

Proof. By induction on the derivation of $\mathtt{U}(\phi, \psi) = \langle \mathcal{P}, \mathcal{C} \rangle$. Note that the existence of a scheme substitution satisfying the constraints in $\langle \mathcal{P}, \mathcal{C} \rangle$ is decidable. □

Theorem 6 (U Completeness). *If $s(\phi) \equiv s(\psi)$ then there exists $\mathcal{H}$ such that $\mathtt{U}(\phi, \psi) = \mathcal{H}$ and s satisfies $\mathcal{H}$.*

Proof. By induction on the shape of ϕ and ψ. □

3.3 The Algorithm

The type inference algorithm defined in Table 3 proves statement of the shape

$$\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{H} \rangle$$

where Ψ is a type scheme context, U is a linear type scheme and $\mathcal{H}$ is a constraints sequence. The type inference algorithm uses the procedure `Unify`, defined in Table 4, that is just an extension of the unification algorithm `U` to type scheme contexts and type schemes.

It is worth noticing the difference between this algorithm and the type inference algorithm for simple types. The latter generates a principal typing, which is a typing for the input term, and for which all and only the typings derivable for the same term are derivable, through substitutions. If the input term cannot be typed, then the algorithm fails. In the current setting, our algorithm generates a sort of an abstract representation of all the typings for the input term `M`, in the sense that, if the constraint sequence $\mathcal{H}$ can be satisfied by a scheme substitution s, then $s(\Psi) \vdash \mathtt{M} : s(U)$ is a typing for `M`, and moreover all typings for `M` can be built from $\mathbf{\Pi}(\mathtt{M})$ by a scheme substitution satisfying $\mathcal{H}$, plus some applications of rules dealing with the modality. If the constraints are not satisfiable, then `M` cannot be typed.

Table 3. Type Inference Algorithm

$$\mathbf{\Pi}(\mathtt{x}) = \mathtt{let}\ a, p\ \mathtt{be\ fresh\ in}\ \ \langle \{\mathtt{x} :!^{p} a\}, a, \{\emptyset, \emptyset\} \rangle$$

$$\begin{array}{l}
\mathbf{\Pi}(\lambda\mathtt{x}.\mathtt{M}) = \mathtt{let}\ \mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{H} \rangle\ \mathtt{in} \\
\qquad \mathtt{let}\ \Psi = \Psi' \odot \{\mathtt{x} :!^{s_1} V_1, \ldots, \mathtt{x} :!^{s_n} V_n\} \mathtt{in\ let}\ a, r\ \mathtt{be\ fresh\ in} \\
\qquad\qquad \mathtt{if}\ n = 0\ \mathtt{then}\ \ \langle \Psi', !^{r} a \multimap U, \mathcal{H} \rangle \\
\qquad\qquad \mathtt{if}\ n = 1\ \mathtt{then}\ \ \langle \Psi', !^{r} V_1 \multimap U, \mathcal{H} \uplus \{r \geq s_1\} \rangle \\
\qquad\qquad \mathtt{if}\ n > 1\ \mathtt{then}\ \ \langle \Psi', !^{r} V_1 \multimap U, \mathcal{H} \uplus \{r > s_1, \ldots, r > s_n\} \rangle
\end{array}$$

$$\begin{array}{l}
\mathbf{\Pi}(\mathtt{MN}) = \mathtt{let}\ \mathbf{\Pi}(\mathtt{M}) = \langle \Psi_{\mathtt{M}}, U, \mathcal{H}_{\mathtt{M}} \rangle\ \mathtt{and}\ \mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, V, \mathcal{H}_{\mathtt{N}} \rangle\ \ \mathtt{be\ disjoint\ in} \\
\quad \mathtt{let}\ a, q_i, p\ \mathtt{be\ fresh\ in\ let}\ \Psi'_{\mathtt{N}} = \{\mathtt{z} :!^{q_i} V_i \mid \exists \mathtt{z} :!^{p_i} V_i \in \Psi_{\mathtt{N}}\}, \\
\quad \mathcal{H} = \mathtt{Unify}(\Psi_{\mathtt{M}}, \Psi'_{\mathtt{N}}, U, !^{p} V \multimap a)\ \mathtt{in}\ \langle \Psi_{\mathtt{M}} \sqcup \Psi'_{\mathtt{N}}, a, \mathcal{H}_{\mathtt{M}} \uplus \mathcal{H}_{\mathtt{N}} \uplus \mathcal{H} \uplus \{q_i \geq p_i + p\} \rangle
\end{array}$$

We need to prove that the type inference algorithm is well defined.

Theorem 7 (Π Termination). *Let $\mathtt{M} \in \Lambda$. Then there exist Ψ, U and $\mathcal{H}$ such that $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{H} \rangle$.*

Proof. By induction on the structure of `M`, using Theorem 4.

The use of multisets instead of sets as contexts in STA helps in the design of the algorithm, maintaining the correcteness of typing. Note that in the definition of $\mathbf{\Pi}$, in the abstraction case we can freely take only the type scheme of the first occurrence (if any) of the variable to be abstracted since all the type schemes have already been unified. The same holds for the `Unify` procedure. We can now finally prove the main theorems of this section.

Theorem 8 ($\mathbf{\Pi}$ Correctness). *Let* $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{H} \rangle$. *Then, for each scheme substitution* s *satisfying* $\mathcal{H}$,

$$s(\Psi) \vdash \mathtt{M} : s(U)$$

Proof. By induction on the derivation proving $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{H} \rangle$. We will show just the most difficult case, when the term is of the shape `PN`. Consider the case $\mathbf{\Pi}(\mathtt{PN}) = \langle \Psi, U, \mathcal{H} \rangle$. By definition U is a scheme variable a, $\mathbf{\Pi}(\mathtt{P}) = \langle \Psi_{\mathtt{P}}, U_{\mathtt{P}}, \mathcal{H}_{\mathtt{P}} \rangle$, $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, U_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$ and they are all disjoint. Let s be a scheme substitution satisfying $\mathcal{H}$. Since s clearly satisfies $\mathcal{H}_{\mathtt{P}}$ and $\mathcal{H}_{\mathtt{N}}$ then by induction we have both $s(\Psi_{\mathtt{P}}) \vdash \mathtt{P} : s(U_{\mathtt{P}})$ and $s(\Psi_{\mathtt{N}}) \vdash \mathtt{N} : s(U_{\mathtt{N}})$. By definition $\mathtt{U}(U_{\mathtt{P}}, !^p U_{\mathtt{N}} \multimap a) = \mathcal{H}'$ with $\mathcal{H}' \subseteq \mathcal{H}$, so since s satisfies $\mathcal{H}$, by Theorem 5: $s(U_{\mathtt{P}}) \equiv s(!^p U_{\mathtt{N}}) \multimap s(a)$. Let $\Psi'_{\mathtt{N}} = \{\mathtt{z} :! ^{q_i} V_i \mid \exists \mathtt{z} :! ^{p_i} V_i \in \Psi_{\mathtt{N}}\}$. Then clearly $s(\Psi) = s(\Psi_{\mathtt{P}} \sqcup \Psi'_{\mathtt{N}}) = s(\Psi_{\mathtt{P}}), s(\Psi'_{\mathtt{N}})$. So, let $s(p) = k$. Then, the following derivation can be built

$$\dfrac{s(\Psi_{\mathtt{P}}) \vdash \mathtt{P} : s(!^p U_{\mathtt{N}}) \multimap s(a) \qquad \dfrac{\dfrac{\dfrac{s(\Psi_{\mathtt{N}}) \vdash \mathtt{N} : s(U_{\mathtt{N}})}{!^k s(\Psi_{\mathtt{N}}) \vdash \mathtt{N} :! ^k s(U_{\mathtt{N}})}\,(sp)^k}{s(\Psi'_{\mathtt{N}}) \vdash \mathtt{N} :! ^k s(U_{\mathtt{N}})}\,(m)^*}{s(\Psi_{\mathtt{P}}), s(\Psi'_{\mathtt{N}}) \vdash \mathtt{PN} : s(a)}\,(\multimap E)$$

□

Table 4. `Unify` procedure

$$\begin{array}{l}\mathtt{Unify}(\Phi, \Psi, \phi, \psi) = \mathtt{let}\ \mathtt{x}_1, \ldots, \mathtt{x}_m = \mathrm{dom}(\Phi) \cap \mathrm{dom}(\Psi),\ \forall 1 \leq i \leq m \\ \qquad \Phi(\mathtt{x}_i) = \{!^{s_1} a_1, \ldots, !^{s_n} a_n\},\ \Psi(\mathtt{x}_i) = \{!^{r_1} b_1, \ldots, !^{r_k} b_k\}, \\ \qquad \mathtt{U}(\phi, \psi) = \langle \mathcal{P}_0, \mathcal{C}_0 \rangle,\ \mathtt{U}(a_1, b_1) = \langle \mathcal{P}_i, \mathcal{C}_i \rangle \\ \qquad \mathtt{in} \langle \bigcup_{j=0}^{m} \mathcal{P}_j, \bigcup_{j=0}^{m} \mathcal{C}_j \rangle,\end{array}$$

Theorem 9 ($\mathbf{\Pi}$ Completeness). *Let* $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{H} \rangle$. *If* $\mathfrak{A} \vdash \mathtt{M} : \sigma$, *then there exists a scheme substitution* s *satisfying* $\mathcal{H}$ *such that*

$$\Sigma \rhd s(\Psi) \vdash \mathtt{M} : s(U)$$

Moreover, the sequent $\mathfrak{A} \vdash \mathtt{M} : \sigma$ *can be obtained from* Σ *by a (maybe empty) sequence of applications of the rules* (w), (m) *and* (sp).

Proof. By induction on the derivation Π proving $\mathfrak{A} \vdash \mathtt{M} : \sigma$. We will show just the case where Π ends as

$$\dfrac{\Sigma' \rhd \mathfrak{A} \vdash \mathtt{N} : \sigma \multimap A \quad \Theta' \rhd \mathfrak{B} \vdash \mathtt{P} : \sigma \quad \mathfrak{A} \approx \mathfrak{B}}{\mathfrak{A}, \mathfrak{B} \vdash \mathtt{NP} : A}\,(\multimap E)$$

Let $\mathbf{\Pi}(\mathtt{NP}) = \langle \Psi, U, \mathcal{H} \rangle$. Then, there are disjoint $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, U_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$, $\mathbf{\Pi}(\mathtt{P}) = \langle \Psi_{\mathtt{P}}, U_{\mathtt{P}}, \mathcal{H}_{\mathtt{P}} \rangle$. By induction, there are scheme substitutions $s_{\mathtt{N}}$ and $s_{\mathtt{P}}$ satisfying respectively $\mathcal{H}_{\mathtt{N}}$ and $\mathcal{H}_{\mathtt{P}}$, such that $\Sigma'' \rhd s_{\mathtt{N}}(\Psi_{\mathtt{N}}) \vdash \mathtt{N} : s_{\mathtt{N}}(U_{\mathtt{N}})$ and $\Theta'' \rhd s_{\mathtt{P}}(\Psi_{\mathtt{P}}) \vdash \mathtt{P} : s_{\mathtt{P}}(U_{\mathtt{P}})$ and Σ' and Θ' can be obtained respectively from Σ'' and Θ'' by some applications of the rules (w), (m) and/or (sp).

Since $\mathcal{H}_{\mathtt{N}}$ and $\mathcal{H}_{\mathtt{P}}$ are disjoint, we can build a scheme substitution s' satisfying both, just acting as each one of the previous substitutions on the corresponding domain. By definition of $\mathbf{\Pi}$, $\Psi = \Psi_{\mathtt{N}} \sqcup \Psi'_{\mathtt{P}}$ where, if $\Psi_{\mathtt{P}} = \mathtt{x}_1 :!^{p_1} V_1, \ldots, \mathtt{x}_n :!^{p_n} V_n$, then $\Psi'_{\mathtt{P}} = \mathtt{x}_1 :!^{q_1} V_1, \ldots, \mathtt{x}_n :!^{q_n} V_n$ for fresh $q_1, \ldots, q_n$. Moreover, for fresh a and p, if $\mathtt{Unify}(\Psi_{\mathtt{N}}, \Psi'_{\mathtt{P}}, U_{\mathtt{N}}, !^p U_{\mathtt{P}} \multimap a) = \mathcal{H}'$ then $U \equiv a$, and $\mathcal{H} = \mathcal{H}_{\mathtt{N}} \uplus \mathcal{H}_{\mathtt{P}} \uplus \mathcal{H}' \uplus \{q_i \geq p_i + p\}$. Since a and p are fresh, we can choose s' satisfying also $s'(U_{\mathtt{N}}) \equiv \sigma \multimap A \equiv s'(!^p U_{\mathtt{P}} \multimap a)$. Hence in particular by Theorem 6 s' satisfies $\mathcal{H}'$. Moreover, since $q_1, \ldots, q_n$ are fresh, it is easy to extend s' to a scheme substitution $s = s'[q_1 \mapsto s(p_1) + s(p), \ldots, q_n \mapsto s(p_n) + s(p)]$. Clearly s satisfies $\mathcal{H}$. Let $s(p) = k$. Then we can build the following derivation

$$\dfrac{\Sigma' \rhd s(\Psi_{\mathtt{N}}) \vdash \mathtt{N} : s(U_{\mathtt{N}}) \qquad \dfrac{\Theta' \rhd s(\Psi_{\mathtt{P}}) \vdash \mathtt{P} : s(U_{\mathtt{P}})}{!^k s(\Psi_{\mathtt{P}}) \vdash \mathtt{P} :!^k s(U_{\mathtt{P}})}\,(sp)^k}{s(\Psi_{\mathtt{N}}), !^k s(\Psi_{\mathtt{P}}) \vdash \mathtt{NP} : s(\forall t.a)}\,(\multimap E)$$

and $\mathfrak{A}, \mathfrak{B} \vdash \mathtt{NP} : A$ can be obtained from it by a sequence of applications of the rules (w), (m) and (sp). □

In the following we will give some examples, and in the next section we will discuss the constraints resolution in them. These example are useful both to understand the behaviour of the algorithm and to compare the typability power of STA and other type assignment systems. Namely the first term $(\mathtt{2})$ is typable in STA and in simple type assignment system, the second term $(\mathtt{222})$ is typable in the simple type assignment system but untypable in STA, and the third one $(\mathtt{2}(\mathtt{yz}))$ is typable in STA but untypable in the propositional fragment of DLAL.

Example 10

1. Let $\mathtt{2} \equiv \lambda \mathtt{s}.\lambda \mathtt{z}.\mathtt{s}(\mathtt{sz})$. Then $\mathbf{\Pi}(\mathtt{2}) = \langle \emptyset, U, \langle \mathcal{P}, \mathcal{C} \rangle \rangle$ where

$$\begin{aligned} U &= !^{r_5} a_2 \multimap (!^{r_4} a_1 \multimap b_2) \\ \mathcal{P} &= \{a_2 =!^{q_1} a_1 \multimap b_1, a_3 =!^{q_2} b_1 \multimap b_2, a_2 = a_3\} \\ \mathcal{C} &= \{r_1 \geq p_1 + q_1, r_2 = p_3, r_2 \geq p_2 + q_2, r_3 \geq r_1 + q_2, r_4 \geq r_3, r_5 > p_3\}\rangle \end{aligned}$$

2. A more involved example is related to the term $\mathtt{222}$. Then, we obtain $\mathbf{\Pi}(\mathtt{222}) = \langle \emptyset, U, \langle \mathcal{P}, \mathcal{C} \rangle \rangle$ where

$$\begin{aligned} U &= a'' \\ \mathcal{P} &= \mathcal{P}^0 \cup \mathcal{P}^1 \cup \mathcal{P}^2 \cup \{a_2^1 =!^{r_5^2} a_2^2 \multimap (!^{r_4^2} a_1^2 \multimap b_2^2), a' =!^{r_4^1} a_1^1 \multimap b_2^1, \\ &\qquad a' =!^{p''} (!^{r_5^0} a_2^0 \multimap (!^{r_4^0} a_1^0 \multimap b_2^0)) \multimap a'' \ \} \\ \mathcal{C} &= \mathcal{C}^0 \cup \mathcal{C}^1 \cup \mathcal{C}^2 \cup \{p' = r_5^1\} \end{aligned}$$

and $\mathcal{P}^i = \{a_2^i =!^{q_1^i} a_1^i \multimap b_1^i, a_3^i =!^{q_2^i} b_1^i \multimap b_2^i, a_2^i = a_3^i\}$ while $\mathcal{C}^i = \{r_1^i \geq p_1^i + q_1^i, r_2^i = p_3^i, r_2^i \geq p_2^i + q_2^i, r_3^i \geq r_1^i + q_2^i, r_4^i \geq r_3^i, r_5^i > p_3^i\}$.

3. Let us consider now the term $\mathtt{2(yz)}$. The application of the algorithm produces: $\mathbf{\Pi}(\mathtt{2(yz)}) = \langle\{\mathtt{y} :!^r c, \mathtt{z} :!^s d\}, U, \langle\mathcal{P}^*, \mathcal{C}^*\rangle\rangle$, where:

$$\begin{aligned} U &= f \\ \mathcal{P}^* &= \mathcal{P} \cup \{a_2 = e, c =!^t d \multimap e, f =!^{r_4} a_1 \multimap b_2\} \\ \mathcal{C}^* &= \mathcal{C} \cup \{r_5 = t', r \geq r' + t', s \geq s' + t', s' \geq t + s''\} \end{aligned}$$

where $\mathcal{P}$ and $\mathcal{C}$ are defined as in point 1 of this example.

4 Constraints Resolution

Let $\mathbf{\Pi}(\mathtt{M}) = \langle\Psi, U, \mathcal{H}\rangle$, where $\mathcal{H} = \langle\mathcal{P}, \mathcal{C}\rangle$. The resolution of the constraints in $\mathcal{H}$ is splitted in two phases. The first one is the application of the standard Robinson resolution [10] to $\mathcal{P}$, so obtaining a new set of constraints, that can be in its turn splitted in a set $\mathcal{P}'$ of constraints on schemes, and $\mathcal{C}'$ of constraints on exponentials. Then the second phase is to find a scheme substitution satisfying the constraints $\mathcal{P}'$ and $\mathcal{C} \cup \mathcal{C}'$. Some examples can clarify the procedure.

Example 11

1. Let us continue Example 10.1, i.e., $\mathbf{\Pi}(\mathtt{2})$. Then, the application of the Robinson resolution to the set $\mathcal{P}$ and $\mathcal{C}$ generates $\mathcal{P}' = \{a_2 =!^{q_1} a_1 \multimap b_1, a_1 = b_1, b_1 = b_2, a_2 = a_3\}$ and $\mathcal{C}' = \{q_1 = q_2\}$ respectively. The substitution

$$s = s'[a_1, b_1, b_2 \mapsto \alpha; a_2, a_3 \mapsto \alpha \multimap \alpha; p_1, p_2, p_3, q_1, q_2, r_1, r_2, r_3, r_4 \mapsto 0; r_5 \mapsto 1]$$

   satisfies the constraints $\mathcal{P}'$, $\mathcal{C}$, $\mathcal{C}'$ for all s', and generates the typing $\emptyset \vdash 2 : !(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$. Hence the term is typable.
2. Let us continue Example 10.2, i.e., $\mathbf{\Pi}(\mathtt{222})$. The application of the Robinson resolution is boring but easy, applied to $\mathcal{P}$ it produces a solvable set of constraints on type schemes and the final type is defined through the type scheme equation

$$a'' =!^{q_1^2}(!^{q_1^0} a_1^0 \multimap a_1^0) \multimap (!^{q_1^0} a_1^0 \multimap a_1^0).$$

   But Robinson algorithm changes also the set of constraints on exponentials $\mathcal{C}$ into the set $\mathcal{C}' = \mathcal{C} \cup \{q_1^0 = r_4^0, q_1^2 = r_5^0, r_4^1 = p'', q_1^2 = r_4^2, q_1^2 = q_2^2, q_1^1 = q_2^1, q_1^0 = q_2^0, q_1^1 = r_5^2\}$ which can be simplified in $\mathcal{C}'' = \{r_5^2 > r_2^2 \geq p_2^2, r_1^1 \geq p_1^1 + q_1^1, p' > r_2^1 \geq p_2^1 + q_1^1, r_4^1 \geq r_3^1 \geq r_1^1 + q_1^1, q_1^2 > r_2^0, q_1^2 = 0\}$. This set is clearly not satisfiable since the last two constraints are contradictory, and so the term is not typable.
3. Let us continue Example 10.3, i.e. $\mathbf{\Pi}(\mathtt{2(yz)})$. The application of Robinson resolution to $\mathcal{P}^*$ gives a set $\mathcal{P}^{**} = \mathcal{P}' \cup \{c =!^t d \multimap e, a_2 = e, f =!^{r_4} a_1 \multimap b_2\}$ and the set of exponential constraints becomes $\mathcal{C}^{**} = \mathcal{C}' \cup \mathcal{C} \cup \{r_5 = t', r \geq r' + t', s \geq s' + t', s' \geq t + s''\}$, where $\mathcal{P}'$and $\mathcal{C}'$ are defined as at point 1 of this example while $\mathcal{C}$ is defined as in Example 10.1. Let s be the substitution at point 1 of this example; then the substitution:

$$s^* = s[c \mapsto (\alpha \multimap \alpha) \multimap \alpha \multimap \alpha; d \mapsto \alpha \multimap \alpha; r', s', t, s'' \mapsto 0; r, s, t' \mapsto 1]$$

satisfies the constraints in $\mathcal{P}^{**}$ and $\mathcal{C}^{**}$ and generates the typing:

$$\mathtt{y} :!((\alpha \multimap \alpha) \multimap \alpha \multimap \alpha), \mathtt{z} :!(\alpha \multimap \alpha) \vdash \mathtt{2(yz)} : \alpha \multimap \alpha.$$

Note that the term $\mathtt{2(yz)}$ is not typable in DLAL due to the presence of two free variables that must be duplicated.

4.1 Type Inference Complexity

It can be shown that our algorithm works in polynomial time. In particular it is easy to verify that the construction of $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{H} \rangle$ can be done in time polynomial in $|\mathtt{M}|$.

Let $\mathcal{H} = \langle \mathcal{P}, \mathcal{C} \rangle$. The application of Robinson resolution to $\mathcal{P}$, generating $\mathcal{P}'$ and $\mathcal{C}'$, is polynomial in the number of both the scheme variables and exponentials in $\mathcal{P}$. The solution of the constraints in $\mathcal{P}'$ can be done through the standard algorithm working on the dag representation of schemes, and so it is polynomial in the number of scheme variables in $\mathcal{P}'$, which coincides with the number of scheme variables in $\mathcal{P}$.

As far as the exponential resolution task, i.e., the problem of solving the constraints in $\mathcal{C} \cup \mathcal{C}'$, is concerned, apparently it seems more difficult, since the problem of solving integer inequalities is in general NP-complete [11]. Nevertheless, following the method shown in [12], we can solve the problem over rational, which takes time polynomial in the number of exponentials. Clearly the set of solutions is closed under multiplication by positive integers. Now an integer solution can be obtained simply multiplying a rational solution by a suitable integer.

It is easy to check that the number of symbols in the constraints generated by Π is polynomial in $|\mathtt{M}|$. So the type inference problem for the propositional fragment can be decided in polynomial time in the size of the term.

5 Type Inference for the Full System

5.1 Schemes, Substitutions and Constraints

Definition 12. *The grammar of* type schemes $\mathcal{T}$*, given in Definition 1, is extended as follows*

$$U, V, Z ::= a \mid \phi \multimap U \mid [t].a \mid [t].\phi \multimap U \ \textit{(Linear type schemes)} \quad \phi, \psi ::= U \mid !^p U$$

where t, u, v belong to a countable set of sequence variables*.*

The notation $[t]$ does not introduce bound variables. Note that schemes of the shape $[t].[u].U$ are not allowed. $\mathrm{FV}(\phi)$ now denotes the set of linear scheme variables, exponentials and sequence variables occurring in ϕ. Two type schemes ϕ, ψ are *disjoint* if $\mathrm{FV}(\phi) \cap \mathrm{FV}(\psi) = \emptyset$.

A *scheme substitution* s is extended to map sequence variables to sequences of type variables. Namely the application of s to a type scheme is extended by the following rule

$$s([t].U) = \begin{cases} s(U) & \text{if } [t \mapsto \varepsilon] \in s \\ \forall \boldsymbol{\alpha}.s(U) & \text{if } [t \mapsto \boldsymbol{\alpha}] \in s \end{cases}$$

As in the propositional case, a type scheme is an abstract representation of all the types that can be obtained from it by a scheme substitution., e.g., the type scheme $[t].([u].b) \multimap a$ represents the set $\{\forall \boldsymbol{\alpha}.(\forall \boldsymbol{\beta}.A) \multimap B, (\forall \boldsymbol{\beta}.A) \multimap B, \forall \boldsymbol{\alpha}.A \multimap B \mid A, B \in \mathbf{T}\}$. The notion of *type scheme context* and its notation can be straightforwardly adapted from the one for the propositional fragment.

A *constraints sequence* $\mathcal{H}$ is a triple $\langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle$ of constraints sets, where $\mathcal{P}$ and $\mathcal{C}$ are as in Subsection 3.1, and $\mathcal{Q}$ is a set of equations of the shape $t = u$ or $t = \varepsilon$, where t, u are sequence variables. $\mathcal{Q}$ is satisfied by a scheme substitution s if $s(t) = s(u)$ $(s(t) = \varepsilon)$, for every $t = u$ $(t = \varepsilon)$ in it.

Table 5. Unification Algorithm

$$\frac{}{\mathtt{U}(a,a) = \langle \emptyset, \emptyset, \emptyset \rangle}\ (\mathtt{U}_0) \qquad \frac{}{\mathtt{U}(a,b) = \langle \{a = b\}, \emptyset, \emptyset \rangle}\ (\mathtt{U}_1) \qquad \frac{a \notin \mathrm{FV}(\phi \multimap U)}{\mathtt{U}(a, \phi \multimap U) = \langle \{a = \phi \multimap U\}, \emptyset, \emptyset \rangle}\ (\mathtt{U}_2)$$

$$\frac{\mathtt{U}(a,U) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}{\mathtt{U}(a, [t].U) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \cup \{t = \epsilon\} \rangle}\ (\mathtt{U}_3) \qquad \frac{\mathtt{U}(a,V) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}{\mathtt{U}(a, !^p V) = \langle \mathcal{P}, \mathcal{C} \cup \{p = 0\}, \mathcal{Q} \rangle}\ (\mathtt{U}_4)$$

$$\frac{\mathtt{U}(\phi,\psi) = \langle \mathcal{P}_1, \mathcal{C}_1, \mathcal{Q}_1 \rangle \quad \mathtt{U}(U,V) = \langle \mathcal{P}_2, \mathcal{C}_2, \mathcal{Q}_2 \rangle}{\mathtt{U}(\phi \multimap U, \psi \multimap V) = \langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{Q}_1 \cup \mathcal{Q}_2 \rangle}\ (\mathtt{U}_5)$$

$$\frac{\mathtt{U}(\phi \multimap U, V) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}{\mathtt{U}(\phi \multimap U, [t].V) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \cup \{t = \varepsilon\} \rangle}\ (\mathtt{U}_6) \qquad \frac{\mathtt{U}(\phi \multimap U, V) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}{\mathtt{U}(\phi \multimap U, !^p V) = \langle \mathcal{P}, \mathcal{C} \cup \{p = 0\}, \mathcal{Q} \rangle}\ (\mathtt{U}_7)$$

$$\frac{\mathtt{U}(U,V) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}{\mathtt{U}([t].U, [u].V) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \cup \{t = u\} \rangle}\ (\mathtt{U}_8) \qquad \frac{\mathtt{U}([t].U, V) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}{\mathtt{U}([t].U, !^p V) = \langle \mathcal{P}, \mathcal{C} \cup \{p = 0\}, \mathcal{Q} \rangle}\ (\mathtt{U}_9)$$

$$\frac{\mathtt{U}(\psi,\phi) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}{\mathtt{U}(\phi,\psi) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}\ (\mathtt{U}_{10}) \qquad \frac{\mathtt{U}(U,V) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle}{\mathtt{U}(!^p U, !^q V) = \langle \mathcal{P}, \mathcal{C} \cup \{p = q\}, \mathcal{Q} \rangle}\ (\mathtt{U}_{11})$$

5.2 Unification Algorithm

In Table 5 we present a unification algorithm $\mathtt{U}$ extending the one presented in the propositional case. $\mathtt{U}$ proves judgments of the shape

$$\mathtt{U}(\phi,\psi) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle$$

where ϕ and ψ are the two schemes that must be unified and $\langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle$ is a constraint sequence. Since the notation $[t]$ does not introduce bound variables in type schemes, we can consider it as a first order symbol. Then the unification problem we are considering is an instance of first order unification. As in the propositional case we have the following easy results.

Theorem 13 (U Termination). *Let $\phi, \psi \in \mathcal{T}$ be disjoint. Then, there exist $\mathcal{P}, \mathcal{C}$ and $\mathcal{Q}$ such that $\mathtt{U}(\phi, \psi) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle$.*

Theorem 14 (U Correctness). *Let $\phi, \psi \in \mathcal{T}$. If $\mathtt{U}(\phi, \psi) = \mathcal{H}$ then, for every substitution s satisfying $\mathcal{H}$*

$$s(\phi) \equiv s(\psi)$$

Proof. By induction on the derivation of $\mathtt{U}(\phi, \psi) = \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle$ noting that $\mathcal{Q}$ contains equalities of the shape $t = u$ or $t = \varepsilon$, hence the existence of a substitution satisfying this kind of constraints is decidable. □

We need now to prove that the algorithm U is also complete. The design of the type inference algorithm will be such that we need just to prove the completeness for the $\equiv$ relation of types. This agrees with the fact proved in [13] that typing in System F does not need the explicit use of α-rule.

Theorem 15 (U Completeness). *If $s(\phi) \equiv s(\psi)$ then there exists $\mathcal{H}$ such that $\mathtt{U}(\phi, \psi) = \mathcal{H}$ and s satisfies $\mathcal{H}$.*

Proof. By straighforward induction on the shape of ϕ and ψ. We will show just the case when $\phi \equiv [t].U$ and $\psi \equiv [u].V$. Let $s(\psi) \equiv s(\phi) = \forall \boldsymbol{\alpha}.\sigma$. Then $s(U) \equiv s(V)$, and by induction $\mathtt{U}(U, V) = \mathcal{H}'$. By rule $\mathtt{U}_8$, $\mathtt{U}([t].U, [u].V) = \langle \mathcal{H} \cup \{t = u\} \rangle$. So $s' = s[t \mapsto \boldsymbol{\alpha}, u \mapsto \boldsymbol{\alpha}]$ is the desired substitution. □

Remark. Note that a stronger completeness property holds for U, namely if $s(\phi)$ and $s(\psi)$ are α-equivalent, then there exists $\mathcal{H}$ such that $\mathtt{U}(\phi, \psi) = \mathcal{H}$ and there is a scheme substitution s' satisfying $\mathcal{H}$ such that $s'(\phi) \equiv s'(\psi)$, and $s'(\phi)$ is α-equivalent to both $s(\phi)$, $s(\psi)$. In fact, if $s(\phi)$ and $s(\psi)$ are α-equivalent, it is always possible to build a substitution s' such that $s'(\phi) \equiv s'(\psi)$, by renaming the bound variables, and then Theorem 15 can be applied.

Table 6. Type Inference Algorithm

$$\boldsymbol{\Pi}(\mathtt{x}) = \mathtt{let}\ u, t, a, b, p\ \mathtt{be\ fresh\ in}\ \ \langle \{\mathtt{x} :!^{p}[t].a\}, [u].b, \{([t].a, b)\}, [u \mapsto \{[t].a\}], \{\emptyset, \emptyset, \emptyset\} \rangle$$

$$\begin{array}{l}\boldsymbol{\Pi}(\lambda \mathtt{x}.\mathtt{M}) = \mathtt{let}\ \boldsymbol{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle\ \mathtt{in} \\ \quad \mathtt{let}\ \Psi = \Psi' \odot \{\mathtt{x} :!^{s_1} V_1, \ldots, \mathtt{x} :!^{s_n} V_n\}\ , \mathcal{I} = \textsc{range}(\Psi')\ \mathtt{in\ let}\ u, t, a, r\ \mathtt{be\ fresh\ in} \\ \quad\ \mathtt{if}\ n = 0\ \mathtt{then}\ \ \langle \Psi', [u].!^{r}([t].a) \multimap U, \mathcal{G}, \mathcal{F} + [u \mapsto \mathcal{I}], \mathcal{H} \rangle \\ \quad\ \mathtt{else\ if}\ n = 1\ \mathtt{then}\ \ \langle \Psi', [u].!^{r} V_1 \multimap U, \mathcal{G}, \mathcal{F} + [u \mapsto \mathcal{I}], \mathcal{H} \uplus \{r \geq s_1, \} \rangle \\ \quad\ \mathtt{else\ if}\ n > 1\ \mathtt{then}\ \ \langle \Psi', [u].!^{r} V_1 \multimap U, \mathcal{G}, \mathcal{F} + [u \mapsto \mathcal{I}], \mathcal{H} \uplus \{r > s_1, \ldots, r > s_n\} \rangle\end{array}$$

$$\begin{array}{l}\boldsymbol{\Pi}(\mathtt{MN}) = \mathtt{let}\ \boldsymbol{\Pi}(\mathtt{M}) = \langle \Psi_{\mathtt{M}}, U, \mathcal{G}_{\mathtt{M}}, \mathcal{F}_{\mathtt{M}}, \mathcal{H}_{\mathtt{M}} \rangle\ \mathtt{and}\ \boldsymbol{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, V, \mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle\ \ \mathtt{be\ disjoint\ in} \\ \qquad \mathtt{let}\ u, t, a, b, q_i, p\ \mathtt{be\ fresh\ in}\ \ \mathtt{let}\ \Psi'_{\mathtt{N}} = \{\mathtt{z} :!^{q_i} V_i \mid \exists \mathtt{z} :!^{p_i} V_i \in \Psi_{\mathtt{N}}\}, \\ \qquad\qquad \mathcal{I} = \textsc{range}(\Psi_{\mathtt{M}} \sqcup \Psi_{\mathtt{N}}),\ \mathcal{H} = \mathtt{Unify}(\Psi_{\mathtt{M}}, \Psi'_{\mathtt{N}}, U, !^{p} V \multimap [t].a)\ \mathtt{in} \\ \langle \Psi_{\mathtt{M}} \sqcup \Psi'_{\mathtt{N}}, [u].b, \mathcal{G}_{\mathtt{M}} \cup \mathcal{G}_{\mathtt{N}} \cup \{([t].a, b)\}, \mathcal{F}_{\mathtt{M}} + \mathcal{F}_{\mathtt{N}} + [u \mapsto \mathcal{I}], \mathcal{H}_{\mathtt{M}} \uplus \mathcal{H}_{\mathtt{N}} \uplus \mathcal{H} \uplus \{q_i \geq p_i + p\} \rangle\end{array}$$

5.3 The Algorithm

The Type Inference Algorithm follows the same lines of the type inference algorithm for System F designed by Ronchi Della Rocca and Giannini in [4]. In order to define it, we need to introduce some further notions.

Definition 16. *The* containment relation $\leq$ *between soft types is the relation defined as follows* $\forall \boldsymbol{\alpha}.A \leq A[\boldsymbol{B}/\boldsymbol{\alpha}]$, *for some* B.

Note that $\sigma \leq \tau$ corresponds to the fact that to a term $\mathtt{M}$ of type σ we can assign also the type τ by some applications of the rule $(\forall E)$. The relation $\leq$ is clearly decidable. Remembering that $\boldsymbol{\alpha}$ could be an empty sequence, $\leq$ is obviously reflexive. Moreover, it is transitive, hence a preorder. Note that $\forall \boldsymbol{\alpha}.\tau \multimap \sigma \leq \tau_1 \multimap \sigma_1$ implies $\forall \boldsymbol{\alpha}.\tau \leq \tau_1$ and $\forall \boldsymbol{\alpha}.\sigma \leq \sigma_1$, while in general the converse does not hold.

A *scheme system* $\mathcal{G}$ is a set of pairs of type schemes. A set of *binding constraints* $\mathcal{F}$ is a function from sequence variables to finite sets of schemes.

Definition 17. *Let* s *be a scheme substitution.*

- s *satisfies a scheme system* $\mathcal{G} = \{(U_1, V_1), \ldots, (U_n, V_n)\}$ *if and only if* $s(U_i) \leq s(V_i)$, $(1 \leq i \leq n)$.
- s *satisfies a binding constraints* $\mathcal{F} = \{u_1 \mapsto \Gamma_1, \ldots, u_n \mapsto \Gamma_n\}$ *if and only if* $\forall i \leq n, \forall \alpha \in s(u_i), \forall U \in \Gamma_i : \alpha \notin \mathrm{FV}(s(U))$

The type inference algorithm defined in Table 6 proves statement of the shape

$$\boldsymbol{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$$

where Ψ is a type scheme assignment context, U is a linear type scheme, $\mathcal{G}$ is a scheme system, $\mathcal{F}$ is a set of binding constraints and $\mathcal{H}$ is a constraints sequence. The type inference algorithm call the `Unify` procedure, defined in Table 7, on contexts and schemes which need to be unified through the unification algorithm.

Theorem 18 ($\boldsymbol{\Pi}$ Termination). *Let* $\mathtt{M} \in \Lambda$. *Then* $\boldsymbol{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$

Proof. By induction on the structure of $\mathtt{M}$. It is easy to verify that the schemes which need to be unified by the algorithm are always disjoint, so Theorem 14 applies. □

Table 7. `Unify` procedure

$$\begin{array}{l}
\mathtt{Unify}(\Phi, \Psi, \phi, \psi) = \mathtt{let}\ \mathtt{x}_1, \ldots, \mathtt{x}_m = \mathrm{dom}(\Phi) \cap \mathrm{dom}(\Psi),\ \forall 1 \leq i \leq m \\
\qquad \Phi(\mathtt{x}_i) = \{!^{s_1} V_1, \ldots, !^{s_n} V_n\},\ \Psi(\mathtt{x}_i) = \{!^{r_1} U_1, \ldots, !^{r_k} U_k\}, \\
\qquad \mathtt{U}(\phi, \psi) = \langle \mathcal{P}_0, \mathcal{C}_0, \mathcal{Q}_0 \rangle,\ \mathtt{U}(V_1, U_1) = \langle \mathcal{P}_i, \mathcal{C}_i, \mathcal{Q}_i \rangle \\
\qquad \mathtt{in} \langle \bigcup_{j=0}^{m} \mathcal{P}_j, \bigcup_{j=0}^{m} \mathcal{C}_j, \bigcup_{j=0}^{m} \mathcal{Q}_j \rangle
\end{array}$$

Finally we can now prove the main theorems of this section.

Theorem 19 (Π Correctness). *Let* $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. *Then, for each substitution* s *satisfying* $\mathcal{G}$, $\mathcal{F}$ *and* $\mathcal{H}$

$$s(\Psi) \vdash \mathtt{M} : s(U)$$

Proof. By induction on the derivation proving $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. We will show just the most difficult case, when the term $\mathtt{M}$ is of the shape $\mathtt{PN}$.

Consider the case $\mathbf{\Pi}(\mathtt{PN}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. By hypothesis $\mathbf{\Pi}(\mathtt{P}) = \langle \Psi_{\mathtt{P}}, U_{\mathtt{P}}, \mathcal{G}_{\mathtt{P}}, \mathcal{F}_{\mathtt{P}}, \mathcal{H}_{\mathtt{P}} \rangle$, $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, U_{\mathtt{N}}, \mathcal{G}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$ and $U \equiv [u].b$. Let s be a substitution satisfying $\mathcal{G}$, $\mathcal{F}$ and $\mathcal{H}$. By induction hypothesis since s clearly satisfies $\mathcal{G}_{\mathtt{P}}$, $\mathcal{G}_{\mathtt{N}}$, $\mathcal{F}_{\mathtt{P}}$, $\mathcal{F}_{\mathtt{N}}$, $\mathcal{H}_{\mathtt{P}}$ and $\mathcal{H}_{\mathtt{N}}$, then we have both $s(\Psi_{\mathtt{P}}) \vdash \mathtt{P} : s(U_{\mathtt{P}})$ and $s(\Psi_{\mathtt{N}}) \vdash \mathtt{N} : s(U_{\mathtt{N}})$.

Moreover by definition $\mathtt{U}(U_{\mathtt{P}}, !^p U_{\mathtt{N}} \multimap [t].a) = \mathcal{H}'$ with $\mathcal{H}' \subseteq \mathcal{H}$, so since s satisfies $\mathcal{H}$ by Theorem 14: $s(U_{\mathtt{P}}) \equiv s(!^p U_{\mathtt{N}}) \multimap s([t].a)$. Let $\Psi'_{\mathtt{N}} = \{\mathtt{z} :!^{q_i} V_i \mid \exists \mathtt{z} :!^{p_i} V_i \in \Psi_{\mathtt{N}}\}$. Then clearly $s(\Psi) = s(\Psi_{\mathtt{P}} \sqcup \Psi'_{\mathtt{N}}) = s(\Psi_{\mathtt{P}}), s(\Psi'_{\mathtt{N}})$. Moreover since by hypothesis s satisfies $\mathcal{G}$, then in particular $s([t].a) \leq s(b)$. So, let $s(u) = \boldsymbol{\alpha}$ and $s(p) = k$. Then, the conclusion follows by the derivation

$$\dfrac{\dfrac{s(\Psi_{\mathtt{P}}) \vdash \mathtt{P} : s(!^p U_{\mathtt{N}}) \multimap s([t].a) \quad \dfrac{\dfrac{\dfrac{s(\Psi_{\mathtt{N}}) \vdash \mathtt{N} : s(U_{\mathtt{N}})}{!^k s(\Psi_{\mathtt{N}}) \vdash \mathtt{N} :!^k s(U_{\mathtt{N}})}\,(sp)^k}{s(\Psi'_{\mathtt{N}}) \vdash \mathtt{N} :!^k s(U_{\mathtt{N}})}\,(m)^*}{s(\Psi_{\mathtt{P}}), s(\Psi'_{\mathtt{N}}) \vdash \mathtt{PN} : s([t].a)}\,(\multimap E)}{s(\Psi_{\mathtt{P}}), s(\Psi'_{\mathtt{N}}) \vdash \mathtt{PN} : s(b)}\,(\forall E)^*}{s(\Psi_{\mathtt{P}}), s(\Psi'_{\mathtt{N}}) \vdash \mathtt{PN} : \forall \boldsymbol{\alpha}.s(b)}\,(\forall I)^*$$

Note that we have freely applied the $(\forall I)$ rule over variables in $\boldsymbol{\alpha}$ since s satisfies the binding constraints $\mathcal{F}$. □

Theorem 20 (Π Completeness). *Let* $\mathbf{\Pi}(\mathtt{M}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. *If* $\mathfrak{A} \vdash \mathtt{M} : \sigma$ *then there exists a substitution* s *satisfying* $\mathcal{G}, \mathcal{F}$ *and* $\mathcal{H}$ *such that*

$$\Sigma \rhd s(\Psi) \vdash \mathtt{M} : s(U)$$

Moreover, the sequent $\mathfrak{A} \vdash \mathtt{M} : \sigma$ *can be obtained from* Σ *by a (maybe empty) sequence of applications of the rules* (w), (m) *and* (sp).

Proof. By induction on the derivation Π proving $\mathfrak{A} \vdash \mathtt{M} : \sigma$. We consider here the two most difficult cases. Let Π ends as

$$\dfrac{\Sigma \rhd \mathfrak{A} \vdash \mathtt{N} : \sigma \multimap A \quad \Theta \rhd \mathfrak{B} \vdash \mathtt{P} : \sigma \quad \mathfrak{A} \approx \mathfrak{B}}{\mathfrak{A}, \mathfrak{B} \vdash \mathtt{NP} : A}\,(\multimap E)$$

Let $\mathbf{\Pi}(\mathtt{NP}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$, $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi_{\mathtt{N}}, U_{\mathtt{N}}, \mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}, \mathcal{H}_{\mathtt{N}} \rangle$ and $\mathbf{\Pi}(\mathtt{P}) = \langle \Psi_{\mathtt{P}}, U_{\mathtt{P}}, \mathcal{G}_{\mathtt{P}}, \mathcal{F}_{\mathtt{P}}, \mathcal{H}_{\mathtt{P}} \rangle$. By definition of $\mathbf{\Pi}$, $\Psi = \Psi_{\mathtt{N}} \sqcup \Psi'_{\mathtt{P}}$ where, if $\Psi_{\mathtt{P}} = \mathtt{x}_1 :!^{p_1} V_1, \ldots, \mathtt{x}_n :!^{p_n} V_n$, then $\Psi'_{\mathtt{P}} = \mathtt{x}_1 :!^{q_1} V_1, \ldots, \mathtt{x}_n :!^{q_n} V_n$ for fresh $q_1, \ldots, q_n$. Moreover, for fresh u, t, a, b and p, if $\mathcal{I} = \textsc{range}(\Psi)$ and $\mathtt{Unify}(\Psi_{\mathtt{N}}, \Psi'_{\mathtt{P}}, U_{\mathtt{N}}, !^p U_{\mathtt{P}} \multimap [t].a) = \mathcal{H}'$ then $U \equiv [u].b$, $\mathcal{G} = \mathcal{G}_{\mathtt{N}} \cup \mathcal{G}_{\mathtt{P}} \cup \{([t].a, b)\}$, $\mathcal{F} = \mathcal{F}_{\mathtt{N}} + \mathcal{F}_{\mathtt{P}} + [u \mapsto \mathcal{I}]$ and $\mathcal{H} = \mathcal{H}_{\mathtt{N}} \uplus \mathcal{H}_{\mathtt{P}} \uplus \mathcal{H}' \uplus \{q_i \geq p_i + p\}$.

By induction hypothesis there exists a scheme substitution $s_{\mathtt{N}}$ satisfying $\mathcal{G}_{\mathtt{N}}, \mathcal{F}_{\mathtt{N}}$ and $\mathcal{H}_{\mathtt{N}}$ such that $\Sigma' \rhd s_{\mathtt{N}}(\Psi_{\mathtt{N}}) \vdash \mathtt{N} : s_{\mathtt{N}}(U_{\mathtt{N}})$ and a substitution $s_{\mathtt{P}}$ satisfying $\mathcal{G}_{\mathtt{P}}, \mathcal{F}_{\mathtt{P}}$

and $\mathcal{H}_{\mathtt{P}}$ such that $\Theta' \rhd s_{\mathtt{P}}(\Psi_{\mathtt{P}}) \vdash \mathtt{P} : s_{\mathtt{P}}(U_{\mathtt{P}})$ and Σ and Θ can be obtained from Σ' and Θ' by a sequence of applications of the rules (w), (m) and (sp). This implies that $U_{\mathtt{N}}$ and $!^p U_{\mathtt{P}} \multimap [t].a$ are unifiable from Theorem 15.

Since $\mathbf{\Pi}(\mathtt{N})$ and $\mathbf{\Pi}(\mathtt{P})$ are disjoint we can build a substitution s' acting as $s_{\mathtt{N}}$ on schemes in $\mathbf{\Pi}(\mathtt{N})$ and as $s_{\mathtt{P}}$ on schemes in $\mathbf{\Pi}(\mathtt{P})$. Note that $s'(U_{\mathtt{N}}) \equiv \sigma \multimap A \equiv s'(!^p U_{\mathtt{P}} \multimap [t].a)$, where t and a are fresh. Hence in particular s' satisfies $\mathcal{H}'$.

Since $u, b, q_1, \ldots, q_n$ are fresh, it is easy to extend s' to a substitution $s = s'[b \mapsto s([t].a), u \mapsto \epsilon, q_1 \mapsto s(p_1) + s(p), \ldots, q_n \mapsto s(p_n) + s(p)]$. Clearly s satisfies $\mathcal{G}, \mathcal{F}$ and $\mathcal{H}$. If $s(p) = k$, then the following derivation can be built

$$\dfrac{\Sigma' \rhd s(\Psi_{\mathtt{N}}) \vdash \mathtt{N} : s(U_{\mathtt{N}}) \qquad \dfrac{\Theta' \rhd s(\Psi_{\mathtt{P}}) \vdash \mathtt{P} : s(U_{\mathtt{P}})}{!^k s(\Psi_{\mathtt{P}}) \vdash \mathtt{P} :!^k s(U_{\mathtt{P}})}\ (sp)^k}{s(\Psi_{\mathtt{N}}), !^k s(\Psi_{\mathtt{P}}) \vdash \mathtt{NP} : s([t].a)}\ (\multimap E)$$

and $\mathfrak{A}, \mathfrak{B} \vdash \mathtt{NP} : A$ can be obtained from it by a sequence of applications of the rules (w), (m) and (sp).

Consider the case where Π ends as

$$\dfrac{\Sigma \rhd \mathfrak{A} \vdash \mathtt{N} : \forall\alpha.A}{\mathfrak{A} \vdash \mathtt{N} : A[B/\alpha]}\ (\forall E)$$

Let $\mathbf{\Pi}(\mathtt{N}) = \langle \Psi, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$. By induction hypothesis there is s satisfying $\mathcal{G}, \mathcal{F}$ and $\mathcal{H}$ such that $\Theta \rhd s(\Psi) \vdash \mathtt{N} : s(U)$ and Σ is derivable from Θ by applying a sequence of rule $(w), (m)$ and (sp). So in particular we have $s(U) \equiv \forall\alpha.A$ and by an inspection of the rules it is easy to verify that $U \equiv [u].V$ for some V and fresh u. Moreover $A \equiv \forall\boldsymbol{\beta}.C$ for some C. Hence in particular $s = s'[u \mapsto \alpha\boldsymbol{\beta}]$ for some substitution s'. Let $a_1, \ldots, a_n$ be such that $s'(a_i) = C_i[\alpha]$, where $C_i[\alpha]$ denotes a type C_i, where α occurs free $(1 \leq i \leq n)$. Then $s_1 = s'[u \mapsto \boldsymbol{\beta}, a_1 \mapsto C_1[B], \ldots, a_n \mapsto C_n[B]]$ and s_1 does the intended work, since the Hygiene Condition. Moreover since u is fresh it is easy to verify that s_1 satisfies $\mathcal{G}, \mathcal{F}$ and $\mathcal{H}$. □

5.4 Examples

Example 21

1. It is easy to verify that $\mathbf{\Pi}(\lambda\mathtt{x}.\mathtt{xx}) = \langle \emptyset, U, \mathcal{G}, \mathcal{F}, \mathcal{H} \rangle$ where

$$\begin{array}{l} U = [w].!^r([t_1].a_1) \multimap [v].c \qquad \mathcal{G} = \{([t_1].a_1, b_1), ([t_2].a_2, b_2), ([t].a, c)\} \\ \mathcal{F} = \{u_1 \mapsto \{[t_1].a_1\}, u_2 \mapsto \{[t_2].a_2\}, v \mapsto \{[t_1].a_1, [t_2].a_2\}\} \\ \mathcal{H} = \langle \{b_1 =!^q([u_2].b_2) \multimap [t].a, a_1 = a_2\}, \{u_1 = \epsilon, t_1 = t_2\}, \\ \qquad \{p_1 = p_3, p_3 \geq p_2 + q, r > p_1, r > p_3\} \rangle \end{array}$$

   The substitution $s = s'[a_1 \mapsto \alpha, a_2 \mapsto \alpha, b_1 \mapsto (\forall\beta.\beta) \multimap \gamma, b_2 \mapsto \beta, c \mapsto \gamma, a \mapsto \gamma, t_1 \mapsto \alpha, t_2 \mapsto \alpha, u_1 \mapsto \epsilon, u_2 \mapsto \beta, v \mapsto \gamma, w \mapsto \epsilon, t \mapsto \epsilon, p_1 \mapsto 0, p_2 \mapsto 0, p_3 \mapsto 0, q \mapsto 0, r \mapsto 1]$ satisfies $\mathcal{G}$, $\mathcal{F}$ and $\mathcal{H}$. Hence the term is typable.
2. It is boring but easy to obtain the constraints in $\mathbf{\Pi}((\lambda\mathtt{x}.\mathtt{xx})\mathtt{2}) = \langle \emptyset, U, \mathcal{G}, \mathcal{F}, \langle \mathcal{P}, \mathcal{C}, \mathcal{Q} \rangle \rangle$. Making the substitutions in $\mathcal{P}$ and $\mathcal{Q}$ we obtain

$U = [z].d \qquad \mathcal{G} = \{([t_2].a_2, c), ([t_z].a_z, b_z), ([t_{sz}].a_{sz}, b_{sz}), ([t_{s^2z}].a_{s^2z}, b_{s^2z}),$
$([t_s].a_s, !^{p_{sz}}([u_z].b_z) \multimap [t_{sz}].a_{sz}), ([z_1].c, d),$
$([t_1].(!^s([t_s].a_s) \multimap ([v].!^r([t_z].a_z) \multimap [u_{s^2z}].b_{s^2z})), b_2),$ (1)
$([t_1].(!^s([t_s].a_s) \multimap ([v].!^r([t_z].a_z) \multimap [u_{s^2z}].b_{s^2z})), !^{q_1}([u_2].b_2) \multimap [t_2].a_2),$ (2)
$([t_s].a_s, !^{p_2}([u_{sz}].b_{sz}) \multimap [t_{s^2z}].a_{s^2z})$ (3)$\}$
$\mathcal{F} = \{u_1, u_2, z_1 \mapsto \{[t_1].!^s([t_s].a_s) \multimap ([v].!^r([t_z].a_z) \multimap [u_{s^2z}].b_{s^2z})\},$
$u_z \mapsto \{[t_z].a_z\}, u_{sz}, u_{s^2z} \mapsto \{[t_z].a_z, [t_s].a_s\}, v \mapsto \{[t_s].a_s\}\}$
$\mathcal{C} = \{r_1 > q_1, r \geq p_{sz} + p_2, s > p_2\}$

The equation (1) implies that b_2 is of the shape

$$!^{s_1}[w_1].b_2^1 \multimap [w_2].!^{s_2}[w_2']b_2^2 \multimap [w_3].b_2^3$$

Moreover it implies that each substitution s satisfying the constraints must be such that $s(s_1) = s(s), s(s_2) = s(r)$ while equation (2) implies $s(s) = s(q_1)$. Remembering that $\mathcal{G}$ is a semi-unification set, equations (1), (2) and (3) imply that $s(t_s) = \epsilon$ and $s(a_s) =!^{s(p_2)}A \multimap B$. Substituting this in equation (2) we have $s(s) = s(p_2)$ but this is in contrast with the constraints in $\mathcal{H}$. Note that this term is typable in System F.

3. Note that the term $\mathtt{2}(\mathtt{yz})$ of Example 10.3 is also typable in the full STA system and in System F but it is still not typable in DLAL due again to the presence of the two free variables.

6 Conclusion

We proved that the type inference problem for STA is decidable in polynomial time in the length of the input term if we restrict ourselves to consider just the propositional fragment. For the whole system we conjecture that the problem is undecidable since the presence of the second order quantifier. Nevertheless we showed an algorithm generating all the constraints that need to be satisfied in order to type a given term. It would be possible to follow the same method as in [4] for System F. Namely, for every $n \in \mathbb{N}$ we can define a bounded type containment relation $\leq_{\mathcal{T}}^n$ such that $\forall \boldsymbol{a}.A \leq_{\mathcal{T}}^n C$ if and only if $C \equiv A[\boldsymbol{B}/\boldsymbol{\alpha}]$ and the variables in $\boldsymbol{\alpha}$ occur in the syntax tree of A at a depth less or equal to n.

Then, we can define a countable set of type assignment systems STA^n which is a complete stratification of the system STA. For each $n \in \mathbb{N}$, the system STA^n is obtained by replacing the $(\forall E)$ rule in Table 1 by the following rule:

$$\frac{\Gamma \vdash \mathtt{M} : A \quad A \leq_{\mathcal{T}}^n B}{\Gamma \vdash \mathtt{M} : B} \ (n\text{-}\forall E)$$

In every STA^n the type inference problem is decidable. We leave the checking of the undecidability of the conjecture and the design of the stratified system for future investigations.

References

1. Gaboardi, M., Ronchi Della Rocca, S.: A soft type assignment system for λ-calculus. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 253–267. Springer, Heidelberg (2007)
2. Lafont, Y.: Soft linear logic and polynomial time. Theoretical Computer Science 318(1-2), 163–180 (2004)
3. Coppo, M., Dezani-Ciancaglini, M., Venneri, B.: Principal type schemes and lambda-calculus semantics. In: Seldin, J.P., Hindley, J.R. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pp. 535–560. Academic Press, Inc., New York (1980)
4. Giannini, P., Ronchi Della Rocca, S.: A type inference algorithm for a stratified polymorphic type discipline. Information and Computation 109(1/2), 115–173 (1994)
5. Coppola, P., Dal Lago, U., Ronchi Della Rocca, S.: Elementary affine logic and the call by value lambda calculus. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 131–145. Springer, Heidelberg (2005)
6. Wells, J.B.: Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In: Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science (LICS 1994), pp. 176–185. IEEE Computer Society, Los Alamitos (1994)
7. Baillot, P., Terui, K.: Light types for polynomial time computation in lambda-calculus. In: Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2004), pp. 266–275. IEEE Computer Society, Los Alamitos (2004)
8. Asperti, A.: Light affine logic. In: Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science (LICS 1998), pp. 300–308. IEEE Computer Society, Los Alamitos (1998)
9. Gaboardi, M.: Linearity: an Analytic Tool in the study of Complexity and Semantics of Programming Languages. PhD thesis, Università degli Studi di Torino - Institut National Polytechnique de Lorraine (2007)
10. Robinson, J.A.: Machine-oriented logic based on resolution principle. Journal of the ACM 12, 23–41 (1965)
11. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
12. Baillot, P., Terui, K.: A feasible algorithm for typing in elementary affine logic. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 55–70. Springer, Heidelberg (2005)
13. Kfoury, A.J., Ronchi Della Rocca, S., Tiuryn, J., Urzyczyn, P.: Alpha-conversion and typability. Information and Computation 150(1), 1–21 (1999)