Unification for λ-calculi without propagation rules

Flávio L. C. de Moura*

Departamento de Ciência da Computação
Universidade de Brasília, Brazil
flaviomoura@unb.br

Abstract. We present a unification procedure for calculi with explicit substitutions (ES) without propagation rules. The novelty of this work is that the unification procedure was developed for the calculi with ES that belong to the paradigm known as "act at a distance", i.e. explicit substitutions are not propagated to the level of variables, as usual. The unification procedure is proved correct and complete, and enjoy a simple form of substitution, called *grafting*, instead of the standard capture avoiding variable substitution.

1 Introduction

Unification is an important operation extensively used in Computer Science whose goal is to find a substitution, when it exists, to identify terms in a certain theory. In particular, it is a central operation in automating higher-order reasoning. The correct framework for higher-order unification is the simply typed λ-calculus where this operation is known to be undecidable [**?**,**?**]. In addition, different unifiers of a given problem can be completely independent from each other, i.e. there is no unicity of solutions.

ES calculi internalize the substitution operation by extending the grammar of the λ-calculus and have been extensively studied during the last decades [**?**,**?**,**?**,**?**,**?**,**?**,**?**,**?**]. It has been mainly used as an intermediate formalism between the theory of the λ-calculus and its implementation. In this context many different approaches were proposed because it was surprisingly difficult to develop a formalism preserving important properties related to the simulation of the λ-calculus.

In this work we present a unification procedure for a family of extensions of the simply typed λ-calculus with explicit substitutions (ES) that act at a distance. This new approach to ES, which has motivations in the study of Proof-Nets [**?**,**?**], satisfy all expected properties for an ES calculus, such as preservation of strong normalization (PSN), confluence on open terms and simulation of one step β-reduction. This approach differs from the usual ES calculi because the explicit substitution is not percolated over terms, and the starting rule of the calculi can traverse ES. In this sense substitutions "act at a distance" or "do not

* Author partially supported by FAPDF.

have propagation rules". Several variants of calculi with ES at a distance have been proposed for different purposes, such as implicit computational complexity [**?**], the theory of abstract standardization [**?**,**?**], abstract machines [**?**], study of equational extensions of the λ-calculus [**?**], etc.

During the unification procedure, each term is transformed into its η-long normal form, which is a well known notion for λ-terms and some (meta)-confluent ES calculi. Nevertheless, the characterization of the η-long normal form for a calculus without propagation rules is not straightforward because explicit substitutions can appear everywhere in the term. In this sense, this work improves [**?**] as follows:

1. We define the notion of binding contexts that allows a precise characterization of the structure of the normal form and of the η-long normal form of a metaterm in calculi without propagation rules.
2. Renaming of bound variables is not taken for free. In fact, the cost of α-conversion needs to be explicitly considered in any implementation of a unification procedure for a calculus that works modulo α-conversion.

An extended version of this work is available at `http://flaviomoura.mat.br`.

2 Explicit substitutions without propagation rules

ES calculi without propagation rules include a family of calculi that allow a substitution to be performed even if it is not at the level of variables. The grammar for all calculi (with metaterms) in this family is defined as follows:

$$t, u ::= x \mid X_{\Delta} \mid t\ u \mid \lambda x.t \mid t[x/u] \tag{1}$$

where x is an ordinary variable, X_{Δ} is a metavariable with support set Δ (a set of ordinary variables), $t\ u$ is an application, $\lambda x.t$ is an abstraction and $t[x/u]$ is a term with an explicit substitution. Metavariables must come with a minimum amount of information in order to guarantee that some basic operations (like replacement of metavariables by metaterms) are sound in a typing context [**?**]. For the sake of clarity, we write X instead of $X_{\{\}}$. As we will see, this approach also simplifies the presentation of the unification rules, and permits to keep a clear separation between the substitutions generated by the unification procedure and the ones generated by the calculus itself during reduction. The separation between ordinary variables and metavariables allows an important simplification in the unification procedure because the substitution generated by the unification procedure is separated from the substitution generated by the calculus itself. Although the calculi considered here are not first-order in the sense that substitutions of variables need to take into account renamings of bound variables in order to avoid capture, the substitution generated by the unification procedure is first-order (*grafting*) [**?**] because metavariables are parameterized by a set of variables that is fundamental for the good behaviour of the procedure. As said before, the construction $t[x/u]$ denotes a term with an explicit substitution, and

both $\lambda x.t$ and $t[x/u]$ bind x in t, i.e. t is the scope of x in both $\lambda x.t$ and $t[x/u]$, and in this case x is called a *bound* variable. For instance, both occurrences of x in the term $(x\ Y_{\{x\}})[x/u]$ are bounded by the explicit substitution. If a variable is not under the scope of an abstraction or an explicit substitution then it is called *free*. Therefore, we work with three sorts of variables: free variables, bound variables and metavariables. In addition, the set of free variables is divided into two kinds: the ones occurring in the support set of metavariables, and the real variables, which are defined as follows:

Definition 1. *The set of free variables occurring in the support sets of metavariables in a metaterm t, denoted by $\mathtt{fm}(t)$, and the set of free (real) variables occurring in the metaterm t, denoted by $\mathtt{fr}(t)$, are inductively defined as follows:*

- $\mathtt{fm}(X_\Delta) = \Delta$
- $\mathtt{fm}(x) = \{\}$
- $\mathtt{fm}(u\ v) = \mathtt{fm}(u) \cup \mathtt{fm}(v)$
- $\mathtt{fm}(\lambda x.u) = \mathtt{fm}(u) \backslash \{x\}$
- $\mathtt{fm}(u[y/v]) = (\mathtt{fm}(u) \backslash \{y\}) \cup \mathtt{fm}(v)$

- $\mathtt{fr}(X_\Delta) = \{\}$
- $\mathtt{fr}(x) = \{x\}$
- $\mathtt{fr}(u\ v) = \mathtt{fr}(u) \cup \mathtt{fr}(v)$
- $\mathtt{fr}(\lambda x.u) = \mathtt{fr}(u) \backslash \{x\}$
- $\mathtt{fr}(u[y/v]) = (\mathtt{fr}(u) \backslash \{y\}) \cup \mathtt{fr}(v)$

The set of free variables of a term t is defined as $\mathtt{fv}(t) = \mathtt{fm}(t) \cup \mathtt{fr}(t)$.

The number of occurrences of the (free) variable x in the term t is denoted by $|t|_x$, as defined above. So, for instance, $|(X_{\{x,y\}}\ x)[y/\lambda z.X_{\{x,z\}}]|_x = 3$.

The operational semantics of the calculi is given in terms of one-hole contexts, and two special classes of contexts called *substitution contexts* [**?**] and *binding contexts*:

Contexts	$C ::= \langle\cdot\rangle \mid C\ t \mid t\ C \mid \lambda x.C \mid C[x/t] \mid t[x/C]$
Biding Contexts	$B ::= \langle\cdot\rangle \mid B[x/t] \mid \lambda x.B$
Substitution Contexts	$L ::= \langle\cdot\rangle \mid L[x/t]$

We write $C\langle t\rangle$ for the term obtained by replacing the hole $\langle\cdot\rangle$ of C by t, thus *e.g.* $(\langle\cdot\rangle y)\langle x\rangle = (x\ y)$ and $(\lambda x.\langle\cdot\rangle)\langle x\rangle = \lambda x.x$. We write $C[\![u]\!]$ when the free variables of u are not captured by the context C, thus for example, $C[\![x]\!]$ denotes the term $(x\ y)$ if $C = \langle\cdot\rangle y$, and $\lambda y.x$, if $C = \lambda x.\langle\cdot\rangle$.

Action at a distance is characterized by the fact that ES are not percolated over terms, and variables can be substituted by terms even if the corresponding explicit substitution is not at the level of the variable itself. As usual for ES calculi, there is one rule to start the simulation of a β-reduction and a set of rules to complete the simulation of the β-step. The rule presented below is the starting rule for calculi acting at a distance, and its name comes from d*istance* B*eta*:

$$L\langle\lambda x.t\rangle\ u \mapsto_{\mathtt{dB}} L\langle t[x/u]\rangle \tag{2}$$

where L is a substitution context. Calculi with explicit substitutions that have (2) as starting rule are called *calculi without propagation rules*. It is interesting to

note how the dB-rule generalizes β-reduction since dB-redexes include application whose left-hand side are not abstractions.

Binding contexts will be used to characterize normal forms. In fact, the first difficult in building a unification procedure for these calculi was to find an adequate notation for terms in normal form without forcing substitutions to be propagated over them. In [**?**], an ongoing work accepted for short presentation, the early stages of this work was presented assuming that ES do not occur outside the external abstractions of a normal form, which is possible since the operational semantics of the calculi is not changed. In subsection 2.2, we present a precise and elegant characterization of normal forms for calculi without propagation rules.

In the next subsection we shortly present some calculi without propagation rules. They differ in the way substitution is controlled as follows: the first calculus, called the *linear substitution calculus* ($\lambda_{\mathtt{lsub}}$) [**?**], performs substitutions one at a time. The second calculus, called the *substitution calculus* ($\lambda_{\mathtt{sub}}$) [**?**], performs the whole substitution in one step, i.e. it splits the (non-terminating) β-reduction of the λ-calculus into two (terminating) rules. Finally, the third calculus, known as the *structural λ-calculus* ($\lambda_{\mathtt{str}}$) [**?**], duplicates ES by introducing new names for them until the variable of the substitution has just one occurrence in the term, which is then executed.

2.1 The Calculi with ES at a distance

As explained before, all the calculi start the simulation of a β-reduction by rule (2), which introduces an explicit substitution. The execution and control of the explicit substitution can be done in different ways, which give rise to different calculi. The following tables in addition to rule (2) form the rules of the $\lambda_{\mathtt{lsub}}$-,$\lambda_{\mathtt{sub}}$- and $\lambda_{\mathtt{str}}$-calculi:

lsub	
$C[\![x]\!][x/u] \mapsto_{\mathtt{ls}} C[\![u]\!][x/u]$	
$t[x/u] \mapsto_{\mathtt{w}} t,$	if $\lvert t\rvert_x = 0$

sub	
$t[x/u] \mapsto_{\mathtt{sub}} t\{x/u\},$	if $x \notin \mathtt{fm}(t)$ or $x \in \mathtt{fr}(t)$

str	
$t[x/u] \mapsto_{\mathtt{c}} t_{\langle x\rangle y}[x/u][y/u]$	if $\lvert t\rvert_x > 1$
$t[x/u] \mapsto_{\mathtt{d}} t\{x/u\}$	if $\lvert t\rvert_x = 1$ and $x \notin \mathtt{fm}(t)$
$t[x/u] \mapsto_{\mathtt{w}} t$	if $\lvert t\rvert_x = 0$

In what follows, we write $\lambda_{\mathtt{dB}}$-calculus to refer to any of these calculi. As usual, $\rightarrow$ denotes the contextual closure of $\mapsto$, $\twoheadrightarrow$ the reflexive-transitive closure of $\rightarrow$ and $=_{\lambda_{\mathtt{dB}}}$ the equivalence relation generated by $\rightarrow$. In rule ls, $C[\![x]\!][x/u]$ denotes a term that contains at least one occurrence of the variable x and an explicit substitution that waits to substitute x by u. After a possible renaming to avoid capture of variables, the rule ls substitutes the variable x by u and leaves the explicit substitution $[x/u]$ at the very same place because it can still be used by other occurrences of x in the term. After the substitution of all occurrences of x in the term, the rule w can be applied to get rid of the explicit substitution.

In the $\lambda_{\mathtt{sub}}$-calculus, the $\mathtt{sub}$-rule transforms in one step the explicit substitution into the implicit metasubstitution, as long as x does not occur in the support set of a metavariable, or it occurs as a real variable, in t. In the $\lambda_{\mathtt{str}}$-calculus, $t_{\langle x|y\rangle}$ denotes the non-deterministic replacement of i $(1 \leq i \leq |t|_x - 1)$ occurrences of x in t by a fresh variable y. A formal definition of the function $_{-\langle -|-\rangle}$ is given in what follows. Initially, we define an auxiliary relation that non-determinitically replaces exactly k occurrences of the variable x in t by the fresh variable y, denoted by $t^k_{\langle x|y\rangle}$.

Definition 2. *The non-deterministic replacement of $k \geq 0$ occurrences of the variable x by the fresh variable y in the metaterm t, denoted by $t^k_{\langle x|y\rangle}$, is inductively defined as follows:*

- $t^k_{\langle x|y\rangle} = t$, *if* $x \notin \mathtt{fv}(t)$ *or* $k = 0$, *otherwise*
- $x^k_{\langle x|y\rangle} = y$
- $X_{\Delta}{}^k_{\langle x|y\rangle} = X_{(\Delta\setminus\{x\})\cup\{y\}}$
- $\lambda z.u^k_{\langle x|y\rangle} = \lambda z.u^k_{\langle x|y\rangle}$
- $(u\ v)^k_{\langle x|y\rangle} = u^i_{\langle x|y\rangle}\ v^j_{\langle x|y\rangle}$, *where* $i + j = k$.
- $u[z/v]^k_{\langle x|y\rangle} = u^i_{\langle x|y\rangle}[z/v^j_{\langle x|y\rangle}]$, *where* $i + j = k$.

According to this definition, the structural λ-calculus can be adapted as follows:

$\mathtt{str}$	
$t[x/u] \mapsto_{\mathtt{c}} t^k_{\langle x|y\rangle}[x/u][y/u]$	if $\lvert t\rvert_x > 1$ and $0 < k < \lvert t\rvert_x$
$t[x/u] \mapsto_{\mathtt{d}} t\{x/u\}$	if $\lvert t\rvert_x = 1$ and $x \notin \mathtt{fm}(t)$
$t[x/u] \mapsto_{\mathtt{w}} t$	if $\lvert t\rvert_x = 0$

The metasubstitution, i.e. the substitution operation of the λ-calculus can be extended to the grammar of calculi with explicit substitutions as follows:

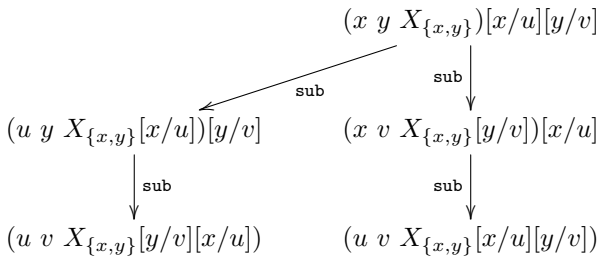
Definition 3. *Let t, u be metaterms. We inductively define the metasubstitution of x by u in t, written $t\{x/u\}$, as follows [?]:*

- $x\{x/u\} = u$
- $y\{x/u\} = y$
- $X_{\Delta}\{x/u\} = \begin{cases} X_{\Delta}[x/u], & \text{if } x \in \Delta \\ X_{\Delta}, & \text{if } x \notin \Delta \end{cases}$
- $(t_1\ t_2)\{x/u\} = t_1\{x/u\}\ t_2\{x/u\}$
- $(\lambda y.t_1)\{x/u\} = \lambda y.t_1\{x/u\}$, *if* $y \notin \mathtt{fv}(u)$
- $t_1[y/t_2]\{x/u\} = t_1\{x/u\}[y/t_2\{x/u\}]$ *if* $y \notin \mathtt{fv}(u)$

In the cases $(\lambda y.t_1)\{x/u\}$ and $t_1[y/t_2]\{x/u\}$ above, a renaming of bound variables before the propagation of the substitution can be necessary to avoid capture of variables.

Nevertheless, for the particular case of the substitution calculus presented above, this notion of metasubstitution leads to a non-confluent calculus. In fact, assuming that $x \notin \mathtt{fv}(v)$ and $y \notin \mathtt{fv}(u)$, one has the following non-joinable

divergence:



One way to close this diagram is to allow permutation of independent substitutions, which is usually formalized via an equation of the following form:

$$t[x/u][y/v] \equiv t[y/v][x/u], \text{ if } x \notin \mathtt{fv}(v) \text{ and } y \notin \mathtt{fv}(u) \tag{3}$$

This approach is adopted in [**?**] in order to prove metaconfluence for the structural λ-calculus. In order to recover metaconfluence for the substitution calculus, we use a different approach based on a new notion of metasubstitution that do not change the position of an explicit substitution in a term for the case of metavariables. This new notion is given by the following definition, and more details can be found in [**?**].

Definition 4 (Metasubstitution). *Let t, u be metaterms. The* **metasubstitution** *$t\{x/u\}$ is defined as follows:*

$$t\{x/u\} = \begin{cases} t\{\!\{x/u\}\!\}[x/u], & \text{if } x \in \mathtt{fm}(t) \\ t\{\!\{x/u\}\!\}, & \text{if } x \notin \mathtt{fm}(t) \end{cases}$$

where $t\{\!\{x/u\}\!\}$ is inductively defined as follows:

$$\begin{array}{lll} t\{\!\{x/u\}\!\} & = t & \text{if } x \notin \mathtt{fr}(t), \text{ otherwise} \\ \bullet\ x\{\!\{x/u\}\!\} & = u & \\ \bullet\ (\lambda y.v)\{\!\{x/u\}\!\} & = \lambda y.v\{\!\{x/u\}\!\} & \text{if } x \neq y \ \&\ y \notin \mathtt{fv}(u) \\ \bullet\ (t_1\ t_2)\{\!\{x/u\}\!\} & = t_1\{\!\{x/u\}\!\}\ t_2\{\!\{x/u\}\!\} & \\ \bullet\ t_1[y/t_2]\{\!\{x/u\}\!\} & = t_1\{\!\{x/u\}\!\}[y/t_2\{\!\{x/u\}\!\}] & \text{if } x \neq y \ \&\ y \notin \mathtt{fv}(u) \end{array}$$

Using this new notion of metasubstitution, note that the previous divergence is no longer generated because the explicit substitution does not percolate over the term.

Despite the fact that calculi without propagation rules are not first order in the sense that they use a notion of substitution that renames bound variables in order to avoid capture, the substitution generated by the unification procedure is still first order. This is possible due to two facts:

1. Metavariables and (ordinary) variables belong to different classes, so that the substitution generated by the unification procedure are separated from the ones generated by the reduction relation.

2. The support set of metavariables contains the variables that can occur free in the term that will replace this metavariable, and in addition, metavariables parameterized by a set of variables allow us to express the functional dependency of the arguments w.r.t. the corresponding abstractions in a very straightforward and simple way.

We define *grafting* as a first order substitution whose domain is the set of metavariables:

Definition 5 (Grafting). *Let t and u be metaterms, and Δ a set of variables with $\mathtt{fv}(u) \subseteq \Delta$. We inductively define the grafting of u for X_Δ in t, denoted by $t[\![X_\Delta/u]\!]$, as follows:*

- $y[\![X_\Delta/u]\!] = y$
- $X_{\Delta'}[\![X_\Delta/u]\!] = \begin{cases} u & \text{if } \Delta \subseteq \Delta' \\ X_{\Delta'} & \text{otherwise.} \end{cases}$
- $Y_{\Delta'}[\![X_\Delta/u]\!] = Y_{\Delta'}$, *if* $X \neq Y$
- $(t_1\ t_2)[\![X_\Delta/u]\!] = (t_1[\![X_\Delta/u]\!])\ (t_2[\![X_\Delta/u]\!])$
- $(\lambda x.t_1)[\![X_\Delta/u]\!] = \lambda x.(t_1[\![X_\Delta/u]\!])$
- $t_1[x/t_2][\![X_\Delta/u]\!] = (t_1[\![X_\Delta/u]\!])[x/t_2[\![X_\Delta/u]\!]]$

In this way, the metavariable $X_{\{x,y,z\}}$ can be replaced by metaterms containing x, y and z as free variables, but no more than that. So, for instance,

$$(\lambda x.x\ X_{\{x,y\}})[\![X_{\{x\}}/x]\!] = \lambda x.x\ x.$$

We show next, grafting and reduction commute.

Lemma 1. *Let t, u be metaterms and X_Δ a metavariable with $\mathtt{fv}(u) \subseteq \Delta$. For $R \in \{\mathtt{dB}, \mathtt{sub}, \mathtt{lsub}, \mathtt{str}\}$, $t \to_R t'$ implies $t[\![X_\Delta/u]\!] \to_R t'[\![X_\Delta/u]\!]$*

Lemma 2. *Let t, u, v be metaterms and X_Δ a metavariable with $\mathtt{fv}(u) \subseteq \Delta$. Then $t\{x/v\}[\![X_\Delta/u]\!] = t[\![X_\Delta/u]\!]\{x/v[\![X_\Delta/u]\!]\}$.*

Corollary 1. *Grafting and reduction commute.*

2.2 Typing rules

The right environment for unification is the simple type theory, in which simple types are defined by the following grammar:

$$\sigma ::= A \mid \sigma \to \sigma$$

where A range over a denumerable set of base types. Type variables will range over $\sigma, \tau, \gamma, \ldots$ with subscripts when necessary, and the constructor $\to$ of a functional type is right associative. Therefore, $\sigma \to \tau \to \gamma$ means $\sigma \to (\tau \to \gamma)$, and every typable metaterm t has a type of the form $\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to A$ $(n \geq 0)$, where A is the *target type* of t, written $\mathtt{tt}(t)$. If t is a metaterm of type σ, then we write $\mathtt{ty}(t) = \sigma$. A type assignment is a pair of the form $x : \sigma$, where x is a variable and σ is a type. The domain of a set of type assignments $\Gamma = \{x_1 : \sigma_1, x_2 : \sigma_2, \ldots, x_n : \sigma_n\}$, denoted by $Dom(\Gamma)$, is the set

$\{x_1, x_2, \ldots, x_n\}$, whose variables are assumed to be pairwise distinct. We define *type contexts* as finite sets of type assignments. We assume that variables in a type context have at most one assignment, and the type contexts can be extended by adding new assignments, as long as this condition is preserved. If Σ is a type context then the restriction of Σ to the variables in the set S, denoted by $\Sigma|_S$, is defined as the set $\{x : \sigma \mid x \in S \text{ and } (x : \sigma) \in \Sigma\}$.

The fact that the term t has type σ in the type context Σ is expressed by the type judgment $\Sigma \vdash t : \sigma$. In addition, to each metavariable X_Δ we associate a unique type σ_X. We assume that for each type σ there is an infinite set of metavariables X_Δ such that $\sigma_X = \sigma$ and the typing rules are given by the following rules:

$$\frac{(x : \sigma) \in \Sigma}{\Sigma \vdash x : \sigma}\ (\textsc{Var}) \qquad \frac{\Delta \subseteq Dom(\Sigma)}{\Sigma \vdash X_\Delta : \sigma_X}\ (\textsc{MVar})$$

$$\frac{\Sigma \vdash t : \sigma \to \tau \qquad \Sigma \vdash u : \sigma}{\Sigma \vdash t\ u : \tau}\ (\textsc{App})$$

$$\frac{\Sigma \cup \{x : \sigma\} \vdash t : \tau}{\Sigma \vdash \lambda x.t : \sigma \to \tau}\ (\textsc{Abs}) \qquad \frac{\Sigma \cup \{x : \sigma\} \vdash t : \tau \qquad \Sigma \vdash u : \sigma}{\Sigma \vdash t[x/u] : \tau}\ (\textsc{Clos})$$

We present some properties of this typing system:

Lemma 3. *Let Σ be a type context, t a metaterm and σ a type. If $\Sigma' \supset \Sigma$ is another type context, then $\Sigma \vdash t : \sigma \implies \Sigma' \vdash t : \sigma$.*

Proof. By induction on $\Sigma \vdash t : \sigma$.

Lemma 4. *Let t be a metaterm such that $\Sigma \vdash t : \sigma$ for some type σ and type context Σ. Then $\Sigma|_{\mathtt{fv}(t)} \vdash t : \sigma$.*

Proof. By induction on $\Sigma \vdash t : \sigma$.

Lemma 5. *Let t be a metaterm such that $\Sigma \vdash t : \sigma$ for some type σ and type context Σ. If $t \to_{\mathtt{dB}} t'$ or $t \to_{\mathtt{w}} t'$ then $\Sigma \vdash t' : \sigma$.*

Lemma 6. *Let Σ and Σ' be type contexts such that its union is still a type context, and t, u be metaterms. If x is a fresh variable of type γ, $x \notin \mathtt{fm}(t)$, $\Sigma \cup \{x : \gamma\} \vdash t : \sigma$ and $\Sigma' \vdash u : \gamma$ then $\Sigma \cup \Sigma' \vdash t\{\!\{x/u\}\!\} : \sigma$*

Proof. By induction on t.

Lemma 7. *Let Σ and Σ' be type contexts such that its union is still a type context, and t, u be metaterms. If $x \in \mathtt{fm}(t)$, $\Sigma \cup \{x : \gamma\} \vdash t : \sigma$ and $\Sigma' \vdash u : \gamma$ then $\Sigma \cup \{x : \gamma\} \cup \Sigma' \vdash t\{\!\{x/u\}\!\} : \sigma$*

Proof. By induction on t.

Proposition 1. *Let Σ and Σ' be type contexts such that its union is still a type context. If t and u are metaterms such that $\Sigma \cup \{x : \gamma\} \vdash t : \sigma$ and $\Sigma' \vdash u : \gamma$ then $\Sigma \cup \Sigma' \vdash t\{x/u\} : \sigma$.*

Corollary 2. *Let Σ and Σ' be type contexts such that its union is still a type context. If C is a context, x a variable of type γ in the type context Σ, u is a metaterm such that $\Sigma' \vdash u : \gamma$ and $\Sigma \cup \{x : \gamma\} \vdash C[\![x]\!] : \sigma$ then $\Sigma \cup \Sigma' \vdash C[\![u]\!] : \sigma$.*

Theorem 1 (Subject reduction for the $\lambda_{\mathtt{lsub}}$-calculus). *Let t be a metaterm such that $\Sigma \vdash t : \sigma$ for some type σ and type context Σ. If $t \to_{\lambda_{\mathtt{lsub}}} t'$ then $\Sigma \vdash t' : \sigma$.*

Theorem 2 (Subject reduction for the $\lambda_{\mathtt{sub}}$-calculus). *Let t be a metaterm such that $\Sigma \vdash t : \sigma$ for some type σ and type context Σ. If $t \to_{\lambda_{\mathtt{sub}}} t'$ then $\Sigma \vdash t' : \sigma$.*

Lemma 8. *Let t be a metaterm. If $\Sigma \cup \{x : \gamma\} \vdash t : \sigma$ then $\Sigma \cup \{x : \gamma\} \cup \{y : \gamma\} \vdash t^k_{\langle x|y\rangle} : \sigma$, for all $0 \leq k \leq |t|_x$ and fresh variable y.*

Corollary 3. *If $|t|_x > 1$ and $\Sigma \cup \{x : \sigma_1\} \vdash t : \sigma$ then $\Sigma \cup \{x : \sigma_1\} \cup \{y : \sigma_1\} \vdash t_{\langle x|y\rangle} : \sigma$, where y is a fresh variable.*

Theorem 3 (Subject reduction for the $\lambda_{\mathtt{str}}$-calculus). *Let t be a metaterm such that $\Sigma \vdash t : \sigma$ for some type σ and type context Σ. If $t \to_{\lambda_{\mathtt{str}}} t'$ then $\Sigma \vdash t' : \sigma$.*

Our unification algorithm works with terms in *η-long normal form* (`lnf`) that is defined in what follows.

By definition, a term is in normal form if it has no reduct. The following lemma gives the general structure of a term in normal form.

Lemma 9 (Characterization of normal forms). *These terms can be characterized by the following structure:*

$$B\langle L_n\langle \cdots L_1\langle L_0\langle a\rangle\ t_1\rangle \cdots t_n\rangle\rangle$$

where

- *a is either a variable or a metavariable;*
- *$L_0 = \langle\cdot\rangle$ if a is not a metavariable;*
- *the terms $t_1, t_2, \ldots, t_n$ are in normal form;*
- *every substitution of the form $[x/u]$ occurring in B or L_i $(0 \leq i \leq n)$ is such that x occurs in the support set of a metavariable that is in the scope of this substitution, and u is in normal form.*

Since the calculi considered here are metaconfluent, normal forms are unique, and we write $(t)\!\downarrow$ to denote the normal form of the term t.

Definition 6 (η-long normal form). *Let $t = B\langle L_n\langle \cdots L_1\langle L_0\langle a\rangle\ t_1\rangle \cdots t_n\rangle\rangle$ be a term in normal form of type $\sigma_1 \to \cdots \to \sigma_m \to A$ with A atomic and $n, m \geq 0$. The η-long normal form (*`lnf`*) of t, written $(t)\updownarrow$, is given by*

$$\lambda x_1 \ldots x_m.B'\langle L'_n\langle \cdots L'_1\langle L'_0\langle a\rangle\ (t_1)\updownarrow\rangle \cdots (t_n)\updownarrow\rangle\rangle\ (x_1)\updownarrow \ldots (x_m)\updownarrow$$

where

- *L'_i $(0 \leq i \leq n)$ is obtained from L_i by taking each substitution of the form $[x/u]$ in L_i and replacing it by $[x/(u)\updownarrow]$.*
- *B' is obtained from B by preserving each abstractor of the form λx, and replacing each substitution of the form $[x/u]$ in B by $[x/(u)\updownarrow]$.*

Lemma 10. *The Definition 6 is well-defined.*

Definition 7 (Heading). *Let $t = B\langle L_n\langle \cdots L_1\langle L_0\langle a\rangle\ t_1\rangle \cdots t_n\rangle\rangle$ be a term in η-long normal form. The heading of t is defined as $(B\langle L_n\langle \cdots L_1\langle L_0\langle a\rangle\rangle \ldots\rangle\rangle)\downarrow$.*

In this way, if $\lambda x.(\lambda y.a\ X_{\{z\}})[z/u]$ is a term in η-long normal form then it can be written as $B_2\langle B_1\langle B_0\langle a\ X_{\{z\}}\rangle\rangle\rangle$ with $B_2 = \lambda x.\langle\cdot\rangle$, $B_1 = \langle\cdot\rangle[z/u]$ and $B_0 = \lambda y.\langle\cdot\rangle$. By definition, its heading is equal to $(\lambda x.(\lambda y.a)[z/u])\downarrow = \lambda x.\lambda y.a$.

Definition 8. *A metaterm in* `lnf` *is flexible if its head is a metavariable. Otherwise, it is rigid.*

3 Unification

In this section we present the unification procedure. It receives as argument a unification problem that is defined in what follows.

Definition 9. *A unification problem P is a finite set of pairs, also called disagreement pairs, of the form $t =^? u$, where t and u are terms in* `lnf` *of the same type. A disagreement pair $t =^? u$ is called trivial if $t =_\alpha u$, i.e. if t and u differ only by the name of bound variables. A grafting σ is a solution of the disagreement pair $t =^? u$, if $t\sigma =_{\lambda_{\mathsf{dB}}} u\sigma$. A grafting σ is a solution of the unification problem P if it is a solution of each disagreement pair of P. A metavariable is unsolved w.r.t. the unification problem P if it appears in either a flexible-rigid or a rigid-rigid pair in P, otherwise, i.e. if it either appears only in flexible-flexible pairs or does not appear at all in P, it is said to be solved w.r.t. the unification problem P. A unification problem is pre-solved if it does not contain unsolved metavariables, i.e. if it contains only flexible-flexible pairs. A unification problem without solution will be denoted by $\bot$. The set of unifiers of a unification problem P is denoted by $\mathcal{U}_{\lambda_{\mathsf{dB}}}(P)$*

The unification rules are presented in Table 1. For a given unification problem P, one starts with the pair $P; \epsilon$, where ϵ is the empty grafting. The rules are applied non-deterministically to the current unification problem until no more

$\{t =^? t'\} \cup Q; \theta \overset{\texttt{Trivial}}{\Longrightarrow} Q; \theta,$ if $t =_\alpha t'$
$\{t =^? t'\} \cup Q; \theta \overset{\texttt{Fail}}{\Longrightarrow} \bot$, if $t =^? t'$ is a rigid-rigid equation with headings that are not α-equivalent
$\{t =^? t'\} \cup Q; \theta \overset{\texttt{Dec}}{\Longrightarrow} (\bigcup_{i=1}^{n} \{B\langle L_n\langle \ldots L_i\langle t_i\rangle \ldots\rangle\rangle =^? B'\langle L'_n\langle \ldots L'_i\langle t'_i\rangle \ldots\rangle\rangle\}) \cup Q; \theta$ if $t =^? t'$ is a rigid-rigid equation, $t = B\langle L_n\langle \ldots L_2\langle L_1\langle a\ t_1\rangle\ t_2\rangle \ldots t_n\rangle\rangle$, $t' = B'\langle L'_n\langle \ldots L'_2\langle L'_1\langle b\ t'_1\rangle\ t'_2\rangle \ldots t'_n\rangle\rangle$ and $(B\langle a\rangle)\downarrow =_\alpha (B'\langle b\rangle)\downarrow$
$P; \theta \overset{\texttt{Exp}}{\Longrightarrow} (P\theta')\updownarrow; \theta\theta'$ if there is a flexible-rigid equation in P with the head of the flexible term equal to X_Δ with $\texttt{ty}(X_\Delta) = \sigma_1 \to \cdots \to \sigma_n \to A$ and $n > 0$, and where $\theta' = [\![X_\Delta / \lambda x_1 \ldots x_n . Y_{\Delta \cup \{x_1, \ldots, x_n\}}]\!]$ and Y is a fresh metavariable
$P; \theta \overset{\texttt{Imit}}{\Longrightarrow} (P\theta')\updownarrow; \theta\theta'$ if there is a flexible-rigid equation with the head of the flexible term equal to X_Δ (atomic metavariable of type A), and the head of the rigid term equal to a free variable, say a, of type $\sigma_1 \to \cdots \to \sigma_n \to A$ ($n \geq 0$), where $\theta' = [\![X_\Delta / a\ Y_{1\,\Delta} \ldots Y_{n\,\Delta}]\!]$ and $Y_1, \ldots, Y_n$ are fresh metavariables
$P; \theta \overset{\texttt{Select}}{\Longrightarrow} (P\theta')\updownarrow; \theta\theta'$ if there is a flexible-rigid equation with the flexible term equal to $B\langle X_\Delta\rangle$, for a binding context B and $\texttt{ty}(X_\Delta) = A$ (atomic), and an ES $[y/u]$ in B with $\texttt{ty}(y) = \sigma_1 \to \cdots \to \sigma_n \to A$ ($n \geq 0$) and $\texttt{fv}(u) \subseteq \Delta$ such that either u is flexible, or the heading of $B\langle u\rangle$ is α-equivalent to the heading of the rigid term of this equation, where $\theta' = [\![X_\Delta / y\ Y_{1\,\Delta} \ldots Y_{n\,\Delta}]\!]$ and $Y_1, \ldots, Y_n$ are fresh metavariables

Table 1. Unification Rules

rules apply, and then one reaches either $\perp$ (no solution is generated) or a pre-solved form $P';\sigma$. During the process to reach a pre-solved form, the procedure transforms the original problem by incrementally building partial graftings that will compose a solution to the original problem. In addition, each rule that can be applied the current unification problem gives rise to a new branch that potentially will generate a solution to the original problem.

The rules `Imit` and `Select` are generalizations of the imitation and projection rules of Huet's procedure for the simply typed λ-calculus [**?**]. A failure in the unification process will be characterized by unification problems containing rigid-rigid pairs with headings that are not α-equivalent.

The case of flexible-flexible pairs are usually not treated explicitly because they are always solvable and their solutions are not unique. In addition, flexible-flexible pairs are not relevant for most refutation methods [**?**], and hence our procedure consists in searching for pre-solved forms of the original problem.

Lemma 11. *The application of the unification rules to well-typed pairs gives rise only to well-typed pairs.*

Proof. By rule analysis.

3.1 Unification Procedure

In this section we detail the general behavior of the unification procedure, and we write $\Longrightarrow \equiv \overset{\texttt{Trivial}}{\Longrightarrow} \cup \overset{\texttt{Fail}}{\Longrightarrow} \cup \overset{\texttt{Dec}}{\Longrightarrow} \cup \overset{\texttt{Exp}}{\Longrightarrow} \cup \overset{\texttt{Imit}}{\Longrightarrow} \cup \overset{\texttt{Select}}{\Longrightarrow}$. Given a unification problem P, one builds a tree, called *unification tree* of P (cf. [**?**]), as follows:

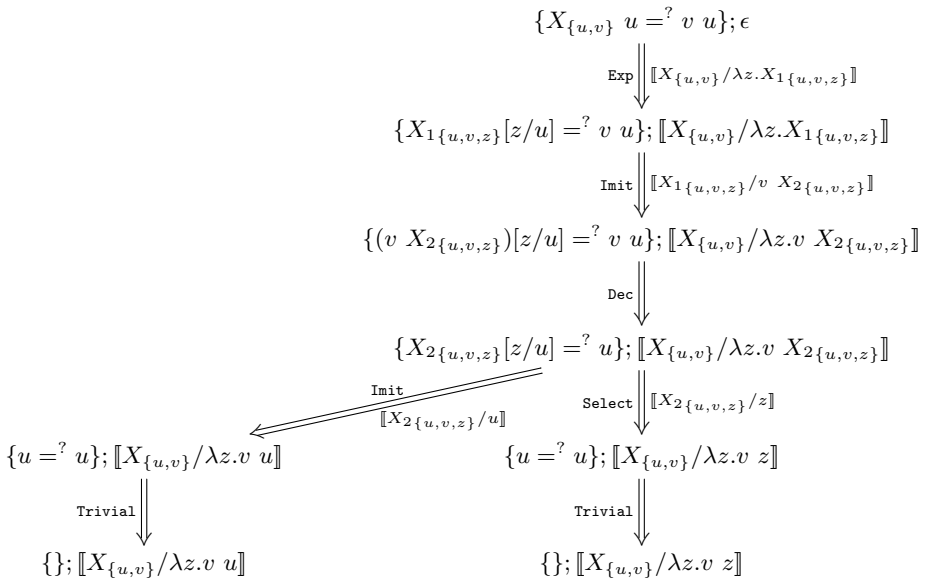
1. The root of the tree is labeled with $P;\epsilon$
2. For each node, one non-deterministically selects a pair in P and applies a rule by adding a child node labeled with the new unification problem P'. In this case, we write $P;\sigma \overset{r}{\Longrightarrow} P\theta;\sigma\theta$.
3. The procedure stops when no more rule applies. A success branch is obtained if the procedure stops at a pre-solved form. Otherwise, the procedure generates a fail branch.

Example 1. Let P be the unification problem containing the sole pair: [1]

$$\{X_{\{u,v\}_{A\to A}}\ u_A =^? v_{A\to A}\ u_A\}$$

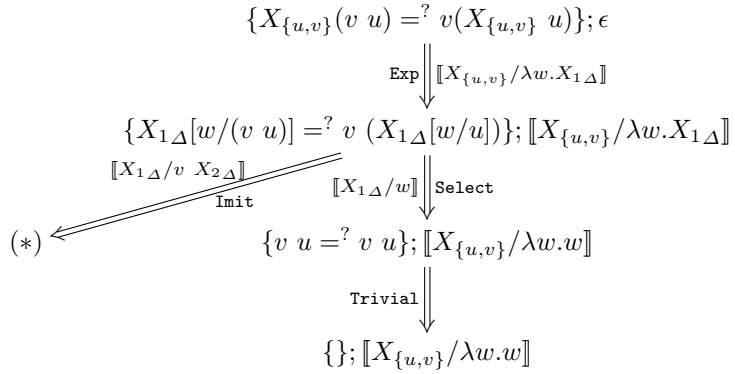
[1] We write the type information only in the initial terms of the examples for readability.

The unification tree is as follows where non-determinism corresponds to "or" branches:



Example 2. Consider the unification problem

$$\{X_{\{u,v\}\,A\to A}(v_{A\to A}\ u_A) =^?\ v_{A\to A}(X_{\{u,v\}\,A\to A}\ u_A)\}$$



where $\Delta = \{u, v, w\}$ and $(*)$ is given by

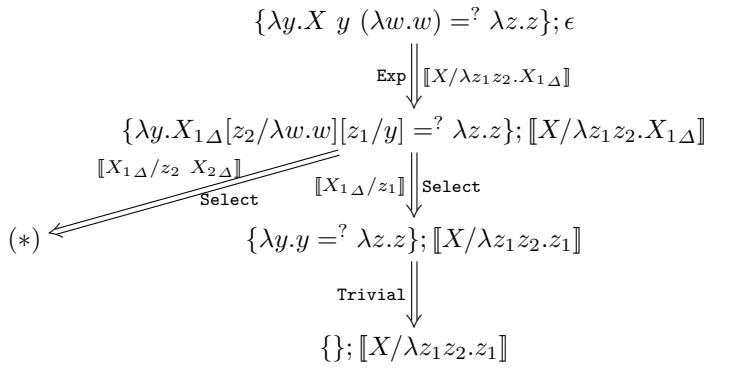
$$\{(v\ X_{2\,\Delta})[w/(v\ u)] =^?\ v((v\ X_{2\,\Delta})[w/u])\};[\![X_{\{u,v\}}/\lambda w.v\ X_{2\,\Delta}]\!]$$

which reduces to $\{X_{2\,\Delta}[w/(v\ u)] =^?\ v(X_{2\,\Delta}[w/u])\};[\![X_{\{u,v\}}/\lambda w.v\ X_{2\,\Delta}]\!]$ after an application of `Dec`. From this point, one can again apply both `Imit` and `Select`, and find out that this problem has infinitely many solutions: $[\![X_{\{u,v\}}/\lambda w.x^n\ w]\!]$ $(n \geq 0)$, where $x^0\ w = w$, and $x^{n+1}\ w = x(x^n\ w)$.

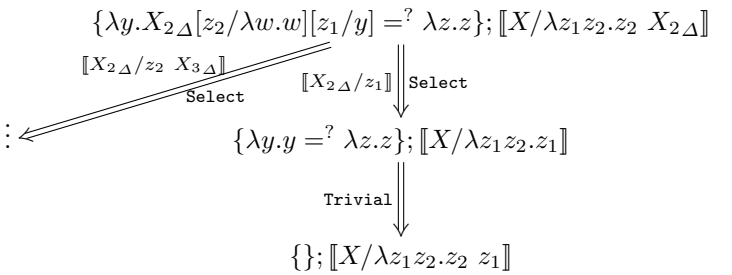
Example 3. Let

$$P\{\lambda y_A.X_{A\to(A\to A)\to A}\ y_A\ (\lambda w_A.w_A) =^? \lambda z_A.z_A\}$$

be a unification problem.



where $(*)$ is as follows:



where $\Delta = \{z_1, z_2\}$. This problem has infinite solutions. In fact, all Church numerals of the form $\lambda xy.y^k\ x$ $(k \geq 0)$ are solutions.

Correctness and Completeness of the Unification Procedure In this subsection, we state the correctness and completeness theorems of the presented unification procedure.

Lemma 12. *Let P be a unification problem. If $P;\theta \overset{R}{\Longrightarrow} P';\theta'$ then $\mathcal{U}_{\lambda_{\mathsf{dB}}}(P') = \mathcal{U}_{\lambda_{\mathsf{dB}}}(P)$, where $R \in \{\mathtt{Trivial}, \mathtt{Fail}, \mathtt{Dec}\}$.*

Lemma 13. *Let P be a solvable unification problem. If $P;\theta \overset{R}{\Longrightarrow} P';\theta'$ then $\mathcal{U}_{\lambda_{\mathsf{dB}}}(P') \subseteq \mathcal{U}_{\lambda_{\mathsf{dB}}}(P)$, where $R \in \{\mathtt{Exp}, \mathtt{Imit}, \mathtt{Select}\}$.*

Theorem 4. *Let P be a solvable unification problem. If $P;\theta \Longrightarrow^* P';\theta'$ then $\mathcal{U}_{\lambda_{\mathsf{dB}}}(P') \subseteq \mathcal{U}_{\lambda_{\mathsf{dB}}}(P)$.*

Proof. By induction on the length of the derivation $P;\theta \Longrightarrow^* P';\theta'$.

Theorem 5 (Correctness of the unification procedure). *Let P be a non-trivial unification problem, and θ a grafting. If $P;\theta \Longrightarrow^* P';\theta'$ with P' in pre-solved form, then $\theta' \in \mathcal{U}_{\lambda_{dB}}(P)$.*

Theorem 6 (Completeness of the unification procedure). *Let P be a unification problem with solution σ. Then there exists a derivation of $P;\epsilon$ to a pre-solved form $P';\theta$ such that $\sigma = \theta\gamma$, for some grafting γ.*

4 Conclusion and Future Work

Higher-order unification and matching are important operations extensively used in Mathematics and Computer Science that have already been studied in the context of ES [**?**,**?**,**?**,**?**] but the substitution operation in all the calculi considered so far acts by proximity. This means that the substitution is propagated over the terms, while in calculi that act at a distance, they are not. In spite of having the desired good properties, such as confluence, full composition and PSN, these calculi are useful to study the λ-calculus rather than to implement it [**?**,**?**]. In particular, the proof of confluence for the calculi with metaterms is simple and elegant.

In this work we presented a unification procedure for a family calculi of ES without propagation rules. Our procedure is not just an adaptation of Huet's [**?**] or Snyder's [**?**] one because it does not develop as many fail branches (which can be computationally expensive) as Huet's does, but it is still correct and complete. In addition, a specialized notion of η-long normal form was defined, and is central for the unification procedure. Unification in calculi that act at a distance is also interesting because grafting and reduction permute, which permits the presentation of a procedure that is simpler than the known unification procedures for the λ-calculus [**?**,**?**,**?**] because grafting is used instead of substitution, and also simpler than the traditional unification approach to ES [**?**] because no transformation, like precooking, is required.

As future work, we will study higher-order matching in these formalisms [**?**,**?**]. This would form an interesting framework to study problems like the decidability of higher-order matching which is still an open question after more than 30 years of investigation [**?**,**?**,**?**,**?**,**?**,**?**].

Acknowledgements

I want to thank the anonymous referees for comments and suggestions on this work.