A Quantitative Version of Simple Types

Daniele Pautasso
Dipartimento di Informatica, University of Torino, Italy

Simona Ronchi Della Rocca
Dipartimento di Informatica, University of Torino, Italy

Abstract

This work introduces a quantitative version of the simple type assignment system, starting from a suitable restriction of non-idempotent intersection types. The resulting system is decidable and has the same typability power as the simple type system; thus, assigning types to terms supplies the very same qualitative information given by simple types, but at the same time can provide some interesting quantitative information. It is well known that typability for simple types is equivalent to unification; we prove a similar result for the newly introduced system. More precisely, we show that typability is equivalent to a unification problem which is a non-trivial extension of the classical one: in addition to unification rules, our typing algorithm makes use of an expansion operation that increases the cardinality of multisets whenever needed.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases λ-calculus, intersection types, unification

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.29

Acknowledgements We would like to thank Delia Kesner and Antonio Bucciarelli for many useful discussions about the topic of this paper. We are also grateful to the anonymous referees for their careful reading and suggestions.

1 Introduction

Simple types. Simple type assignment for λ-calculus [8, 13] is a way of assigning types, which are formulas of minimal intuitionistic logic, to λ-terms. It enjoys important properties: to be typable implies strong normalization, and both typability and inhabitation are decidable – typability being the problem of determining, given a term, whether it is possible to assign a type to it, and inhabitation the problem of determining, given a type, if there is a term to which it can be assigned. Considering the λ-calculus as a general paradigm for functional programming languages, types are program specifications, so the decidability of typability provides tools for proving program correctness, that of inhabitation for program synthesis. Simple type assignment is the basis of typed functional languages, like ML and Haskell.

Quantitative intersection types. Recently the scientific community interest turned to a quantitative approach for program semantics, and from this point of view non-idempotent intersection types are a powerful tool. Intersection types have been introduced in [9], in order to increase the typability power of simple type assignment systems, but quite early they turned out to be useful in characterizing qualitative semantic properties of λ-calculus like solvability and strong normalization, and in describing models of λ-calculus in various settings [2, 20]. Intersection types are built by adding to the connective $\to$ of simple types an intersection connective $\wedge$ which enjoys associativity, commutativity and idempotency, i.e., $A \wedge A = A$; in other words, an intersection of types can be seen as a notation for a set of types. A variant of intersection types, where intersection is no more idempotent (so that intersection of types becomes a notation for multisets), was first designed by [10], and later used by De Carvalho [12], for the purpose of studying the complexity of reduction.

Indeed non-idempotency has a quantitative flavour: non-idempotent intersection types have been used to design relational models of λ-calculus [17, 5], to characterize polynomial time computational complexity [4], and to compute the exact number of reductions to reach the normal form of a term [14]. The typability problem is undecidable both in idempotent and non-idempotent intersection type assignment systems; on the other hand, the inhabitation problem is decidable if we consider non-idempotent intersection [6], while it is undecidable for the idempotent case [22].

A question naturally arises: *is there a quantitative version of simple types?* Rephrasing it in a more technical way: *is there a restriction of the non-idempotent intersection type system with the same typability power as the simple type system?*

Contribution of this paper. In this paper we give a positive answer: we introduce a non-idempotent intersection type assignment system which is a restriction of the one defined in [7], and prove that the typability problem is decidable. The key idea is to restrict the multiset formation to uniform types, two types being uniform if they differ only in the cardinality of the multisets occurring in it. Assigning types to terms in such a system, which we name system $\mathcal{U}$, supplies the very same qualitative information given by simple types, but at the same time provides some interesting quantitative information.

In our analysis we take the well-known equivalence between typability in simple types and unification as a starting point, and prove a similar result for system $\mathcal{U}$. More precisely, we show that typability is equivalent to an extension of the classical unification problem: in addition to unification rules, our algorithm makes use of an operation, called expansion, which increases the cardinality of multisets whenever needed. Indeed, in the simple type system all the derivations for a given term share the same structure (as trees), and differ from each other only in the types occurring in their nodes. But, in the intersection type system, derivations for the same term can differ both in the previous sense and in the structure of the derivation. So unification is used to match the types in the nodes of a derivation, while the expansion modifies its structure.

Related works. Typability for intersection types has been intensively studied. A first approach, introducing the notion of expansion, can be found in [10]; the result was then extended to a more general case in [21, 19]. Both type systems considered in the aforementioned works enjoy an approximation theorem, where approximants are head normal forms in the λ-calculus extended with a constant Ω. This property allows the principal pair to be defined by induction on the structure of terms. We cannot use a similar technique here, since there is no syntactical characterization of simply typable terms. A different methodology has been explored in [16], by enriching types with pointers to the subtypes that can be modified by the expansion. In our work this role is played by a system of constraints, which also keeps track of the restrictions on the construction of multisets.

In the literature, some decidable restrictions of idempotent intersection types have been proposed, namely in [11] and [15]. In both cases, however, they are obtained by limiting the shape of types, and do not supply subsets of terms with interesting properties.

Paper organization. In Section 2 some well-known and essential facts about the simply typed λ-calculus are presented, together with the notion of principal derivation. In Section 3 we introduce the type assignment system $\mathcal{U}$ of uniform intersection types, extend the notion of principal derivation to it, and design a typing algorithm. Section 4 contains some clarifying examples and Section 5 the main result of the paper. A short conclusion is in Section 6. The most technical proof can be found in the Appendix.

$$\frac{}{\Gamma, x : A \vdash x : A}\ (\mathtt{var}) \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ (\to_{\mathtt{I}})$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A \qquad \Gamma \smile \Delta}{\Gamma \cup \Delta \vdash MN : B}\ (\to_{\mathtt{E}})$$

Figure 1 The system $\mathcal{S}$.

2 Simple types

We briefly recall the λ-calculus grammar and the simple type assignment system. Terms and term contexts of the λ-calculus are generated by the following grammars, respectively:

$$M, N, P ::= x \mid \lambda x.M \mid MN \qquad\qquad \mathtt{C} ::= \square \mid \lambda x.\mathtt{C} \mid \mathtt{C}M \mid M\mathtt{C}$$

where x ranges over a countable set of variables. $\mathtt{FV}(M)$ denotes the set of free variables of M. We will use for λ-calculus the notations in [1].

▶ **Definition 1.** *The set $\mathcal{T}_{\mathcal{S}}$ of* simple types *is defined by the following grammar:*

$$A, B, C \quad ::= \quad a \mid A \to B$$

where a ranges over a countable set of type variables.

A *context* is a set of pairs $x : A$, where x is a term variable and $A \in \mathcal{T}_{\mathcal{S}}$; contexts are ranged over by Γ, Δ. If $x : A \in \Gamma$, then $\Gamma(x) = A$; the domain of a context Γ is $\mathtt{dom}(\Gamma) = \{x \mid x : A \in \Gamma\}$; the writing $\Gamma \smile \Delta$ means that Γ and Δ agree, i.e. if $x \in \mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta)$ then $\Gamma(x) = \Delta(x)$; if S is a set of variables, then $\Gamma|_S$ denotes the restriction of $\mathtt{dom}(\Gamma)$ to S; moreover Γ, Δ is short for $\Gamma \cup \Delta$ in case $\mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta) = \emptyset$.

The simple type assignment system $\mathcal{S}$ is a set of rules proving statements (typings) of shape $\Gamma \vdash M : A$, where Γ is a context, M a term and $A \in \mathcal{T}_{\mathcal{S}}$. The rules are shown in Figure 1. A *derivation* is a tree of rules, such that its leaves are applications of rule ($\mathtt{var}$) and the root is its conclusion. Derivations are ranged over by Π, Σ, Θ. We write $\Gamma \vdash M : A$ as a shorthand for the existence of a derivation proving $\Gamma \vdash M : A$, and when we want to name a particular derivation with such conclusion we write $\Pi \triangleright \Gamma \vdash M : A$. The system enjoys two important properties:

▶ **Theorem 2** (subject reduction). *$\Gamma \vdash M : A$ and $M \to_\beta N$ imply $\Gamma \vdash N : A$.*

▶ **Theorem 3** (strong normalization). *$\Gamma \vdash M : A$ implies M is strongly normalizing.*

The principal derivation

System $\mathcal{S}$ is decidable, and it enjoys the principal typing property, i.e., each typing for a term M is obtained from the principal one by substitution and weakening [3, 13].

Let $\mathtt{Var}(A)$ be the set of type variables occurring in the type A; we say that A and B are disjoint (notation $A * B$) if $\mathtt{Var}(A) \cap \mathtt{Var}(B) = \emptyset$. Both the definition of $\mathtt{Var}$ and the notion of disjointness can be extended to contexts and derivations in the standard way. A type A is fresh w.r.t. a derivation Π if $A * B$ for each B occurring in Π. Throughout the work we will frequently use indexed types, where indexes are natural numbers, and the symbols I, J will denote sets of indexes.

$$\frac{}{\Pi \triangleright x : a \vdash_{\mathtt{p}} x : a}\ (\mathtt{var}) \qquad (E_\Pi, V_\Pi) = (\emptyset, \emptyset)$$

$$\frac{\Sigma \triangleright \Gamma, x : a \vdash_{\mathtt{p}} M : A}{\Pi \triangleright \Gamma \vdash_{\mathtt{p}} \lambda x.M : a \to A}\ (\to_{\mathtt{I}}) \qquad (E_\Pi, V_\Pi) = (E_\Sigma, V_\Sigma)$$

$$\frac{\Sigma \triangleright \Gamma \vdash_{\mathtt{p}} M : A \quad x \notin \mathtt{dom}(\Gamma) \quad a \text{ fresh}}{\Pi \triangleright \Gamma \vdash_{\mathtt{p}} \lambda x.M : a \to A}\ (\to_{\mathtt{I}})_\emptyset \qquad (E_\Pi, V_\Pi) = (E_\Sigma, V_\Sigma)$$

$$\frac{\Sigma_1 \triangleright \Gamma \vdash_{\mathtt{p}} M : a \to B \quad \Sigma_2 \triangleright \Delta \vdash_{\mathtt{p}} N : A \quad \Sigma_1 * \Sigma_2}{\Pi \triangleright \Gamma \cup \Delta \vdash_{\mathtt{p}} MN : B}\ (\to_{\mathtt{E1}})$$

$$E_\Pi = E_{\Sigma_1} \cup E_{\Sigma_2} \cup \{a \doteq A\}$$
$$V_\Pi = V_{\Sigma_1} \cup V_{\Sigma_2} \cup \{\Gamma(x) \doteq \Delta(x) \mid x \in \mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta)\}$$

$$\frac{\Sigma_1 \triangleright \Gamma \vdash_{\mathtt{p}} M : a \quad \Sigma_2 \triangleright \Delta \vdash_{\mathtt{p}} N : A \quad b \text{ fresh} \quad \Sigma_1 * \Sigma_2}{\Pi \triangleright \Gamma \cup \Delta \vdash_{\mathtt{p}} MN : b}\ (\to_{\mathtt{E2}})$$

$$E_\Pi = E_{\Sigma_1} \cup E_{\Sigma_2} \cup \{a \doteq A \to b\}$$
$$V_\Pi = V_{\Sigma_1} \cup V_{\Sigma_2} \cup \{\Gamma(x) \doteq \Delta(x) \mid x \in \mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta)\}$$

Figure 2 Principal derivation for $\mathcal{S}$.

A pseudo-derivation is a tree of rules assigning simple types to terms; we extend to pseudo-derivations all the notations for derivations. To each pseudo-derivation Π is associated a system of constraints (E_Π, V_Π), where E_Π and V_Π are sets of equations between types, written $A \doteq B$; we will drop subscripts from E_Π and V_Π when they are clear from the context.

▶ **Definition 4.** *The principal derivation for a term M, denoted by* $\mathtt{PD}(M)$*, is a pair $(\Pi, (E_\Pi, V_\Pi))$, where Π is a pseudo-derivation with subject M and (E_Π, V_Π) is the associated system of constraints, as defined in Fig. 2.* $\mathtt{PD}(M)$ *is unique, modulo renaming of type variables.*

▶ **Definition 5.** *Let $S = \{A_i \doteq B_i \mid i \in I\}$ be a set of equations between types.*

- *S is solvable if there is a substitution θ, replacing type variables by types, such that $\theta(A_i) = \theta(B_i)$ for all $i \in I$.*
- *S is in solved form iff:*
  - *every A_i is a variable a_i, and all a_i are distinct;*
  - *no left-hand side a_i appears in some right-hand side B_j for all $i, j \in I$.*
- *If $S = \{a_i \doteq A_i \mid i \in I\}$ is in solved form, $\vec{S}$ such that $\vec{S}(a_i) = A_i$ is the most general substitution solving it, also called most general unifier (m.g.u.).*
- *S is in unsolvable form iff it contains a circular equation, i.e. an equation $a \doteq A$ such that a occurs in A.*

The rules reducing an equational system either to solved or unsolvable form are given in Fig. 3; they are directly obtained from the classical Robinson's unification algorithm [18]. We will write $\mathtt{Unify}(S)$ for the procedure applying to S the rules of Fig. 3 as much as possible. A substitution θ can be extended to types and pseudo-derivations in the standard way.

$$\frac{S \cup \{A \doteq A\}}{S}\ \textit{erase} \qquad \frac{S \cup \{A \to B \doteq C \to D\}}{S \cup \{A \doteq C\} \cup \{B \doteq D\}}\ \textit{decompose}$$

$$\frac{S \cup \{A \to B \doteq a\}}{S \cup \{a \doteq A \to B\}}\ \textit{swap} \qquad \frac{S \cup \{a \doteq A\} \quad a \notin \mathtt{Var}(A) \quad a \in \mathtt{Var}(S)}{S[A/a] \cup \{a \doteq A\}}\ \textit{substitute}$$

Figure 3 Unification rules for simple types.

▶ Remark 6. A principal derivation $\mathtt{PD}(M) = (\Pi, (E, V))$ essentially becomes a derivation in $\mathcal{S}$ under a substitution θ solving the system of constraints; due to the differences between system $\mathcal{S}$ and the rules in Fig. 2, however, some minor adjustments are needed. To ease the notation we will leave such transformations implicit and just write $\theta(\Pi)$ for the derivation in which all $(\to_{\mathtt{I}})_{\emptyset}$ rules have been replaced by $(\to_{\mathtt{I}})$ rules (by suitably exploiting weakening in the axioms to introduce the abstracted dummy variables), and the distinction between rules $(\to_{\mathtt{E1}})$ and $(\to_{\mathtt{E2}})$ has been dropped, yielding rule $(\to_{\mathtt{E}})$.

The following result, first proved in [3], is well known:

▶ **Theorem 7.** *Let* $\mathtt{PD}(M) = (\Pi, (E, V))$*, where* $\Pi \triangleright \Gamma \vdash_{\mathtt{p}} M : A$*.*

1. *For every substitution* θ *which is a solution of* $E \cup V$*,* $\theta(\Pi) \triangleright \theta(\Gamma) \vdash M : \theta(A)$*.*
2. *For every derivation* $\Sigma \triangleright \Delta \vdash M : B$ *there is a substitution* θ*, solution of* $E \cup V$*, such that* $\theta(\Gamma) = \Delta|_{\mathtt{FV}(M)}$ *and* $\theta(A) = B$*.*

▶ Remark 8. In the literature the property speaks about principal pair, not principal derivation. The result is the same; here the presentation through the derivation is useful for the comparison with intersection types.

3 Uniform intersection types

We now define the uniform intersection type assignment system, based on the notion of uniform multiset. A multiset is an unordered list of elements; $\uplus$ denotes the union of multisets taking into account the multiplicities, and $|\sigma|$ denotes the cardinality of the multiset σ. The system is obtained from the system $\mathcal{H}_{e,w}$ in [6] by requiring that multisets contain only equivalent types.

▶ **Definition 9.**

- *Intersection types* ($\mathcal{I}$) *are defined by the following grammar:*

$$\begin{array}{lll} (\textbf{\textit{Intersection Types}}) & A, B, C & ::= \ a \mid \sigma \to A \\ (\textbf{\textit{Multisets}}) & \sigma, \tau & ::= \ [A_1, \ldots, A_n] \quad (n \geq 1) \end{array}$$

where a *ranges over a countable set of type variables.*

- *Equivalence relation on intersection types:*

$$\begin{array}{l} a \sim a \ \textit{ for all variables } a; \\ \sigma \to A \sim \tau \to B \ \textit{ iff } \ \sigma \sim \tau \textit{ and } A \sim B; \\ [A_1, ..., A_m] \sim [B_1, ..., B_n] \ \textit{ iff } \ \forall i, j. A_i \sim B_j \quad (1 \leq i \leq m, 1 \leq j \leq n) \end{array}$$

- *Uniform intersection types* ($\mathcal{T}_{\mathcal{U}}$) *are defined by the following grammar:*

$$\begin{array}{lll} (\textbf{\textit{Unif. Int. Types}}) & A, B, C & ::= \ a \mid \sigma \to A \\ (\textbf{\textit{Uniform Multisets}}) & \sigma, \tau & ::= \ [A_1, ..., A_n] \quad \forall i, j. A_i \sim A_j \quad (1 \leq i, j \leq n) \end{array}$$

$$\frac{A \in \sigma}{\Gamma, x : \sigma \vdash_{\mathtt{i}} x : A}\ (\mathtt{var}) \qquad \frac{\Gamma, x : \sigma \vdash_{\mathtt{i}} M : A}{\Gamma \vdash_{\mathtt{i}} \lambda x.M : \sigma \to A}\ (\to_{\mathtt{I}})$$

$$\frac{\Gamma \vdash_{\mathtt{i}} M : [A_1, ..., A_n] \to B \quad (\Delta_i \vdash_{\mathtt{i}} N : A_i)_{1 \le i \le n} \quad \forall i.\Gamma \sim \Delta_i \quad \forall ij.\Delta_i \sim \Delta_j}{\Gamma \uplus_{1 \le i \le n} \Delta_i \vdash_{\mathtt{i}} MN : B}\ (\to_{\mathtt{E}})$$

Figure 4 The system $\mathcal{U}$.

For the sake of brevity, we well speak about types and multisets in both cases, when being uniform or not is clear from the context. Remark that in both grammars the empty multiset is not allowed. The types $A = [[a, a, a] \to b, [a] \to b] \to c$ and $B = [[a, a] \to b] \to c$ are uniform, whereas $[a, [a] \to b] \to c$ is not; moreover, observe that $A \sim B$.

The system $\mathcal{U}$, assigning types in $\mathcal{T}_{\mathcal{U}} \subset \mathcal{I}$ to terms, is shown in Fig. 4. We use the same notations as for simple types, but note that now a context assigns uniform multisets to term variables. If σ and τ are multisets, $\sigma \subseteq \tau$ means $\exists \rho.\tau = \sigma \uplus \rho$. If Γ and Δ are two contexts, $\Gamma \sim \Delta$ means that $\forall x \in \mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta).\Gamma(x) \sim \Delta(x)$. The system is quantitative in the following sense:

▶ **Property 10.** *Let* $\Pi \rhd \Gamma, x : \sigma \vdash_{\mathtt{i}} M : A$.

- *The number of free occurrences of* x *in* M *is bounded by* $|\sigma|$.
- *If* n *is the maximal cardinality of a multiset in* Π*, then every variable in* M*, either free or bound, has a number of occurrences* $\le n$.

Both the following properties are inherited from [7].

▶ **Theorem 11** (Subject reduction). $\Gamma \vdash_{\mathtt{i}} M : A$ *and* $M \to_\beta N$ *imply* $\Gamma \vdash_{\mathtt{i}} N : A$.

▶ **Theorem 12** (Strong normalization). $\Gamma \vdash_{\mathtt{i}} M : A$ *implies* M *is strongly normalizing.*

3.1 Principal derivations

We adapt the notion of principal derivation to system $\mathcal{U}$. In this setting, we will deal both with sets of equations and sets of equivalences between intersection types (multisets), written respectively as $A \doteq B$ ($\sigma \doteq \tau$), and $A \approx B$ ($\sigma \approx \tau$). A pseudo-derivation is a tree of rules assigning intersection types to terms. From now on multisets in a pseudo-derivation will be considered as lists, and consequently also the minor premises of the rules ($\to_{\mathtt{E1}}$) and ($\to_{\mathtt{E2}}$) will be ordered; this is essential to limit the complexity of the algorithm `Solve` we will discuss in the following section. To each pseudo-derivation Π we associate a system of constraints (E_Π, V_Π), where E_Π is a set of equations between intersection types and V_Π is a set of equivalence constraints between type variables.

▶ **Definition 13.**

- *A principal derivation for a term* M *is a pair* $\mathtt{PD}_{\mathtt{i}}(M) = (\Pi, (E_\Pi, V_\Pi))$*, where* Π *is a pseudo-derivation with subject* M *and* (E_Π, V_Π) *is the associated system of constraints, as defined in Fig. 5.*
- *The minimal principal derivation for a term* M *is a principal derivation for* M *obtained by posing* $n = 0$ *whenever dealing with rule* ($\to_{\mathtt{I}}$) *of Fig. 5, and* $n = 1$ *whenever dealing with rules* $(\to_{\mathtt{I}})_\emptyset$ *and* ($\to_{\mathtt{E2}}$)*. Since the minimal principal derivation is unique, modulo renaming of type variables, we will refer to it as* $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M)$.

$$\frac{}{\Pi \rhd x : [a] \vdash_{\mathtt{q}} x : a}\ (\mathtt{var}) \qquad (E_\Pi, V_\Pi) = (\emptyset, \emptyset)$$

$$\frac{\Sigma \rhd \Gamma, x : \sigma \vdash_{\mathtt{q}} M : A \qquad a_1, ..., a_n \text{ fresh}}{\Pi \rhd \Gamma \vdash_{\mathtt{q}} \lambda x.M : \sigma \uplus [a_1, ..., a_n] \to A}\ (\to_{\mathtt{I}}) \qquad \begin{array}{l} E_\Pi = E_\Sigma \\ V_\Pi = V_\Sigma \cup \{a \approx b \mid a, b \in \sigma \uplus [a_1, ..., a_n]\} \end{array}$$

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathtt{q}} M : A \quad x \notin \mathtt{dom}(\Gamma) \quad a_1, ..., a_n \text{ fresh}}{\Pi \rhd \Gamma \vdash_{\mathtt{q}} \lambda x.M : [a_1, ..., a_n] \to A}\ (\to_{\mathtt{I}})_\emptyset \qquad \begin{array}{l} E_\Pi = E_\Sigma \\ V_\Pi = V_\Sigma \cup \{a \approx b \mid a, b \in [a_1, ..., a_n]\} \end{array}$$

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathtt{q}} M : [a_1, ..., a_n] \to B \quad (\Sigma_i \rhd \Delta_i \vdash_{\mathtt{q}} N : A_i)_{1 \le i \le n} \quad \forall i. \Sigma * \Sigma_i \quad \forall ij. \Sigma_i * \Sigma_j\ (i \neq j)}{\Pi \rhd \Gamma \uplus_{1 \le i \le n} \Delta_i \vdash_{\mathtt{q}} MN : B}\ (\to_{\mathtt{E1}})$$

$$\begin{array}{l} E_\Pi = E_\Sigma \cup_{1 \le i \le n} E_{\Sigma_i} \cup \{a_i \doteq A_i\}_{1 \le i \le n} \\ V_\Pi = V_\Sigma \cup_{1 \le i \le n} V_{\Sigma_i} \cup \{a \approx b \mid x \in \mathtt{dom}(\Gamma) \cap_{1 \le i \le n} \mathtt{dom}(\Delta_i), a \in \Gamma(x), b \in \Delta_i(x)\} \end{array}$$

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathtt{q}} M : a \quad (\Sigma_i \rhd \Delta_i \vdash_{\mathtt{q}} N : A_i)_{1 \le i \le n} \quad b \text{ fresh} \quad \forall i. \Sigma * \Sigma_i \quad \forall ij. \Sigma_i * \Sigma_j\ (i \neq j)}{\Pi \rhd \Gamma \uplus_{1 \le i \le n} \Delta_i \vdash_{\mathtt{q}} MN : b}\ (\to_{\mathtt{E2}})$$

$$\begin{array}{l} E_\Pi = E_\Sigma \cup_{1 \le i \le n} E_{\Sigma_i} \cup \{a \doteq [A_1, ..., A_n] \to b\} \\ V_\Pi = V_\Sigma \cup_{1 \le i \le n} V_{\Sigma_i} \cup \{a \approx b \mid x \in \mathtt{dom}(\Gamma) \cap_{1 \le i \le n} \mathtt{dom}(\Delta_i), a \in \Gamma(x), b \in \Delta_i(x)\} \end{array}$$

Figure 5 Principal derivations for $\mathcal{U}$.

▶ **Definition 14.** *Let us denote by ψ a substitution $\mathtt{Var} \to \mathcal{I}$, and by ϕ a substitution $\mathtt{Var} \to \mathcal{T}_\mathcal{U}$. Substitutions are extended to multisets and pseudo-derivations in the standard way.*

- *ψ solves $E = \{A_i \doteq B_i \mid i \in I\} \cup \{\sigma_j \doteq \tau_j \mid j \in J\}$ if $\psi(A_i) = \psi(B_i)$ for all $i \in I$ and $\psi(\sigma_j) = \psi(\tau_j)$ for all $j \in J$;*
- *ϕ satisfies $V = \{A_i \approx B_i \mid i \in I\} \cup \{\sigma_j \approx \tau_j \mid j \in J\}$ if $\phi(A_i) \sim \phi(B_i)$ for all $i \in I$ and $\phi(\sigma_j) \sim \phi(\tau_j)$ for all $j \in J$;*
- *E is solvable w.r.t. V if there is a substitution ϕ solving E and satisfying V.*

In order to solve a system of constraints, we extend to intersection types the unification rules for simple types: Fig. 6 introduces two sets of rules for solving sets of equations or equivalences (replacing $\simeq$ either by $\doteq$ or by $\approx$, respectively). All the rules but *decomposeM* are the same for the two cases; observe that, in particular, the definition of *decomposeM*$_{\doteq}$ for multisets relies on the fact that multisets are now ordered. Let $\mathtt{QuasiUnify}_{\doteq}(S)$ (resp. $\mathtt{QuasiUnify}_{\approx}(S)$) denote the application of the rules in Fig. 6, replacing $\simeq$ by $\doteq$ (resp. by $\approx$), to the set S as much as possible. Moreover, if S is a set of equations, let $S/_{\approx}$ denote the set of equivalence constraints obtained replacing $\doteq$ by $\approx$. The writing $S/_{\doteq}$ is used for the dual transformation.

▶ **Definition 15.**

- *A set of constraints between intersection types $S = \{A_1 \simeq B_1, \ldots, A_n \simeq B_n, \sigma_1 \simeq \tau_1, \ldots, \sigma_m \simeq \tau_m\}$, where $\simeq \in \{\doteq, \approx\}$, is in solved form iff:*
  - *$m = 0$, i.e. there is no constraint involving multisets;*
  - *every A_i is a variable a_i, and all variables a_i are distinct;*
  - *no left-hand side a_i appears in some right-hand side B_j $(1 \le i, j \le n)$.*

$$\frac{S \cup \{A \simeq A\}\ (\{\sigma \simeq \sigma\})}{S}\ \textit{erase} \qquad \frac{S \cup \{\sigma \to A \simeq \tau \to B\}}{S \cup \{\sigma \simeq \tau\} \cup \{A \simeq B\}}\ \textit{decomposeT}$$

$$\frac{S \cup \{\sigma \to A \simeq a\}}{S \cup \{a \simeq \sigma \to A\}}\ \textit{swap} \qquad \frac{S \cup \{a \simeq A\} \quad a \notin \mathtt{Var}(A) \quad a \in \mathtt{Var}(S)}{S[A/a] \cup \{a \simeq A\}}\ \textit{substitute}$$

$$\frac{S \cup \{[A_1, \dots, A_n] \doteq [B_1, \dots, B_n]\}}{S \cup \{A_i \doteq B_i\}_{1 \le i \le n}}\ \textit{decomposeM}_{\doteq}$$

$$\frac{S \cup \{\sigma \approx \tau\}}{S \cup \{A \approx B \mid A \in \sigma, B \in \tau\}}\ \textit{decomposeM}_{\approx}$$

Figure 6 Unification rules for intersection types.

- *A constraint of the form $a \simeq A$ such that a occurs in A is a* circular constraint.
- *An equation between multisets $\sigma \doteq \tau$ such that $|\sigma| \neq |\tau|$ is a* critical equation.
- *A set S is in unsolvable form if it contains at least one circular constraint.*
- *A set of equations S is in blocked form if it contains at least one critical equation.*

▶ **Property 16.**
1. *Let S be a set of equations. Then $\mathtt{QuasiUnify}_{\doteq}(S)$ outputs a set of equations either in solved, unsolvable, or blocked form. Moreover, $\mathtt{QuasiUnify}_{\doteq}(S) = S'$ in solved form if and only if $\vec{S'}$ is the m.g.u. of S.*
2. *Let S be a set of equivalences. Then $\mathtt{QuasiUnify}_{\approx}(S)$ outputs a set of equivalences either in solved or unsolvable form. Moreover, $\mathtt{QuasiUnify}_{\approx}(S) = S'$ in solved form implies $\vec{S'}$ satisfies S.*

Proof.
1. From the fact that the rules of Fig. 6, when $\simeq$ is replaced by $\doteq$, are precisely the Robinson's unification rules.
2. From the previous point and the fact that, when $\simeq$ is replaced by $\approx$, multisets can be decomposed even if they have different cardinalities. ◀

3.2 Expansion

In order to deal with a set of equations in blocked form, we introduce an operation called expansion, working on principal derivations. Intuitively, an expansion modifies a principal derivation by replicating the minor premises of a rule $(\to_{\mathtt{E}i})$, increasing the cardinality of multisets, and updating the associated constraints accordingly.

▶ **Definition 17.**
- *Let $(\Pi, (E, V))$ be a principal derivation for M, and let x be a bound variable occurring in M. This means that in Π there is a rule:*

$$\frac{\Gamma, x : \sigma \vdash_{\mathtt{q}} N : B \qquad \sigma \subseteq \tau}{\Gamma \vdash_{\mathtt{q}} \lambda x.N : \tau \to B}\ (\to_{\mathtt{I}})$$

If there is also a subsequent rule:

$$\frac{\Gamma' \vdash_{\mathtt{q}} P : \tau \to B \qquad (\Delta_i \vdash_{\mathtt{q}} Q : C_i)_{1 \le i \le |\tau|}}{\Gamma' \uplus_{1 \le i \le |\tau|} \Delta_i \vdash_{\mathtt{q}} PQ : B}\ (\to_{\mathtt{E1}})$$

then we say that x is an applied bound variable in Π, and this last rule is called the application rule for x. Otherwise x is an unapplied bound variable in Π. We denote $\mathcal{B}$ the set of applied bound variables in Π.

- *Let $(\Pi, (E, V))$ be a principal derivation for M, and let x, y be applied bound variables in Π. If $M = \mathtt{C}[\lambda x.\mathtt{C}'[\lambda y.N]]$, then define $y < x$.*

▶ **Definition 18.** *An expansion, denoted $\mathtt{Expand}(\sigma, n)$, has two parameters: a multiset σ and a natural number $n \geq 1$. Applying $\mathtt{Expand}(\sigma, n)$ to a principal derivation $(\Pi, (E, V))$ means to perform the steps described below. There are four cases:*

1. *If there is a rule $(\to_{\mathtt{I}})$ with conclusion $\Gamma \vdash_{\mathtt{q}} \lambda x.N : \sigma \to A$, and x is an unapplied bound variable in Π, then:*
   - **a.** *replace, in the conclusion of the rule $(\to_{\mathtt{I}})$ and in the subsequent rules, the multiset σ by $\sigma \uplus [a_1, ..., a_n]$ where all the a_i are fresh;*
   - **b.** *recompute the system of constraints (E, V) according to Fig. 5, taking into account the increased cardinality.*
2. *If $\sigma = [A_1, ..., A_m]$ and in Π there is an elimination rule of shape*

$$\frac{\Gamma \vdash_{\mathtt{q}} P : B \qquad (\Sigma_i \triangleright \Delta_i \vdash_{\mathtt{q}} Q : A_i)_{1 \leq i \leq m}}{\Xi = \Gamma \uplus_{1 \leq i \leq m} \Delta_i \vdash_{\mathtt{q}} PQ : C} \ (\to_{\mathtt{E}j})$$

   perform the following actions:
   - **a.** *update the structure of Π by adding n disjoint copies of $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(Q)$ as the rightmost premises of the $(\to_{\mathtt{E}j})$ rule;*
   - **b.** *compute the new conclusion context Ξ' by taking into account the newly added premises, and replace Ξ by Ξ' in the conclusion of the rule $(\to_{\mathtt{E}j})$ and in all subsequent rules;*
   - **c.** *if $B = [a_1, ..., a_m] \to C$, i.e. if the rule was the application rule for some variable x, then replace, in the conclusion of the $(\to_{\mathtt{I}})$ rule abstracting x and in the subsequent rules, the multiset $[a_1, ..., a_m]$ by $[a_1, ..., a_m, a_{m+1}, ..., a_{m+n}]$, where $a_{m+1}, ..., a_{m+n}$ are fresh;*
   - **d.** *let $\mathcal{B}$ be the set of applied bound variables in Π, and let $\mathcal{L} = \mathcal{B} \cap \mathtt{dom}(\Delta_i)$.*
     While $\mathcal{L} \neq \emptyset$ do:
     - *let $\mathtt{min}(\mathcal{L})$ be the minimum element of $\mathcal{L}$ according to the $<$ relation of Definition 17, and let its application rule be:*

$$\frac{\Phi \vdash_{\mathtt{q}} S : \tau \to D \qquad (\Theta_i \triangleright \Psi_i \vdash_{\mathtt{q}} T : D_i)_{1 \leq i \leq s}}{\Upsilon = \Phi \uplus_{1 \leq i \leq s} \Psi_i \vdash_{\mathtt{q}} ST : D} \ (\to_{\mathtt{E1}})$$

       the modifications until this point certainly increased $|\tau|$, so $|\tau| > s$. Identify in τ the $|\tau| - s$ type variables that were introduced by the expansion, i.e. the set $\mathcal{F} = \{a \mid a \in \tau, a \notin \mathtt{Var}(V)\}$;
     - *update the structure of Π by adding $|\tau| - s$ disjoint copies of $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(T)$ as premises of the $(\to_{\mathtt{E1}})$ rule, matching them to the type variables in $\mathcal{F}$;*
     - *compute the new conclusion context Υ' by taking into account the newly added premises, and replace Υ by Υ' in the conclusion of the rule $(\to_{\mathtt{E1}})$ and in all subsequent rules;*
     - *update $\mathcal{L}$ by posing $\mathcal{L} = (\mathcal{L} - \mathtt{min}(\mathcal{L})) \cup (\mathcal{B} \cap \mathtt{dom}(\Theta_i))$;*
   - **e.** *recompute the system of constraints (E, V) according to Fig. 5, taking into account the new structure of the pseudo-derivation Π.*

3. *If $\sigma = [a_1, ..., a_m]$ and in Π there is a rule:*

$$\frac{\Gamma \vdash_{\mathtt{q}} P : [a_1, ..., a_m] \to B \qquad (\Sigma_i \triangleright \Delta_i \vdash_{\mathtt{q}} Q : A_i)_{1 \le i \le m}}{\Gamma \uplus_{1 \le i \le m} \Delta_i \vdash_{\mathtt{q}} PQ : B} \ (\to_{\mathtt{E1}})$$

then perform $\mathtt{Expand}([A_1, ..., A_m], n)$.

4. *If none of the above conditions is met, the expansion behaves as the identity, i.e.* $\mathtt{Expand}(\sigma, n)(\Pi, (E, V)) = (\Pi, (E, V))$

▶ **Lemma 19.** *Let $(\Pi, (E, V))$ be a principal derivation for M. For all σ, n, the expansion $\mathtt{Expand}(\sigma, n)(\Pi, (E, V))$ terminates, returning a principal derivation for M.*

Proof. Both in cases (1) and (4) the expansion obviously stops without altering the structure of Π, and the result is coherent with the definition of principal derivation. Case (3) reduces to (2), hence we limit our analysis to such case. Clearly the structure of the output derivation differs from the input one, as some minor premises of elimination rules are extended with multiple copies of the original subderivations; however, since such copies introduce only fresh variables, all the new subderivations are disjoint, in agreement with the definition of principal derivation. We now prove that the number of while iterations is finite. The key observation is the following one: choosing the minimal element of $\mathcal{L}$ ensures that any applied bound variable is considered at most once, because a variable cannot re-enter the set $\mathcal{L}$ after being selected as $\mathtt{min}(\mathcal{L})$ and then discarded. To see this, let $x, y \in \mathcal{B}$; moreover let Σ be the subderivation whose conclusion is the $(\to_{\mathtt{I}})$ rule abstracting x, and let Σ' be the subderivation whose conclusion is the $(\to_{\mathtt{I}})$ rule abstracting y. Indeed, $y < x$ means that Σ' is a subderivation of Σ. Hence modifications performed by the expansion when the while loop selects y as $\mathtt{min}(\mathcal{L})$ may increment the premise of x in the context (note that x is possibly free in the subderivation being replicated), but not the other way around. We conclude that the number of while iterations is bounded by $|\mathcal{B}|$. ◀

Thanks to the expansion, the notion of solvability can be extended to principal derivations. We stress that the considerations made in Remark 6 apply also to the present case, and therefore we adopt the same conventions.

▶ **Definition 20.** *Let $\mathtt{PD_i}(M) = (\Pi \triangleright \Gamma \vdash_{\mathtt{q}} M : A, (E, V))$. A solution of $\mathtt{PD_i}(M)$ is a pair $(\bar{e}, \phi)$, where $\bar{e}$ is sequence of expansions such that $\bar{e}(\Pi, (E, V)) = (\Pi' \triangleright \Gamma' \vdash_{\mathtt{q}} M : A', (E', V'))$, and $\phi : \mathtt{Var} \to \mathcal{T_U}$ is a solution of E' w.r.t. V', i.e. $\phi(\Pi') \triangleright \phi(\Gamma') \vdash_{\mathtt{i}} M : \phi(A')$.*

For the sake of simplicity, from now on if $\bar{e}(\Pi, (E, V)) = (\Pi', (E', V'))$ we will also denote Π' by $\bar{e}(\Pi)$, E' by $\bar{e}(E)$ and V' by $\bar{e}(V)$.

3.3 The algorithm

In this subsection we present an algorithm $\mathtt{Solve}$ which tries to solve the system of constraints generated by $\mathtt{PD_i^{min}}(M)$. A solution, if it exists, is found by interleaving unification rules and expansions. In $\mathtt{Solve}$ we make use of the following notations:

- Let S be a set of equivalences. $\mathtt{Fail}(S) = \mathtt{true}$ iff S is in unsolvable form.
- Let S be a set of equations. $\mathtt{Blocked}(S) = \mathtt{true}$ iff S is in blocked form.

Informally, the algorithm behaves as follows. $\mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$ is used to check if the constraints of V can be respected by E; in case of failure, there is no solution. Otherwise, the algorithm tries to solve E. If a blocked form is reached, a critical equation is dealt with using the operation of expansion. At every expansion, the system of constraints (E, V) is

Algorithm 1 The algorithm `Solve`. Input: a term M.

```
function Solve($M$)
    $(\Pi, (E, V)) \leftarrow \mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M)$
    $C \leftarrow \mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$
    if Fail($C$) then FAIL
    end if
    $E^* \leftarrow \mathtt{QuasiUnify}_{\doteq}(E)$
    while Blocked($E^*$) do
        choose $(\sigma \doteq \tau) \in E^*$ such that $|\sigma| \neq |\tau|$
        if $|\sigma| > |\tau|$ then
            $n \leftarrow |\sigma| - |\tau|$
            $(\Pi, (E, V)) \leftarrow \mathtt{Expand}(\tau, n)(\Pi, (E, V))$
        else
            $n \leftarrow |\tau| - |\sigma|$
            $(\Pi, (E, V)) \leftarrow \mathtt{Expand}(\sigma, n)(\Pi, (E, V))$
        end if
        $E^* \leftarrow \mathtt{QuasiUnify}_{\doteq}(E)$
    end while
    $V^* \leftarrow \mathtt{QuasiUnify}_{\approx}(\vec{E^*}(V))$
    $S \leftarrow \mathtt{QuasiUnify}_{\doteq}(E^* \cup V^*/_{\doteq})$
    return $(\Pi, \vec{S})$
end function
```

recomputed, then $\mathtt{QuasiUnify}_{\doteq}(E)$ is applied again; the loop stops as soon as a solved form is reached. Note that since the system of constraints is recomputed after each expansion, the only purpose of the calls to $\mathtt{QuasiUnify}_{\doteq}(E)$ in the while loop is to expose critical equations, thus guiding the expansions. Lastly, the final calls to $\mathtt{QuasiUnify}_{\approx}$ and $\mathtt{QuasiUnify}_{\doteq}$ generate the unifying substitution. The algorithm is non-deterministic, as critical equations are randomly chosen. Remark that, in an actual implementation, the efficiency of `Solve` could be improved by avoiding unnecessary and time consuming steps, such as recomputing the system of constraints from scratch after each expansion and repeating the whole unification procedure: this would require keeping track of the modified/added constraints only. Here we favoured both clarity and brevity of exposition over efficiency; we leave more detailed considerations about the time complexity, as well as possible optimizations, to future work. In what follows we prove that the algorithm behaves correctly.

▶ Remark 21. `Solve` does not find all possible solutions of $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M)$, since expansions are used only when strictly necessary (i.e. when the set of equations E^* is in blocked form). Note also that not all cases considered in Definition 18 are actually needed for the purposes of the algorithm: in particular case 4 never applies, but has been included for completeness.

Let A be an intersection type. The syntactic tree of A, written $T(A)$, is a tree defined inductively as follows: if $A = a$, then $T(a)$ is a single node named a; otherwise, if $A = [A_1, ..., A_n] \to B$, $T(A)$ is an ordered tree whose root, labelled A, has $n+1$ children, namely $T(A_1), ..., T(A_n)$ and $T(B)$. Two subtypes of A and B are *corresponding* if they occur at the same path in $T(A)$ and $T(B)$.

▶ **Lemma 22.** *Let* $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M) = (\Pi \triangleright \Gamma \vdash_{\mathtt{q}} M : A, (E, V))$ *and* $C = \mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$. *If C is in unsolvable form then* $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M)$ *is unsolvable.*

Proof. C in unsolvable form means that it contains a circular constraint, let it be $b \approx B$, such that b occurs in B. If E is unsolvable, i.e. b and B are corresponding subtypes of two equated intersection types, then the result is obtained by observing that no amount of expansions can get rid of the circular equation: indeed, an expansion can add new constraints, but not erase the existing ones. On the other hand, assume E solvable, i.e. there is a sequence s of expansions and applications of unification rules such that $s(E)$ is in solved form. Then the circularity has been introduced by the constraints in V, that is, b and B must be equivalent to each other. But by definition of $\sim$, it does not exist ϕ such that $\phi(b) \sim \phi(B)$, because if b occurs in B, $\phi(b)$ and $\phi(B)$ have syntactic trees of different depths, for all ϕ; again, no amount of expansions can get rid of the circularity. ◀

▶ **Theorem 23** (Partial Correctness). *Let* $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi \rhd \Gamma \vdash_{\mathtt{q}} M : A, (E, V))$ *and assume* $\mathtt{Solve}(M)$ *terminates. If the result is FAIL then* M *is not typable in* $\mathcal{U}$*; otherwise the algorithm outputs* $(\Sigma \rhd \Gamma' \vdash_{\mathtt{q}} M : A', \phi)$ *such that* $\phi(\Sigma) \rhd \phi(\Gamma') \vdash_{\mathtt{i}} M : \phi(A')$.

Proof. Let $C = \mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$. Then C is either in solvable or unsolvable form. In the latter case $\mathtt{Solve}$ immediately returns *FAIL*, and the result follows by Lemma 22. If C is in solved form, the algorithm continues by performing an interleaved sequence s of unification rules and expansions. First we show that, for all sequences s, both $\mathtt{QuasiUnify}_{\approx}(s(E)/_{\approx} \cup s(V))$ and $\mathtt{QuasiUnify}_{\doteq}(s(E))$ do no not contain circular constraints. The unification rules play no role in the introduction or elimination of circular constraints, hence we limit our analysis to the sequence of expansions $\bar{e}$ belonging to s. It is easy to check that no circularity can be introduced as a result of an expansion: indeed, each expansion creates disjoint replicas of sub-pseudo-derivations which were part of $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M)$, thus it can only originate fresh copies of constraints that already existed from the beginning. We conclude by observing that the equations that can be generated during a call to $\mathtt{QuasiUnify}_{\doteq}(\bar{e}(E))$ form a subset of the constraints that can be generated during a call to $\mathtt{QuasiUnify}_{\approx}(\bar{e}(E)/_{\approx} \cup \bar{e}(V))$. The last thing to do is proving that if the check on C does not fail, then $\mathtt{Solve}$ outputs a pair (Σ, ϕ) with the desired properties. Since we assumed that $\mathtt{Solve}(M)$ terminates, the number of iterations of the while loop is finite; this means that the algorithm builds a principal derivation $(\Sigma, (E_\Sigma, V_\Sigma))$ such that $E^*_\Sigma = \mathtt{QuasiUnify}_{\doteq}(E_\Sigma)$ is in solved form. Clearly $\mathtt{dom}(\vec{E^*_\Sigma}) \cap \mathtt{Var}(\vec{E^*_\Sigma}(V_\Sigma)) = \emptyset$, so $V^* = \mathtt{QuasiUnify}_{\approx}(\vec{E^*_\Sigma}(V_\Sigma))$ induces a substitution $\vec{V^*}$ whose domain and codomain are disjoint from the domain of $\vec{E^*_\Sigma}$, that is, $(\mathtt{dom}(\vec{V^*}) \cup \mathtt{cod}(\vec{V^*})) \cap \mathtt{dom}(\vec{E^*_\Sigma}) = \emptyset$. Therefore $S = \mathtt{QuasiUnify}_{\doteq}(E^*_\Sigma \cup V^*/_{\doteq})$ is in solved form. As $\vec{E^*_\Sigma}$ solves E^*_Σ and $\vec{V^*}$ satisfies V_Σ, it is easy to verify that $\phi = \vec{S}$ is a solution of E_Σ w.r.t V_Σ. Then $\phi(\Sigma)$ is a correct derivation, because (E_Σ, V_Σ) consists of all and only the constraints needed to convert a pseudo-derivation into an actual derivation. ◀

The following two results guarantee that the algorithm does not perform an infinite sequence of expansions and unification rules. Moreover they show that expansions and β-reductions are closely related.

▶ **Lemma 24.** *If* M *is in normal form then* $\mathtt{Solve}(M)$ *stops without performing any expansion.*

Proof. By induction on M. Recall that a normal form is defined by the following grammar:

$$M, N \quad ::= \quad \lambda x.M \mid x \underbrace{M...M}_{n} \quad n \geq 0$$

If $M = x$ the proof is trivial, as $E = \emptyset$. $M = \lambda x.N$ follows by induction, as no new equation is added. Lastly, consider the case $M = xM_1...M_n$ for $n > 0$. Let $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(x) = (\Pi_0 \rhd x : [a_0] \vdash_{\mathtt{q}} x : a_0, (\emptyset, \emptyset))$ and $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M_i) = (\Pi_i, (E_i, V_i))$ where $\Pi_i \rhd \Gamma_i \vdash_{\mathtt{q}} M_i : A_i$ $(1 \leq i \leq n)$. Then $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi, (E, V))$, where $E = \bigcup_{1 \leq i \leq n} E_i \cup \{a_0 \doteq [A_1] \to a_1,\ a_1 \doteq [A_2] \to a_2,\ ...,\ a_{n-1} \doteq [A_n] \to a_n\}$ and $a_1, \ldots, a_n$ are fresh. If the check on $\mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$ fails, the algorithm immediately stops; otherwise it computes $\mathtt{QuasiUnify}_{\doteq}(E)$. Since by induction E_i does not generate critical equations, neither does E, because a_0 does not occur in E_i and $a_1, \ldots, a_n$ are fresh. As $\mathtt{QuasiUnify}_{\doteq}(E)$ does not contain critical equations, the algorithm exits the while loop without performing any expansion. ◀

▶ **Theorem 25** (Partial Termination). *Let M be strongly normalizing. Then* $\mathtt{Solve}(M)$ *terminates.*

Proof. The proof is in the Appendix. ◀

4 Some examples

▶ **Example 26.** In order to understand the necessity and the behaviour of expansions, consider the term $M = ((\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp))(tz)$. The associated minimal principal derivation is $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi_M, (E, V))$, where:

- Π is:

$$\dfrac{\dfrac{\dfrac{\dfrac{y : [a_1] \vdash_{\mathtt{q}} y : a_1 \qquad x : [b_1] \vdash_{\mathtt{q}} x : b_1}{y : [a_1], x : [b_1] \vdash_{\mathtt{q}} yx : c_1} \qquad x : [b_2] \vdash_{\mathtt{q}} x : b_2}{y : [a_1], x : [b_1, b_2] \vdash_{\mathtt{q}} yxx : d_1}}{y : [a_1] \vdash_{\mathtt{q}} \lambda x.yxx : [b_1, b_2] \to d_1}}{\Pi \rhd\ \vdash_{\mathtt{q}} \lambda y.\lambda x.yxx : [a_1] \to [b_1, b_2] \to d_1}$$

  Σ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{w : [e_1] \vdash_{\mathtt{q}} w : e_1 \qquad p : [f_1] \vdash_{\mathtt{q}} p : f_1}{w : [e_1], p : [f_1] \vdash_{\mathtt{q}} wp : g_1} \qquad p : [f_2] \vdash_{\mathtt{q}} p : f_2}{w : [e_1], p : [f_1, f_2] \vdash_{\mathtt{q}} wpp : h_1}}{w : [e_1], p : [f_1, f_2] \vdash_{\mathtt{q}} \lambda r.wpp : [k_1] \to l_1}}{\Sigma \rhd w : [e_1] \vdash_{\mathtt{q}} \lambda p.\lambda r.wpp : [f_1, f_2] \to [k_1] \to l_1}$$

  Θ_1 is:

$$\dfrac{t : [m_1] \vdash_{\mathtt{q}} t : m_1 \qquad z : [n_1] \vdash_{\mathtt{q}} z : n_1}{\Theta_1 \rhd t : [m_1], z : [n_1] \vdash_{\mathtt{q}} tz : o_1}$$

  Θ_2 is a disjoint instance of Θ_1, and lastly Π_M is:

$$\dfrac{\dfrac{\Pi \qquad \Sigma}{\vdash_{\mathtt{q}} (\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp) : [b_1, b_2] \to d_1} \qquad \Theta_1 \qquad \Theta_2}{\Pi_M \rhd w : [e_1], t : [m_1, m_2], z : [n_1, n_2] \vdash_{\mathtt{q}} ((\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp))(tz) : d_1}$$

- $E = \{a_1 \doteq [b_1] \to c_1,\ c_1 \doteq [b_2] \to d_1,\ e_1 \doteq [f_1] \to g_1,\ g_1 \doteq [f_2] \to h_1,\ a_1 \doteq [f_1, f_2] \to [k_1] \to l_1, m_1 \doteq [n_1] \to o_1,\ m_2 \doteq [n_2] \to o_2,\ b_1 \doteq o_1, b_2 \doteq o_2\}$.
- $V = \{b_1 \approx b_2,\ f_1 \approx f_2,\ m_1 \approx m_2,\ n_1 \approx n_2\}$.

Applying the unification rules to E, we obtain equation $[b_1] \to c_1 \doteq [f_1, f_2] \to [k_1] \to l_1$; decomposing yields the critical equation $[b_1] \doteq [f_1, f_2]$. Therefore we need to perform $\texttt{Expand}(\sigma, n)$ with $\sigma = [b_1]$ and $n = 1$. Under the action of the expansion, the derivation Π becomes:

$$\dfrac{\dfrac{\dfrac{\dfrac{y : [a_1] \vdash_{\mathtt{q}} y : a_1 \qquad x : [b_1] \vdash_{\mathtt{q}} x : b_1 \qquad x : [b_3] \vdash_{\mathtt{q}} x : b_3}{y : [a_1], x : [b_1, b_3] \vdash_{\mathtt{q}} yx : c_1} \qquad x : [b_2] \vdash_{\mathtt{q}} x : b_2}{y : [a_1], x : [b_1, b_3, b_2] \vdash_{\mathtt{q}} yxx : d_1}}{y : [a_1] \vdash_{\mathtt{q}} \lambda x.yxx : [b_1, b_3, b_2] \to d_1}}{\Pi \triangleright \vdash_{\mathtt{q}} \lambda y.\lambda x.yxx : [a_1] \to [b_1, b_3, b_2] \to d_1}$$

Consequently Π_M becomes:

$$\dfrac{\dfrac{\Pi \qquad \Sigma}{\vdash_{\mathtt{q}} (\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp) : [b_1, b_3, b_2] \to d_1} \qquad \Theta_1 \quad \Theta_2}{\Pi_M \triangleright w : [e_1], t : [m_1, m_2], z : [n_1, n_2] \vdash_{\mathtt{q}} ((\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp))(tz) : d_1}$$

At this point Π_M is no longer a principal derivation for M, as its last rule is incorrect. But the expansion is not terminated yet: there is one applied bound variable in the context of the duplicated subderivation, namely x. Therefore we enter the while loop with $\mathcal{L} = \{x\}$, and since the application rule for x is the last rule of Π_M, we update the derivation as follows:

$$\dfrac{\dfrac{\Pi \qquad \Sigma}{\vdash_{\mathtt{q}} (\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp) : [b_1, b_3, b_2] \to d_1} \qquad \Theta_1 \quad \Theta_3 \quad \Theta_2}{\Pi_M \triangleright w : [e_1], t : [m_1, m_3, m_2], z : [n_1, n_3, n_2] \vdash_{\mathtt{q}} ((\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp))(tz) : d_1}$$

where Θ_3 is yet another disjoint instance of Θ_1. No applied bound variable is in the context of Θ_i, so $\mathcal{L} = \emptyset$ and we exit the while loop. Lastly we recompute the constraints, which now are:

- $E = \{a_1 \doteq [b_1, b_3] \to c_1,\ c_1 \doteq [b_2] \to d_1,\ e_1 \doteq [f_1] \to g_1,\ g_1 \doteq [f_2] \to h_1,\ a_1 \doteq [f_1, f_2] \to [k_1] \to l_1, m_1 \doteq [n_1] \to o_1,\ m_2 \doteq [n_2] \to o_2,\ m_3 \doteq [n_3] \to o_3,\ b_1 \doteq o_1, b_2 \doteq o_2,\ b_3 \doteq o_3\}$.
- $V = \{b_1 \approx b_2,\ b_2 \approx b_3,\ f_1 \approx f_2,\ m_1 \approx m_2,\ m_2 \approx m_3,\ n_1 \approx n_2,\ n_2 \approx n_3, \dots\}$.

The reader can infer the omitted equivalence constraints by considering the equivalence relation induced by V. Note that all the constraints generated by Θ_i have been duplicated. Now E can be reduced to solved form by decomposing equation $[b_1, b_3] \doteq [f_1, f_2]$.

▶ **Example 27.** Let $M = \lambda x.xx$, which is not simply typable. $\texttt{PD}^{\texttt{min}}_{\texttt{i}}(M) = (\Pi_M, (E, V))$ where:

$$\dfrac{\dfrac{x : [a] \vdash_{\mathtt{q}} x : a \qquad x : [b] \vdash_{\mathtt{q}} x : b}{x : [a, b] \vdash_{\mathtt{q}} xx : c}}{\Pi_M \triangleright \vdash_{\mathtt{q}} \lambda x.xx : [a, b] \to c}$$

$E = \{a \doteq [b] \to c\}$ and $V = \{a \approx b\}$. Clearly E is unsolvable w.r.t. V.

5 From simple types to intersection types and viceversa

In this section we will prove the main result of the paper, namely the correspondence between the two systems $\mathcal{S}$ and $\mathcal{U}$. First we define a translation from intersection types to simple types, which erases the multisets in $\mathcal{I}$.

▶ **Definition 28.** *The collapse translation t from $\mathcal{I}$ to $\mathcal{T}_\mathcal{S}$ is defined by induction on the size of types in the following way (recall that multisets are now ordered):*

$$\begin{array}{lcl} t(a) & = & a; \\ t(\sigma \to A) & = & t(\sigma) \to t(A); \\ t([A_1, ..., A_n]) & = & t(A_1). \end{array}$$

The following property is easy to prove, by induction on types.

▶ **Property 29.** *$A \sim B$ implies $t(A) = t(B)$.*

Observe that in the translation of an (ordered) multiset we could have chosen any of its elements, in a nondeterministic way. However, always choosing the first one allows us to easily extend the translation to pseudo-derivations and derivations.

▶ **Definition 30.**

- *$t(.)$ is extended to contexts by posing $t(\Gamma) = \{x : t(\sigma) \mid x : \sigma \in \Gamma\}$.*
- *$t(.)$ is extended to pseudo-derivations in the following way:*
  - *if $\Pi \rhd x : [a] \vdash_{\mathtt{q}} x : a$, then $t(\Pi) \rhd x : t([a]) \vdash_{\mathtt{p}} x : t(a)$;*
  - *the case the subject is $\lambda x.M$ is straightforward.*
  - *let Π be:*

$$\frac{\Gamma \vdash_{\mathtt{q}} M : A \qquad (\Delta_i \vdash_{\mathtt{q}} N : B_i)_{1 \leq i \leq n}}{\Gamma \uplus_{1 \leq i \leq n} \Delta_i \vdash_{\mathtt{q}} MN : C} \ (\to_{\mathtt{E}j})$$

then $t(\Pi)$ is:

$$\frac{t(\Gamma) \vdash_{\mathtt{p}} M : t(A) \qquad t(\Delta_1) \vdash_{\mathtt{p}} N : t(B_1)}{t(\Gamma) \cup t(\Delta_1) \vdash_{\mathtt{p}} MN : t(C)} \ (\to_{\mathtt{E}j})$$

- *$t(.)$ can be extended to principal derivations by posing $t(\Pi, (E, V)) = (t(\Pi), (E_{t(\Pi)}, V_{t(\Pi)}))$, where the system of constraints $(E_{t(\Pi)}, V_{t(\Pi)})$ is computed from the pseudo-derivation $t(\Pi)$ according to Fig. 2, as usual.*
- *$t(.)$ is extended to derivations in the same way as for pseudo-derivations.*

▶ **Property 31.** *Let $\mathtt{PD_i}(M) = (\Pi, (E, V))$. For each sequence of expansion $\bar{e}$, $\mathtt{PD}(M) = (t(\bar{e}(\Pi)), (E_{t(\bar{e}(\Pi))}, V_{t(\bar{e}(\Pi))}))$.*

Proof. In case $\bar{e}$ is the empty sequence the proof follows by induction on the definition of $\mathtt{PD}(M)$ and $\mathtt{PD_i}(M)$, recalling that $\mathtt{PD}(M)$ is defined modulo renaming of type variables. The general case follows from the definition of expansion. ◀

We are now able to prove the decidability of $\mathcal{U}$, and the fact that the systems $\mathcal{S}$ and $\mathcal{U}$ have the same typability power.

▶ **Theorem 32.** *The system $\mathcal{U}$ is decidable.*

Proof. Let $\mathtt{PD_i^{min}}(M) = (\Pi, (E, V))$ and $\mathtt{PD}(M) = (t(\Pi), (E_{t(\Pi)}, V_{t(\Pi)}))$. We already proved that $\mathtt{Solve}(M)$ terminates with a correct result in case M is strongly normalizing (Theorems 23 and 25). If M is not strongly normalizing, then it is not simply typable, i.e. $\mathtt{Unify}(E_{t(\Pi)} \cup V_{t(\Pi)})$ fails. Observe that $\mathtt{QuasiUnify}_{\approx}$ decomposes multiset equivalences by introducing equivalence constraints between all possible pairs of elements, hence it generates a circular constraint exactly when $\mathtt{Unify}$ does. We conclude that the failure of $\mathtt{Unify}(E_{t(\Pi)} \cup V_{t(\Pi)})$ implies the failure of the check on $\mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$ performed at the beginning of the algorithm $\mathtt{Solve}$. ◀

▶ **Corollary 33.** $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M)$ *is solvable if and only if* $\mathtt{PD}(M)$ *is solvable.*

Proof. Let $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M) = (\Pi, (E, V))$. $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M)$ is solvable if and only if (by Theorem 23) the check on $\mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$ succeeds if and only if (looking at the proof of Theorem 32) $\mathtt{PD}(M)$ is solvable. ◀

The main result

The main result of this paper is the correspondence between the derivations in the two systems $\mathcal{S}$ and $\mathcal{U}$: from one side the translation of every derivation in $\mathcal{U}$ is a derivation in $\mathcal{S}$, and on the other side for each derivation Π in $\mathcal{S}$ there is an infinite set of derivations in $\mathcal{U}$ from which Π can be obtained through the translation.

▶ **Theorem 34.**

1. $\Gamma \vdash_{\mathtt{i}} M : A$ *implies* $t(\Gamma) \vdash M : t(A)$*;*
2. $\Pi \triangleright \Gamma \vdash M : A$ *implies there exists* $\Pi_{\mathtt{i}} \triangleright \Gamma_{\mathtt{i}} \vdash_{\mathtt{i}} M : A_{\mathtt{i}}$ *such that* $\Gamma = t(\Gamma_{\mathtt{i}})$ *and* $A = t(A_{\mathtt{i}})$*.*

Proof.

1. Easy, by definition of collapse translation.
2. First we consider the case $\mathtt{dom}(\Gamma) = \mathtt{FV}(M)$. Let $\mathtt{PD}(M) = (\Sigma, (E_\Sigma, V_\Sigma))$; clearly $\Pi \triangleright \Gamma \vdash M : A$ implies that there is θ m.g.u. of $E_\Sigma \cup V_\Sigma$, and there is θ' such that $\theta' \circ \theta(\Sigma) = \Pi$. Moreover, by Corollary 33, $\mathtt{Solve}(M) = (\Sigma_{\mathtt{i}}, \phi)$; let $(E_{\Sigma_{\mathtt{i}}}, V_{\Sigma_{\mathtt{i}}})$ be the system of constraints associated to $\Sigma_{\mathtt{i}}$. It is easy to verify that $r \circ t \circ \phi(\Sigma_{\mathtt{i}}) = \theta(\Sigma)$ for some renaming of type variables $r : \mathtt{Var} \to \mathtt{Var}$ (trivially extended to derivations): this follows from the fact that both systems of constraints, namely (E_Σ, V_Σ) for simple types and $(E_{\Sigma_{\mathtt{i}}}, V_{\Sigma_{\mathtt{i}}})$ for uniform intersection types, contain all and only the constraints strictly needed to type M in the corresponding type system. Then, choosing any substitution $\phi' : \mathtt{Var} \to \mathcal{T}_{\mathcal{U}}$ such that $t \circ \phi' = \theta'$, one can build $\Pi_{\mathtt{i}} = \phi' \circ r \circ \phi(\Sigma_{\mathtt{i}})$. In case there is $x : B \in \Gamma$ such that $x \notin \mathtt{FV}(M)$, starting from the above construction and suitably exploiting ($\mathtt{var}$) rules it is always possible to obtain a derivation $\Pi_{\mathtt{i}}$ such that $x : \sigma \in \Gamma_{\mathtt{i}}$ and $t(\sigma) = B$. ◀

6 Conclusion

Starting from a non-idempotent intersection type assignment system which is undecidable, as it characterises strong normalisation, we have built a decidable restriction of it. More precisely, we designed an algorithm `Solve` that, given in input a λ-term, outputs either FAIL, if the term cannot be typed, or the most general typing for it. System $\mathcal{U}$ has the same typability power as the simple type assignment system, and the derivations in the two systems are related through a translating function. Furthermore, the system is quantitative, in the sense that some information about the number of occurrences of a variable in the subject can be deduced from a derivation in system $\mathcal{U}$. In the future we plan to use this new system to reason about properties related to the complexity of reductions of simply typed terms. Moreover we would like to investigate the existence of further decidable restrictions with interesting properties. From a technical point of view, we also aim at studying the time and space complexity of `Solve` in more detail, in order to come up with an efficient implementation.

References

1 Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundation of mathematics*. North-Holland, Amsterdam, 1984.

2 Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter model and the completeness of type assignment. *J. Symb. Log.*, 48(4):931–940, 1983. doi:10.2307/2273659.

3 Choukri-Bey Ben-Yelles. *Type-assignment in the lambda-calculus; syntax and semantics.* PhD thesis, University of Wales Swansea, 1979.

4 Erika De Benedetti and Simona Ronchi Della Rocca. A type assignment for lambda-calculus complete for fptime and strong normalization. *Inf. Comput.*, 248(195-214):195–214, 2016. doi:10.1016/j.ic.2015.12.012.

5 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL 2007*, volume 4646, pages 298–312, 2007.

6 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Inhabitation for non-idempotent intersection types. *Log. Methods Comput. Sci.*, 14(3), 2018. doi:10.23638/LMCS-14(3:7)2018.

7 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.

8 Alonzo Church. A formulation of simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

9 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Form. Log.*, 21(4):685–693, 1980.

10 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Math. Log. Q.*, 27(2-6):45–58, 1981.

11 Mario Coppo and Paola Giannini. Principal types and unification for a simple intersection type system. *Inf. Comput.*, 122(1):70–96, 1995.

12 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

13 Roger J. Hyndley. *Basic Simple Type theory.* Hoepli, 1997.

14 Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In *TCS*, LNCS, 2014.

15 Assaf J. Kfoury and Joe B. Wells. Principality and decidable type inference for finite-rank intersection types. In Andrew W. Appel and Alex Aiken, editors, *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*, pages 161–174. ACM, 1999.

16 Assaf J. Kfoury and Joe B. Wells. Principality and type inference for intersection types using expansion variables. *Theor. Comput. Sci.*, 311(1-3):1–70, 2004.

17 Luca Paolini, Mauro Piccolo, and Simona Ronchi Della Rocca. Essential and relational models. *Mathematical Strucures in Computer Science*, 27, 2017.

18 John A. Robinson. A machine-oriented logic based on the resolution principle. *J. Asoc. for Computing Machinery 12 (1965)*, 12:23–41, 1965.

19 Simona Ronchi Della Rocca. Principal type scheme and unification for intersection type discipline. *Theor. Comput. Sci.*, 59:181–209, 1988. doi:10.1016/0304-3975(88)90101-6.

20 Simona Ronchi Della Rocca and Luca Paolini. *The Parametric Lambda Calculus - A Metamodel for Computation.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

21 Simona Ronchi Della Rocca and Betti Venneri. Principal type schemes for an extended type theory. *Theor. Comput. Sci.*, 28:151–169, 1984. doi:10.1016/0304-3975(83)90069-5.

22 Pawel Urzyczyn. The emptiness problem for intersection types. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*, pages 300–309. IEEE Computer Society, 1994. doi:10.1109/LICS.1994.316059.

A Proof of Theorem 25

Proof (Hint). In order to develop some useful intuitions, we start by considering M that reduces to N in one step, by reducing one of its innermost redexes; we denote this fact by writing $M \to_\beta^{\mathtt{in}} N$. Let $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M) = (\Pi_M, (E_M, V_M))$, $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(N) = (\Pi_N, (E_N, V_N))$, $M = \mathtt{C}[(\lambda x.P)Q]$ and $N = \mathtt{C}[P[Q/x]]$. Assume $\mathtt{QuasiUnify}_\approx(E_M/_\approx \cup V_M)$ is in solved form, termination being obvious in the other case. We will prove that there is a run of $\mathtt{Solve}(M)$ performing an interleaved sequence s of unification rules and expansions such that $E_N \subset s(E_M)$ and $V_N \subset \bar{e}(V_M)$, where $\bar{e}$ is the sequence of expansions belonging to s. At the same time we will show that $s(E_M) - E_N$ is in solved form and cannot generate critical equations. The proof depends on the occurrences of x in P.

x does not occur in P. Let Π_M contain subderivations $\Pi^j_{(\lambda x.P)Q}$ $(j \in J)$ of shape:

$$(\Pi^j_{(\lambda x.P)Q}) \qquad \dfrac{\dfrac{\Gamma^j \vdash_{\mathtt{q}} P : B^j}{\Pi^j_{\lambda x.P} \rhd \Gamma^j \vdash_{\mathtt{q}} \lambda x.P : [a^j] \to B^j} \qquad \Pi^j_Q \rhd \Delta^j \vdash_{\mathtt{q}} Q : A^j}{\Gamma^j \uplus \Delta^j \vdash_{\mathtt{q}} (\lambda x.P)Q : B^j}$$

where $(\Pi^j_{\lambda x.P}, (E^j_{\lambda x.P}, V^j_{\lambda x.P}))$ and $(\Pi^j_Q, (E^j_Q, V^j_Q))$ are disjoint instances of $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(\lambda x.P)$ and $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(Q)$ respectively. Moreover let $E_M = E' \cup_{j\in J} (E^j_{\lambda x.P} \cup E^j_Q \cup E^j_{\mathrm{app}})$ and $V_M = V' \cup_{j\in J} (V^j_{\lambda x.P} \cup V^j_Q \cup V^j_{\mathrm{app}})$, where $E^j_{\mathrm{app}} = \{a^j \doteq A^j\}$ and $V^j_{\mathrm{app}} = \{b^j \approx c^j \mid y \in \mathtt{dom}(\Gamma^j) \cap \mathtt{dom}(\Delta^j), b^j \in \Gamma^j(y), c^j \in \Delta^j(y)\}$. First, observe that $E_N = E' \cup_{j\in J} E^j_P = E' \cup_{j \in J} E^j_{\lambda x.P}$ and $V_N = V' \cup_{j\in J} V^j_P \subset V_M$. Since Q is in normal form, by Lemma 24, for all j we know that $E^{j*}_Q = \mathtt{QuasiUnify}_{\doteq}(E^j_Q)$ is in solved form. Let $\bar{u}$ be the sequence of unification rules performed by all $\mathtt{QuasiUnify}_{\doteq}(E^j_Q)$ $(j \in J)$; clearly each a^j occurs only in E^j_{app}, hence $\bar{u}(E_M) - E_N = \bigcup_{j\in J} E^{j*}_Q \cup E^j_{\mathrm{app}}$ is in solved form and cannot generate critical equations. Note that in this case $\bar{e}$ is the empty sequence.

x occurs in P. Let Π_M contain subderivations $\Pi^j_{(\lambda x.P)Q}$ $(j \in J)$ ending by:

$$(\Pi^j_{(\lambda x.P)Q}) \qquad \dfrac{\dfrac{\Gamma^j, x : [a^j_i]_{i\in I^j} \vdash_{\mathtt{q}} P : B^j}{\Gamma^j \vdash_{\mathtt{q}} \lambda x.P : [a^j_i]_{i\in I^j} \to B^j} \qquad (\Delta^j_i \vdash_{\mathtt{q}} Q : A^j_i)_{i\in I^j}}{\Gamma^j \uplus_{i\in I^j} \Delta^j_i \vdash_{\mathtt{q}} (\lambda x.P)Q : B^j}$$

where I^j is a suitable set of indexes. Focusing on the occurrences of x in functional position in P, each $\Pi^j_{(\lambda x.P)Q}$ contains disjoint subderivations of shape:

$$(\Sigma^j_i) \qquad \dfrac{x : [a^j_i] \vdash_{\mathtt{q}} x : a^j_i \qquad \Psi^j_i \vdash_{\mathtt{q}} R_i : C^j_i}{\{x : [a^j_i]\} \uplus \Psi^j_i \vdash_{\mathtt{q}} xR_i : e^j_i}$$

where $i \in I^j_{\mathrm{fun}} \subseteq I^j$ (recall that, by construction, each application uses a different premise on x). The set of equations generated by the last rule of all the aforementioned Σ^j_i is:

$$E_{\mathrm{fun}} = \{a^j_i \doteq [C^j_i] \to e^j_i \mid j \in J, i \in I^j_{\mathrm{fun}}\}$$

Focusing instead on the occurrences of x in argument position, each $\Pi^j_{(\lambda x.P)Q}$ contains disjoint subderivations of shape:

$$(\Theta^j_k) \qquad \dfrac{\Phi^j_k \vdash_{\mathtt{q}} S_k : D^j_k \qquad (x : [a^j_i] \vdash_{\mathtt{q}} x : a^j_i)_{i\in I^j_k}}{\Phi^j_k \uplus_{i\in I^j_k} \{x : [a^j_i]\} \vdash_{\mathtt{q}} S_k x : F^j_k}$$

where $k \in K^j$ and the sets I^j_k form a partition of indexes, i.e. $I^j_k \cap I^j_{k'} = \emptyset$ if $k \neq k'$ and $\bigcup_{k\in K^j} I^j_k = I^j_{\text{arg}} = (I^j - I^j_{\text{fun}})$; observe that $|K^j| \leq |I^j_{\text{arg}}|$. Indeed, either the type of S_k is an arrow type, let it be $D^j_k = [d^j_i]_{i\in I^j_k} \to F^j_k$, or both $D^j_k = d^j_k$ and $F^j_k = f^j_k$ are type variables. Let $K^j_{\text{arr}} = \{k \in K^j \mid \text{the type of } S_k \text{ is an arrow type}\}$ and $K^j_{\text{var}} = K^j - K^j_{\text{arr}}$; note that $k \in K^j_{\text{var}}$ implies I^j_k is a singleton, hence K^j_{var} and $\bigcup_{k\in K^j_{\text{var}}} I^j_k$ can be identified. The last rule of all subderivations Θ^j_k generate equations:

$$E_{\text{arg}} = \{d^j_i \doteq a^j_i \mid j \in J,\, k \in K^j_{\text{arr}},\, i \in I^j_k\} \cup \{d^j_k \doteq [a^j_k] \to f^j_k \mid j \in J,\, k \in K^j_{\text{var}}\}$$

The sequence s varies depending on whether A^j_i is a type variable or not. In the former case no expansion is needed, i.e. s consists of unification rules only. We now analyse both cases in detail.

A^j_i is a variable. A^j_i is a variable means that Q is not an abstraction. Let $A^j_i = b^j_i$ and $E_M = E' \cup E_{\text{fun}} \cup E_{\text{arg}} \cup \{a^j_i \doteq b^j_i \mid j \in J,\, i \in I^j\}$. Consider the sequence of substitutions $\bar{u}$ replacing each a^j_i by b^j_i. These substitutions modify only the components E_{fun} and E_{arg}, which become:

$$E^*_{\text{fun}} = \{b^j_i \doteq [C^j_i] \to e^j_i \mid j \in J,\, i \in I^j_{\text{fun}}\}$$
$$E^*_{\text{arg}} = \{d^j_i \doteq b^j_i \mid j \in J,\, k \in K^j_{\text{arr}},\, i \in I^j_k\} \cup \{d^j_k \doteq [b^j_k] \to f^j_k \mid j \in J,\, k \in K^j_{\text{var}}\}$$

Observe that $E_N = E' \cup E^*_{\text{fun}} \cup E^*_{\text{arg}} \subset \bar{u}(E_M)$, because in Π_N the subderivations corresponding to Σ^j_i and Θ^j_k are, respectively:

$$(\Sigma'^j_i) \qquad \frac{\Delta^j_i \vdash_{\mathtt{q}} Q : b^j_i \qquad \Psi'^j_i \vdash_{\mathtt{q}} R_i[Q/x] : C^j_i}{\Delta^j_i \uplus \Psi'^j_i \vdash_{\mathtt{q}} Q(R_i[Q/x]) : e^j_i}$$

$$(\Theta'^j_k) \qquad \frac{\Phi'^j_k \vdash_{\mathtt{q}} S_k[Q/x] : D^j_k \qquad (\Delta^j_i \vdash_{\mathtt{q}} Q : b^j_i)_{i\in I^j_k}}{\Phi'^j_k \uplus_{i\in I^j_k} \Delta^j_i \vdash_{\mathtt{q}} (S_k[Q/x])Q : F^j_k}$$

The various a^j_i do not occur in Π_N, thus we also have $V_N = V_M - \{a^j_i \approx a^j_l \mid j \in J,\, i, l \in I^j\}$. We conclude that in this case $s = \bar{u}$, i.e. the sequence consists of the unification rules transforming $E_{\text{fun}} \cup E_{\text{arg}} \cup \{a^j_i \doteq b^j_i \mid j \in J,\, i \in I^j\}$ into $E^*_{\text{fun}} \cup E^*_{\text{arg}} \cup \{a^j_i \doteq b^j_i \mid j \in J,\, i \in I^j\}$. It is straightforward to check that $\bar{u}(E_M) - E_N = \{a^j_i \doteq b^j_i \mid j \in J,\, i \in I^j\}$ is in solved form, and that all variables a^j_i do not occur outside of $\bar{u}(E_M) - E_N$. Hence $\bar{u}(E_M) - E_N$ cannot play any role in generating critical equations.

A^j_i is not a variable. Since Q is in normal form, A^j_i not a variable means that Q is an abstraction; hence the reduction $\mathtt{C}[(\lambda x.P)Q] \to^{\mathtt{in}}_{\beta} \mathtt{C}[P[Q/x]]$ may generate new redexes. Let $A^j_i = \sigma^j_{i,0} \to \ldots \to \sigma^j_{i,q} \to c^j_i$ for some $q \geq 0$, and let $|\sigma^j_{i,p}| = n^j_{i,p}$ $(0 \leq p \leq q)$. For $j \in J, i \in I^j_{\text{fun}}$ and $p^j_i \leq q$, let $\sigma^j_{i,0}\ldots,\sigma^j_{i,p^j_i}$ be associated to applied bound variables in Π_N; then the application of unification rules generates one critical equation for each $\sigma^j_{i,p}$ such that $p \leq p^j_i$ and $n^j_{i,p} > 1$. During the while loop, `Solve` can choose exclusively these critical equations and perform the appropriate expansions. For any given $j \in J,\, i \in I^j_{\text{fun}}$, a single expansion $\mathtt{Expand}([C^j_i], n^j_{i,0} - 1)$ transforms Σ^j_i into:

$$(\widehat{\Sigma}^j_i) \qquad \frac{x : [a^j_i] \vdash_{\mathtt{q}} x : a^j_i \qquad (\Psi^j_{i,m} \vdash_{\mathtt{q}} R_i : C^j_{i,m})_{1\leq m\leq n^j_{i,0}}}{\{x : [a^j_i]\} \uplus_{1\leq m\leq n^j_{i,0}} \Psi^j_{i,m} \vdash_{\mathtt{q}} xR_i : e^j_i}$$

Similarly, with the convention that $e^j_i = e^j_{i,1}$, an expansion for $1 \leq p \leq p^j_i$ transforms a subderivation like:

$$(\Omega^j_{i,p}) \qquad \frac{\Xi^j_{i,p} \vdash_{\mathtt{q}} U_{i,p} : e^j_{i,p} \qquad \Upsilon^j_{i,p} \vdash_{\mathtt{q}} V_{i,p} : G^j_{i,p}}{\Xi^j_{i,p} \uplus \Upsilon^j_{i,p} \vdash_{\mathtt{q}} U_{i,p}V_{i,p} : e^j_{i,p+1}}$$

into:

$$(\widehat{\Omega}^j_{i,p}) \qquad \frac{\Xi^j_{i,p} \vdash_{\mathtt{q}} U_{i,p} : e^j_{i,p} \qquad (\Upsilon^j_{i,p,m} \vdash_{\mathtt{q}} V_{i,p} : G^j_{i,p,m})_{1 \leq m \leq n^j_{i,p}}}{\Xi^j_{i,p} \uplus_{1 \leq m \leq n^j_{i,p}} \Upsilon^j_{i,p,m} \vdash_{\mathtt{q}} U_{i,p}V_{i,p} : e^j_{i,p+1}}$$

Observe that each espansion may increase the cardinality of the set of indexes I^j: think, for instance, of the term $P = x(xP')(P''x)$ when $Q : [a, a'] \to [b, b'] \to c$. This originates a sequence of expansions $\bar{e}$ and an increasing sequence $I^j_{e1}, I^j_{e2}, I^j_{e3}, \ldots$ of sets of indexes. Let $\widehat{I}^j$ denote the last set of the sequence: such a set must exist because functional occurrences of x may increment, to their right, the number of subderivations with subject x, but not vice versa. Let $\widehat{\Pi}_M = \bar{e}(\Pi_M)$ be derivation obtained after all the aforementioned expansions, and let $(\widehat{E}_M, \widehat{V}_M) = (\bar{e}(E_M), \bar{e}(V_M))$ be its associated constraints (w.l.o.g., we ignore the unification rules performed between expansions: they are not relevant in our proof because the system of constraints is recomputed after each expansion). In the same way as before, for each $j \in J$ it is possible to identify $\widehat{I}^j_{\text{fun}} \subseteq \widehat{I}^j$ and the partition $\{\widehat{I}^j_k \mid k \in \widehat{K}^j\}$ such that $\bigcup_{k \in \widehat{K}} \widehat{I}^j_k = \widehat{I}^j_{\text{arg}} = (\widehat{I}^j - \widehat{I}^j_{\text{fun}})$; with these subsets of indexes, the equations related to all occurrences of x in functional and argument position can be expressed respectively as:

$$\widehat{E}_{\text{fun}} = \{a^j_i \doteq [C^j_{i,0}, \ldots, C^j_{i,n^j_{i,0}}] \to e^j_i \mid j \in J,\, i \in \widehat{I}^j_{\text{fun}}\}$$
$$\widehat{E}_{\text{arg}} = \{d^j_i \doteq A^j_i \mid j \in J,\, k \in \widehat{K}^j_{\text{arr}},\, i \in \widehat{I}^j_k\} \cup \{d^j_k \doteq [A^j_k] \to f^j_k \mid j \in J,\, k \in \widehat{K}^j_{\text{var}}\}$$

Moreover, the equalities generated by the last elimination rule of the various $\widehat{\Omega}^j_{i,p}$ are (recall that $e^j_i = e^j_{i,1}$):

$$\widehat{E}_{\widehat{\Omega}} = \{e^j_{i,p} \doteq [G_{i,p,1}, \ldots, G_{i,p,n^j_{i,p}}] \to e^j_{i,p+1} \mid j \in J,\, i \in \widehat{I}^j_{\text{fun}},\, 1 \leq p \leq p^j_i\}$$

Now let $\sigma^j_{i,p} = [b^j_{i,p,1}, ..., b^j_{i,p,n^j_{i,p}}]$ $(0 \leq p \leq p^j_i)$ and observe that $\{a^j_i \doteq A^j_i \mid j \in J,\, i \in \widehat{I}^j\} \subset \widehat{E}_M$. Thanks to the now agreeing multiset cardinalities, the call to $\mathtt{QuasiUnify}_{\doteq}(\widehat{E}_M)$ on line 16 of the algorithm can finally apply a sequence $\bar{u}$ of unification rules that perform all substitutions involving the various $e^j_{i,p}$ and replace a^j_i by A^j_i, then decompose the resulting equations. Such a sequence transforms $\widehat{E}_{\text{fun}}$ and $\widehat{E}_{\text{arg}}$ into:

$$\widehat{E}^*_{\text{fun}} = \{b^j_{i,0,1} \doteq C^j_{i,1}, \ldots, b^j_{i,0,n^j_{i,0}} \doteq C^j_{i,n^j_{i,0}} \mid j \in J,\, i \in \widehat{I}^j_{\text{fun}}\}$$
$$\widehat{E}^*_{\text{arg}} = \{d^j_i \doteq A^j_i \mid j \in J,\, k \in \widehat{K}^j_{\text{arr}},\, i \in \widehat{I}^j_k\} \cup \{d^j_k \doteq [A^j_k] \to f^j_k \mid j \in J,\, k \in \widehat{K}^j_{\text{var}}\}$$

These equations are also part of E_N because, by construction, in Π_N the subderivations corresponding to $\widehat{\Sigma}^j_i$ and $\widehat{\Theta}^j_k$ $(j \in J,\, i \in \widehat{I}^j,\, k \in \widehat{K}^j)$ are, respectively:

$$(\Sigma'^j_i) \qquad \frac{\Delta^j_i \vdash_{\mathtt{q}} Q : \sigma^j_{i,0} \to \ldots \to \sigma^j_{i,q} \to c^j_i \qquad (\Psi'^j_{i,m} \vdash_{\mathtt{q}} R_i[Q/x] : C^j_{i,m})_{1 \leq m \leq n^j_{i,0}}}{\Delta^j_i \uplus_{1 \leq m \leq n^j_{i,0}} \Psi'^j_{i,m} \vdash_{\mathtt{q}} Q(R_i[Q/x]) : \sigma^j_{i,1} \to \ldots \to \sigma^j_{i,q} \to c^j_i}$$

where $\sigma^j_{i,0} = [b^j_{i,0,1}, ..., b^j_{i,0,n^j_{i,0}}]$, and

$$(\Theta'^j_k) \qquad \frac{\Phi'^j_k \vdash_{\mathtt{q}} S_k[Q/x] : D^j_k \qquad (\Delta^j_i \vdash_{\mathtt{q}} Q : A^j_i)_{i \in \widehat{I}^j_k}}{\Phi'^j_k \uplus_{i \in I^j_k} \Delta^j_i \vdash_{\mathtt{q}} (S_k[Q/x])Q : F^j_k}$$

Moreover, focusing on the equations that involve $\sigma^j_{i,p}$ $(1 \leq p \leq p^j_i)$, one can see that applying the multiset decompositions in $\bar{u}$ produces:

$$E_\sigma = \{b_{i,p,1} \doteq G_{i,p,1}, \dots, b_{i,p,n^j_{i,p}} \doteq G_{i,p,n^j_{i,p}} \mid j \in J,\, i \in \widehat{I}^j_{\text{fun}},\, 1 \leq p \leq p^j_i\}$$

Clearly the above equations are part of E_N too, as Π_N contains subderivations of shape:

$$(\Omega'^j_{i,p}) \quad \frac{\Xi'^j_{i,p} \vdash_{\mathtt{q}} U_{i,p}[Q/x] : \sigma^j_{i,p} \to \dots \to \sigma^j_{i,q} \to c^j_i \qquad (\Upsilon'^j_{i,p,m} \vdash_{\mathtt{q}} V_{i,p}[Q/x] : G^j_{i,p,m})_{1 \leq m \leq n^j_{i,p}}}{\Xi'^j_{i,p} \uplus_{1 \leq m \leq n^j_{i,p}} \Upsilon'^j_{i,p,m} \vdash_{\mathtt{q}} (U_{i,p}V_{i,p})[Q/x] : \sigma^j_{i,p+1} \to \dots \to \sigma^j_{i,q} \to c^j_i}$$

Recall that $\widehat{E}_M = \bar{e}(E_M)$ and $\widehat{V}_M = \bar{e}(V_M)$. Hence, ignoring the unification rules performed between expansions, we have $E_N = E' \cup \widehat{E}^*_{\text{fun}} \cup \widehat{E}^*_{\text{arg}} \cup E_\sigma \subset \bar{u}(\widehat{E}_M) = \bar{u} \circ \bar{e}(E_M)$; moreover the various a^j_i do not occur in Π_N, so $V_N = \bar{e}(V_M) - \{a^j_i \approx a^j_l \mid j \in J,\, i, l \in \widehat{I}^j\}$. Summarising, if A^j_i is not a variable the sequence $s = \bar{u} \circ \bar{e}$ consists of a sequence of expansions $\bar{e}$ transforming (E_M, V_M) into $(\widehat{E}_M, \widehat{V}_M)$, followed by the unification rules $\bar{u}$ eventually leading to a superset of E_N. We conclude by observing that $s(E_M) - E_N = \bar{u}(\widehat{E}_{\widehat{\Omega}}) \cup \{a^j_i \doteq A^j_i \mid j \in J,\, i \in I^j\}$ is in solved form, and that all variables a^j_i and $e^j_{i,p}$ do not occur outside of $s(E_M) - E_N$. Thus, once again, equations belonging to $s(E_M) - E_N$ cannot play any role in generating critical equations.

The special case $P = x$. This is the only way x can occur in P, but neither in functional nor argument position. A sequence $s = \bar{u} \circ \bar{e}$ of expansions and unification rules similar to the one described above may be necessary also in this case, because $P[Q/x] = Q$ can generate new redexes in N if Q is an abstraction.

Now consider an innermost strategy, say the rightmost-innermost one; thanks to the fact that $s(E_M) - E_N$ cannot generate critical equations, termination of `Solve`(M) (more precisely, of a deterministic version that always performs the expansions associated to the rightmost-innermost redex) follows by induction on the length of the rightmost-innermost reduction from M to its normal form, using Lemma 24. The extension of this result to all possile sequences of expansions can then be obtained by observing that a critical equation always originates from a redex appearing in one of the reduction sequences from M to its normal form. Therefore, taking into account that:

- M is strongly normalizing, so the length of all reduction sequences from M to its normal form is finite;
- a β-reduction corresponds to a finite number ≥ 0 of expansions on $\mathtt{PD_i}(M)$;
- different critical equations are generated in different subtrees of $\mathtt{PD_i}(M)$, hence the order in which expansions are performed does not matter;

we can conclude that the number of expansions performed by the algorithm is always finite, i.e. `Solve`(M) terminates. ◀