Combining Pattern *E*-unification Algorithms*

Alexandre Boudet and Evelyne Contejean

LRI, CNRS UMR 8623, Bât. 490, Université Paris-Sud, Centre d'Orsay,
91405 Orsay Cedex, France

Abstract. We present an algorithm for unification of higher-order patterns modulo combinations of disjoint first-order equational theories. This algorithm is highly non-deterministic, in the spirit of those by Schmidt-Schauß [20] and Baader-Schulz [1] in the first-order case. We redefine the properties required for elementary pattern unification algorithms of pure problems in this context, then we show that some theories of interest have elementary unification algorithms fitting our requirements. This provides a unification algorithm for patterns modulo the combination of theories such as the free theory, commutativity, one-sided distributivity, associativity-commutativity and some of its extensions, including Abelian groups.
Keywords. Combination of unification algorithms – Pattern equational unification

Introduction

Patterns have been defined by Miller [18] in order to provide a compromise between simply-typed lambda-terms for which unification is known to be undecidable [12, 9] and mere first-order terms which are deprived of any abstraction mechanism. A *pattern* is a term of the simply-typed lambda-calculus in which the arguments of a free variable are all pairwise distinct bound variables. Patterns are close to first-order terms in that the free variables (with their bound variables as only permitted arguments) are at the leaves. Under this rather drastic restriction, unification becomes decidable and unitary:

Theorem 1 ([18]). *It is decidable whether two patterns are unifiable, and there exists an algorithm which computes a most general unifier of any two unifiable patterns.*

Yet, patterns are useful in practice for defining higher-order pattern rewrite systems [17, 6], or for defining functions by cases in functional programming languages. Some efforts have been devoted to the study of languages combining functional programming (lambda-calculus) and algebraic programming (term rewriting systems) [15, 13, 7].

* This research was supported in part by the EWG CCL, and the RNRT project Calife.

In this paper we provide a nondeterministic algorithm for combining elementary equational patterns unification algorithms. This is the object of section 2. In sections 3 and 4, we show that such elementary unification algorithms exist for theories such as the free theory, commutativity, one-sided distributivity, as well as associativity-commutativity and its common extensions including Abelian groups.

Our method does not consist of using a first-order unification algorithm for the combined equational theories extended to the case of patterns. Such an approach has been used by Qian & Wang [19], but considering a first-order unification algorithm as a black box leads to incompleteness (see example in [5]). What we need here is a *pattern* unification algorithm for each of the theories to be combined *plus* a combination algorithm for the elementary E_i-pattern unification algorithm. Evidence of this need is that contrarily as in the first-order case, the unifier set of the pure equation $\lambda xy.Fxy = \lambda xy.Fyx$ modulo the free theory (no equational axioms) changes if one adds say a commutative axiom $x + y = y + x$ (see [19]). The requirements we have on elementary pattern unification algorithms are very much in the spirit of those needed in the first-order case, yet there are relevant differences due in particular to the possible presence of equations with the same free variable at the head on both sides (like $\lambda xy.Fxy = \lambda xy.Fyx$).

The worst difficulties, for the combination part as well as for the elementary unification algorithms, come from such equations which have no minimal complete sets of E-unifiers, even for theories which are finitary unifying in the first-order case. On top of this, the solutions of such equations introduce terms which are not patterns. For this reason, we will never attempt to solve such equations explicitly, but we will keep them as *constraints*. The output of our algorithm is a (DAG-) solved form constrained by some equations of the above form and *compatible* with them. As the algorithm by Baader and Schulz [1], ours can be used for combining decision procedures.

1 Preliminaries

We assume the reader is familiar with simply-typed lambda-calculus, and equational unification. Some background is available in *e.g.* [10, 14] for lambda-calculus and E-unification.

1.1 Patterns and equational theories

Miller [18] 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: $\lambda xyz.f(H(x,y),H(x,z))$ and $\lambda x.F(\lambda z.x(z))$[1] 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. We shall use the following notations: the sequence of variables $x_1,\ldots,x_n$ will be written $\overline{x_n}$ or even $\overline{x}$ if n is not relevant. Hence $\lambda x_1\cdots\lambda x_n.s$ will be written $\lambda\overline{x_n}.s$, or even $\lambda\overline{x}.s$. If in a same expression $\overline{x}$ appears several times it denotes the same sequence of variables. If π is a permutation of $\{1,\ldots,n\}$, $\overline{x}^\pi$ denotes the sequence $x_{\pi(1)},\ldots,x_{\pi(n)}$. In the following, we shall use either $\lambda\overline{x}.F(\overline{x}^\pi)$ or the α-equivalent term $\lambda\overline{y}^\varphi.F(\overline{y})$, where $\varphi=\pi^{-1}$, in order to denote $\lambda x_1\cdots\lambda x_n.F(x_{\pi(1)},\ldots,x_{\pi(n)})$. 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 variables of a term t are denoted by $\mathcal{FV}(t)$.

$t|_p$ 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 [10], the β and η rules being respectively oriented as follows: $(\lambda x.M)N\to_\beta M\{x\mapsto N\}$ and $F\to_{\eta\uparrow}\lambda\overline{x_n}.F(\overline{x_n})$ if the type of F is $\alpha_1\to\ldots\to\alpha_n\to\alpha$, 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 $t\updownarrow^\eta_\beta$.

A *substitution* σ is a mapping from a finite set of variables to terms of the same type, written $\sigma=\{X_1\mapsto t_1,\ldots,X_n\mapsto t_n\}$. The set $\{X_1,\ldots,X_n\}$ is called the *Domain* of σ and denoted by $\mathcal{D}\mathrm{om}(\sigma)$.

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 $\mathcal{F}$, $=_E$ is the least congruence[2] containing all the identities $l\sigma=r\sigma$ where $l=r\in E$ and σ is a suitably typed substitution. $=_{\eta\beta E}$ is then the least congruence containing $=_E$, $=_\eta$ and $=_\beta$.

The following is a key theorem by 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 ([7]). *Let E be an equational theory and s and t two terms. Then $s=_{\eta\beta E}t\iff s\updownarrow^\eta_\beta=_E t\updownarrow^\eta_\beta$.*

---

[1] 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.

[2] compatible *also* with application and λ-abstraction in our context.

1.2 Unification problems

Unification problems are formulas built-up using only the equality predicate $=$ (between *terms*), conjunctions, disjunctions and existential quantifiers. The solutions of $s = t$ are the substitutions σ such that $s\sigma =_{\eta\beta E} t\sigma$. This definition extends the natural way to unification problems. We restrict our attention to problems of the form $(\exists \overline{X})\ s_1 = t_1 \wedge \cdots \wedge s_n = t_n$, the only disjunctions being implicitly 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 explicitly 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 (by using α-conversion if necessary) because the two terms have to be in η-long β-normal form and of the same type. 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 1. *A* flexible *pattern is a term of the form $\lambda\overline{x}.F(\overline{y})$ where F is a free variable and $\{\overline{y}\} \subseteq \{\overline{x}\}$. A* flex-flex *equation is an equation between two flexible patterns. 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 $F \notin \mathcal{FV}(s) \cup \{\overline{x_k}\}$. A variable is* solved *in a unification problem if it occurs only once as the left-hand side of a quasi-solved equation.*

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 often write a quasi-solved equation in the form $F = \lambda\overline{y_n}.s$ instead of $\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.s$.

Definition 2. *A* DAG-solved form *is a 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 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	$\lambda\overline{x_k}.F(\overline{y_n}) = \lambda\overline{x_k}.s \ \wedge\ P \quad \rightarrow\ F = \lambda\overline{y_n}.s \ \wedge\ P$
Replacement	$F = \lambda\overline{y_n}.s \ \wedge\ P \quad \rightarrow\ F = \lambda\overline{y_n}.s \ \wedge\ P\{F \mapsto \lambda\overline{y_n}.s\}$ if F has a free occurrence in P.
EQE	$(\exists F)\ F = t \ \wedge\ P \quad \rightarrow\ P$ if F has no free occurrence in P.

1.3 Combinations of equational theories

As in the first-order case [22, 16, 8, 5], we will consider a *combination of equational theories*. We assume that $E_0, \ldots, E_n$ are equational theories over disjoint signatures $\mathcal{F}_0, \ldots, \mathcal{F}_n$, and we will provide a unification algorithm for the theory E presented by the reunion of the presentations, provided an *elementary E_i-unification algorithm* for patterns is known for each E_i. As in the first-order case, we have some further assumptions on elementary E_i-unification that will be made precise later.

Definition 3. *The* theory *of a bound variable, or of a free algebraic symbol* i.e., *a symbol which does not appear in any axiom is the free theory E_0. The theory of an algebraic symbol $f \in \mathcal{F}_i$ is E_i. A variable F is E_i-instantiated in a unification problem P if it occurs in a quasi-solved equation $F = s$ where the head of s has theory E_i. A variable F is E_i-instantiated by a substitution σ if the head of $F\sigma$ has theory E_i.*

We first give two well-known rules in order to obtain equations between *pure terms* only.

Definition 4. *The subterm u of the term $t[u]_p$ is an* alien subterm *in t if it occurs as an argument of a symbol from a different theory from its head symbol.*

VA	$\lambda\overline{x}.s[u]_p = \lambda\overline{x}.t \quad \rightarrow\ \exists F\ \lambda\overline{x}.s[F]_p = \lambda\overline{x}.t \ \wedge\ \lambda\overline{y}.F(\overline{y}) = \lambda\overline{y}.u$ if u is an alien subterm and $\{\overline{y}\} = \{\overline{x}\} \cap \mathcal{FV}(u)$ and F is a new variable of appropriate type.
Split	$\lambda\overline{x}.\gamma(\overline{s}) = \lambda\overline{x}.\delta(\overline{t}) \ \rightarrow \exists F\ \lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.\gamma(\overline{s}) \ \wedge\ \lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.\delta(\overline{t})$ if γ and δ are not free variables and belong to different theories, where F is a new variable of appropriate type.

The above rules obviously terminate and yield a problem which is equivalent to the original problem and where all the equations are *pure*,

i.e., containing only symbols from a same theory. We can now split a unification problem into several pure subproblems:

Definition 5. *A unification problem P will be written in the form*

$$P \equiv (\exists \overline{X})\ P_F \wedge P_V \wedge P_0 \wedge P_1 \wedge \cdots \wedge P_n \qquad \textit{where}$$

- *P_F is the set of equations of the form $\lambda\overline{x_n}.F(\overline{x_n}) = \lambda\overline{x_n}.F(\overline{x_n}^{\pi})$, where π is a permutation of $\{1,\ldots,n\}$. Such equations will be called* frozen[3].
- *P_V is the set of equations of the form $\lambda\overline{x}.F(\overline{y}) = \lambda\overline{x}.G(\overline{z})$, where F and G are different free variables.*
- *$P_0,\ldots,P_n$ are pure problems in the theories $E_0,\ldots,E_n$ respectively.*

2 A combination algorithm

In this section, we will present a non-deterministic algorithm, one step beyond that by Baader and Schulz [1], who extend the method initiated by Schmidt-Schauß [20] for unification in combinations of equational theories. In particular, we will guess a projection for each free variable, and then restrict our attention to *constant-preserving substitutions*, like we suggested in [4].

The aim of these rules is to guess in advance the form of a unifier, and to make the algorithm fail when the solutions of the problem do not correspond to the choices that are being considered currently. The drawback of such an approach is a blow-up in the complexity, but it allows to avoid recursion, hence guaranteeing termination.

After the variable abstraction step, we want to guess, for each free variable F of arity k (*i.e.*, whose η-long form is $\lambda\overline{x_k}.F(x_1,\ldots,x_k)$) which of the bound variables $x_1,\ldots,x_k$ will effectively participate in the solution.

Definition 6. *A* constant-preserving substitution *is a substitution σ such that for all $F \in Dom(\sigma)$ if $F\sigma\downarrow^{\eta}_{\beta} = \lambda\overline{x_k}.s$ then every variable of $\overline{x_k}$ has a free occurrence in s. A* projection *is a substitution of the form*

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

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

[3] These equations are always trivially solvable, for example by $F \mapsto \lambda\overline{x_n}.C$, where C is a variable of the appropriate base type, but we will never solve them explicitly because they have no minimal complete sets of unifiers and their solutions introduce terms which are not patterns, see [19, 5].

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 E-solution. In particular, the equation* $\lambda\overline{x}.F(\overline{y}) = \lambda\overline{x}.G(\overline{z})$*, where* $\{\overline{y}\}$ *and* $\{\overline{z}\}$ *are not the same set, has no constant-preserving E-solution.*

This will allow us to choose (and apply) a projection for the free variables and to discard in the sequel the problems whose solutions are not constant-preserving.

2.1 Non-deterministic choices

At this point, we will guess through "don't know" nondeterministic choices some properties of the solutions. The idea is that once a choice has been made, some failure rules will apply when the solutions of the current problem correspond to other choices. This method initiated by Schmidt-Schauß [20] is obviously correct because the solutions of the problems that are discarded this way correspond to another choice and will be computed in the corresponding branch.

The following transformations have to be successively applied to the problem:

C.1 Choose a projection for the free variables
We first guess for a given solution σ, the projection of which σ will be an instance by a constant-preserving substitution, in the conditions of lemma 2. This is achieved by applying nondeterministically the following rule to some of the free variables F of the problem:

Project $P \quad \rightarrow \ (\exists F')\ F = \lambda\overline{x_n}.F'(\overline{y_k}) \wedge P\{F \mapsto \lambda\overline{x_n}.F'(\overline{y_k})\}$
where F has arity n and F' is a new variable and $\{\overline{y_k}\} \subset \{\overline{x_n}\}$

After this step, we can restrict our attention to constant-preserving solutions.

C.2 Choose some flex-flex equations
We now guess the equations of the form $\lambda\overline{x}.F(\overline{y}) = \lambda\overline{x}.G(\overline{z})$ that will be satisfied by a solution σ. This is done by applying the following rule to some pairs $\{F, G\}$ of the free variables of the problem:

$\mathbf{FF}_{\neq} \quad P \quad \rightarrow \ F = \lambda\overline{x}.G(\overline{x}^{\pi}) \ \wedge \ P\{F \mapsto \lambda\overline{x}.G(\overline{x}^{\pi})\}$
where π is a permutation of $\{1, \ldots, n\}$, F has type $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau$, G has type $\tau_{\pi(1)} \rightarrow \cdots \rightarrow \tau_{\pi(n)} \rightarrow \tau$, $F \neq G$ and F and G occur in P.

We restrict the application of this rule to pairs of variables of the same arity and of the same type (up to a permutation of the types of the arguments), because after applying **Project**, the only flex-flex equations admitting constant-preserving solutions are of this form.

C.3 Choose the permutations on the arguments of the variables
For each free variable F of type $\tau_1 \to \cdots \to \tau_n \to \tau$, we choose the group of permutations $\mathrm{Perm}(F)$ such that a solution σ satisfies $\lambda\overline{x_n}.F\sigma(\overline{x_n}) =_{\eta\beta E} \lambda\overline{x_n}.F\sigma(\overline{x_n}^{\pi})$ for each $\pi \in \mathrm{Perm}(F)$. For this, we apply the following rule to some of the free variables F of the problem:

FF$_=$ $\quad P \quad \to \lambda\overline{x_n}.F(\overline{x_n}) = \lambda\overline{x_n}.F(\overline{x_n}^{\pi}) \wedge P$
where F is a free variable of P of type $\tau_1 \to \cdots \to \tau_n \to \tau$ and π is a permutation such that $\tau_{\pi(i)} = \tau_i$ for $1 \leq i \leq n$.

C.4 Apply as long as possible the following transformation:

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

After **Project** has been applied, the arity of the values of the variables is fixed, hence two variables may be identified only if they have the same arity. Note that $F = \lambda\overline{y_n}.G(\overline{z_n})$ is solved after the application of **Coalesce**. After this step, we have an equivalence relation on the variables, and a notion of representative:

Definition 7. *Two variables F and G are* identified *in P if they appear in an equation $\lambda\overline{x}.F(\overline{y}) = \lambda\overline{x}.G(\overline{z})$ of P. The relation $=_V$ is the least equivalence containing any pair of identified variables.*
Assume that **Coalesce** *has been applied as long as possible to P. In an equivalence class of $=_V$, only a single variable may occur more than once in P. When such a variable exists, it is chosen as a* representative *for all variables in that class. Otherwise, the representative is chosen arbitrarily in the equivalence class.*

C.5 Choose a theory for the representatives
We now guess for each representative F, the theory E_i such that F is allowed to have the head symbol of its value by a solution in E_i. Again, this was already done by Schmidt-Schauß [20] and by Baader and Schulz.

C.6 Choose an ordering on representatives
Finally, we guess a total strict ordering compatible with the occur-check relation defined by $F{<}G$ if $G\sigma$ is a proper subterm of $F\sigma$. Choose a total ordering $<_{oc}$ on the representatives of the variables of the problem. This is exactly what Baader and Schulz do in the first-order case [1], reflecting the fact that if σ is a finite solution, then the occur-check relation must be acyclic.

2.2 Solving pure subproblems

We make now precise our assumptions on the elementary E_i unification algorithms. First, we take note of the fact that there is not much to do with the frozen equations of P_F:

Frozen Equations
Although they are always trivially solvable in the free theory, we will never try to solve the equations of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$ of P_F. These equations will be kept as *constraints* because they do not have finite complete sets of unifiers even for theories which have finitary first-order unification, and their solutions introduce terms which are not patterns. Here is an example by Qian and Wang:

Example 1 ([19]). Consider the equation $\lambda xy.F(x,y) = \lambda xy.F(y,x)$ 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 the above equation. 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.

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 [21]. All we can do is to make sure that the frozen equations in P_F are *compatible* with a (DAG-) solved form of the problem:

Definition 8. *Given a conjunction P_F of flex-flex equations of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$, we will write $P_F \models s =_{\eta\beta E} t$ if $s \updownarrow^\eta_\beta = t \updownarrow^\eta_\beta$ can be proved using the axioms of E and the equations $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$ of P_F, where F is treated like a free algebraic symbol. A substitution σ is* compatible *with P_F if for all equation $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$ of P_F, $P_F \models \lambda\overline{x}.F\sigma(\overline{x}) =_{\eta\beta E} \lambda\overline{x}.F\sigma(\overline{x}^\pi)$.*

Lemma 4. *If a substitution σ (seen as a conjunction of equations) is compatible with P_F as defined above, then the E-solutions of $\sigma \wedge P_F$ are the substitutions $\sigma\theta$, where θ is an E-solution of P_F.*

Definition 9. *Given a unification problem P, with no flex-flex equations and a conjunction P_F of (frozen) equations of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$, a* constrained E-solved form *of P is $P' \wedge P'_F$ where*

- *P' is a solved form with mgu σ, containing no equations of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$.*
- *P'_F contains P_F plus some equations of the form $\lambda\overline{x}.G(\overline{x}) = \lambda\overline{x}.G(\overline{x}^\pi)$, where G is a new variable not E-instantiated in P'.*

– $P'_F \models s\sigma =_{\eta\beta E} t\sigma$ *for every equation* $s=t$ *of* P.
– σ *is compatible with* P'_F.

In this case, σ *constrained by* P'_F*, denoted by* $\sigma\,|P'_F$ *is called a* constrained E-unifier *of* $P \wedge P_F$*. The* solutions *of* $\sigma\,|P'_F$ *are the substitutions of* $\sigma\theta$ *where* θ *is a solution of* P'_F.

Definition 10 (Solve rule for elementary theories).
A **Solve** *rule for the theory* E_i *is an algorithm that takes as input a problem* P_i*, pure in* E_i *and a conjunction* P_F *of frozen equations (as in definition 5) and that returns* P'_i *and* P'_{iF} *such that*

1. P'_i *is a solved form with mgu a constant-preserving substitution* σ[4].
2. P'_i *has no flex-flex equations.*
3. P'_{iF} *contains the equations of* P_F *plus some only flexible-flexible equations of the form* $\lambda\overline{x}.F(\overline{x})=\lambda\overline{x}.F(\overline{x}^\pi)$*, where* $F \notin \mathcal{FV}(P_i) \cup Dom(\sigma)$.
4. F *can be* E_i*-instantiated by* σ *only if* E_i *has been chosen as the theory of* F *at the step* **C.5**
5. $F\sigma$ *can be of the form* $\lambda\overline{x}.c[G(\cdots)]$*, where* $F, G \in \mathcal{FV}(P_i)$ *and another theory than* E_i *has been chosen for* G *at the step* **C.5**, *only if* $F <_{oc} G$*, for the ordering chosen at the step* **C.6**.
6. σ *is compatible with* P'_F.
7. $P'_{iF} \models \lambda\overline{x}.s =_{E_i} \lambda\overline{x}.t$ *for all the equations* $\lambda\overline{x}.s =_{E_i} \lambda\overline{x}.t$ *of* P_i.

Proposition 1. *Let* $s=t$ *be an equation, pure in the theory* E_i*, and let* σ *be an* E*-solution of* $s=t$*. Then there exists a set of equations* P_{perm} *of the form* $\lambda\overline{x}^\pi.F(\overline{x})=\lambda\overline{x}^\varphi.F(\overline{x})$*, and two substitutions* σ_{E_i} *and* θ *such that*

– $\sigma =_E \sigma_{E_i}\theta$.
– σ_{E_i} *is pure in the theory* E_i,
– θ *is an* E*-solution of* P_{perm}.
– $P_{perm} \models s\sigma_{E_i} =_{\eta\beta E_i} t\sigma_{E_i}$,
– *if* $F \in \mathcal{D}\mathrm{om}(\sigma)$ *and there exists a permutation* π *such that* $\lambda\overline{x}.F(\overline{x})\sigma =_{\eta\beta E} \lambda\overline{x}.F(\overline{x}^\pi)\sigma$*, then* $P_{perm} \models \lambda\overline{x}.F(\overline{x})\sigma_{E_i} =_{\eta\beta E_i} \lambda\overline{x}.F(\overline{x}^\pi)\sigma_{E_i}$.

The result is obtained by using Theorem 2 and adapting the proof of Theorem 5.1 of [3], which is the corresponding theorem in the first-order case.

[4] The correctness will be preserved if one allows non-constant-preserving substitutions, but the redundancy of the complete sets of unifiers will be increased in this case.

The algorithm

ALGORITHM FOR PATTERN UNIFICATION MODULO $E_0 \cup \cdots \cup E_n$

1. Apply as long as possible the rules **VA** and **Split** of section 1.
2. Perform successively the steps **C.1** to **C.6**.
3. Apply a **Solve** rule for theory E_i to each P_i accordingly to definition 10.
4. Return $P'_0 \wedge P'_1 \wedge \cdots \wedge P'_n \wedge P_F \wedge \bigwedge_{1 \leq i \leq n} P'_{iF}$.

Theorem 3. *Given an equational theory $E = E_0 \cup \cdots \cup E_n$, where the E_is are defined over disjoint signatures $\mathcal{F}_0, \ldots, \mathcal{F}_n$ and a unification problem P, containing only algebraic symbols of $\mathcal{F}_0 \cup \cdots \cup \mathcal{F}_n$,*

- *The above algorithm returns a constrained DAG-E-solved form of P.*
- *Every E-unifier of P is a solution of a constrained DAG-solved form computed by the above algorithm.*

Corollary 1. *Unifiability of higher-order patterns is decidable in combinations of theories having a* **Solve** *rule.*

3 A Solve rule for some syntactic theories

In [4], we show how to do pattern unification for a narrow class of theories: a subset of the simple syntactic theories. For lack of space, we just give here some hints on how to design a **Solve** rule for the free theory, the theory of left-distributivity LD and the commutativity C. These three theories are *simple* theories, *i.e.*, they have no equality between a term and one of its proper subterms. As it is well-known from the works on first-order unification, compound cycles or theory conflicts cannot be solved in such theories. It is easy to show that the following two rules are correct for simple theories:

Clash $F = s \quad \rightarrow \bot$
if F is E_i-instantiated and E_j, $j \neq i$ has been chosen for F at **C.5**.

Cycle $F = c[G] \quad \rightarrow \bot$
if c is a non-empty context and $F \not<_{oc} G$ for the ordering chosen at **C.6**.

Now, the free theory and LD have their symbols *decomposable* (*i.e.*, $f(s_1, \ldots, s_n) =_E f(t_1, \ldots, t_n)$ iff $s_i =_E t_i$) and C enjoys a similar property: $s_1 + s_2 =_C t_1 + t_2$ iff $s_1 =_C t_1 \wedge s_2 =_C t_2$ or $s_1 =_C t_2 \wedge s_2 =_C t_1$. Hence, the rules for testing the compatibility of a solved form with a frozen equation are:

Fail $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^{\pi}) \quad \rightarrow \bot$
if F is not a new variable and $\pi \notin \mathrm{Perm}(F)$ as chosen at the step **C.3**.
Dec-Propagate
$F = \lambda\overline{x}.f(\overline{s_n}) \wedge \lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^{\pi}) \quad \rightarrow$
$\quad F = \lambda\overline{x}.f(\overline{s_n}) \bigwedge_{1 \leq i \leq n} \lambda\overline{x}.s_i = \lambda\overline{x}^{\pi}.s_i$
if f is a decomposable constant or a bound variable.
C-Propagate
$F = \lambda\overline{x}.s_1 + s_2 \wedge \lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^{\pi}) \quad \rightarrow$
$\quad F = \lambda\overline{x}.s_1 + s_2 \ \wedge \ ((\lambda\overline{x}.s_1 = \lambda\overline{x}^{\pi}.s_1 \ \wedge \ \lambda\overline{x}.s_2 = \lambda\overline{x}^{\pi}.s_2)$
$\qquad\qquad \vee \ (\lambda\overline{x}.s_1 = \lambda\overline{x}^{\pi}.s_2 \ \wedge \ \lambda\overline{x}.s_2 = \lambda\overline{x}^{\pi}.s_1))$
if $+$ is a commutative algebraic symbol.

The **Mutate** rule of [4], together with the rule **Coalesce** and a failure rule when two (non-new) variables are identified allow us to compute a solved form satisfying the conditions 1 to 3 and 7 of definition 10. The first of the two above sets of rules allows us to fulfill conditions 4 and 5, and the second, condition 6.

4 From AC to Abelian Groups

In this section, we consider the associativity-commutativity, AC and some of its usual extensions ACU (AC with unit), AG (the Abelian groups) and ACUN (ACU with nilpotence). For lack of space, we only give the flavor of a **Solve** rule for these theories. Some more details can be found for AC in our previous paper [5].

In the first order case, the unification algorithm consists of counting the number of times an immediate subterm from another theory occurs in each side of an equation: both sides must have the same number of occurrences. We associate with each algebraic variable x an integer variable x_t representing the number of times the value of x contains the term t as an immediate subterm, and we translate each equation between two terms into a linear equation over the integers. These linear equations have to be solved over different integer domains depending on the considered theory. Then the solutions for the unification problem are built from the integer solutions, modulo some restrictions, in order to get some "well-formed" terms, and a complete set of unifiers. Thanks to Theorem 2, the same approach can also be used in the pattern case, as shown in [5] for the AC case. The main difference comes from the bound variables: if $\lambda\overline{x}.F(\overline{x})$ introduces a term $t(\overline{x})$, then $\lambda\overline{x}.F(\overline{x}^{\pi})$ introduces $t(\overline{x}^{\pi})$, and we do not know *a priori* whether $t(\overline{x})$ and $t(\overline{x}^{\pi})$ are equal or not. This is exemplified below:

4.1 An example of AC(+)-unification problem

Consider the equation $\mathcal{E} \equiv \lambda\overline{x_3}.2F(\overline{x_3}) + F(\overline{x_3}^{\pi}) + 9G(\overline{x_3}) = \lambda\overline{x_3}.2H(\overline{x_3})$ where $\pi = \{1 \mapsto 2; 2 \mapsto 3; 3 \mapsto 1\}$, to be solved modulo AC(+). If F introduces α times the term $t(\overline{x_3})$, then F^{π} introduces α times the term $t(\overline{x_3}^{\pi})$. If $t(\overline{x_3})$ and $t(\overline{x_3}^{\pi})$ are distinct, we have to count also the number of times F introduces $t(\overline{x_3}^{\pi})$, and finally, we have to count the number of $t(\overline{x_3}^{\pi^2})$. Then we can stop since π^3 is the identity. Let us denote by $\mathcal{G}(\pi)$ the group of permutations generated by π. The above unification problem is translated into 2 subsystems:

$$S_{\mathcal{G}(\pi)} = 2\alpha \ + \alpha \ + 9\beta \ = 2\gamma \qquad S_{\{id\}} = \begin{cases} 2\alpha'_{id} + \alpha'_{\pi^2} + 9\beta'_{id} = 2\gamma'_{id} \\ 2\alpha'_{\pi} + \alpha'_{id} + 9\beta'_{\pi} = 2\gamma'_{\pi} \\ 2\alpha'_{\pi^2} + \alpha'_{\pi} + 9\beta'_{\pi^2} = 2\gamma'_{\pi^2} \end{cases}$$

where F (resp. G, H) introduces α (resp. β, γ) times a term $t(\overline{x_3})$ such that $\lambda\overline{x_3}.t(\overline{x_3}) =_E \lambda\overline{x_3}.t(\overline{x_3}^{\pi})$, and α'_{φ} (resp. β'_{φ}, γ'_{φ}) times $s(\overline{x_3}^{\varphi})$, $\varphi = id, \pi, \pi^2$ where $\lambda\overline{x_3}.s(\overline{x_3}) \neq_E \lambda\overline{x_3}.s(\overline{x_3}^{\pi})$. These two systems are solved over *non-negative* integers, and as in the first order case, the unifiers are built from the Diophantine solutions, with the main difference that in the pattern case the introduced variables L_is corresponding to a solution of $S_{\mathcal{G}(\pi)}$ are constrained by $\mathrm{Perm}(L_i) = \mathcal{G}(\pi)$.

In the same spirit, this can be done in the extensions of AC, such as ACU, AG and ACUN, where the equations are solved by counting how many times a variable introduces a given term.

4.2 Handling the additional constraints for a Solve rule

Let us assume now that the problem to be solved is a part of a combination problem and have to be solved modulo the conditions of definition 10. Each variable F comes with some additional assumptions, such as the theory in which it can be instantiated, and its group of permutations $\mathrm{Perm}(F)$ as defined in section 3. These constraints are also translated into linear constraints over integers. Indeed the constraint $\mathrm{Perm}(H) = \mathcal{H}$ corresponds to the equations $\lambda\overline{x_3}.H(\overline{x_3}) = \lambda\overline{x_3}.H(\overline{x_3}^{\psi})$, where $\psi \in \mathcal{H}$, and these equations are translated into a system of linear equations over variables exactly in the same way as before, except that we have to consider all subgroups of the permutation group generated by π and $\mathcal{H}$ as possible invariants for an introduced term.

A constraint like "the variable G cannot be instantiated in the considered theory" means here that the number of terms from another theory introduced by its value is exactly one. Only one among all of the integer variables corresponding to G has to be equal to 1, the others being null.

A constraint like $H \not<_{oc} G$ will be treated in a second step, after one has built the solutions to the unification problem. If we get a solution where G occurs in the value of H, this is due to the fact that G is (equal to a new variable which is) associated with a solution which has some non-zero values for some integer variables corresponding to H. In the AC and ACU cases, such a integer solution has to be discarded, while in the AG and ACUN cases, this problem can be fixed by computing a particular solution such that these integer variables are null.

In all the 4 cases of AC, ACU, AG and ACUN, we are able to get a solved form which satisfies the additional hypotheses of the **Solve** rule.

5 Conclusion

We believe that with the emergence of higher-order rewriting and higher-order logic programming, there will be a use for pattern unification modulo equational theories. The algorithm that we proposed here is meant to provide a decidability result: it will not behave satisfactorily in practice, due to the heavy nondeterminism. It will be necessary to investigate how to reduce the nondeterminism as we did in the first-order case in [2, 3]. Another issue of interest will be to develop matching algorithms which should be dramatically more efficient in practice.

Although our method for elementary unification works well for the *AC*-like theories, we do not have a general method for ensuring the compatibility of a unifier with an equation of the form $\lambda xy.F(x, y) = \lambda xy.F(y, x)$. For instance, the known methods for unification in Boolean rings do not use equations over the integers, and such equations do not translate naturally as shown in the previous section. Actually, we conjecture that there exists a theory with decidable unification of problems with linear constant restriction (the equivalent in the first-order case of our **Solve** rule [1]) and undecidable pattern unification.

References

1. Franz Baader and Klaus Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *Journal of Symbolic Computation*, 21(2), February 1996.
2. Alexandre Boudet. *Unification dans les Mélanges de Théories équationnelles.* Thèse de doctorat, Université Paris-Sud, Orsay, France, February 1990.
3. Alexandre Boudet. Combining unification algorithms. *Journal of Symbolic Computation*, 16:597–626, 1993.
4. Alexandre Boudet. Unification of higher-order patterns modulo simple syntactic equational theories. *Discrete Mathematics and Theoretical Computer Science*, 4(1):11–30, 2000.

5. 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.
6. 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.
7. Val Breazu-Tannen. Combining algebra and higher-order types. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, July 1988.
8. François Fages. Associative-commutative unification. *Journal of Symbolic Computation*, 3(3), June 1987.
9. Warren D. Goldfarb. Note on the undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
10. R. Hindley and J. Seldin. *Introduction to Combinators and λ-calculus*. Cambridge University Press, 1986.
11. Jieh Hsiang and Michaël Rusinowitch. On word problems in equational theories. In Thomas Ottmann, editor, *14th International Colloquium on Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71, Karlsruhe, Germany, July 1987. Springer-Verlag.
12. Gérard Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots \omega$. Thèse d'Etat, Univ. Paris 7, 1976.
13. Jean-Pierre Jouannaud. Executable higher-order algebraic specifications. In C. Choffrut and M. Jantzen, editors, *Proc. 8th Symp. on Theoretical Aspects of Computer Science, Hamburg, LNCS 480*, pages 16–25, February 1991.
14. 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.
15. Jean-Pierre Jouannaud and Mitsuhiro Okada. Executable higher-order algebraic specification languages. In *Proc. 6th IEEE Symp. Logic in Computer Science, Amsterdam*, pages 350–361, 1991.
16. M. Livesey and Jörg H. Siekmann. Unification of bags and sets. Research report, Institut fur Informatik I, Universität Karlsruhe, West Germany, 1976.
17. Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, February 1998.
18. 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.
19. Zhenyu Qian and Kang Wang. Modular AC-Unification of Higher-Order Patterns. In Jean-Pierre Jouannaud, editor, *First International Conference on Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 105–120, München, Germany, September 1994. Springer-Verlag.
20. M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *Journal of Symbolic Computation*, 1990. Special issue on Unification.
21. Jörg H. Siekmann. Unification theory. *Journal of Symbolic Computation*, 7(3 & 4), 1989. Special issue on unification, part one.
22. M. E. Stickel. A complete unification algorithm for associative-commutative functions. In *Proceedings 4th International Joint Conference on Artificial Intelligence, Tbilissi (USSR)*, pages 71–76, 1975.