Discrete Mathematics and Theoretical Computer Science **4**, 2000, 011–030

Unification of Higher-order Patterns modulo Simple Syntactic Equational Theories

Alexandre Boudet[†]

LRI, Bât. 490, Université Paris-Sud, 91405 Orsay Cedex, France
email : boudet@lri.fr

received December 6, 1999, *revised January 11, 2000*, *accepted March 15, 2000*.

We present an algorithm for unification of higher-order patterns modulo simple *syntactic equational theories* as defined by Kirchner [14]. The algorithm by Miller [17] for pattern unification, refined by Nipkow [18] is first modified in order to behave as a first-order unification algorithm. Then the *mutation rule* for syntactic theories of Kirchner [13, 14] is adapted to pattern E-unification. If the syntactic algorithm for a theory E terminates in the first-order case, then our algorithm will also terminate for pattern E-unification. The result is a DAG-solved form plus some equations of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x^\pi})$, where $\overline{x^\pi}$ is a permutation of $\overline{x}$. When all function symbols are *decomposable* these latter equations can be discarded, otherwise the compatibility of such equations with the solved form remains open.

Keywords: Unification, Higher-order unification

1 Introduction

Unification is a crucial mechanism in logic programming and automated theorem proving. Unification modulo an equational theory E has been introduced by Plotkin [20] and has become an area of research of its own. With the emergence of higher-order logic programming and rewrite systems [17, 18, 16], the issue of higher-order unification is of growing interest. Higher-order unification is known to be undecidable [10, 8], but Miller has shown that the unification problem is decidable for *patterns*, which are terms of the simply-typed lambda-calculus in which the arguments of a free variable are always distinct bound variables[‡]. Patterns allow to define higher-order functions using pattern matching, as well as interesting higher-order rewrite systems. The aim of the present work is to apply the methods initiated by Kirchner for first-order E-unification to the case of pattern E-unification. This requires to adapt the *mutation* rule to the case of patterns.

In practice, patterns are very similar to first-order terms because the condition that the arguments of the free variables are pairwise distinct bound variables forbids to have free variables "in the middle" of the terms. The free variables (with their restricted kind of arguments) are *at the leaves* of a pattern. The syntactic theories have been defined by Kirchner [13, 14] as those collapse-free equational theories which

[†]This research was supported in part by the EWG CCL, and the "GDR de programmation du CNRS".

[‡] Actually, even the decidability of higher-order matching is still open beyond order 4 [10, 7, 19].

admit a finite presentation such that every equational theorem can be proved by using at most one axiom at the root. This property provides us with complete (non-deterministic) top-down strategies for searching proofs or unifiers. One may guess which axiom applies at the root and then pursue the search in the subterms.

A difficulty with pattern unification is that one needs to introduce new variables. For instance, the most general unifier of $\lambda xyz.F(x,y) = \lambda xyz.G(z,x)$, where the free variables are F and G is $\sigma = \{F \mapsto \lambda xy.H(x), G \mapsto \lambda xy.H(y)\}$, where H is a new free variable. We will see that not only the solving of such flexible-flexible equations (*i.e.*, having a free variable at the top on both sides) require to introduce new variables. On the other hand, new variables are not needed for first-order unification. Our ultimate goal is to take advantage of the resemblance of patterns with first-order terms for lifting the methods that have been developed for two decades for first-order E-unification. In the present paper we want to present an algorithm for pattern unification modulo syntactic theories which behaves exactly as in the first-order case, hence yielding in particular a terminating algorithm whenever the first-order algorithm terminates. For this, we will introduce a preliminary non-deterministic step in which a *projection* such as the above substitution σ is chosen. After this step, a first goal is achieved with an algorithm which does not introduce further new variables and whose (non-failure) rules behave as in first-order unification (in a sense that will be made precise later). Then, we will adapt Kirchner's *mutation* rule to the case of pattern unification. The flexible-flexible equations with the same head variable on both sides (like $\lambda xy.F(x,y) = \lambda xy.F(y,x)$) are *frozen*. Such equations are always solvable (by a projection), but we do not know how to test their compatibility with the rest of the problem in general.We will give an interpretation I of pattern unification problems in terms of first-order unification problems such that if a rule applies to P yeldinq Q, the corresponding rule of the first-order algorithm will apply to $I(P)$ yielding $I(Q)$. Finally, we will show how to handle the equations like $\lambda xy.F(x,y) = \lambda xy.F(y,x)$ in the case where all the function symbols are *decomposable*.

2 Preliminaries

We assume the reader is familiar with simply-typed lambda-calculus, and equational unification. Some background is available in *e.g.* [9, 12] for lambda-calculus and E-unification.

2.1 Patterns and equational theories

Given a set $\mathcal{B}$ of *base types*, the set $\mathcal{T}$ of all *types* is the closure of $\mathcal{B}$ under the (right-associative) function space constructor $\to$. The *simply-typed lambda-terms* are generated from a set $\biguplus_{\tau\in\mathcal{T}} V_\tau$ of *typed variables* and a set $\biguplus_{\tau\in\mathcal{T}} C_\tau$ of *typed constants* using the following construction rules:

$$\frac{x \in V_\tau}{x : \tau} \qquad \frac{c \in C_\tau}{c : \tau} \qquad \frac{s : \tau\to\tau' \quad t : \tau}{(s\,t) : \tau'} \qquad \frac{x : \tau \quad s : \tau'}{\lambda x.s : \tau\to\tau'}$$

The order of a base type is 1, and the order of an arrow type $\tau\to\tau'$ is the maximum of the order of τ plus 1 and the order of τ'. The order of a term is the order if its type.

We shall use the following notations: $\lambda x_1\cdots\lambda x_n.s$ will be written $\lambda\overline{x_n}.s$, or even $\lambda\overline{x}.s$ if n is not relevant. If in a same expression $\overline{x}$ appears several times it denotes the same sequence of variables. The curly-bracketed expression $\{\overline{x_n}\}$ denotes the (multi) set $\{x_1,\ldots,x_n\}$. In addition, we will use the notation $t(u_1,\ldots,u_n)$ or $t(\overline{u_n})$ for $(\cdots(t\;u_1)\cdots u_n)$. The free (resp. bound) variables of a term t are denoted by $\mathcal{FV}(t)$ (resp. $\mathcal{BV}(t)$). The *positions* of a term t are words over $\{0,1\}$, Λ is the empty word (denoting the root position) and $t|_p$ is the subterm of t at position p. The notation $t[u]_p$ stands for a term t with a subterm u at position p, $t[u_1,\ldots,u_n]$ for a term t having subterms $u_1,\ldots,u_n$.

Unless otherwise stated, we assume that the terms are in η-long β-normal form [9], the β and η rules being respectively oriented as follows:
$(\lambda x.M)N \rightarrow_\beta M\{x \mapsto N\}$ (only the free occurrences of x are replaced by N), $F \rightarrow_{\eta\uparrow} \lambda\overline{x_n}.F(\overline{x_n})$ if the type of F is $\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \alpha$, and α is a base type. In this case, F is said to have *arity n*.
The η-long β-normal form of a term t is denoted by $t\updownarrow_\beta^\eta$.

A *substitution* σ is a mapping from a finite set of variables to terms of the same type, written $\sigma = \{x_1 \mapsto t_1,\ldots,x_n \mapsto t_n\}$.

Miller [17] has defined the *patterns* as those terms of the simply-typed lambda-calculus in which the arguments of a free variables are (η-equivalent to) pairwise distinct bound variables. For instance, $\lambda xyz.f(H(x,y),H(x,z))$ and $\lambda x.F(\lambda z.x(z))$§ are patterns while $\lambda xy.G(x,x,y)$, $\lambda xy.H(x,f(y))$ and $\lambda xy.H(F(x),y)$ are not patterns. Patterns have useful applications in higher-order logic programming [17], pattern rewrite systems [18, 16, 4], or definitions of functions by cases in functional programming languages.

It is known that higher-order unification and even second-order unification are undecidable [10, 8]. On the contrary, patterns have decidable and unitary unification :

Theorem 1 ([17]) *Pattern unification is decidable, and there exists an algorithm that computes a most general unifier of any solvable pattern unification problem.*

The equational theories we consider here are the usual first-order equational theories: given a set E of (unordered) first-order axioms built over a signature $\mathcal{F}$, there is an *elementary equational proof* $s \leftrightarrow_E t$ if there exist an axiom $l = r \in E$, a position p of s and a substitution θ such that $s|_p = l\theta$ and $t = s[r\theta]_p$. If $p = \Lambda$, we call this proof a Λ-*step*. The *equational theory* $=_E$ *generated by* E is the reflexive transitive closure $\overset{*}{\leftrightarrow}_E$ of $\leftrightarrow_E$.

The following is a key theorem due to Tannen. It allows us to restrict our attention to $=_E$ for deciding η-β-E-equivalence of terms in η-long, β-normal form :

Theorem 2 ([5]) *Let E be an equational theory and s and t two terms. Then* $s =_{\eta\beta E} t \Longleftrightarrow s\updownarrow_\beta^\eta =_E t\updownarrow_\beta^\eta$.

2.2 Unification problems

Definition 1 *Unification problems are inductively defined as follows:*

- $\top$ *(the trivial unification problem) and* $\perp$ *(the unsolvable unification problem) are unification problems.*
- *An equation* $s = t$ *where s and t are patterns of the same type is a unification problem.*
- *If P and Q are unification problems and X is a variable, then* $P \wedge Q$, $P \vee Q$ *and* $(\exists X)\ P$ *are unification problems.*

Any substitution is a solution of $\top$, $\perp$ *has no solutions and the* σ *is a solution of* $s = t$ *if* $s\sigma =_{\eta\beta E} t\sigma$. *The solutions of* $P \wedge Q$ *(resp.* $P \vee Q$ *) are the intersection (resp. the union) of the solutions of P and Q. A substitution* σ *is a solution of* $(\exists X)\ P$ *if there exists a solution of P identical to* σ *except maybe on X.*

§ We will always write such a pattern in the (η-equivalent) form $\lambda x.F(x)$, where the argument of the free variable F is *indeed* a bound variable.

As usual, we restrict our attention to the problems of the form

$$(\exists \overline{X})\ s_1 = t_1 \wedge \cdots \wedge s_n = t_n$$

the only disjunctions being implicitely introduced by the non-deterministic rules.

Terminology In the following, *free variable* denotes an occurrence of a variable which is not λ-bound and *bound variable* an occurrence of a variable which is λ-bound. To specify the status of a free variable with respect to existential quantifications, we will explicitely write *existentially quantified* or *not existentially quantified*. In the sequel, upper-case F, G, X,... will denote free variables, a, b, f, g,... constants, and x, y, z, x_1,... bound variables.

Without loss of generality, we assume that the left-hand sides and right-hand sides of the equations have the same prefix of λ-bindings. This is made possible because the two terms have to be of the same type, and by using α-conversion if necessary. In other terms, we will assume that the equations are of the form $\lambda\overline{x}.s = \lambda\overline{x}.t$ where s and t do not have an abstraction at the top.

Definition 2 *An equation is* quasi-solved *if it is of the form* $\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.s$ *and* $\mathcal{FV}(s) \cap \{\overline{x_k}\} \subseteq \{\overline{y_n}\}$ *and* $F \notin \mathcal{FV}(s)$.

Rather than computing substitutions, we will compute DAG-solved forms, from which it is trivial to extract solved form which represents its own mgu.

Lemma 1 *If the equation* $\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.s$ *is quasi-solved, then it is has the same solutions as* $\lambda\overline{y_n}.F(\overline{y_n}) = \lambda\overline{y_n}.s$ *and (by* η*-equivalence) as* $F = \lambda\overline{y_n}.s$. *A most general unifier of such an equation is* $\{F \mapsto \lambda\overline{y_n}.s\}$.

For the sake of readability, we will write a quasi-solved equation in the form $F = \lambda\overline{y_n}.s$ instead of $\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.s$ in the following definition and in the rules **Merge** and **Check*** of the next section.

Definition 3 *A* DAG-solved form *is a unification problem of the form*

$$(\exists Y_1 \cdots Y_m)\ X_1 = s_1 \wedge \cdots \wedge X_n = s_n$$

where for $1 \leq i \leq n$, X_i *and* s_i *have the same type, and* $X_i \neq X_j$ *for* $i \neq j$ *and* $X_i \notin \mathcal{FV}(s_j)$ *for* $i \leq j$.

A solved form *is a unification problem of the form*

$$(\exists Y_1 \cdots Y_m)\ X_1 = s_1 \wedge \cdots \wedge X_n = s_n$$

where for $1 \leq i \leq n$, X_i *and* s_i *have the same type,* X_i *is not existentially quantified, and* X_i *has exactly one occurrence.*

A solved form is obtained from a DAG-solved form by applying as long as possible the rules

Quasi-solved
$\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.s \wedge P \quad\Rightarrow\quad F = \lambda\overline{y_n}.s \wedge P$

Replacement
$F = \lambda\overline{y_n}.s \wedge P \quad\Rightarrow\quad F = \lambda\overline{y_n}.s \wedge P\{F \mapsto \lambda\overline{y_n}.s\}$
if F has a free occurrence in P.

EQE
$(\exists F)\ F = t \wedge P \quad\Rightarrow\quad P$
if F has no free occurrence in P.

2.3 Syntactic equational theories

Claude Kirchner [13] has defined the *syntactic theories* as those collapse-free equational theories which admit a finite presentation such that every equational proof can be performed by applying at most once an axiom at the root. Such a property provides complete top-down strategies for equational proofs or unification. At first, the unification community was not aware of the existence of many syntactic theories besides commutativity and its variants. Kirchner and Klay noticed that it is enough for a theory E to be syntactic that every equation of the form $f(x_1,\ldots,x_n) = g(y_1,\ldots,y_m)$ has a finite complete set of E-unifiers $\Sigma_{f,g}$ [15]. The permutative theories like commutativity, or more generally the theories presented by axioms of the form $f(x_1,\ldots,x_n) = f(x_{\pi(1)},\ldots,x_{\pi(n)})$ where π is a permutation of $(1,\ldots,n)$ are syntactic, and the algoritm of figure 1 terminates for such theories. The theories of associativity, associativity-commutativity left-distributivity are syntactic, but the algorithm does not terminate in general. Arnborg and Tidén give a criterion which allows to avoid non-termination in the case of left-distributivity by detecting unsolvable problems [24]. Boudet and Contejean give a criterion for pruning the search space and discarding some non-minimal solutions which ensures the termination while preserving the completeness in the case of associativity-commutativity [2].

Definition 4 *An equational theory is* syntactic *if it possesses a finite* resolvent presentation *E. A set E of equations is a* resolvent presentation *if every E-equality proof can be performed using the axioms of E with at most one Λ-step. $E_{f,g}$ is the set of the axioms of E of the form $f(s_1,\ldots,s_n) = g(t_1,\ldots,t_m)$.*

In the following, we assume that the set of first-order axioms E is a resolvent presentation. In addition, we require that E-is a *simple* theory, that is a theory containing no equalities of the form $s =_E u$ where u is a strict subterm of s.

Figure 1 gives a set of rules for first-order unification modulo simple syntactic theories. The reader is referred to *e.g.* [13, 14, 12, 2] for some background on syntactic theories.

3 Free pattern unification revisited

In this section, we propose a modification of Miller's algorithm [17], refined by Nipkow [18] for pattern unification. We introduce a preliminary non-deterministic step in which we choose those arguments of the free variables which will effectively participate in the solution, and those that will be eliminated by a projection. After this step, we may assume that the value a free variable by a solution σ will effectively

Trivial
$s = s \wedge P \quad \Rightarrow \quad P$

Merge
$x = s \wedge x = t \quad \Rightarrow \quad x = s \wedge s = t$
if $x \in \mathcal{X}$ and $s,t \notin \mathcal{X}$

Var-Rep (Coalesce)
$(\exists z_1, \ldots, z_n) \;\; x = y \wedge P \quad \Rightarrow \quad (\exists z_1, \ldots, z_n) \;\; x = y \wedge P\{x \mapsto y\}$
if $x, y \in V(P)$

Mutate
$f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)$
$\quad \Rightarrow \quad (\exists \mathcal{V}(f(l_1, \ldots, l_n) = g(r_1, \ldots, r_m))) \;\; \bigwedge_{1 \le i \le n} s_i = l_i \bigwedge_{1 \le j \le m} t_j = r_j$
where $f(l_1, \ldots, l_n) = g(r_1, \ldots, r_m) \in E_{f,g}$

Check*
$x_1 = t_1[x_2]_{p_1} \wedge x_2 = t_2[x_3]_{p_2} \wedge \cdots \wedge x_n = t_n[x_1]_{p_n} \quad \Rightarrow \quad \perp$
if some $p_i \neq \Lambda$

Fig. 1: A set of rules for unification modulo simple syntactic theories

depend on each of its arguments, forbidding any further projection. The price to pay is an exponential blowup in the complexity, and the loss of minimality of the algorithm. On the other hand, the complexity of equational unification algorithms is already at least exponential for most of the theories of interest. The advantage of this approach is that the simplification rules can be modified in order to avoid introducing new variables which are needed precisely for possible projections. The resulting algorithm, after the preliminary non-deterministic step mimics closely a first-order unification algorithm. There is no need then for a new termination proof, and the algorithm will extend as in the first-order case to deal with syntactic equational theories.

In this section, the terms we consider are built over a set of typed variables and a set $\mathcal{FC}$ of typed *free constants*, that is constants which are not constrained by any equational theory. We give an example to show a crucial difference of pattern unification with first-order unification.

Example 1 *Consider the equation*

$$\lambda x_1x_2x_3.F(x_1,x_2) = \lambda x_1x_2x_3.a(G(x_3),b(x_2,x_3),H(x_1),s[x_1,x_2,x_3])$$

Nipkow's algorithm transforms this equation into

$$\begin{array}{rl} (\exists L_1 \cdots L_4) & F = \lambda x_1x_2.a(L_1(x_1,x_2),\ldots,L_4(x_1,x_2)) \\ \wedge & \lambda x_1x_2x_3.L_1(x_1,x_2) = \lambda x_1x_2x_3.G(x_3) \\ \wedge & \lambda x_1x_2x_3.L_2(x_1,x_2) = \lambda x_1x_2x_3.b(x_2,x_3) \\ \wedge & \lambda x_1x_2x_3.L_3(x_1,x_2) = \lambda x_1x_2x_3.H(x_1) \\ \wedge & \lambda x_1x_2x_3.L_4(x_1,x_2) = \lambda x_1x_2x_3.s[x_1,x_2,x_3] \end{array}$$

The first equation will be propagated in the rest of the problem. The second equation will be solvable by mapping both L_1 and G onto a new 0-ary variable. The third equation is not solvable since L_2 does not have x_3 as one of its arguments. The fourth equation has solution $\{L_3 \mapsto \lambda xy.H(x)\}$, and the last equation will be solvable or not, depending on the context s.

The above example shows that when the head of the left-hand side of an equation, is a free variable, one cannot say whether this equation is solvable even if the right-hand side does not contain the left-hand side, without traversing it all. In first-order unification, an equation of the form $x = s$ is solved if x does not occur in s. Note that even if the equation is solvable, one may *need* new variables to express the solution.

Figure 2 gives a non-deterministic algorithm for pattern unification. It is two-fold: in a first step, a projection is choosen nondeterministically which removes some of the bound variables under each free variable. In a second step, some rules are applied as long as possible which recall some well-known rules for first-order unification (see *e.g.* [12]). It has to be noticed that after the first step, no new variables are added. Our algorithm will fail when encountering an equation like that of the above example after the projection step because the sets of bound variables occurring in both sides of the equation are not the same.

Example 2 *Consider the equation*

$$\lambda xyz.F(y,z) = \lambda xyz.G(x,z,y)$$

1. APPLY THE FOLLOWING RULE FOR EVERY FREE VARIABLE F OF P:

Project
$P \quad \Rightarrow \; F = \lambda \overline{x_n} F'(y_1, \ldots, y_k) \; \wedge \; P\{F \mapsto \lambda \overline{x_n} F'(y_1, \ldots, y_k)\}$
where F has arity n and F' is a new variable and $\{y_1, \ldots, y_k\} \subseteq \{x_1, \ldots, x_n\}$

2. APPLY AS LONG AS POSSIBLE THE RULES:

Fail
$\lambda \overline{x_k}.s = \lambda \overline{x_k}.t \; \wedge \; P \quad \Rightarrow \; \bot$
if $\mathcal{FV}(s) \cap \overline{x_k} \neq \mathcal{FV}(t) \cap \overline{x_k}$

FF=
$\lambda \overline{x_k}.F(\overline{y_n}) = \lambda \overline{x_k}.F(\overline{z_n}) \; \wedge \; P \quad \Rightarrow \; \bot$
if $\overline{y_n} \neq \overline{z_n}$.

Trivial
$s = s \; \wedge \; P \quad \Rightarrow \; P$

Decompose
$\lambda \overline{x_k}.a(s_1, \ldots, s_n) = \lambda \overline{x_k}.a(t_1, \ldots, t_n) \; \wedge \; P$
$\quad \Rightarrow \; \lambda \overline{x_k}.s_1 = \lambda \overline{x_k}.t_1 \; \wedge \; \cdots \; \wedge \; \lambda \overline{x_k}.s_n = \lambda \overline{x_k}.t_n \; \wedge \; P$
if $a \in \mathcal{FC} \cup \{\overline{x_k}\}$

FF≠ (Coalesce)
$\lambda \overline{x_k}.F(\overline{y_n}) = \lambda \overline{x_k}.G(\overline{z_n}) \; \wedge \; P \quad \Rightarrow \; \lambda \overline{y_n}.F(\overline{y_n}) = \lambda \overline{y_n}.G(\overline{z_n}) \; \wedge \; P\{F \mapsto \lambda \overline{y_n}.G(\overline{z_n})\}$
if $F \neq G$ and $F, G \in \mathcal{FV}(P)$ and $\overline{y_n}$ is a permutation of $\overline{z_n}$

Merge
$F = s \; \wedge \; F = t \; \wedge \; P \quad \Rightarrow \; F = s \; \wedge \; s = t \; \wedge \; P$
if $|s| \leq |t|$

Clash
$\lambda \overline{x_k}.a(s_1, \ldots, s_n) = \lambda \overline{x_k}.b(t_1, \ldots, t_m) \; \wedge \; P \quad \Rightarrow \; \bot$
if $a, b \in \mathcal{FC} \cup \{\overline{x_k}\}$ and $a \neq b$

Check*
$F_1 = s_1[F_2(\cdots)] \; \wedge \; F_2 = s_2[F_3(\cdots)] \; \wedge \; \cdots \; \wedge \; F_n = s_n[F_1(\cdots)] \; \wedge \; P \quad \Rightarrow \; \bot$
if one $s_i[\cdot]$ is not the empty context

Fig. 2: Non-deterministic algorithm for pattern unification

The values of F and G by a solution σ must depend on the same arguments and Nipkow's algorithm will return the solution

$$\{F \mapsto \lambda yz.H(y,z), G \mapsto \lambda xyz.H(z,y)\}$$

Our algorithm will make, (among others) the choice to keep all the arguments of F, but only the last two arguments of G. The problem obtained after the first projection step will be

$$F = \lambda xy.F'(x,y) \wedge G = \lambda xyzG'(y,z) \wedge \lambda xyz.F'(y,z) = \lambda xyz.G'(z,y)$$

The third equation is then transformed into

$$F' = \lambda yz.G'(z,y)$$

yielding a DAG solved form. It has to be noticed that no new variable is introduced in this latter transformation.

If a bad choice is made, the algorithm will fail: assume that both F and G keep all their arguments, the problem obtained after the projection step will be

$$F = \lambda xy.F'(x,y) \wedge G = \lambda xyz.G'(x,y,z) \wedge \lambda xyz.F'(y,z) = \lambda xyz.G'(x,z,y)$$

The above problem is obviously solvable, but we forbid ourselves any further projection after the initial step, hence no solution is computed here.

Finally, some choices may lead to solutions which are less general than the mgu. In our example this happens when the preliminary projections make F and G depend only on their last argument. The DAG solved form computed will be

$$F = \lambda xy.F'(y) \wedge G = \lambda xyz.G'(z) \wedge F' = \lambda y.G'(y)$$

which is strictly less general than the mgu.

Definition 5 *A* constant-preserving substitution *is a substitution σ such that for all $F \in Dom(\sigma)$ if $F\sigma \updownarrow_\beta^\eta = \lambda\overline{x_k}.s$ then every variable of $\overline{x_k}$ has a free occurrence in s. A* projection *is a substitution of the form*

$$\sigma = \{F \mapsto \lambda\overline{x_k}.F'(\overline{y_m}) \mid F \in Dom(\sigma),\ \{\overline{y_m}\} \text{ subset of } \{\overline{x_k}\}\}$$

The correctness of the failure rules is given by the following straightforward lemmas:

Lemma 2 *For every substitution σ, there exist a projection π and a constant-preserving substitution θ such that $\sigma \updownarrow_\beta^\eta = (\pi\theta) \updownarrow_\beta^\eta$.*

Lemma 3 *The equation $\lambda\overline{x}.s = \lambda\overline{x}.t$ where $\{\overline{x}\} \cap \mathcal{FV}(s) \neq \{\overline{x}\} \cap \mathcal{FV}(t)$ has no constant-preserving solution. The equation $\lambda\overline{x}.F(\overline{y}) = \lambda\overline{x}.F(\overline{z})$, where $\overline{y}$ and $\overline{z}$ are not the same sequence, has no constant-preserving solution.*

Proposition 1 *The non-deterministic algorithm of figure 2 is sound and complete for pattern unification. The irreducible problems are DAG-solved forms.*

Proof: The lemma 2 shows that it is correct to first guess a projection and then restrict one's attention to constant-preserving substitutions. The lemma 3 shows that the failure rules **Fail** and **FF=** are complete with respect to constant-preserving substitutions. The rules **Decompose**, **Clash** and **Check*** are already used by Nipkow. The rule **FF≠ (Coalesce)** preserves the sets of solutions: $\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.G(\overline{z_n})$, where $\overline{z_n}$ is a permutation of $\overline{y_n}$ and $\{\overline{y_n}\} \subseteq \{\overline{x_k}\}$, has the same solutions as $\lambda\overline{y_n}.F(\overline{y_n}) = \lambda\overline{y_n}.G(\overline{z_n})$, and by η-equivalence, as $F = \lambda\overline{y_n}.G(\overline{z_n})$. Since $=_{\eta\beta}$ is a congruence, F can be replaced by $\lambda\overline{y_n}.G(\overline{z_n})$ in the rest of the problem. The rules **Trivial** and **Merge** are correct since $=_{\eta\beta}$ is an equivalence. A case analysis shows that if a problem is not in a DAG solved form, then some rule must apply. □

The termination of the algorithm will follow from the termination of the first-order rules. This will be shown in section 5.

4 Unification modulo syntactic theories

4.1 Mutation

In this section, we give a *mutation* rule for pattern unification modulo *simple* equational theories. We first introduce the notion of $\overline{x_k}$-variant.

Definition 6 *An* $\overline{x_k}$-variant *of a first-order axiom* $l = r$ *is* $l\sigma = r\sigma$ *where* σ *maps every variable* Y *of* $l = r$ *onto* $Y'(\overline{y_n})$ *where*

1. $\overline{y_n}$ *is a subsequence of* $\overline{x_k}$.
2. Y' *has type* $\tau_1 \to \cdots \to \tau_n \to \tau$ *if* $y_1,\ldots,y_n$ *have types* $\tau_1,\ldots,\tau_n$ *and* Y *has type* τ.

Example 3 *Consider the first-order axiom* $X+Y = Y+X$ *(C) (where* X *and* Y *have type say* `Nat`*). A* z_1,z_2,z_3*-variant of C, where* z_1,z_2,z_3 *are of type* `Nat` *is* $X'(z_1,z_2)+Y'(z_3) = Y'(z_3)+X'(z_1,z_2)$*, with* X' *of type* `Nat` $\to$ `Nat` $\to$ `Nat` *and* Y' *of type* `Nat` $\to$ `Nat`.

The *mutation* rule is the following:

Mutate
$\lambda\overline{x_k}.f(s_1,\ldots,s_n) = \lambda\overline{x_k}.g(t_1,\ldots,t_m)$
$\Rightarrow\ (\exists\mathcal{V}(f(\overline{l_n}) = g(\overline{r_m})))\ \bigwedge_{1\leq i\leq n}\lambda\overline{x_k}.s_i = \lambda\overline{x_k}.l_i \bigwedge_{1\leq j\leq m}\lambda\overline{x_k}.t_j = \lambda\overline{x_k}.r_j$
where $f(l_1,\ldots,l_n) = g(r_1,\ldots,r_m)$ is an $\overline{x_k}$-variant of an equation of $E_{f,g}$

We give an example of the use of the rule **Mutate**.

Example 4 *Let* E *be the theory of left-distributivity, presented by the axiom* $(V_1 * V_2)+(V_1 * V_3) = V_1 * (V_2+V_3)$ *(LD), and consider the equation*

$$\lambda xyz.F(x,y,z)+F(x,z,y) = \lambda xyz.G(x,y,z)*H(x,y,z)$$

to be solved modulo E. Let us choose to apply **Project** *with the projection* $\{G \mapsto \lambda xyz.G'(x)\}$*. Now, we may restrict our attention to the constant-preserving solutions of the problem*

$$\lambda xyz.F(x,y,z)+F(x,z,y) = \lambda xyz.G'(x)*H(x,y,z)$$

Let us apply **Mutate** *with the xyz-variant*

$$(V_1(x) * V_2(x,y,z)) + (V_1(x) * V_3(x,y,z)) = V_1(x) * (V_2(x,y,z) + V_3(x,y,z))$$

of (LD). The resulting problem is

$$\begin{array}{rl} (\exists V_1 V_2 V_3) & \lambda xyz.F(x,y,z) = \lambda xyz.V_1(x) * V_2(x,y,z) \\ \wedge & \lambda xyz.F(x,z,y) = \lambda xyz.V_1(x) * V_3(x,y,z) \\ \wedge & \lambda xyz.G'(x) = \lambda xyz.V_1(x) \\ \wedge & \lambda xyz.H(x,y,z) = \lambda xyz.V_2(x,y,z) * V_3(x,y,z) \end{array}$$

Merge *applies to the first two equations, yielding the problem*

$$\begin{array}{rl} (\exists V_1 V_2 V_3) & \lambda xyz.F(x,y,z) = \lambda xyz.V_1(x) * V_2(x,y,z) \\ \wedge & \lambda xyz.V_1(x) * V_2(x,y,z) = \lambda xyz.V_1(x) * V_3(x,z,y) \\ \wedge & \lambda xyz.G'(x) = \lambda xyz.V_1(x) \\ \wedge & \lambda xyz.H(x,y,z) = \lambda xyz.V_2(x,y,z) * V_3(x,y,z) \end{array}$$

Decompose *applies to the second equation which is equivalent to*

$$\lambda xyz.V_1(x) = \lambda xyz.V_1(x) \ \wedge \ \lambda xyz.V_2(x,y,z) = \lambda xyz.V_3(x,z,y)$$

The equation $\lambda xyz.V_1(x) = \lambda xyz.V_1(x)$ is removed by **Trivial***, and* **Coalesce** *applies to the second. The problem to be solved is now*

$$\begin{array}{rl} (\exists V_1 V_2 V_3) & \lambda xyz.F(x,y,z) = \lambda xyz.V_1(x) * V_3(x,z,y) \\ \wedge & \lambda xyz.V_2(x,y,z) = \lambda xyz.V_3(x,z,y) \\ \wedge & \lambda xyz.G'(x) = \lambda xyz.V_1(x) \\ \wedge & \lambda xyz.H(x,y,z) = \lambda xyz.V_3(x,z,y) * V_3(x,y,z) \end{array}$$

EQE *removes the useless existentially quantified variable V_2, yielding*

$$\begin{array}{rl} (\exists V_1 V_3) & \lambda xyz.F(x,y,z) = \lambda xyz.V_1(x) * V_3(x,z,y) \\ \wedge & \lambda xyz.G'(x) = \lambda xyz.V_1(x) \\ \wedge & \lambda xyz.H(x,y,z) = \lambda xyz.V_3(x,z,y) * V_3(x,y,z) \end{array}$$

The reader can now check that the substitution

$$\{F \mapsto \lambda xyz.V_1(x) * V_3(x,z,y), G' \mapsto \lambda xyz.V_1(x), H \mapsto \lambda xyz.V_3(x,z,y) * V_3(x,y,z)\}$$

is a constant-preserving solution of

$$\lambda xyz.F(x,y,z) + F(x,z,y) = \lambda x.G'(x) * H(x,y,z)$$

modulo the left-distributivity.

Lemma 4 **Mutate** *preserves the sets of constant-preserving solutions.*

Proof: The soundness is straightforward: a constant-preserving solution of the right-hand side of the rule is a constant-preserving solution of its left-hand side. We show the completeness: consider the equation

$$\lambda\overline{x_k}.f(s_1,\ldots,s_n) = \lambda\overline{x_k}.g(t_1,\ldots,t_m)$$

where f and g are algebraic constants, to be solved in a syntactic theory E. By Tannen's theorem, a solution σ in η-long, β-normal form must satisfy

$$\lambda\overline{x_k}.f(s_1\sigma\updownarrow_\beta^\eta,\ldots,s_n\sigma\updownarrow_\beta^\eta) =_E \lambda\overline{x_k}.g(t_1\sigma\updownarrow_\beta^\eta,\ldots,\sigma\updownarrow_\beta^\eta t_m)$$

Since E is syntactic, there exists $f(l_1,\ldots,l_n) = g(r_1,\ldots,r_m) \in E_{f,g}$[¶] and a substitution θ such that $s_i\sigma\updownarrow_\beta^\eta = l_i\theta$ for $1 \leq i \leq n$ and $t_i\sigma\updownarrow_\beta^\eta = r_i\theta$ for $1 \leq i \leq m$. In other words, σ is an E-solution of

$$(\exists\mathcal{F}\mathcal{V}(f(\overline{l_n}) = g(\overline{r_m}))) \bigwedge_{1\leq i\leq n} s_i = l_i \bigwedge_{1\leq i\leq m} t_i = r_i$$

hence of

$$(\exists\mathcal{F}\mathcal{V}(f(\overline{l_n}) = g(\overline{r_m}))) \bigwedge_{1\leq i\leq n} \lambda\overline{x_k}.s_i = \lambda\overline{x_k}.l_i \bigwedge_{1\leq i\leq m} \lambda\overline{x_k}.t_i = \lambda\overline{x_k}.r_i$$

Now, we do not *need* to guess an $\overline{x_k}$-variant of $f(\overline{l_n}) = g(\overline{r_m})$ for the correctness of the rule, but guessing which bound variables will occur as arguments of the variables of the axioms will allow the algorithm to fail when encountering an equation $\lambda\overline{x}.s = \lambda\overline{x}.t$ with $\{\overline{x}\} \cap \mathcal{F}\mathcal{V}(s) \neq \{\overline{x}\} \cap \mathcal{F}\mathcal{V}(t)$. □

4.2 The algorithm

The algorithm of figure 2 has to be adapted in presence of an equational theory. First, the rule **Clash** has to be modified. Indeed, an equation may be solvable if the heads of its left-hand-side and righ-hand side are different algebraic constants, by applying **Mutate** (see figure 3).

The rule **Decompose** can be removed if one assumes that for every constant f of arity n, $f(x_1,\ldots,x_n) = f(x_1,\ldots,x_n) \in E_{f,f}$.

More interesting is the case of the flexible-flexible equations with the same heads. It has been noticed by Qian and Wang that although such equations are always solvable by a projection, they do not have finite complete sets of AC-unifiers.

Example 5 ([21]) *Consider the equation* $e \equiv \lambda xy.F(x,y) = \lambda xy.F(y,x)$ *in the AC-theory of* $+$*. For* $m \geq 0$*, the substitution*

$$\sigma_m = \{F \mapsto \lambda xy.G_m(H_1(x,y)+H_1(y,x),\ldots,H_m(x,y)+H_m(y,x))\}$$

is an AC-unifier of e. On the other hand, every solution of e is an instance of some σ_i*. In addition* σ_{n+1} *is strictly more general than* σ_n*.*

[¶] Or possibly $f(X_1,\ldots,X_n) = f(X_1,\ldots,X_n)$ if $f = g$.

1. APPLY THE FOLLOWING RULE FOR EVERY FREE VARIABLE F OF P:

Project
$P \quad \Rightarrow \ F = \lambda\overline{x_n}F'(y_1,\dots,y_k) \ \wedge \ P\{F \mapsto \lambda\overline{x_n}F'(y_1,\dots,y_k)\}$
where F has arity n and F' is a new variable and $\{y_1,\dots,y_k\} \subseteq \{x_1,\dots,x_n\}$

2. APPLY AS LONG AS POSSIBLE THE RULES:

Fail
$\lambda\overline{x_k}.s = \lambda\overline{x_k}.t \ \wedge \ P \quad \Rightarrow \ \bot$
if $\mathcal{FV}(s) \cap \overline{x_k} \neq \mathcal{FV}(t) \cap \overline{x_k}$

FF=
$\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.F(\overline{z_n}) \ \wedge \ P \quad \Rightarrow \ \bot$
if $\{\overline{y_n}\} \neq \{\overline{z_n}\}$.

Freeze
$(\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.F(\overline{z_n}) \wedge P) \wedge P_F \quad \Rightarrow \ P \wedge (\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.F(\overline{z_n}) \wedge P_F)$
if $\overline{y_n}$ is a permutation of $\overline{z_n}$.

Trivial
$s = s \ \wedge \ P \quad \Rightarrow \ P$

FF$\neq$ (Coalesce)
$\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.G(\overline{z_n}) \ \wedge \ P \quad \Rightarrow \ \lambda\overline{y_n}.F(\overline{y_n}) = \lambda\overline{y_n}.G(\overline{z_n}) \ \wedge \ P\{F \mapsto \lambda\overline{y_n}.G(\overline{z_n})\}$
if $F \neq G$ and $F,G \in \mathcal{FV}(P)$ and $\overline{y_n}$ is a permutation of $\overline{z_n}$

Mutate
$\lambda\overline{x_k}.f(s_1,\dots,s_n) = \lambda\overline{x_k}.g(t_1,\dots,t_m)$
$\quad \Rightarrow \ (\exists\mathcal{V}(f(\overline{l_n}) = g(\overline{r_m}))) \ \bigwedge_{1\leq i\leq n}\lambda\overline{x_k}.s_i = \lambda\overline{x_k}.l_i \bigwedge_{1\leq j\leq m}\lambda\overline{x_k}.t_j = \lambda\overline{x_k}.r_j$
where $f(l_1,\dots,l_n) = g(r_1,\dots,r_m)$ is an $\overline{x_k}$-variant of an equation of $E_{f,g}$.

Merge
$F = s \ \wedge \ F = t \ \wedge \ P \quad \Rightarrow \ F = s \ \wedge \ s = t \ \wedge \ P$
if $|s| \leq |t|$

Clash
$\lambda\overline{x_k}.a(s_1,\dots,s_n) = \lambda\overline{x_k}.b(t_1,\dots,t_m) \ \wedge \ P \quad \Rightarrow \ \bot$
if $a \in \overline{x_k}$ or $b \in \overline{x_k}$, a and b are not a variable and $a \neq b$

Check*
$F_1 = s_1[F_2(\cdots)] \ \wedge \ F_2 = s_2[F_3(\cdots)] \ \wedge \ \cdots \ \wedge \ F_n = s_n[F_1(\cdots)] \ \wedge \ P \quad \Rightarrow \ \bot$
if one $s_i[\cdot]$ is not the empty context

Fig. 3: Algorithm for pattern unification modulo simple syntactic theories

Hence, AC-unification of patterns is not only infinitary, but *nullary*, in the sense that some problems do not have *minimal* complete sets of AC-unifiers [23].

As Qian and Wang, and as in [3], we keep these equations unaltered : the syntax of unification problems is slightly modified by distinguishing the conjunction P_F of *frozen equations* that will never be modified by the simplification rules. The rule **Freeze** ignores the flexible-flexible equations with same heads and *freezes* them by storing them in P_F. This is made necessary by the fact that even if P_0 does not contain such equations at the beginning, some may appear by applying the other rules. There are still no constant-preserving solutions of $\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.F(\overline{z_n})$ if $\{\overline{y_n}\}$ and $\{\overline{z_n}\}$ are not the same set, hence the rule **FF**= of figure 2 is replaced by the two rules **Freeze** and **FF**= of figure 3. We do not go into further detail now concerning frozen equations because first, we do not know how to handle them in general, and second, they will just lead to failure when all function symbols are decomposable as, for instance in the case of one-sided distributivity (see section 6).

Figure 3 presents our algorithm for pattern unification modulo a simple syntactic equational theory E. The fact that E is a simple theory is needed to preserve the completeness of the rules **Clash** and **Check***, which are not correct in general. The rule **Check*** is correct as a corollary of the following lemma :

Lemma 5 *The equation $\lambda\overline{x}.F(\overline{y}) = \lambda\overline{x}.s[F(\overline{z})]_p$ has no E-solution if E is a simple theory and $p \neq \Lambda$.*

Proof: By contradiction. Assume that σ is a solution in η-long β-normal form, and let $F\sigma = \lambda\overline{v}.t[\overline{v}]$. We have

$$\lambda\overline{x}.F\sigma(\overline{y}) = \lambda\overline{x}.\lambda\overline{v}.t[\overline{v}](\overline{y}) =_\beta \lambda\overline{x}.t[\overline{y}]$$

with $\lambda\overline{x}.t[\overline{y}]$ in η-long β-normal form. On the other side, we have

$$(\lambda\overline{x}.s[F(\overline{z})]_p)\sigma = \lambda\overline{x}.s\sigma[\lambda\overline{v}.t[\overline{v}](\overline{z})]_p =_\beta \lambda\overline{x}.(s\sigma)\updownarrow_\beta^\eta[t[\overline{z}]]_p$$

There is a proof of $\lambda\overline{x}.t[\overline{y}] =_E \lambda\overline{x}.(s\sigma)\updownarrow_\beta^\eta[t[\overline{z}]]_p$, hence there is a proof of $t[\overline{y}] =_E (s\sigma)\updownarrow_\beta^\eta[t[\overline{z}]]_p$. But *a fortiori*, there is a proof of the above identity where all the occurrences of bound variables have been replaced by a constant a. Let t' (resp. s') be t (resp. $(s\sigma)\updownarrow_\beta^\eta$), where all the occurrences of bound variables have been replaced by a. We have $t' =_E s'[t']_p$ with $p \neq \Lambda$, which is impossible for a simple theory E. □

The rule **Clash** is correct as a corollary of the following lemma :

Lemma 6 *The equation $\lambda\overline{x}.x_i(\overline{s}) = \lambda\overline{x}.a(\overline{t})$ has no E-solution if $x_i \in \{\overline{x}\}$ and E is a simple theory and a is a bound variable different than x_i or a constant.*

Proof: By contradiction : assume σ is a solution. If a is a bound variable x_j $(i \neq j)$, then we would have a proof of $\lambda\overline{x}.x_i(\overline{s\sigma\updownarrow_\beta^\eta}) = \lambda\overline{x}.x_j(\overline{t\sigma\updownarrow_\beta^\eta})$, hence of $x_i(\overline{s\sigma\updownarrow_\beta^\eta} = x_j(\overline{t\sigma\updownarrow_\beta^\eta})$, which is impossible since neither x_i nor x_j appear in the axioms of E. If a is an algebraic constant f, thet there would be a proof of $x_i(\overline{s\sigma\updownarrow_\beta^\eta} = f(\overline{t\sigma\updownarrow_\beta^\eta})$ which is again impossible since a simple theory admits no identities with a function symbol at the top on one side only. □

It is now easy to show that our rules, after the projection step, mimic exactly those of figure 1, except for the more numerous failure cases due to the restriction to constant-preserving solutions.

5 Patterns as first-order terms

We give now an interpretation of pattern unification problems in terms of first-order unification problems. It just consists of forgetting the arguments of the free variables which are replaced by first-order variables, and the lambda-bindings, and of replacing the bound variables by new free constants.

Definition 7 *For $n \in N$, let f^n be a free function symbol of arity n. The interpretation $I(t)$ of a pattern t is the first-order term obtained from t by*

1. *replacing the flexible subterms of the form $X(\overline{x})$ by v^X, where X is a free variable and v^X is a first-order variable associated with X,*

2. *replacing the subterms $a(\overline{s})$ by $f^n(\overline{s})$, where a is a bound variable of arity n.*

3. *replacing the subterms of the form $\lambda x.s$ by s.*

The interpretation $I(P)$ of a pattern unification problem P is the first-order unification problem obtained from P by replacing every term t by its interpretation $I(t)$. The frozen part P_F is ignored.

We will now take advantage of the fact that the algorithms of figures 2 and 3 are complete (The rules preserve the solutions and the constant-preserving solutions respectively).

Lemma 7 *Assume that a pattern unification problem P is transformed into Q by an application of one of the non-failure rules of figure 3 (except the rule* **Project***). Then there is a rule* **R** *of the algorithm of figure 1 such that $I(Q)$ is obtained from $I(P)$ by an application of* **R***.*

Proof: The rules **Merge** and **Mutate** of figure 3 correspond to those of figure 1. The rules **Freeze** and **Trivial** of figure 3 correspond to the rule **Trivial** of figure 1. The rule **FF≠ (Coalesce)** corresponds to the rule **Var-Rep (Coalesce)** of figure 1. □

Corollary 1 *If the algorithm of figure 1 terminates for a given theory E, so does our pattern unification algorithm.*

Corollary 2 *If the first-order unification problem $I(P)$ has no solution, then P has no solution.*

Proof: It is enough to notice that if P is irreducible by the rules of figure 3, then so is $I(P)$ by the rules of figure 1. In both cases the irreducible problems are either $\perp$, or a DAG-solved form. The result follows by contradiction. □

This last result allows one to use some criteria for the first-order case such as the one given by Arnborg and Tidén for one-sided distributivity [24]. For this theory the syntactic algorithm does not terminate, but the authors give a criterion allowing to discard some unsolvable problems, providing a terminating algorithm.

6 Frozen equations

The algorithm that we have presented so far transforms a unification problem into a problem of the form $P \wedge P_F$, where P is a DAG-solved form and P_F is a conjunction of frozen equations of the form $\lambda\overline{x}.F(\overline{y}) = \lambda\overline{x}.F(\overline{z})$, where $\overline{z}$ is a permutation of $\overline{y}$. There remains to check whether there exists an instance of the mgu of P that satisfies the equations of P_F. The problem arises when the variable F has a "value" in P, that is, when there is an equation of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.t$ in P.

Example 6 *Assume that $+$ is an associative-commutative function symbol and that $P \wedge P_F$ is $F = \lambda xy.x+a+y \wedge \lambda xy.F(x,y) = \lambda xy.F(y,x)$. Then, the mgu $\{F \mapsto \lambda xy.x+a+y\}$ of P satisfies P_F, and we are done. Assume now that $P \wedge P_F$ is $F = \lambda xy.H(x,y) \wedge \lambda xy.F(x,y) = \lambda xy.F(y,x)$, and that $+$ is a commutative function symbol. Then the substitution $\{H \mapsto \lambda xy.H'(x,y)+H'(y,x)\}$ will map the mgu $\{F \mapsto \lambda xy.H(x,y)\}$ of P onto $\{F \mapsto \lambda xy.H'(x,y)+H'(y,x)\}$ which satisfies P_F. Finally, if f is a free function symbol and $P \wedge P_F$ is $F = \lambda xy.f(x,y) \wedge \lambda xy.F(x,y) = \lambda xy.F(y,x)$, then there is no instance of the mgu $\{F \mapsto \lambda xy.f(x,y)\}$ of P satisfying P_F.*

We do not have a general solution to this problem, but we propose a straightforward rule which may cause non-termination in general, and a method that will work for theories such as left-distributivity where the function symbols of the theory are *decomposable*.

The obvious rule for computing the solutions of P that satisfy P_F is the following :

F-Merge
$F = \lambda\overline{x}.t \wedge \lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x^\pi}) \wedge P$
$\Rightarrow\ F = \lambda\overline{x}.t \wedge \lambda\overline{x}.t = \lambda\overline{x^\pi}.t \wedge P$
where $\overline{x^\pi}$ is a permutation of $\overline{x}$

Of course, we cannot guarantee the termination since the solving of the new equation $\lambda\overline{x}.t = \lambda\overline{x^\pi}.t$ can yield new flexible flexible equations making it necessary to apply **F-Merge** again, and so on. Actually, we do conjecture that it is not decidable in general, given a theory E with decidable first-order unification, whether an equation of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x^\pi})$, where $\overline{x^\pi}$ is a permutation of $\overline{x}$, has a *non-trivial E-solution*, that is a solution which is not a solution modulo the empty theory.

Definition 8 *A function symbol f is* decomposable *if*

$$f(s_1,\ldots,s_n) =_E f(t_1,\ldots,t_n) \Longleftrightarrow s_1 =_E t_1 \& \cdots \& s_n =_E t_n$$

Arnborg and Tidén [24] have shown that the axion of left-distributivity forms a resolvent presentation. This implies that both $+$ and $*$ are decomposable, since there is no axiom with $+$ (or $*$) at the top on both sides.

Proposition 2 *We call* trivial *an E-solution of an equation which is also a solution modulo the empty theory. Assume that E is such that all the function symbols are decomposable. Then, the equations of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x^\pi})$ have no non-trivial E-solutions.*

Proof: By contradiction, and induction on the structure of the value $F\sigma$ of F by the alledged non-trivial solution σ in η-long β-normal form. If $F\sigma$ is of the form $\lambda\overline{x}.\alpha(t_1,\ldots,t_n)$, where α is a free variable or a bound variable, we have $\lambda\overline{x}.\alpha(t_1,\ldots,t_n) =_E \lambda\overline{x^\pi}.\alpha(t_1,\ldots,t_n)$, hence $\lambda\overline{x}.t_i = \lambda\overline{x^\pi}.t_i$ which is impossible by the induction hypothesis. The same holds if $F\sigma$ is of the form $\lambda\overline{x}.f(t_1,\ldots,t_n)$, where f is a decomposable constant. $\square$

When all function symbols are decomposable, the only solutions to flexible-flexible equations with the same head on both sides are projections as in the non-equational case. Since we are interested in constant-preserving solutions, we can replace the **Freeze** rule by the following failure rule.

F-Fail
$\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.F(\overline{z_n}) \wedge P \quad \Rightarrow \quad \bot$
if $\overline{y_n}$ is a permutation of $\overline{z_n}$ other than the identity.

Theorem 3 *Assume E is a simple syntactic equational theory such that every function symbol is decomposable. Assume that the algorithm given in figure 2 terminates for E in the first-order case. Then, the algorithm of figure 3, where the rule* **Freeze** *has been replaced by* **F-Fail** *terminates and implements a complete pattern E-unification algorithm.*

7 Conclusion

We believe that with the emergence of higher-order rewriting, higher-order logic programming and functional-algebraic programming languages, equational pattern unification will be useful. It is certainly not a good idea to perform a non-deterministic projection step for standard pattern unification. We have used this trick because when one is interested in constant-preserving substitutions, the equations with one free variable on one side behave as in the first order case. Either they have a suitable sequence of arguments and the equation is quasi-solved, or there is no constant-preserving solution.

Surprisingly, the main difficulty comes from equations like $\lambda xy.F(x,y) = \lambda xy.F(y,x)$. In the empty theory, such equations cause no problem, but in the equational case, it is the only "higher-order" problem we have encountered. The problem of the existence of non-trivial solutions to such equations could be rephrased as

> Does there exist a first-order term with n variables which is invariant modulo E when some variables are permuted?

We believe that this problem is undecidable in general for theories with decidable unification, but we will try and provide ad-hoc solutions for some familiar theories.

The assumption that E is a simple theory is essentially technical, and could be dropped as it has been done in the first-order case for unification in combinations of equational theories. The mechanisms used for preventing or solving cycles and clashes[22, 1], should be adaptable to the pattern unification context. Then, the syntactic approach could apply to larger classes of equational theories such as the *shallow* theories [6].

References

[1] Alexandre Boudet. Combining unification algorithms. *Journal of Symbolic Computation*, 16:597–626, 1993.

[2] Alexandre Boudet and Evelyne Contejean. “Syntactic” AC-unification. In Jouannaud [11], pages 136–151.

[3] Alexandre Boudet and Evelyne Contejean. AC-unification of higher-order patterns. In Gert Smolka, editor, *Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, pages 267–281, Linz, Austria, October 1997. Springer-Verlag.

[4] Alexandre Boudet and Evelyne Contejean. About the Confluence of Equational Pattern Rewrite Systems. In C. and H. Kirchner, editors, *15th International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 88–102. Springer-Verlag, 1998.

[5] Val Breazu-Tannen. Combining algebra and higher-order types. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, July 1988.

[6] Hubert Comon, Marianne Haberstrau, and Jean-Pierre Jouannaud. Syntacticness, cycle-syntacticness and shallow theories. *Information and Computation*, 111(1):154–191, May 1994.

[7] Gilles Dowek. Third order matching is decidable. In *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 2–10. IEEE Comp. Soc. Press, IEEE Comp. Society Press, 1992.

[8] Warren D. Goldfarb. Note on the undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[9] R. Hindley and J. Seldin. *Introduction to Combinators and λ-calculus*. Cambridge University Press, 1986.

[10] Gérard Huet. *Résolution d'équations dans les langages d'ordre* $1,2,\ldots\omega$. Thèse d'Etat, Univ. Paris 7, 1976.

[11] Jean-Pierre Jouannaud, editor. *First International Conference on Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, München, Germany, September 1994. Springer-Verlag.

[12] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT-Press, 1991.

[13] Claude Kirchner. Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories equationnelles. Thèse d'Etat, Univ. Nancy, France, 1985.

[14] Claude Kirchner. Computing unification algorithms. In *Proc. 1st IEEE Symp. Logic in Computer Science, Cambridge, Mass.*, pages 206–216, 1986.

[15] Claude Kirchner and Francis Klay. Syntactic theories and unification. In *Proc. 5th IEEE Symp. Logic in Computer Science, Philadelphia*, June 1990.

[16] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, February 1998.

[17] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In P. Schroeder-Heister, editor, *Extensions of Logic Programming*. LNCS 475, Springer Verlag, 1991.

[18] T. Nipkow. Higher order critical pairs. In *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.

[19] Vincent Padovani. *Filtrage d'ordre supérieur*. PhD thesis, Université de Paris VII, 1996.

[20] Gordon Plotkin. Building-in equational theories. *Machine Intelligence*, 7, 1972.

[21] Zhenyu Qian and Kang Wang. Modular AC-Unification of Higher-Order Patterns. In Jouannaud [11], pages 105–120.

[22] M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *Journal of Symbolic Computation*, 1990. Special issue on Unification.

[23] Jörg H. Siekmann. Unification theory. *Journal of Symbolic Computation*, 7(3 & 4), 1989. Special issue on unification, part one.

[24] Erik Tidén and Stefan Arnborg. Unification problems with one-sided distributivity. *Journal of Symbolic Computation*, 3(1–2):183–202, 1987.