On Type Inference

in the

Intersection Type Discipline

Gérard Boudol and Pascal Zimmer

INRIA Sophia Antipolis
BP 93 – 06902 Sophia Antipolis Cedex, France

Abstract

We introduce a new unification procedure for the type inference problem in the intersection type discipline. We show that unification exactly corresponds to reduction in an extended λ-calculus, where one never erases arguments that would be discarded by ordinary β-reduction. We show that our notion of unification allows us to compute a principal typing for any strongly normalizing λ-expression.

1. Introduction

Type inference – say, for any λ-calculus based model –, as it is now presented in textbooks (see for instance [18], p. 136), generally proceeds as follows:

1. Assign a type to the expression and each subexpression. For any compound expression or variable, use a type variable.
2. Generate a set of constraints on types, reflecting the fact that, if a function is applied to an argument, then the type of the argument must agree with the type of the domain of the function.
3. Solve these constraints.

This design of a type inference algorithm was first (as far as we can see) proposed by J. Morris in his thesis [20]. At the first step of this procedure, a decision has to be taken, in order to build the type of a function, that is an abstraction $\lambda x M$: in which way do we consider the collection of type variables $t_1, \ldots, t_m$ assigned to the various occurrences of x in M as a type? There are various possibilities, which are not unrelated:

- Simple (monomorphic, possibly recursive) types: a variable x has only one type. That is, one has the constraint that the t_i's are equal (with or without "occur check").
- Generalized (polymorphic) types: the constraint here is that x is only used in M with types which are *instances* of the type of the domain of $\lambda x M$.
- Intersection types: the collection $t_1, \ldots, t_m$ is considered as a type, interpreted as the *conjunction* of the t_i's.

- Subtyping: x is only used in M with types which are *subtypes* of the type of the domain of $\lambda x M$.

(The question, and thus the possible answers, would be different regarding the "let" construct, that is $(\lambda x M N)$, where the abstraction $\lambda x M$ does not have to be explicitly typed, see for instance [11, 17].)

In this paper we are interested in type inference for the intersection type discipline, introduced by Coppo and Dezani [8], and independently by Pottinger [21] (see [2, 4] for a complete review of various systems with intersection types). There is no algorithm for deciding typability in this system, called "system $\mathcal{D}$" in [16], since this is equivalent to strong normalizability. However, one can compute a *principal* typing for any typable expression [9, 16, 24], that is a typing from which any other typing for the given expression can be derived, by means of suitable operations, among which the most important one is *expansion* (for an explanation of this notion, see for instance [3]). Type inference can be achieved by normalizing the expression, and then typing the normal form, but obviously this cannot be extended to a language where one may wish to type non-terminating programs. Ronchi proposed in [23] a direct procedure, based on a generalized unification mechanism. This was later revisited by Kfoury [13], and then Kfoury and Wells [14], who used explicit *expansion variables*, in order to provide a better understanding of the operation of expansion, and showed that type inference is decidable for subsystems with a bounded rank restriction.

In this paper we introduce a new way of solving the typing constraints that arise from type inference for intersection types. To give an idea of our generalized unification procedure, let us recall that the constraints to solve which are attached to an application node (MN) have the form

$$(\tau \to t) = \sigma$$

where t is the type variable assigned (at step 1) to the node, τ is the type of the argument N, and σ is the type of M. When M is a function $\lambda x M'$ – that is, when the application is a *redex* –, the latter has the form $(t_1, \ldots, t_m \to \theta)$ where $t_1, \ldots, t_m$ is the conjunction of the type variables assigned to x in M', and θ is the type of M'. Then, mimicking the β-reduction of $(\lambda x M' N)$ into $\{x \mapsto N\}M'$, our generalized unification procedure identifies t with θ, makes m distinct copies of the constraints associated with the argument N, and identifies the t_i's with the appropriate copy of the type τ of N. This, however, is not correct in the case of a β_K-redex, where $m = 0$, since we could then miss to check that some subterms are typable. For instance, it would be wrong to declare that the expression $(\lambda u.\mathsf{F}(uu))\Delta$, where $\mathsf{F} = \lambda x \lambda y\, y$ and $\Delta = \lambda z(zz)$ is typable (in system $\mathcal{D}$). We should rather keep in any case a copy of the constraints associated with the argument in a redex, instead of removing them, as it happens with β-reduction. Then we will show that our unification procedure exactly corresponds to reduction in an *extended* λ-calculus, where one never erases subexpressions that would be discarded by ordinary β-reduction. This calculus, which builds upon Klop's one [15], was introduced in [5]. It uses in particular Klop's construction $[M, N]$ where N is a "discarded" expression. For instance, the expression above reduces to $(\lambda u[\lambda y\, y, (uu)]\Delta)$ in this calculus. In order to perform the appropiate expansions in solving type equations, we shall keep, associated with each equation $(\tau \to t) = \sigma$ corresponding to an application (MN), its *territory*, which is the set of type variables assigned to subexpressions of the argument N. This is generally not directly accessible form the set of equations, because in an expression $[M, N]$, the constraints associated with N are disconnected from those of M.

Our semi-algorithm for type inference has been implemented by the second author of this paper, see [25]. For any λ-expression, it computes, when it exists, its principal typing, in the sense of [9, 16, 24]; more precisely, it computes a proof of the principal typing. Like the one of Kfoury and Wells [14], our semi-algorithm terminates when restricted to types of a bounded rank. Although it is, to our view, simpler, and thus easier to prove correct, it is not less (nor more) complex than Kfoury and Wells' one: indeed, it is shown in [19] that the type inference for system I of [14] is intrinsically as complex as strong normalization. A similar result holds for our type inference procedure, although we do not have to resort to sharing graphs and proof nets, as in [19], to establish a direct correspondence between β-reduction – or more accurately κ-reduction, see below – and the reduction of typing constraints.

When this paper was about to be finished, we became aware of [7], which presents a seemingly similar result. Indeed, a main result of [7] is that the type inference process proposed in that paper corresponds to β-reduction. However, there are two major differences with the results presented here: first, unlike [7], we do not use the notion of an "expansion variable" that was introduced in [13] (see also [14]). Second, [7] deals with an extended intersection type discipline where the type ω, introduced by Sallé in [22], is assigned to any term (this is called "system $\mathcal{D}\Omega$" in [16]), thus making the typability problem trivial, whereas we deal with "system $\mathcal{D}$" where only strongly normalizing λ-terms are typable. Clearly, there cannot exist a correspondence between β-reduction and type inference in system $\mathcal{D}$, because of β_K-redexes, and this is why we use a variant of Klop's calculus instead. We also notice that, although a notion of principal typing does exist for "approximate normal forms" in system $\mathcal{D}\Omega$ (see [9, 24]), such a notion does not seem to exist for all (typable) expressions, and especially for λ-terms with an infinite Böhm tree, like for instance the fixpoint combinator $\Delta(\lambda y\lambda f.f(yyf))$. This contrasts with system $\mathcal{D}$, where a principal typing exists for any typable expression.

2. The Extended λ-Calculus

Our extended λ-calculus is basically Klop's one [15], with some differences that are explained in [5]. The syntax is as follows:

$$M,\ N\ldots\ ::=\ x\ \mid\ \lambda xM\ \mid\ (MN)\ \mid\ [M,N]$$

In the new construction $[M,N]$ which is added to the λ-calculus primitives, M is the main expression, and N is an expression that is discarded when interpreting $[M,N]$ as an ordinary λ-expression. The operation of (capture avoiding) substitution, denoted $\{x\mapsto N\}M$, is defined as usual. To define our notion of *reduction* κ, we allow n to be 0 in an expression $[\cdots[M,N_1]\cdots,N_n]$, in which case this denotes M. We abbreviate this as $[M,N_1,\ldots,N_n]$, and sometimes even $[M,\ldots]$. The reduction $\underset{\kappa}{\rightarrow}$ is then given by the two following axiom schemas:

$$\frac{x\in\mathsf{fv}(M)}{([\lambda xM,\ldots]N)\underset{\kappa}{\rightarrow}[\{x\mapsto N\}M,\ldots]}\qquad\qquad\frac{x\notin\mathsf{fv}(M)}{([\lambda xM,\ldots]N)\underset{\kappa}{\rightarrow}[M,\ldots,N]}$$

It has beeen shown in [5] that in this calculus, strong and weak normalization coincide.

Example. *To illustrate the various notions introduced in our paper, we shall use the λ-term $\mathsf{F}(\lambda u.\Delta(uu))$, that is $(\lambda x\lambda yy\lambda u(\lambda z(zz)(uu))$. For this expression, we have the following reductions:*

$$\begin{array}{rcl}\mathsf{F}(\lambda u.\Delta(uu)) & \underset{\kappa}{\rightarrow} & [\lambda yy, \lambda u.\Delta(uu)] \\ \mathsf{F}(\lambda u.\Delta(uu)) & \underset{\kappa}{\rightarrow} & \mathsf{F}(\lambda u.(uu)(uu))\end{array}$$

Now let us introduce the intersection type system for this calculus. It is convenient to consider types where conjunction does not occur on the right of an arrow (this is not a serious restriction, see [4, 5] for instance). That is, the syntax of types is as follows – where t is any type variable:

$$\begin{array}{rcll}\tau,\ \sigma\ldots & ::= & t \ \mid\ (\pi \to \sigma) & \textit{prime types} \\ \pi,\ \kappa\ldots & ::= & \omega \ \mid\ \tau \ \mid\ (\pi \wedge \kappa) & \textit{types}\end{array}$$

In the type system, we consider types modulo the congruence $\equiv_{\mathrm{UACI}}$ generated by the following equations:

$$(\omega \wedge \pi) = \pi \tag{U}$$

$$((\pi_0 \wedge \pi_1) \wedge \pi_2) = (\pi_0 \wedge (\pi_1 \wedge \pi_2)) \tag{A}$$

$$(\pi_0 \wedge \pi_1) = (\pi_1 \wedge \pi_0) \tag{C}$$

$$(\pi \wedge \pi) = \pi \tag{I}$$

Indeed, we shall most often write prime types as $(\tau_1, \ldots, \tau_n \to \sigma)$ where the order in the sequences $\tau_1, \ldots, \tau_n$ is irrelevant (that is, the sequence $\tau_1, \ldots, \tau_n$ stands for an arbitrary conjunctive combination of these types, and it denotes ω when $n = 0$). We have included the idempotency property (I) mainly for completeness, that is, more precisely, to ensure that the intersection type system we use is a conservative extension of the standard system of simple types (where sequences are restricted to contain only one element). However, it should be pointed out that this idempotency property will not be used in any technical development.

The judgements of the type system have the form $\Gamma \vdash M : \tau$, where Γ is, as usual, a typing context, assigning types to a finite number of λ-variables. We denote by $\Gamma \wedge \Delta$ the conjunction of Γ and Δ, which is defined in the obvious way (that is, pointwise, assuming that $\Gamma(x)$ is ω for any x not in the domain of Γ). The congruence $\equiv_{\mathrm{UACI}}$ is extended pointwise to typing contexts, that is, $\Gamma \equiv_{\mathrm{UACI}} \Delta$ means $\Gamma(x) \equiv_{\mathrm{UACI}} \Delta(x)$ for any x. The rules of the type system are as follows:

$$\frac{}{x : \tau \vdash x : \tau} \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma(x) = \pi}{\Gamma\backslash x \vdash \lambda x M : (\pi \to \sigma)} \qquad \frac{\Gamma \vdash M : \sigma \quad \Delta \vdash N : \tau}{\Gamma \wedge \Delta \vdash [M, N] : \sigma}$$

where $\Gamma\backslash x$ denotes the typing context obtained from Γ by removing the typing assumption about x, if any.

$$\frac{\Gamma \vdash M : (\tau_1, \ldots, \tau_m \to \sigma) \quad \forall i.\Delta_i \vdash N : \tau_i}{\Gamma \wedge \Delta_1 \wedge \cdots \wedge \Delta_m \vdash (MN) : \sigma}\ m > 0 \qquad \frac{\Gamma \vdash M : (\omega \to \sigma) \quad \Delta \vdash N : \tau}{\Gamma \wedge \Delta \vdash (MN) : \sigma}$$

$$\frac{\Gamma \vdash M : \tau \quad \Delta \equiv_{\text{UACI}} \Gamma}{\Delta \vdash M : \tau}$$

For instance, we have:

$$\frac{\dfrac{}{z : \tau_0 \to \tau_1 \vdash z : \tau_0 \to \tau_1} \qquad \dfrac{}{z : \tau_0 \vdash z : \tau_0}}{\dfrac{z : (\tau_0 \to \tau_1) \wedge \tau_0 \vdash (zz) : \tau_1}{\vdash \Delta : ((\tau_0 \to \tau_1) \wedge \tau_0) \to \tau_1}}$$

Example (continued). *The expression* $\mathsf{F}(\lambda u.\Delta(uu))$ *is typable, with type* $\tau \to \tau$, *since* F *is typable, with type* $\sigma \to \tau \to \tau$, *and* $\lambda u.\Delta(uu)$ *is typable, with type*

$$\sigma = (\theta \to \tau_0 \to \tau_1), (\theta \to \tau_0), \theta \to \tau_1$$

We can easily extend the classical result that β-normal forms are typable in such a system (see [8, 16]). To see this, let us first observe that the set $\mathcal{N}$ of *normal forms* (that is, κ-irreducible expressions) P, $Q\ldots$ of our extended λ-calculus is given by the following grammar:

$$\begin{array}{rcl} P,\ Q\ldots & ::= & H \mid \lambda x P \mid [P, Q] \\ H & ::= & x \mid (HP) \mid [H, P] \end{array}$$

We denote by $\mathsf{hv}(H)$ the *head variable* of H, that is

$$\mathsf{hv}(x) = x \qquad \text{and} \qquad \mathsf{hv}(HP) = \mathsf{hv}([H, P]) = \mathsf{hv}(H)$$

Then we define, up to the renaming of type variables, the *canonical typing* of P. This is a pair of a typing context and a type, written $\Gamma \vdash \tau$, inductively given as follows, observing that, if P is an H with head variable x, this has the shape

$$\{x : \tau_1 \to \cdots \tau_n \to t\} \wedge \Gamma' \vdash t$$

where t does not occur in Γ'.

i. $x : t \vdash t$ is the canonical typing of x, where t is any type variable;

ii. if $(\{x : \tau_1 \to \cdots \tau_n \to t\} \wedge \Gamma) \vdash t$ is the canonical typing of H and $\Delta \vdash \tau$ is the canonical typing of P, involving disjoint sets of type variables, then

$$\{x : \tau_1 \to \cdots \tau_n \to \tau \to t\} \wedge \Gamma \wedge \Delta \vdash t$$

is the canonical typing of (HP);

iii. if $\Gamma \vdash \tau$ is the canonical typing of P and $\Gamma(x) = \pi$ (with $\pi = \omega$ if $x \notin \mathsf{dom}(\Gamma)$), then $\Gamma \backslash x \vdash \pi \to \tau$ is the canonical typing of $\lambda x P$;

iv. if $\Gamma \vdash \tau$ and $\Delta \vdash \sigma$ are respectively the canonical typings of P and Q, involving disjoint sets of type variables, then $\Gamma \wedge \Delta \vdash \tau$ is the canonical typing of $[P, Q]$.

It is easy to check that the canonical typing $\Gamma \vdash \tau$ of a normal form P is indeed a valid typing, that is, $\Gamma \vdash P : \tau$ is provable. We also recall that it has been shown (see [9, 24]) that, for any λ-expression M having a β-normal form N, the canonical typing of N is a *principal typing* for M, in the sense that it is a valid typing for M, from which any other typing can be derived, by means of suitable operations. To conclude this section, we notice that one can extend the classical result relating typability and strong normalization:

THEOREM 2.1. *In the extended λ-calculus, an expression is typable if and only if it is strongly normalizable.*

The fact that typability implies strong normalization was established in [5] (the "subject reduction" property only uses AC). Conversely, it is not difficult to check (proving a "subject expansion" property, along the lines given in [1] for instance) that a strongly normalizing expression of the extended λ-calculus is typable.

3. Typing Constraints

In this section we define the constraints that are associated with an expression, in order to perform type inference. The constraints are, as usual, type equations, but they involve types of a restricted shape, that we call *skeletal*. A skeletal type is either a type variable, or a type of the form $(t_1, \ldots, t_m \to \xi)$ where ξ is skeletal (and the t_i's are type variables). The syntax is as follows :

$$\begin{array}{lcll} \xi,\ \zeta \ldots & ::= & t \mid (\phi \to \xi) & \textit{skeletal types} \\ \phi,\ \psi \ldots & ::= & \omega \mid t \mid (\phi \wedge \psi) & \end{array}$$

As we said in the introduction, a first phase of the type inference process consists in assigning types to the expression to type, and its subexpression, assigning (distinct) type variables to compound expressions (MN) and (occurrences of) λ-variables. That is, we start with *annotated expressions* A, $B \ldots$ defined as follows. We simultaneously define the set $\mathcal{A}$ of annotated expressions, together with the set $\mathsf{tyvar}(A)$ of type variables occurring in A. The set $\mathcal{A}$ is inductively defined by:

i. for each λ-variable x and each type variable t, the expression x^t is in $\mathcal{A}$, and $\mathsf{tyvar}(x^t) = \{t\}$;
ii. if $A \in \mathcal{A}$ then $\lambda x A \in \mathcal{A}$ and $\mathsf{tyvar}(\lambda x A) = \mathsf{tyvar}(A)$;
iii. if $A \in \mathcal{A}$ and $B \in \mathcal{A}$ with $\mathsf{tyvar}(A) \cap \mathsf{tyvar}(B) = \emptyset$, and t is a type variable not in $\mathsf{tyvar}(A) \cup \mathsf{tyvar}(B)$ then $(AB)^t \in \mathcal{A}$ and $[A, B] \in \mathcal{A}$ with $\mathsf{tyvar}((AB)^t) = \{t\} \cup \mathsf{tyvar}(A) \cup \mathsf{tyvar}(B)$ and $\mathsf{tyvar}([A, B]) = \mathsf{tyvar}(A) \cup \mathsf{tyvar}(B)$.

EXAMPLE (CONTINUED). *An annotated version of $\mathsf{F}(\lambda u.\Delta(uu))$ is, underlining the type variables corresponding to an application node:*

$$(\lambda x \lambda y y^{t_0} \lambda u (\lambda z (z^{t_1} z^{t_2})^{\underline{t_3}} (u^{t_4} u^{t_5})^{\underline{t_6}})^{\underline{t_7}})^{\underline{t_8}}$$

We define various functions over annotated terms: first, erase is the function that erases the type annotations, producing an expression of the extended λ-calculus from an annotated expression (the definition is obvious). Then typ associates a (skeletal) type with an annotated expression. This is defined as follows, using auxiliary functions Γ_A which, given an annotated expression A, associate a (ϕ) type (that is, a sequence of type variables) with each λ-variable:

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{typ}(x^t) & = & t & \Gamma_{x^t}(y) & = & \begin{cases} t & \text{if } y = x \\ \omega & \text{otherwise} \end{cases} \\[2ex] \mathsf{typ}(\lambda x A) & = & (\Gamma_A(x) \to \mathsf{typ}(A)) & \Gamma_{\lambda x A}(y) & = & \begin{cases} \omega & \text{if } y = x \\ \Gamma_A(y) & \text{otherwise} \end{cases} \\[2ex] \mathsf{typ}((AB)^t) & = & t & \Gamma_{(AB)^t} & = & (\Gamma_A \wedge \Gamma_B) \\ \mathsf{typ}([A, B]) & = & \mathsf{typ}(A) & \Gamma_{[A,B]} & = & (\Gamma_A \wedge \Gamma_B) \end{array}$$

As the notation suggests, in what follows we also consider Γ_A as a typing context associated with A. For instance, we have

$$\mathsf{typ}(\lambda z(z^{t_1}z^{t_2})^{t_3}) = t_1, t_2 \rightarrow t_3$$

With an annotated expression A we finally associate a set of *constraints* to solve in order to type $\mathsf{erase}(A)$. These are, as usual, *type equations* $\mathsf{typ}(A_1) \rightarrow t = \mathsf{typ}(A_0)$ attached to application nodes $(A_0A_1)^t$ in the expression, except that we have to record also the *territory* of the equation, which is the set of type variables that have to be duplicated when the equation is reduced, namely $\mathsf{tyvar}(A_1)$ (1). Then the constraints have the form $(\tau \overset{\perp}{=} \sigma; T)$ where T is a set of type variables. We write $\tau \overset{\perp}{=} \sigma$, instead of $\tau = \sigma$, to remind that the left (resp. right) member of an equation should be considered as *negative* (resp. *positive*), see [6, 12]. The set $\mathcal{E}_A$ of constraints associated with A is defined inductively as follows:

$$\begin{array}{rcl}
\mathcal{E}_{x^t} & = & \emptyset \\
\mathcal{E}_{\lambda x A} & = & \mathcal{E}_A \\
\mathcal{E}_{(AB)^t} & = & \{(\mathsf{typ}(B) \rightarrow t \overset{\perp}{=} \mathsf{typ}(A); \mathsf{tyvar}(B))\} \cup \mathcal{E}_A \cup \mathcal{E}_B \\
\mathcal{E}_{[A,B]} & = & \mathcal{E}_A \cup \mathcal{E}_B
\end{array}$$

Example (continued). *Associated with the annotated version*

$$(\lambda x \lambda y y^{t_0} \lambda u(\lambda z(z^{t_1}z^{t_2})^{\underline{t_3}}(u^{t_4}u^{t_5})^{\underline{t_6}})^{\underline{t_7}})^{\underline{t_8}}$$

of $\mathbf{F}(\lambda u.\Delta(uu))$*, we get the following set of constraints:*

$$\begin{array}{rcl}
\mathcal{E}_0 & = & \{\, (t_4, t_5 \rightarrow t_7) \rightarrow t_8 \overset{\perp}{=} \omega \rightarrow (t_0 \rightarrow t_0)\,;\, \{t_1, \ldots, t_7\}, \\
& & t_6 \rightarrow t_7 \overset{\perp}{=} t_1, t_2 \rightarrow t_3\,;\, \{t_4, t_5, t_6\}, \\
& & t_5 \rightarrow t_6 \overset{\perp}{=} t_4\,;\, \{t_5\}, \\
& & t_2 \rightarrow t_3 \overset{\perp}{=} t_1\,;\, \{t_2\}\,\}
\end{array}$$

There are two kinds of equations in $\mathcal{E}_A$: those of the form $(\xi \rightarrow t) \overset{\perp}{=} t'$ correspond to application nodes in A where the function is a λ-variable or an application, while equations of the shape $(\xi \rightarrow t) \overset{\perp}{=} (t_1, \ldots, t_n \rightarrow \zeta)$ correspond to application nodes where the function to apply is a λ-abstraction, that is to redexes, of the form $([\lambda x A', \ldots]B)^t$. It is worth observing that, given the polarities assigned to the members of an equation, and the fact that in a typing $\Gamma \vdash \tau$ the types in the image of Γ are *negative*, whereas τ is *positive*, we have:

Remark 3.1.

(i) *Each type variable t assigned to an application node has exactly one negative occurrence in an equation of $\mathcal{E}_A$, namely in the equation $(\xi \rightarrow t) \overset{\perp}{=} \zeta$ associated with the node. Moreover, it has at most one positive occurrence, either in $\mathcal{E}_A$, if the application node is a subexpression of another application, or in* $\mathsf{typ}(A)$.

(ii) *Each type variable assigned to a λ-variable x has exactly one negative occurrence, either in $\mathcal{E}_A$, if x is bound by a λ-abstraction which is a subexpression of an application, or in Γ_A, and at most one positive occurrence, in $\mathcal{E}_A$, if x is a subexpression of an application, or in* $\mathsf{typ}(A)$.

(1) Without the construction $[M, N]$, this could be derived from the equations, starting from the set of type variables in $\mathsf{typ}(A_1)$, and including in the territory all the type variables occurring in an equation $\xi \rightarrow t' = \zeta$ whose root t' is already in the territory.

This is in fact an invariant that will be preserved in solving the constraints.

4. $\wedge$-Unification

To solve a set of constraints, we will reduce them, by means of a generalized unification mechanism, which involves the notion of type substitution, that we introduce now. Since the constraints to reduce only involve skeletal types, we shall only consider applying substitutions to this restricted kind of types. A *prime type substitution* is a map S from a finite set $\mathsf{dom}(\mathsf{S})$ of type variables to prime types. If $\mathsf{dom}(\mathsf{S}) = \{t_1, \ldots, t_n\}$ and $\mathsf{S}(t_i) = \tau_i$, we also denote S by $\{t_1 \mapsto \tau_1, \ldots, t_n \mapsto \tau_n\}$. We let $\mathsf{S}(t) = t$ for $t \notin \mathsf{dom}(\mathsf{S})$. The result of applying the substitution S to a (skeletal) type ξ is denoted $\mathsf{S}\xi$ (the definition, by induction on the structure of τ, is the usual one). As a matter of fact, we shall only use $\mathsf{S}\xi$ in the case where S is a *renaming*, assigning (distinct) type variables to type variables. However, we shall also use the application of a substitution S to positive occurrences of type variables in a skeletal type. The resulting type is denoted $\mathsf{S}^+\xi$. Since there is exactly one positive occurrence of a type variable in ξ, the definition of $\mathsf{S}^+\xi$ should be obvious: if $\xi = (\phi_1 \to \cdots (\phi_n \to t) \cdots)$ then $\mathsf{S}^+\xi$ is$(\phi_1 \to \cdots (\phi_n \to \mathsf{S}(t)) \cdots)$. Notice that this is a skeletal type if $\mathsf{S}(t)$ is skeletal. These positive applications of type substitutions are extended, according to the polarities we suggested above (negative on the left, positive on the right) to type equations, as follows:

$$\mathsf{S}^+(\xi \to t \doteq \zeta) \;=\; \mathsf{S}^+\xi \to t \doteq \mathsf{S}^+\zeta$$

Finally, we shall also consider substitutions that assign types (not necessarily prime) to type variables. Obviously, these should only be applied on the left of the arrow, that is, since we are only considering applications to skeletal types, to negative occurrences of type variables. Then, given such a mapping D from a finite set $\mathsf{dom}(\mathsf{D})$ of type variables to types, extended with $\mathsf{D}(t) = t$ for $t \notin \mathsf{dom}(\mathsf{D})$, we define $\mathsf{D}^-\xi$, the result of applying D to the (skeletal) type ξ, and $\mathsf{D}^+\phi$, the type obtained by applying D to ϕ, as follows:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\mathsf{D}^-t & = & t & \mathsf{D}^+\omega & = & \omega \\
\mathsf{D}^-(\phi \to \xi) & = & (\mathsf{D}^+\phi \to \mathsf{D}^-\xi) & \mathsf{D}^+t & = & \mathsf{D}(t) \\
 & & & \mathsf{D}^+(\phi \wedge \psi) & = & (\mathsf{D}^+\phi \wedge \mathsf{D}^+\psi)
\end{array}$$

As a matter of fact, we shall only use this in the case where D is a *duplication*, assigning a conjunction of distinct type variables to type variables. Notice that in this case $\mathsf{D}^-\xi$ is a skeletal type. Again, we extend this to equations, but only when the root of the equation is not affected by D. That is, if $t \notin \mathsf{dom}(\mathsf{D})$, we let:

$$\mathsf{D}^-(\xi \to t \doteq \zeta) \;=\; \mathsf{D}^-\xi \to t \doteq \mathsf{D}^-\zeta$$

Besides type substitutions, we shall also need, in order to solve the constraints, to apply some transformations on the territory of the equations. These are determined by mappings U from a finite set of type variables to finite sets of type variables, which we denote $\mathsf{U} = \{t_1 \mapsto U_1, \ldots, t_n \mapsto U_n\}$. Assuming that, by convention, $\mathsf{U}(t) = \{t\}$ if $t \notin \mathsf{dom}(\mathsf{U})$, these are applied to sets of type variables as follows:

$$\mathsf{U}(T) = \bigcup_{t \in T} \mathsf{U}(t)$$

Finally, identifying a pair of functions with a function returning pairs, we shall use transformations of the form

$$\{t_1 \mapsto (\tau_1; U_1), \ldots, t_n \mapsto (\tau_n; U_n)\}$$

(or $\{t_1 \mapsto (\pi_1; U_1), \ldots, t_n \mapsto (\pi_n; U_n)\}$) which acts as the type substitution $\{t_1 \mapsto \tau_1, \ldots, t_n \mapsto \tau_n\}$ (or, negatively, as $\{t_1 \mapsto \pi_1, \ldots, t_n \mapsto \pi_n\}$) on types and equations, and as $\{t_1 \mapsto U_1, \ldots, t_n \mapsto U_n\}$ on territories ([2]). Type substitutions and term substitutions are related as follows. If $\mathsf{tyvar}(A) \cap \mathsf{tyvar}(B) = \emptyset$, we denote by $\{x^t \mapsto B\}A$ the capture avoiding substitution of x^t by B in A. Then we have:

Lemma 4.1.

(i) $\mathsf{typ}(\{x^t \mapsto B\}A) = \{t \mapsto (\mathsf{typ}(B); \mathsf{tyvar}(B))\}^+ \mathsf{typ}(A)$

(ii) *if* x^t *occurs in* A *then* $\mathcal{E}_{\{x^t \mapsto B\}A} = \mathcal{E}_B \cup \{t \mapsto (\mathsf{typ}(B); \mathsf{tyvar}(B))\}^+ \mathcal{E}_A$.

Proof: by induction on A. ❏

Now we define the notion of *reduction* $\mathcal{E} \rhd \mathcal{E}'$ on sets of constraints. This closely mimicks, as we shall see, the κ-reduction on expressions. A constraint corresponds to a redex $([\lambda x A, \ldots] B)^t$ if it has the form

$$(\xi \to t) \overset{\perp}{=} (\phi \to \zeta); \mathsf{tyvar}(B)$$

where ξ is the type assigned to the argument B, t is the type of the application node, $\phi = t_1, \ldots, t_m$ is the sequence of types of the abstracted variable x in the function A, and ζ is the type of the function body A. As usual, to reduce such an equation, we have to unify t with ζ (reflecting the fact that A, where the substitution of B for x is performed, takes the place of the application node) and ξ with ϕ, but the latter cannot be solved in the usual way (that is, identifying the t_i's). By analogy with β-reduction, solving $\xi \overset{\perp}{=} t_1, \ldots, t_m$ should correspond to substituting the argument B to the m occurrences of the variable x in A. In order to obtain a well-formed annotated term, we have to make m distinct copies of B, annotated with fresh type variables (which are copies of the type variables in the territory $\mathsf{tyvar}(B)$ of the equation). However, we cannot simply replace $\xi \overset{\perp}{=} t_1, \ldots, t_m$ by $\xi^1 \overset{\perp}{=} t_1, \ldots, \xi^m \overset{\perp}{=} t_m$ where $\xi^1, \ldots, \xi^m$ are copies of ξ, because the type variables occurring in ξ may also occur elsewhere in the set of constraints.

Example (continued). *The reduction*

$$\mathsf{F}(\lambda u.\Delta(uu)) \underset{\kappa}{\to} \mathsf{F}(\lambda u.(uu)(uu))$$

should correspond to an "annotated reduction"

$$(\lambda x \lambda y y^{t_0} \lambda u(\lambda z(z^{t_1} z^{t_2})^{\underline{t_3}}(u^{t_4} u^{t_5})^{\underline{t_6}})^{\underline{t_7}})^{\underline{t_8}} \underset{\kappa}{\to} (\lambda x \lambda y y^{t_0} \lambda u((u^{t_4^1} u^{t_5^1})^{\underline{t_6^1}}(u^{t_4^2} u^{t_5^2})^{\underline{t_6^2}})^{\underline{t_3}})^{\underline{t_8}}$$

and to a decomposition of the equation $t_6 \to t_7 \overset{\perp}{=} t_1, t_2 \to t_3$, *with territory* $\{t_4, t_5, t_6\}$, *since it is the redex of type* t_7 *which is reduced. Therefore we should duplicate not only* t_6, *but also* t_4 *and* t_5, *which appear in the duplicated argument* $(u^{t_4} u^{t_5})^{\underline{t_6}}$. *These type variables also*

[2] As far as territories are concerned, the positive or negative application of such transformations is simply $\mathsf{U}(T)$, as defined above.

occur in other equations, namely $(t_4, t_5 \to t_7) \to t_8 \overset{\perp}{=} \omega \to (t_0 \to t_0)$ and $t_5 \to t_6 \overset{\perp}{=} t_4$. We see from the (annotated) κ-reduction that the latter should be simply duplicated, while in the former, we should replace the sequence t_4, t_5, corresponding to the abstraction λu, by $t_4^1, t_4^2, t_5^1, t_5^2$ (modulo the associativity and commutativity axioms AC), in order to obtain the set of constraints associated with $(\lambda x \lambda y y^{t_0} \lambda u((u^{t_4^1} u^{t_5^1})^{\underline{t_6^1}}(u^{t_4^2} u^{t_5^2})^{\underline{t_6^2}})^{\underline{t_3}})^{\underline{t_8}}$, that is

$$\begin{aligned}
\mathcal{E}_1 \quad = \quad \{\, & (t_4^1, t_5^1, t_4^2, t_5^2 \to t_3) \to t_8 \overset{\perp}{=} \omega \to (t_0 \to t_0)\,;\, \{t_3, t_4^1, t_4^2, t_5^1, t_5^2, t_6^1, t_6^2\},\\
& t_6^2 \to t_3 \overset{\perp}{=} t_6^1\,;\, \{t_4^2, t_5^2, t_6^2)\},\\
& t_5^1 \to t_6^1 \overset{\perp}{=} t_4^1\,;\, \{t_5^1\},\\
& t_5^2 \to t_6^2 \overset{\perp}{=} t_4^2\,;\, \{t_5^2\}\,\}
\end{aligned}$$

This is formalized in the following rule, defining the $\wedge$-unification process, which consists in a relation $\mathcal{E} \rhd \mathcal{E}'$ of reduction between sets of constraints. In the following definition, we explicitly record the transformation Θ used in the reduction, which is then denoted $\mathcal{E} \rhd \mathcal{E}'\ [\Theta]$. To state the definition, it is also convenient to introduce the following notations:

$$\begin{aligned}
\mathcal{E} \downarrow T &= \{\, (\xi \to t \overset{\perp}{=} \zeta; U) \mid (\xi \to t \overset{\perp}{=} \zeta; U) \in \mathcal{E} \ \&\ t \in T \,\}\\
\mathcal{E} \uparrow T &= \mathcal{E} - (\mathcal{E} \downarrow T)
\end{aligned}$$

One can see that, when T is the territory of some equation $\xi \to t \overset{\perp}{=} \zeta$ in $\mathcal{E}$, corresponding to an application node $(AB)^t$, then $\mathcal{E} \downarrow T$ is the set of constraints associated with the argument B, that is $\mathcal{E} \downarrow \mathsf{tyvar}(B) = \mathcal{E}_B$.

Definition ($\wedge$-Unification) 4.2. *Let $\mathcal{E}_0 = \{(\xi \to t \overset{\perp}{=} \phi \to \zeta; T)\} \cup \mathcal{E}$ be a set of constraints. Then $\mathcal{E}_0 \rhd \mathcal{E}_1\ [\Theta]$ where*

(i) if $\phi = \omega$ then $\Theta = \{t \mapsto (\zeta; \emptyset)\}^+$ and $\mathcal{E}_1 = \Theta(\mathcal{E})$;

(ii) if $\phi = t_1, \ldots, t_m$ with $m > 0$ then $\Theta = \mathsf{S}^+ \circ \{t \mapsto (\zeta; \emptyset)\}^+ \circ \mathsf{D}^-$ and

$$\mathcal{E}_1 = \mathsf{S}^+\{t \mapsto (\zeta; \emptyset)\}^+\big(\mathsf{D}^-(\mathcal{E} \uparrow T) \cup \bigcup_{1 \leqslant j \leqslant m} \mathsf{R}_j(\mathcal{E} \downarrow T)\big)$$

with, if $T = \{s_1, \ldots, s_n\}$,

$$\begin{aligned}
\mathsf{D} &= \{\, s_i \mapsto s_i^1, \ldots, s_i^m; \{s_i^1, \ldots, s_i^m\}) \mid 1 \leqslant i \leqslant n \,\}\\
\mathsf{R}_j &= \{s_1 \mapsto (s_1^j; \{s_1^j\}), \ldots, s_n \mapsto (s_n^j; \{s_n^j\})\} \qquad (1 \leqslant j \leqslant m)\\
\mathsf{S} &= \{t_1 \mapsto (\mathsf{R}_1\xi; \mathsf{R}_1 T), \ldots, t_m \mapsto (\mathsf{R}_m\xi; \mathsf{R}_m T)\}
\end{aligned}$$

where $s_1^1, \ldots, s_n^1, \ldots, s_1^m, \ldots, s_n^m$ are fresh (not occurring in $\mathcal{E}_0$), distinct type variables.

Example (continued). *Regarding our running example expression $\mathbf{F}(\lambda u.\Delta(uu))$, or more precisely its annotated version $(\lambda x \lambda y y^{t_0} \lambda u(\lambda z(z^{t_1} z^{t_2})^{\underline{t_3}}(u^{t_4} u^{t_5})^{\underline{t_6}})^{\underline{t_7}})^{\underline{t_8}}$, we see that if we select from the set $\mathcal{E}_0$ of associated constraints the equation $t_6 \to t_7 \overset{\perp}{=} t_1, t_2 \to t_3$ to reduce, with territory $\{t_4, t_5, t_6\}$, we obtain, using the notations of the definition (with $T = \{t_4, t_5, t_6\}$ and $\mathcal{E}_0 = \{t_6 \to t_7 \overset{\perp}{=} t_1, t_2 \to t_3; T\} \cup \mathcal{E}$), the following substitutions to apply:*

$$\begin{aligned}
\mathsf{D} &= \{\, t_i \mapsto t_i^1, t_i^2; \{t_i^1, t_i^2\}) \mid 4 \leqslant i \leqslant 6 \,\}\\
\mathsf{R}_1 &= \{t_4 \mapsto (t_4^1; \{t_4^1\}), t_5 \mapsto (t_5^1; \{t_5^1\}), t_6 \mapsto (t_6^1, \{t_6^1\})\}\\
\mathsf{R}_2 &= \{t_4 \mapsto (t_4^2; \{t_4^2\}), t_5 \mapsto (t_5^2; \{t_5^2\}), t_6 \mapsto (t_6^2; \{t_6^2\})\}\\
\mathsf{S} &= \{t_1 \mapsto (t_6^1, \{t_4^1, t_5^1, t_6^1\}), t_2 \mapsto (t_6^2, \{t_4^2, t_5^2, t_6^2\})\}
\end{aligned}$$

Since

$$\begin{array}{rcl} \mathcal{E}\downarrow T & = & \{\, t_5 \to t_6 \stackrel{\perp}{=} t_4\,;\, \{t_5\}\,\} \\ \mathcal{E}\uparrow T & = & \{\, (t_4, t_5 \to t_7) \to t_8 \stackrel{\perp}{=} \omega \to (t_0 \to t_0)\,;\, \{t_1, \ldots, t_7\}, \\ & & \quad t_2 \to t_3 \stackrel{\perp}{=} t_1\,;\, \{t_2\}\,\} \end{array}$$

where $\mathcal{E}\downarrow T$ is the set of constraints associated with the argument $(u^{t_4}u^{t_5})^{\underline{t_6}}$ of the redex, typed t_7, that we are considering for reduction, we get $\mathcal{E}_0 \rhd \mathcal{E}_1$ where

$$\begin{array}{rcl} \mathcal{E}_1 & = & \{\, (t_4^1, t_5^1, t_4^2, t_5^2 \to t_3) \to t_8 \stackrel{\perp}{=} \omega \to (t_0 \to t_0)\,;\, \{t_3, t_4^1, t_4^2, t_5^1, t_5^2, t_6^1, t_6^2\}, \\ & & t_6^2 \to t_3 \stackrel{\perp}{=} t_6^1\,;\, \{t_4^2, t_5^2, t_6^2)\}, \\ & & t_5^1 \to t_6^1 \stackrel{\perp}{=} t_4^1\,;\, \{t_5^1\}, \\ & & t_5^2 \to t_6^2 \stackrel{\perp}{=} t_4^2\,;\, \{t_5^2\}\,\} \end{array}$$

which is the set of constraints associated with $(\lambda x\lambda y y^{t_0}\lambda u((u^{t_4^1}u^{t_5^1})^{\underline{t_6^1}}(u^{t_4^2}u^{t_5^2})^{\underline{t_6^2}})^{\underline{t_3}})^{\underline{t_8}}$, as described above. In $\mathcal{E}_1$ there is only one reducible equation, with root t_8. Decomposing this equation, we apply the case (i) of the definition, and we get

$$\begin{array}{rcl} \mathcal{E}_2 & = & \{\, t_6^2 \to t_3 \stackrel{\perp}{=} t_6^1\,;\, \{t_4^2, t_5^2, t_6^2)\}, \\ & & t_5^1 \to t_6^1 \stackrel{\perp}{=} t_4^1\,;\, \{t_5^1\}, \\ & & t_5^2 \to t_6^2 \stackrel{\perp}{=} t_4^2\,;\, \{t_5^2\}\,\} \end{array}$$

which is the set of constraints associated with $[\lambda y y^{t_0}, \lambda u((u^{t_4^1}u^{t_5^1})^{\underline{t_6^1}}(u^{t_4^2}u^{t_5^2})^{\underline{t_6^2}})^{\underline{t_3}}]$. Notice that in the second step of solving the typing constraints, from $\mathcal{E}_1$ to $\mathcal{E}_2$, we had to apply the substitution $t_8 \mapsto (t_0 \to t_0)$. Since t_8 does not occur in $\mathcal{E}_1$, apart obviously in the reduced equation, this substitution had no effect. However, one should notice that it transforms the type t_8 of the annotated version of $\mathsf{F}(\lambda u.\Delta(uu))$ into $(t_0 \to t_0)$, which is the expected type of this expression, and of its normal form $[\lambda yy, \lambda u.(uu)(uu)]$.

5. Typablity and Typing

Now we show that $\wedge$-unification can be used to characterize typability, by showing that it exactly corresponds to reduction in the extended λ-calculus. First, we show the correspondence between the normal forms.

Lemma (Normal Forms) 5.1. *An expression M of the extended λ-calculus is a κ-normal form if and only if $\mathcal{E}_A$ is irreducible, for any A such that* $\mathsf{erase}(A) = M$.

(The proof is obvious.) The next lemma states the crucial property of $\wedge$-unification. For its proof, we need the the UAC axioms, as one can see for instance when applying our type inference process to the λ-expression $(\lambda y(x(yx))\,x)$.

Lemma (Operational Correspondence) 5.2.

(i) *If $M \underset{\kappa}{\to} N$ and* $\mathsf{erase}(A) = M$ *then there exists B such that* $\mathsf{erase}(B) = N$ *and $\mathcal{E}_A \rhd \mathcal{E}_B$.*

(ii) *If $\mathcal{E}_A \rhd \mathcal{E}$ then there exists B such that $\mathcal{E} = \mathcal{E}_B$ and* $\mathsf{erase}(A) \underset{\kappa}{\to} \mathsf{erase}(B)$.

PROOF (SKETCH):

(i) we prove a more precise statement, namely that if $M \underset{\kappa}{\rightarrow} N$ and $\mathsf{erase}(A) = M$ then there exist B and Θ such that $\mathsf{erase}(B) = N$, $\mathcal{E}_A \rhd \mathcal{E}_B\ [\Theta]$, $\mathsf{typ}(B) \equiv_{\mathrm{UAC}} \Theta(\mathsf{typ}(A))$, $\Gamma_B \equiv_{\mathrm{UAC}} \Theta(\Gamma_A)$ and $\mathsf{tyvar}(B) = \Theta(\mathsf{tyvar}(A))$. We proceed by induction on $M \underset{\kappa}{\rightarrow} N$.

- If $M \underset{\kappa}{\rightarrow} N$ is an axiom, then $M = ([\lambda x M_0, N_1, \ldots, N_k] M_1)$ and $A = ([\lambda x A_0, B_1, \ldots, B_k] A_1)^t$, and $\mathcal{E}_A$ contains the equation $(\xi \rightarrow t) \stackrel{\perp}{=} (\phi \rightarrow \zeta)$ where $\xi = \mathsf{typ}(A_1)$, $\phi = \Gamma_{A_0}(x)$ and $\zeta = \mathsf{typ}(A_0)$. Let $T = \{s_1, \ldots, s_n\} = \mathsf{tyvar}(A_1)$. There are two cases.

(1) If $x \notin \mathsf{fv}(M_0)$, then $N = [M_0, N_1, \ldots, N_k, M_1]$. Since $x \notin \mathsf{fv}(A_0)$, we have $\phi = \omega$. Then, by the definition of $\wedge$-unification, we have

$$\mathcal{E}_A = \{(\xi \rightarrow t \stackrel{\perp}{=} \omega \rightarrow \zeta; T)\} \cup \mathcal{E} \rhd \{t \mapsto (\zeta; \emptyset)\}^+ \mathcal{E} \quad [\{t \mapsto (\zeta; \emptyset)\}^+]$$

Since $\mathcal{E} = \mathcal{E}_{A_0} \cup \mathcal{E}_{B_1} \cup \cdots \cup \mathcal{E}_{B_k} \cup \mathcal{E}_{A_1}$ and t does not occur in $A_0, B_1, \ldots, B_k, A_1$, it is clear that we may let $B = [A_0, B_1, \ldots, B_k, A_1]$.

(2) If $x \in \mathsf{fv}(M_0)$, then $N = [\{x \mapsto M_1\} M_0, N_1, \ldots, N_k]$. Let $\phi = t_1, \ldots, t_m$. Then we have $\mathcal{E}_A \rhd \mathcal{E}\ [\Theta]$, where, using the notations of the Definition 4.2, $\Theta = \mathsf{S}^+ \circ \{t \mapsto (\zeta; \emptyset)\}^+ \circ \mathsf{D}^-$ and

$$\mathcal{E} = \mathsf{S}^+ \circ \{t \mapsto (\zeta; \emptyset)\}^+ \big(\mathsf{D}^-(\mathcal{E}_{[A_0, B_1, \ldots, B_k]}) \cup \bigcup_{1 \leqslant j \leqslant m} \mathsf{R}_j(\mathcal{E}_{A_1})\big)$$

We let $B = [\{\, x^{t_j} \mapsto \mathsf{R}_j A_1 \mid 1 \leqslant j \leqslant m \,\} A_0, B_1, \ldots, B_k]$, and we conclude using the Lemma 4.1.

- If $M = (M_0 M_1)$ and $N = (N_0 M_1)$ with $M_0 \underset{\kappa}{\rightarrow} N_0$ then $A = (A_0 A_1)^t$ with $\mathsf{erase}(A_i) = M_i$ and

$$\mathcal{E}_A = \{(\xi \rightarrow t \stackrel{\perp}{=} \zeta; \mathsf{tyvar}(A_1))\} \cup \mathcal{E}_{A_0} \cup \mathcal{E}_{A_1}$$

where $\xi = \mathsf{typ}(A_1)$ and $\zeta = \mathsf{typ}(A_0)$. By induction hypothesis, there is B_0 and Θ such that, in particular, $\mathsf{erase}(B_0) = N_0$ and $\mathcal{E}_{A_0} \rhd \mathcal{E}_{B_0}\ [\Theta]$. We let $B = (B_0 A_1)^t$, and we use the induction hypotheses to conclude.

- The other cases are similar.

(ii) By induction on A, omitted (see [26]). ❏

THEOREM 5.3. *An expression M of the extended λ-calculus is typable if and only if there is no infinite sequence of reductions from $\mathcal{E}_A$, for any A such that $\mathsf{erase}(A) = M$.*

PROOF: by the Theorem 2.1, if M is typable, then M is strongly normalizing, and therefore, by the previous Lemma, there is no infinite sequence of reductions from $\mathcal{E}_A$ if $\mathsf{erase}(A) = M$. Conversely, if, for A such that $\mathsf{erase}(A) = M$, there is no infinite sequence of reductions from $\mathcal{E}_A$, then M is strongly normalizing, and we conclude using Theorem 2.1 again. ❏

This result provides an alternative solution to a problem raised in [13], of finding a unification-like characterization of strong normalization of λ-terms, without using expansion variables. In the PhD Thesis of the second author [26], it is shown that if we do not distinguish the two cases (i) and (ii) in the Definition 4.2, allowing m to be 0 in the second case, then the reduction of the set of constraints associated with [an annotated version of] an expression M converges if and only if this expression M has a normal form. This corresponds to a characterization of (weakly) normalizing terms in system $\mathcal{D}\Omega$, see [10, 16].

A consequence of the Lemma 5.2 is that, if $\mathcal{E}_A \rhd^* \mathcal{E}$ where $\mathcal{E}$ is irreducible, then there exists B such that $\mathcal{E} = \mathcal{E}_B$ and $\mathsf{erase}(A) \underset{\kappa}{\overset{*}{\rightarrow}} \mathsf{erase}(B)$. By the Lemma 5.1, we know that $\mathsf{erase}(B)$ is a normal form. Now we show how to built the canonical typing of $\mathsf{erase}(B)$ using $\mathcal{E}_B$. To this end, we define a transformation $\rightsquigarrow$ on pairs $(\mathcal{E}, \Gamma \vdash \tau)$, called *simplification* (of typing constraints) and given as follows:

$$(\{\sigma \stackrel{\perp}{=} t; T\} \cup \mathcal{E}, \Gamma \vdash \tau) \ \rightsquigarrow \ \{t \mapsto \sigma\}^-(\mathcal{E}, \Gamma \vdash \tau)$$

where $\{t \mapsto \sigma\}^-$ is only applied to types and equations, not to territories (by the Remark 3.1, this is well defined). Our final result, combined with Theorem 5.3, establishes that $\wedge$-unification and simplification allow us to compute a canonical typing for any strongly normalizing expression:

THEOREM 5.4. *For any normal form P and annotated expression A such that* $\mathsf{erase}(A) = P$ *there exists Γ and τ such that* $(\mathcal{E}_A, \Gamma_A \vdash \mathsf{typ}(A)) \overset{*}{\rightsquigarrow} (\emptyset, \Gamma \vdash \tau)$ *and $\Gamma \vdash \tau$ is the canonical typing of P.*

PROOF (SKETCH): by induction on A. Let us just examine the case where $A = (A_0 A_1)^t$. Notice that $\mathsf{erase}(A_0)$ must be an H, with a head variable x. We have

$$\mathcal{E}_A = \{(\xi \rightarrow t \stackrel{\perp}{=} \zeta; T)\} \cup \mathcal{E}_{A_0} \cup \mathcal{E}_{A_1}$$

where $\xi = \mathsf{typ}(A_1)$ and $\zeta = \mathsf{typ}(A_0)$. Moreover $\Gamma_A = \Gamma_{A_0} \wedge \Gamma_{A_1}$. By induction hypothesis, we have

$$(\mathcal{E}_{A_i}, \Gamma_{A_i} \vdash \mathsf{typ}(A_i)) \ \overset{*}{\rightsquigarrow} \ (\emptyset, \Gamma_i \vdash \tau_i)$$

where $\Gamma_i \vdash \tau_i$ is the canonical typing of $\mathsf{erase}(A_i)$, for $i = 0, 1$, and therefore

$$\Gamma_0 = \{x : \sigma_1 \rightarrow \cdots \sigma_n \rightarrow t'\} \wedge \Gamma'_0$$

with $\tau_0 = t'$. Since τ_i is obtained from $\mathsf{typ}(A_i)$ by a sequence of non trivial type substitutions, we have $\zeta = t'$, and the simplification of $\mathcal{E}_{A_1}$ transforms ξ into τ_1. Then we have

$$\mathcal{E}_A \ \overset{*}{\rightsquigarrow} \ (\{(\tau_1 \rightarrow t \stackrel{\perp}{=} t'); T\}, \Gamma_A \vdash t) \ \rightsquigarrow \ \{t' \mapsto (\tau_1 \rightarrow t)\}^-(\emptyset, \Gamma_A \vdash t)$$

and it is easy to see that $\{t' \mapsto (\tau_1 \rightarrow t)\}^-(\Gamma_A \vdash t)$ is the canonical typing of $P = \mathsf{erase}(A)$. ❑

The Lemma 5.2 and Theorem 5.4 are only existential assertions, and therefore they do not completely specify a semi-algorithm for type inference. Indeed, the algorithm implemented by the second author, which is available from the url mentionned in [25], is more clever than that: it deals with pairs $(\mathcal{E}, \Pi)$, where Π is a "tentative proof of typing", that is a proof in the type system where the typing rule for application is

$$\frac{\Gamma \vdash M : \tau \quad \forall i.\Delta_i \vdash N : \tau_i}{\Gamma \wedge \Delta_1 \wedge \cdots \wedge \Delta_m \vdash (MN) : \sigma} \ m > 0$$

Then the transformations performed by the algorihm do not only operate on the set of constraints (by $\wedge$-unification), but also on the proof of typing part, in such a way that, if we start with $(\mathcal{E}_A, \Pi_A)$ (for a suitably defined Π_A) where $\mathsf{erase}(A)$ is strongly normalizing, then the algorithm ends up with $(\emptyset, \Pi)$ where Π is a valid proof of typing for the initial expression $\mathsf{erase}(A)$. Moreover, the algorithm checks at every step the rank of the generated types, so that if a bound is provided for the rank, the type inference algorithm terminate (we refer to [26] for the details). The algorithm could also more simply deal with pairs $(\mathcal{E}, \Gamma \vdash \tau)$, starting from $(\mathcal{E}_A, \Gamma_A \vdash \mathsf{typ}(A))$.

It is not easy to compare our algorithm with the one of Kfoury and Wells for their system I [14], because the formalisms which are used are quite different. The main differences are that we replace the notion of an expansion variable with the notion of a territory of an equation, and that we perform in an atomic way several operations in one $\wedge$-unification step, while these "micro-steps" are allowed to commute in Kfoury and Wells' algorithm, thus making a precise comparison very difficult. Nevertheless, we strongly believe (see [26] for a thorough discussion) that one $\wedge$-unification step in our algorithm, where the duplication factor is m, corresponds to $m + 2$ (if $m > 0$, otherwise 3) steps in Kfoury and Wells' algorithm.

6. Conclusion

We have presented a new semi-algorithm for inferring principal typings for strongly normalizing λ-expressions in the intersection type discipline. The correctness of our unification mechanism is not too difficult to establish. Although we showed that in the pure λ-calculus our algorithm coincides with (strong) normalization, it still deals with typing constraints rather than with λ-expressions, and is therefore open to generalizations to enriched calculi. In [26] some preliminary results in this direction are obtained, regarding the typing of mutable variables (that is, references à la ML) and of recursion.

References

[1] R. Amadio, P.-L. Curien, *Domains and Lambda-Calculi,* Cambridge Tracts in Computer Science Vol. 46, Cambridge University Press (1998).

[2] S. van Bakel, *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems,* PhD Thesis, Mathematisch Centrum, Amsterdam (1993).

[3] S. van Bakel, *Principal type schemes for the strict type assignment system,* J. of Logic and Computation, Vol. 3 No. 6 (1993) 643-670.

[4] S. van Bakel, *Intersection type assignment systems,* Theoretical Comput. Sci. Vol. 151 No. 2 (1995) 348-435.

[5] G. Boudol, *On strong normalization in the intersection type discipline,* TLCA'03, Lecture Notes in Comput. Sci. 2701 (2003) 60-74.

[6] S. Carlier, *Polar type inference with intersection types and ω,* Workshop on Intersection Types and Related Systems, Electronic Notes in Theoret. Comput. Sci. Vol. 70 (2002).

[7] S. Carlier, J.B. Wells, *Type inference with expansion variables and intersection types in system E and an exact correspondence with β-reduction,* Tech. Rep. HW-MACS-TR-0012, Heriot Watt University (2004).

[8] M. Coppo, M. Dezani-Ciancaglini, *An extension of the basic functionality theory for the λ-calculus,* Notre Dame J. of Formal Logic 21 (1980) 685-693.

[9] M. Coppo, M. Dezani-Ciancaglini, B. Venneri, *Principal type schemes and lambda-calculus semantics,* In To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism (J.R. Hindley and J.P. Seldin, Eds.), Academic Press (1980) 535-560.

[10] M. Coppo, M. Dezani-Ciancaglini, B. Venneri, *Functional characters of solvable terms,* Zeit. Math. Logik Grund. 27 (1981) 45-58.

[11] L. Damas, *Type Assignment in Programming Languages,* PhD Thesis, University of Edimburgh CST-33-85 (1985).

[12] T. Jim, *A polar type system,* Workshop on Intersection Types and Related Systems (2000).

[13] A. J. Kfoury, *Beta-reduction as unification,* in Logic, Algebra and Computer Science, H. Rasiowa Memorial Conference, December 1996 (D. Niwinski, Ed.), Banach Center Publication Vol. 46 (1999) 137-158.

[14] A. J. Kfoury, J.B. Wells, *Principality and decidable type inference for finite-rank intersection types,* Theoretical Comput. Sci. Vol. 311 No. 1 (2004) 1-70.

[15] J.W. Klop, *Combinatory Reduction Systems,* PhD Thesis, Utrecht University. Mathematical Centre Tracts Vol. 127, Mathematisch Centrum, Amsterdam (1980).

[16] J.-L. Krivine, *Lambda-Calcul: Types et Modèles,* Masson, Paris (1990). English translation "Lambda-Calculus, Types and Models", Ellis Horwood (1993).

[17] R. Milner, *A theory of type polymorphism in programming,* J. of Computer and System Sciences Vol. 17 (1978) 348-375.

[18] J.C. Mitchell, *Concepts in Programming Languages,* Cambridge University Press (2003).

[19] P. Møller Neergaard, H.G. Mairson, *Rank bounded intersection: types, potency, and idempotency,* Draft (2003).

[20] J.H. Morris, *Lambda-calculus models of programming languages,* PhD Thesis, MIT (1968).

[21] G. Pottinger, *A type assignment for the strongly normalizable λ-terms,* In To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism (J.R. Hindley and J.P. Seldin, Eds.), Academic Press (1980) 561-577.

[22] P. Sallé, *Une extension de la théorie des types en λ-calcul,* ICALP, Lecture Notes in Comput. Sci. 62 (1978) 398-410.

[23] S. Ronchi Della Rocca, *Principal type scheme and unification for intersection type discipline,* Theoretical Comput. Sci. 59 (1988) 181-209.

[24] S. Ronchi Della Rocca, B. Venneri, *Principal type schemes for an extended type theory,* Theoretical Comput. Sci. 28 (1984) 151-169.

[25] P. Zimmer, *TypI, a type inference interpreter for the intersection type discipline,* `http://www-sop.inria.fr/mimosa/Pascal.Zimmer/typi.html` (2003).

[26] P. Zimmer, *Récursion Généralisée et Inférence de Type avec Intersection,* Thèse, Université de Nice-Sophia Antipolis. Available from the web page of the author (2004).