AC-Unification of Higher-order Patterns*

Alexandre Boudet and Evelyne Contejean

LRI, CNRS URA 410
Bt. 490, Universit Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France

Abstract. We present a complete algorithm for the unification of higher-order patterns modulo the associative-commutative theory of some constants $+_1, \ldots, +_n$. Given an AC-unification problem over higher-order patterns, the output of the algorithm is a finite set DAG solved forms [9], constrained by some flexible-flexible equations with the same head on both sides. Indeed, in the presence of AC constants, such equations are always solvable, but they have no minimal complete set of unifiers [13]. We prove that the algorithm terminates, is sound, and that any solution of the original unification problem is an instance of one of the computed solutions which satisfies the constraints.

Introduction

Higher-order unification is undecidable [8], yet the unification of higher-order patterns, a subset of the terms of the simply-typed λ-calculus is decidable, and useful in practice [11, 12]. The combination of algebraic and functional programming paradigms [10] leads one to investigate higher-order unification modulo equational theories. Unification of higher-order patterns will also be a key to higher-order extensions of membership equational logic described in [5].

The problem we address here is the unification of higher-order patterns modulo the associativity and commutativity of some constant symbols. This was partially achieved by Qian and Wang [13] who used a combination algorithm by Jouannaud and Kirchner [9] for first-order unification algorithms, and an AC1-unification algorithm for solving elementary AC problems. The most surprising result in their paper is that although unification of patterns is unitary in the standard case, in the AC case a single flexible-flexible equation with the same head has no minimal complete set of unifiers. Their algorithm is not complete because AC-unification of pure patterns (*i.e.* involving only one AC constant) is more involved than in the first-order case and one cannot just use an AC or AC1-unification algorithm as a black box in the case of patterns (see the example in Section 3.2). Our main contribution is to present a complete AC-unification algorithm for pure patterns. The paper is organized as follows. The next section briefly recalls some background on patterns and equational theories. Section 2 introduces a *variable abstraction* rule and recalls all but one of

* This research was supported in part by the EWG CCL, the HCM Network CONSOLE, and the "GDR de programmation du CNRS".

Nipkow's rules for pattern unification. Section 3 gives an algorithm for unification of pure AC-patterns, and section 4 shows how one gets an AC-unification algorithm by combining the previous steps using some standard techniques for combining first-order unification algorithms.

1 Preliminaries

We assume the reader is familiar with simply-typed lambda-calculus, and equational unification. Some background is available in *e.g.* [7, 9, 2] for lambda-calculus and unification in (combinations of) first-order equational theories. 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. In addition, we will use the notation $t(u_1, \ldots, u_n)$ or $t(\overline{u_n})$ for $(\cdots(tu_1)\cdots)u_n$. If π is a permutation of $(1, \ldots, n)$, $\overline{x_n}^{\pi}$ stands for the sequence $x_{\pi(1)}, \ldots, x_{\pi(n)}$. The free (resp. bound) variables of a term t are denoted by $\mathcal{FV}(t)$ (resp. $\mathcal{BV}(t)$). The notation $t[u]_p$ stands for a term t with a subterm u at some position p. Upper-case $X, F, G, L, L_1, \ldots$ denote free variables, lower-case $x, x_1, y, z, \ldots$ bound variables, and $a, b, f, g, \ldots$ constants.

1.1 Patterns

Definition 1. *A pattern is a term of the simply-typed λ-calculus in β-normal form in which the arguments of a free variable are η-equivalent to distinct bound variables.*

For instance, $\lambda xyz.f(H(x,y), H(x,z))$ 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 always assume that the terms are in η-long β-normal form [7, 12, 13], the β and η rules being respectively oriented as follows:
$(\lambda x.M)N \to_\beta M\{x \mapsto N\}$ (only the free occurrences of x are replaced by N),
$F \to_\eta \lambda\overline{x_n}.F(\overline{x_n})$ if the type of F is $\alpha_1 \to \ldots \to \alpha_n \to \alpha$, and α is a base type.
The η-long β-normal form of a term t is denoted by $t\downarrow^{\eta}_{\beta}$. Pattern unification is decidable, a result by Miller [11], refined by Nipkow [12]:

Theorem 1. *The unifiability of patterns is decidable and if two patterns are unifiable, there is an algorithm computing a unique most general unifier.*

We define now what we mean by "associative-commutative operators".

1.2 Equational theories, *E*-unification and AC-unification

Let $E = \{l_1 \simeq r_1, \ldots, l_n \simeq r_n\}$ a set of *axioms* such that l_i and r_i are terms of the same type, for $1 \leq i \leq n$. The *equational theory* $=_E$ generated by E is the

[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.

least congruence[2] containing all the instances of the axioms of E. The theory we consider here is the associative-commutative theory of one or more constant operators.

We distinguish some binary constant operators which are associative and commutative (which we denote by $+, +_1, \ldots$), and that we write in infix notation. The other constants are called *free*. Actually, we shall use a flattened representation, and when a term is written in the form $t_1 + \cdots + t_n$, it is implicitly assumed that the top symbol of the t_is is not $+$. For instance, the term $\lambda xyz.+(+(F(x,y),H(y,z)),F(x,y))$ will be written $\lambda xyz.F(x,y)+H(y,z)+F(x,y)$. In addition, we will sometimes write it $\lambda xyz.2F(x,y)+H(y,z)$, with the usual convention that nt stands for $\underbrace{t+\cdots+t}_{n \text{ times}}$, where n is a positive integer.

Definition 2. *The associative-commutative (AC) theory of $+$ is the equational theory presented by $AC(+) = \{(x+y)+z \simeq x+(y+z), x+y \simeq y+x\}$. The theory $AC(+_1,\ldots,+_n)$ is the theory presented by $AC(+_1) \cup \cdots \cup AC(+_n)$. In the sequel, we will refer to AC and $=_{AC}$ when $+_1,\ldots,+_n$ are not relevant.*

Definition 3 (Unification problems). *An* equation *is a pair $< s,t >$ of patterns of the same type, denoted by $s = t$. A unification problem is either $\top$ (the trivial problem), or $\bot$ (the unsolvable problem), or a (disjunction of) conjunction(s) of equations of the form $P \equiv s_1 = t_1 \wedge \cdots \wedge s_n = t_n$. A substitution σ is an E-*unifier *of P if $s_i\sigma$ and $t_i\sigma$ are equivalent modulo $\eta\beta$-equivalence and the theory $=_E$ for $1 \leq i \leq n$, which we write $s_i =_{\beta\eta E} t_i$.*

Since we consider only terms in η-long β-normal form, the following result from Tannen will allow us to restrict our attention to $=_E$ instead of $=_{\beta\eta E}$:

Theorem 2 ([6]). *For any terms u and v, $u =_{\beta\eta E} v$ if and only if $u \updownarrow^{\eta}_{\beta} =_E v \updownarrow^{\eta}_{\beta}$.*

In the case of AC-theories, we can assume without loss of generality that the AC-operators have types of the form $\alpha \to \alpha \to \alpha$ for some type α. Indeed, if it wasn't so, then $(a+b)+c$ and $a+(b+c)$ would not be of the same type. Further, we assume that α is a base type. Nevertheless, we can still define a "higher-order $+$". Consider two terms u and v of type $\sigma \to \alpha$ where α is a base type. Then define $u +_{\sigma\to\alpha} v$ as the term $\lambda x.u(x)+v(x)$ of type $\sigma \to \alpha$.
Under the above assumptions, there cannot be an abstraction below an AC-operator, since otherwise either the term would not be in η-long β-normal form, or the AC-operator would not apply to a base type.

2 Purification

In this section we start by giving a *Variable abstraction* rule which is obviously correct and terminating, and allows us to split the original problem into pure subproblems.

[2] *i.e.*, compatible *also* with application and abstraction, in our context.

Definition 4. *A unification problem P is* pure *in the free theory if it contains no AC symbol. P is* pure *in the AC-theory of $+$ if the only constant occurring in P is the associative-commutative constant $+$.*

Definition 5. *A subterm u of a term t is an* alien *subterm if it occurs immediately under an AC symbol and its head is not a free variable, or if it occurs immediately under a free constant and its head is an AC symbol.*

Applying repeatedly the following rule will yield a problem with no alien subterms.

VA
$\lambda\overline{x}.t[u]_p = \lambda\overline{x}s \quad \rightarrow \lambda\overline{x}.t[H(\overline{y})]_p = \lambda\overline{x}.s \ \wedge \ \lambda\overline{y}.H(\overline{y}) = \lambda\overline{y}.u$
if u is an alien subterm of $t[u]_p$ at position p, and $\overline{y} = \mathcal{FV}(u) \cap \overline{x}$, where H is a new variable.

Clearly, **VA** alone terminates, and we introduce some syntax for the unification problems obtained after applying it as long as possible.

Definition 6. *A unification problem in the theory $AC(+_1, \ldots, +_n)$ will be written in the form*

$$P \equiv P_F \cup P_0 \cup P_1 \cup \cdots \cup P_n$$

where

- *P_0 is pure in the free theory, containing no equations of the form $\lambda\overline{x}.F(x_1, \ldots, x_n) = \lambda\overline{x}.F(x_{\pi(1)}, \ldots, x_{\pi(n)})$ (where π is a permutation of $(1, \ldots, n)$),*
- *P_F contains all the flexible-flexible equations with the same heads described above.*
- *P_i is a pure unification problem in the AC theory of $+_i$, the arguments of $+_i$ being of the form $F(\overline{x})$ where F is a free variable.*

We will also distinguish some equations that will not be written in η-long β-normal form.

Definition 7. *An equation in a unification problem P is* quasi-solved *if it is of the form $F = s$ where F is a free variable and $F \notin \mathcal{FV}(s)$. A quasi-solved equation $F = s$ of a unification problem P is* solved *in P if F has no other free occurrence in P.*

We can now present some of the rules of Nipkow's algorithm [12] which are valid in the presence of $=_{AC}$. Only part of the work is done here since two special cases are not treated, namely the flexible-flexible equations with the same free variable on both sides and the equations where the top constant is associative-commutative. We also introduce some new rules which are specific to AC-unification.

Dec-free
$\lambda\overline{x}.a(s_1,\ldots,s_n) = \lambda\overline{x}.a(t_1,\ldots,t_n) \ \wedge \ P_\emptyset \quad \rightarrow$
$\lambda\overline{x}.s_1 = \lambda\overline{x}.t_1 \ \wedge \ \cdots \ \wedge \ \lambda\overline{x}.s_n = \lambda\overline{x}.t_n \ \wedge \ P_\emptyset$
if a is a free constant symbol or a bound variable of $\overline{x}$.

FR-free
$\lambda\overline{x}.F(\overline{y_n}) = \lambda\overline{x}.a(s_1,\ldots,s_m) \ \wedge \ P_\emptyset \quad \rightarrow$
$\lambda\overline{y_n}.H_1(\overline{y_n}) = \lambda\overline{y_n}.s_1 \ \wedge \ \cdots \ \wedge \ \lambda\overline{y_n}.H_m(\overline{y_n}) = \lambda\overline{y_n}.s_m$
$\wedge \ P_\emptyset\{F \mapsto a(H_1(\overline{y_n}),\ldots,H_m(\overline{y_n}))\}$
$\wedge \ F = \lambda\overline{x}.a(H_1(\overline{y_n}),\ldots,H_m(\overline{y_n}))\}$
If F is a free variable, a a free constant and $F \notin \mathcal{FV}(s_i)$ for $1 \leq i \leq m$, where $H_1,\ldots,H_m$ are new variables.

FF$\neq$
$\lambda\overline{x}.F(\overline{y_n}) = \lambda\overline{x}.G(\overline{z_m}) \ \wedge \ P_\emptyset \quad \rightarrow$
$P_\emptyset\{F \mapsto \lambda\overline{y_n}.H(\overline{v_p}), G \mapsto \lambda\overline{z_m}.H(\overline{v_p})\}$
$\wedge \ F = \lambda\overline{y_n}.H(\overline{v_p}) \ \wedge \ G = \lambda\overline{z_m}.H(\overline{v_p})$
if F and G are different free variables where $\overline{v_p} = \overline{y}_n \cap \overline{z_m}$.

Fail1
$\lambda\overline{x}.a(\overline{s}) = \lambda\overline{x}.b(\overline{t}) \quad \rightarrow$
$\bot$
if a and b are constants or bound variables and $a \neq b$.

Fail2
$\lambda\overline{x}.F(\overline{y}) = \lambda\overline{x}.a(\overline{s}) \quad \rightarrow$
$\bot$
if F is free in $\overline{s}$ or $a \in \overline{x} \setminus \overline{y}$.

Fig. 1. A subset of Nipkow's rules for the unification of higher-order patterns

The rules of figure 1 are a subset of those presented by Nipkow and as such, they terminate. These rules are to be applied to the pure subproblem P_0. Note, however that Nipkow explicitly states that his termination proof relies on the fact that he uses lists, and not multisets (conjunctions in our case). Hence, the conjunction operator must not be considered as associative commutative when applying these rules. It has to be noticed that the rules **FR-free** and **FR**$\neq$ make some solved equations appear. These equations will *not* be put in η-long β-normal form, and no rule of figure 1 will apply to them.

Note that the rule **Fail1** is still correct if a or b is an AC constant due to Tannen's theorem and to the fact that AC-theories are collapse-free.

Some cases are missing in figure 1, which is not surprising, since they require a special treatment in the presence of AC constants. A first case we omitted above is the flexible-flexible case with equal heads. It has been noticed by Qian and Wang that although such equations are always solvable, they do not have finite complete sets of AC-unifiers. That is why we will freeze them and just check for the compatibility of such equation with the rest of the problem. The second case that has not been considered yet is the pure AC subproblems.

Another rule is added to those of figure 1 which just 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.

Freeze
$P_F \wedge (s = t \wedge P_0) \wedge P_1 \wedge \cdots \wedge P_n \quad \rightarrow \; (s = t \wedge P_F) \wedge P_0 \wedge P_1 \wedge \cdots \wedge P_n$
if $s = t$ is a flexible-flexible equation where s and t have the same head variable.

Once the **VA** has been applied as long as possible to the original problem, and the rules of figure 1 plus the rule **Freeze** have been applied as long as possible to the subproblems P_0 and P_F, the problem has the form

$$P_F \;\wedge\; P_0 \;\wedge\; P_1 \;\wedge\; \cdots \;\wedge\; P_n$$

where

- P_F contains only flexible-flexible equations with the same free constant on both sides,
- P_0 is in a DAG solved form, from which a most general unifier is trivially obtained by applying a variable elimination rule.
- P_i is a pure problem in the AC-theory of $+_i$.

We extend the usual notion of DAG solved forms so as to take into account the frozen equations.

Definition 8. *A problem P is in a* DAG solved form *if it is of the form $P' \wedge P_F$ where*

– *P' is a conjunction $F_1 = t_1 \ \wedge \ \cdots \ \wedge \ F_n = t_n$ of quasi-solved equations, containing no cycle as in the premise of the* **Cycle** *rule of figure 2 (section 4), and where each F_i occurs exactly once as a left-hand side of an equation,*
– *P_F is a conjunction of frozen equations of the form $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$ such that if $F = \lambda\overline{x}.u(\overline{x})$ is in P', then $u(\overline{x}) =_{\eta\beta AC} u(\overline{x}^\pi)$.*

What we are left to do is to solve the pure AC subproblems (this is the object of the next section), to recombine the different solved subproblems $P_0, P_1, \ldots, P_n$, and to check the compatibility of the frozen flexible-flexible equations with the rest of the problem. This will be done in section 4.

3 Elementary AC-unification of higher-order patterns

In this section, we show how to handle the two cases that we omitted in the previous section.

3.1 Flexible-flexible equations with the same head variable

Actually, there is not much to do with such equations. Indeed, even though they are always solvable, they do not have finite complete sets of AC-unifiers. This was noticed by Qian and Wang [13] who give the following example:

Example 1 ([13]). 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

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

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.

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 [14].

As Qian and Wang, we keep these equations as constraints, and we will see in section 4, how to check the compatibility of such constraints with the rest of the solutions.

3.2 Elementary AC-unification: an example

Before giving the algorithm for elementary AC-unification, let us develop an example which will help to follow the remainder of this section. Consider the equation

$$\lambda xyz.2F(x,y,z) + F(y,z,x) = \lambda xyz.2G(x,y,z)$$

A solution σ may introduce a term $t(x,y,z)$ which does not depend on the order of its arguments (*i.e.* $t(x,y,z) = t(y,z,x)$). Such a term is introduced 2α times by $2F(x,y,z)\sigma$ and α times by $F(y,z,x)\sigma$. On the other hand $t(x,y,z)$ must be introduced 2β times by $2G(x,y,z)\sigma$ where (α,β) is a positive solution of the

linear Diophantine equation $3n = 2m$. The set of minimal positive solutions of this equation is $\{(2,3)\}$. Let L_1 be a new variable and assume that θ is such that $L_1(x,y,z)\theta = L_1(y,z,x)\theta = L_1(z,x,y)\theta$. Then

$$\{F \mapsto \lambda(x,y,z).2L_1(x,y,z)\theta, G \mapsto \lambda(x,y,z).3L_1(x,y,z)\theta\}$$

is a solution of the original equation.
On the other hand, σ may introduce a term $t(x,y,z)$ which depends on the order of its variables (*i.e.* $t(x,y,z) \neq t(y,z,x)$)[3]. Assume that $F(x,y,z)\sigma$ introduces α_1 times $t(x,y,z)$, α_2 times $t(y,z,x)$ and α_3 times $t(z,x,y)$. In this case, $G(x,y,z)$ introduces respectively $\beta_1, \beta_2, \beta_3$ times the terms $t(x,y,z)$, $t(y,z,x)$ and $t(z,x,y)$ where $(\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3)$ are positive solutions of the system of linear Diophantine equations

$$\begin{aligned} 2n_1 + n_3 &= 2m_1 \\ 2n_2 + n_1 &= 2m_2 \\ 2n_3 + n_2 &= 2m_3 \end{aligned}$$

Each of the above equations ensures that the respective numbers of occurrences of $t(x,y,z)$, $t(y,z,x)$ and $t(z,x,y)$ introduced by σ in both hands of the equation are the same. The set of minimal positive solutions of this system is {(2,0,0,2,1,0),(0,2,0,0,2,1),(0,0,2,1,0,2)} Let us associate the new variables L_2, L_3, L_4 with each of these solutions. A solution of the original problem is

$$\begin{aligned} \{\ F \mapsto \lambda xyz.\ & 2L_1(x,y,z)\theta + 2L_2(x,y,z) + 2L_3(y,z,x) + 2L_4(z,x,y), \\ G \mapsto \lambda xyz.\ & 3L_1(x,y,z)\theta + 2L_2(x,y,z) + L_2(y,z,x) + 2L_3(y,z,x) \\ & + L_3(z,x,y) + L_4(x,y,z) + 2L_4(z,x,y)\} \end{aligned}$$

provided that $L_1(x,y,z)\theta = L_1(y,z,x)\theta = L_1(z,x,y)\theta$. As in the first-order AC-unification, some of the new variables may be omitted as long as one has a well-formed substitution (*i.e.* not mapping a variable onto an "empty" term).

3.3 Elementary AC-unification

We show now how to solve a pure equation modulo AC in the general case. The technique naturally extends to the solving of pure problems as in the first-order case, but the notations are already quite involved, and we prefer to restrict our presentation to the case of a single equation. We can assume without loss of generality that the problem has the form:

$$P \equiv \lambda\overline{x}.\sum_{i=1}^{n_1}\sum_{\pi\in\Pi} a_{i,\pi}F_i(\overline{x}^\pi) = \lambda\overline{x}.\sum_{i=n_1+1}^{n_2}\sum_{\pi\in\Pi} a_{i,\pi}F_i(\overline{x}^\pi)$$

[3] This case was not considered in [13], since AC-unification was a black box: this is why the algorithm is not complete

where Π is a subgroup of the group of permutations over all the variables of $\overline{x}$. Π is actually the subgroup generated by the permutations occuring in the problem[4]. Note that some of the $a_{i,\pi}$s may be equal to zero.

As in the example, a solution of the problem may introduce a term $t(\overline{x})$, which is invariant under some permutations of its arguments. These permutations are a subgroup Π_0 of Π. Let us denote by $P^{\Pi_0}_{Dioph}$ the following linear Diophantine system in $n_2 \times \mathrm{Card}(\Pi)$ natural unknowns $y_{i,\pi}$:

$$P^{\Pi_0}_{Dioph} \equiv \begin{array}{c} \bigwedge_{i=1}^{n_2} \bigwedge_{(\pi_1,\pi_2)|\Pi_0\circ\pi_1=\Pi_0\circ\pi_2} y_{i,\pi_1} = y_{i,\pi_2} \\ \wedge \\ \bigwedge_{\pi'\in\Pi} \sum_{i=1}^{n_1}\sum_{\pi\in\Pi} a_{i,\pi^{-1}\circ\pi'} y_{i,\pi} = \sum_{i=n_1+1}^{n_2}\sum_{\pi\in\Pi} a_{i,\pi^{-1}\circ\pi'} y_{i,\pi} \end{array}$$

and by P_{Dioph} the disjunction $\bigvee_{\Pi_0 \text{ subgroup of } \Pi} P^{\Pi_0}_{Dioph}$. $y_{i,\pi}$ is the number of occurrences of a given term $t(\overline{x})$ in $F_i(\overline{x}^\pi)$. The first part of $P^{\Pi_0}_{Dioph}$ states that t is invariant under the permutations of Π_0, and the second part states that the number of $t(\overline{x}^{\pi'})$s is the same on both sides of the equation.

Definition 9. *Let $\mathcal{P} = \bigcup_{\Pi_0} \mathcal{P}_{\Pi_0}$ be a subset of minimal solutions of the linear Diophantine system P_{Dioph}, where each $\mathcal{P}_{\Pi_0}$ is a subset of minimal solutions of $P^{\Pi_0}_{Dioph}$. $\mathcal{P}$ is said to be* great enough *if*

$$\forall i \in \{1,\ldots,n_2\} \sum_{\Pi_0}\sum_{m\in\mathcal{P}_{\Pi_0}}\sum_{\pi\in\Pi/\Pi_0} m(i,\pi) > 0$$

Proposition 1. *Let $\mathcal{P}$ be any subset of the minimal solutions of P_{Dioph}, which is great enough. Then $\sigma_{\mathcal{P}}$ is a solution of P:*

$$\sigma_{\mathcal{P}} = \{F_i \mapsto \lambda\overline{x}. \sum_{\Pi_0}\sum_{m\in\mathcal{P}_{\Pi_0}}\sum_{\pi'\in\Pi/\Pi_0} m(i,\pi') L_m(\overline{x}^{\pi'})\}$$

where $L_m, m \in \mathcal{P}_{\Pi_0}$ is a new variable constrained by $\forall\pi \in \Pi_0 \;\; L_m(\overline{x}^\pi) = L_m(\overline{x})$.

Proof. Since $\mathcal{P}$ is great enough, $\sum_{\Pi_0}\sum_{m\in\mathcal{P}_{\Pi_0}}\sum_{\pi'\in\Pi/\Pi_0} m(i,\pi')L_m(\overline{x}^{\pi'})$ is a non-empty sum for all i, hence $F_i\sigma_{\mathcal{P}}$ is well-defined. Moreover, we have:

$\sum_{i=1}^{n_1}\sum_{\pi\in\Pi} a_{i,\pi}(F_i\sigma_{\mathcal{P}})(\overline{x}^\pi) =$
$\sum_{i=1}^{n_1}\sum_{\pi\in\Pi} a_{i,\pi}\sum_{\Pi_0}\sum_{m\in\mathcal{P}_{\Pi_0}}\sum_{\pi'\in\Pi/\Pi_0} m(i,\pi')L_m(\overline{x}^{\pi'\circ\pi}) =$
$\sum_{\Pi_0}\sum_{m\in\mathcal{P}_{\Pi_0}}\sum_{\pi\in\Pi}\sum_{i=1}^{n_1}\sum_{\pi''\in\Pi/\Pi_0} a_{i,\pi} m(i,\pi^{-1}\circ\pi'')L_m(\overline{x}^{\pi''}) =$
$\sum_{\Pi_0}\sum_{m\in\mathcal{P}_{\Pi_0}}\sum_{\pi''\in\Pi/\Pi_0}(\sum_{\pi\in\Pi}\sum_{i=1}^{n_1} a_{i,\pi} m(i,\pi^{-1}\circ\pi''))L_m(\overline{x}^{\pi''}) =$
$\sum_{\Pi_0}\sum_{m\in\mathcal{P}_{\Pi_0}}\sum_{\pi''\in\Pi/\Pi_0}(\sum_{\pi'''\in\Pi}\sum_{i=1}^{n_1} a_{i,\pi''\circ\pi'''^{-1}} m(i,\pi'''))L_m(\overline{x}^{\pi''}) =$
since m is a solution of $P^{\Pi_0}_{Dioph}$
$\sum_{\Pi_0}\sum_{m\in\mathcal{P}_{\Pi_0}}\sum_{\pi''\in\Pi/\Pi_0}(\sum_{\pi'''\in\Pi}\sum_{i=n_1+1}^{n_2} a_{i,\pi''\circ\pi'''^{-1}} m(i,\pi'''))L_m(\overline{x}^{\pi''}) =$
$\sum_{i=n_1+1}^{n_2}\sum_{\pi\in\Pi} a_{i,\pi}(F_i\sigma_{\mathcal{P}})(\overline{x}^\pi)$

[4] In the example, Π is equal to $\{\pi_0,\pi_1,\pi_2\}$, where $\pi_0 = \{x\mapsto x, y\mapsto y, z\mapsto z\}$, $\pi_1 = \{x\mapsto y, y\mapsto z, z\mapsto x\}$, $\pi_2 = \{x\mapsto z, y\mapsto x, z\mapsto y\}$

Proposition 2. *$\{\sigma_{\mathcal{P}} \mid \mathcal{P}$ is great enough$\}$ is a complete set of solutions for P.*

Proof. Let σ be a solution of P. Let $\{t_j^\pi\}_{j\in J,\pi\in\Pi}$ be a set of terms which contains all the immediate alien subterms of the $F_i\sigma$, and such that

$$\forall j,k \;\; (\exists \pi,\pi' \;\; t_j^\pi = t_k^{\pi'}) \Leftrightarrow (j=k)$$

$F_i\sigma$ may be written as $\lambda\overline{x}.\sum_{j\in J}\sum_{\pi'\in\Pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'})$ σ is a solution, hence by the theorem of Tannen [6]:

$$\sum_{i=1}^{n_1}\sum_{\pi\in\Pi} a_{i,\pi}(F_i\sigma)(\overline{x}^\pi) =_{AC} \sum_{i=n_1+1}^{n_2}\sum_{\pi\in\Pi} a_{i,\pi}(F_i\sigma)(\overline{x}^\pi)$$

$$\sum_{i=1}^{n_1}\sum_{\pi\in\Pi} a_{i,\pi}(\sum_{j\in J}\sum_{\pi'\in\Pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'\circ\pi})) =_{AC} \sum_{i=n_1+1}^{n_2}\sum_{\pi\in\Pi} a_{i,\pi}(\sum_{j\in J}\sum_{\pi'\in\Pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'\circ\pi}))$$

$$\sum_{j\in J}\sum_{i=1}^{n_1}\sum_{\pi\in\Pi}\sum_{\pi'\in\Pi} a_{i,\pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'\circ\pi}) =_{AC} \sum_{j\in J}\sum_{i=n_1+1}^{n_2}\sum_{\pi\in\Pi}\sum_{\pi'\in\Pi} a_{i,\pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'\circ\pi})$$

Hence, by hypothesis, for all $j\in J$, we have

$$\sum_{i=1}^{n_1}\sum_{\pi\in\Pi}\sum_{\pi'\in\Pi} a_{i,\pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'\circ\pi}) =_{AC} \sum_{i=n_1+1}^{n_2}\sum_{\pi\in\Pi}\sum_{\pi'\in\Pi} a_{i,\pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'\circ\pi})$$

Let Π_j be the subgroup of invariant permutations of t_j *i. e.* $\Pi_j = \{\pi\in\Pi \mid \lambda\overline{x}.t_j(\overline{x}) = \lambda\overline{x}.t_j(\overline{x}^\pi)\}$. Hence we have for all $\pi''\in\Pi$:

$$\sum_{i=1}^{n_1}\sum_{\pi,\pi'\mid\pi'\circ\pi\circ\pi''^{-1}\in\Pi_j} a_{i,\pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi''}) =_{AC} \sum_{i=n_1+1}^{n_2}\sum_{\pi,\pi'\mid\pi'\circ\pi\circ\pi''^{-1}\in\Pi_j} a_{i,\pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi''})$$

$$\sum_{i=1}^{n_1}\sum_{\pi,\pi'\mid\pi'\circ\pi\circ\pi''^{-1}\in\Pi_j} a_{i,\pi}\alpha_{i,j,\pi'} =_{\mathbb{N}} \sum_{i=n_1+1}^{n_2}\sum_{\pi,\pi'\mid\pi'\circ\pi\circ\pi''^{-1}\in\Pi_j} a_{i,\pi}\alpha_{i,j,\pi'}$$

$$\sum_{i=1}^{n_1}\sum_{\pi} a_{i,\pi}(\sum_{\pi'\mid\pi'\circ\pi\circ\pi''^{-1}\in\Pi_j}\alpha_{i,j,\pi'}) =_{\mathbb{N}} \sum_{i=n_1+1}^{n_2}\sum_{\pi} a_{i,\pi}(\sum_{\pi'\mid\pi'\circ\pi\circ\pi''^{-1}\in\Pi_j}\alpha_{i,j,\pi'})$$

$$\sum_{i=1}^{n_1}\sum_{\pi} a_{i,\pi}(\sum_{\pi'\mid\pi'\in\Pi_j\circ\pi''\circ\pi^{-1}}\alpha_{i,j,\pi'}) =_{\mathbb{N}} \sum_{i=n_1+1}^{n_2}\sum_{\pi} a_{i,\pi}(\sum_{\pi'\mid\pi'\in\Pi_j\circ\pi''\circ\pi^{-1}}\alpha_{i,j,\pi'})$$

$$\sum_{i=1}^{n_1}\sum_{\pi} a_{i,\pi'''^{-1}\circ\pi''}(\underbrace{\sum_{\pi'\mid\pi'\in\Pi_j\circ\pi'''}\alpha_{i,j,\pi'}}_{\beta_{i,j,\pi'''}}) =_{\mathbb{N}} \sum_{i=n_1+1}^{n_2}\sum_{\pi} a_{i,\pi'''^{-1}\circ\pi''}(\underbrace{\sum_{\pi'\mid\pi'\in\Pi_j\circ\pi'''}\alpha_{i,j,\pi'}}_{\beta_{i,j,\pi'''}})$$

For all j, $(\beta_{i,j,\pi})_{i,\pi}$ satisfies moreover $\Pi_j\circ\pi_1 = \Pi_j\circ\pi_2 \;\; \Rightarrow \beta_{i,j,\pi_1} = \beta_{i,j,\pi_2}$. Hence $\beta_j = (\beta_{i,j,\pi})_{i,\pi}$ is a solution of $P_{Dioph}^{\Pi_j}$, β_j is a linear combination of some minimal solutions of $P_{Dioph}^{\Pi_j}$.

Let $\mathcal{P} = \bigcup_{\Pi_0} \mathcal{P}_{\Pi_0}$ be the subset of minimal solutions of P_{Dioph} used at least by one β_j, hence one can write $\forall j \;\; \beta_j = \sum_{m \in \mathcal{P}_{\Pi_j}} c_{j,m} m$, where the $c_{j,m}$ are non-negative. We extend the definition of the $c_{j,m}$s for the ms which are minimal solutions of P_{Dioph} but not of $P_{Dioph}^{\Pi_j}$ by 0. Note that σ can be written as:

$$\begin{aligned}
F_i\sigma &= \lambda\overline{x}.\textstyle\sum_{j\in J}\sum_{\pi'\in\Pi}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'}) \\
&= \lambda\overline{x}.\textstyle\sum_{j\in J}\sum_{\overline{\pi'''}\in\Pi/\Pi_j}\sum_{\pi'\in\Pi_j\circ\pi'''}\alpha_{i,j,\pi'}t_j(\overline{x}^{\pi'}) \\
&= \lambda\overline{x}.\textstyle\sum_{j\in J}\sum_{\overline{\pi'''}\in\Pi/\Pi_j}(\sum_{\pi'\in\Pi_j\circ\pi'''}\alpha_{i,j,\pi'})t_j(\overline{x}^{\pi'''}) \\
&= \lambda\overline{x}.\textstyle\sum_{j\in J}\sum_{\overline{\pi'''}\in\Pi/\Pi_j}\beta_{i,j,\pi'''}t_j(\overline{x}^{\pi'''}) \\
&= \lambda\overline{x}.\textstyle\sum_{j\in J}\sum_{\overline{\pi'''}\in\Pi/\Pi_j}(\sum_{m\in\mathcal{P}}c_{j,m}m(i,\pi'''))t_j(\overline{x}^{\pi'''}) \\
&= \lambda\overline{x}.\textstyle\sum_{m\in\mathcal{P}}\sum_{j\in J}\sum_{\overline{\pi'''}\in\Pi/\Pi_j}c_{j,m}m(i,\pi''')t_j(\overline{x}^{\pi'''})
\end{aligned}$$

Let us define θ as $\forall m \in \mathcal{P} \;\; L_m \mapsto \lambda\overline{x}.\sum_{j\in J} c_{j,m}t_j(\overline{x})$. θ is a valid substitution since if $c_{j,m} \neq 0$, the subgroup of invariant permutations of t_j is Π_j, hence $L_m\theta$ satisfies the constraint. It is easy to verify that σ is equal to $\sigma_{\mathcal{P}}\theta$.

We have shown how to solve a pure equation modulo AC, but actually, we solve systems of such equations P_{+_i}, together with the frozen equations in P_F which involve a variable occurring in P_{+_i}. Such frozen equations may be considered as pure equations, without any AC symbols. Obviously Proposition 1 and Proposition 2 are still valid.

4 Recombination

We first present a combination algorithm for combining the solutions of the different solved subproblems. It is closely inspired by our algorithm for (first-order) AC-unification. [1, 4, 3]. There, the termination proof is based on the notion of *shared variables* which we adapt here. With this modification of the notion of shared variables, our proof can be reused exactly as such since it basically relies upon the fact that solving a pure subproblem will not increase the number of shared variables.

4.1 Combination

In the previous section, we have seen how to turn $P_F \wedge P_i$ into an equivalent problem $P'_F \wedge P'_i$ where P'_i is now solved and P'_F may contain additional frozen flexible-flexible equations. Now, it may happen that solving a subproblem P_i yields some solved equations of the form $F = \lambda\overline{x}.G(\overline{x})$. These equations are by definition solved in P_i, but the free variables F and G may occur in some other subproblem P_j. Replacing F by $\lambda\overline{x}.G(\overline{x})$ in the other subproblems may make some of them unsolved.

Definition 10. *Two distinct non-solved free variables are* shared *in P_i if they both occur in P_i and are identifiable outside P_i. Two non-solved free variables F_1 and F_k are* identifiable outside P_i *if there exists a sequence $(F_1, F_2), (F_2, F_3), \ldots, (F_{n-1}, F_k)$ such that for $1 \leq i < k$, both F_i and F_{i+1} have a free occurrence in some problem P_j with $i \neq j$.*

The rules of figure 2 mimic those mentioned above for first-order AC-unification. They have to be applied as follows. If P_0 is not solved, then it will be solved by using the rules of figure 1. The pure AC subproblems will be solved using the algorithm for elementary AC unification of patterns of section 3. In both cases, the flexible-flexible equations with the same free variable on both sides are frozen in P_F. Let us stress that the frozen equations in P_F are never used, nor altered at this stage. In particular, the substitutions are *not* applied to P_F.

Solve
$P_i \quad \rightarrow \; P_i'$
if P_i is not solved, and P_i' is a solved form of P_i.

Variable-Replacement
$F = \lambda\overline{x}.G(\overline{y}) \;\wedge\; P \quad \rightarrow \; F = \lambda\overline{x}.G(\overline{y}) \;\wedge\; P\{F \mapsto \lambda\overline{x}.G(\overline{y})\}$
if both F and G have a free occurrence in P.

Clash
$F = s \;\wedge\; F = t \quad \rightarrow \; \bot$
if s and t have different constant heads.
Cycle
$F_1 = t_1[F_2] \;\wedge\; F_2 = t_2[F_3] \cdots \wedge\; F_n = t_n[F_1] \quad \rightarrow \; \bot$
if there is a constant on the path between the head and $F_{i+1(mod\ n)}$ in some t_i.

Fig. 2. The combination rules

The following lemmas are the same as in the first-order case, and we omit the proofs which translate naturally in our context.

Lemma 1. Solve *does not increase the number of shared variables in any of the subproblems $P_0, \ldots, P_n$.*

Lemma 2. *If* **Variable-Replacement** *makes a solved subproblem P_i unsolved, then it decreases the multiset of the numbers of shared variables in $P_0, \ldots, P_n$.*

The termination proof is then provided by the following measure which compares lexicographically

1. The multiset of the numbers of shared variables in $P_0, \ldots, P_n$,
2. the number of unsolved subproblems in $\{P_0, \ldots, P_n\}$.

Proposition 3. *The rules of figure 2 terminate and yield either the unsolvable problem $\bot$, or a problem*

$$P \equiv P_F \ \wedge \ P_0 \ \wedge \ \cdots \ \wedge \ P_n$$

where

- *P_F contains only flexible-flexible equations with the same head variable on both sides,*
- *P_i is solved for $0 \leq i \leq n$,*
- *each free variable F occurs at most once in $P_0 \ \wedge \ \cdots \ \wedge \ P_n$ as a left-hand side of an equation,*
- *there is no cycle as in the premise of the rule* **Cycle**.

What the above proposition says is that the problem obtained, if one forgets the frozen part P_F, is a DAG-solved form (see [9]), that is a problem from which a unifier is trivially obtained by applying as long as possible a variable elimination rule. We make a technical (yet cost-less) assumption on the rule **Solve** that will ease the termination proof of the next section. If solving a pure subproblem yields a solved equation $F = \lambda\overline{x}.L(\overline{x})$ and a frozen equation $\lambda\overline{x}.L(\overline{x}) = \lambda\overline{x}.L(\overline{x}^\pi)$, where L is a new variable, then L is replaced by F in the rest of the problem, and the frozen equation is replaced by $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$. In other words, we want to avoid to just rename a variable by a variable of a frozen equation.

4.2 Frozen variables

The problem that we have omitted so far is that of the frozen flexible-flexible equations. As we have seen in section 3, such equations have no minimal complete set of unifiers, hence the solution (suggested by Qian and Wang) to keep them as *constraints*. In practice, such equations will never be explicitly solved, but one still needs to test their compatibility with the rest of the problem.

This is achieved by applying the rules of figure 3. By abuse of notations, we write $t(\overline{x})$ and $t(\overline{x}^\pi)$, even if all the variables of $\overline{x}$ do not appear in t so as to be able to apply the permutations. The only rule that may seem to cause non-termination is **Merge**, since **Compatibility** and **Incompatibility** can obviously cause no trouble, and **Propagate** will eventually lead to applying **Compatibility**, **Incompatibility** or **Merge**. But **Merge** may make some previously solved problems unsolved. One has then to apply again as long as possible the rules of figure 2.

Lemma 3. Merge *cannot be applied infinitely many times.*

Proof (Sketched). If solving P_i makes it possible to apply **Merge** to some other problem P_j, then the solving of P_i must have created a new frozen equation $\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^\pi)$, where F appeared free in both P_i and P_j. We call such

Compatibility
$\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^{\pi}) \ \wedge \ F = \lambda\overline{x}.u(\overline{x}) \quad \rightarrow \ F = \lambda\overline{x}.u(\overline{x})$
if $\lambda\overline{x}.u(\overline{x}) =_{\eta\beta AC} \lambda\overline{x}.u(\overline{x}^{\pi})$.

Incompatibility
$\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^{\pi}) \ \wedge \ F = \lambda\overline{x}.u(\overline{x}) \quad \rightarrow \ \bot$
if $\lambda\overline{x}.u(\overline{x})$ is ground and $\lambda\overline{x}.u(\overline{x}) \neq_{\eta\beta AC} \lambda\overline{x}.u(\overline{x}^{\pi})$.

Propagate
$\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^{\pi}) \ \wedge \ F = \lambda\overline{x}.\gamma(t_1(\overline{x}), \ldots, t_n(\overline{x})) \quad \rightarrow$
$\lambda\overline{x}.t_1(\overline{x}) = \lambda\overline{x}.t_1(\overline{x}^{\pi}) \ \wedge \ \cdots \ \wedge \ \lambda\overline{x}.t_n(\overline{x}) = \lambda\overline{x}.t_n(\overline{x}^{\pi})$
if γ is not an AC constant.

Merge
$\lambda\overline{x}.F(\overline{x}) = \lambda\overline{x}.F(\overline{x}^{\pi}) \ \wedge \ F = \lambda(\overline{x}).t_1(\overline{x}) + \cdots + t_n(\overline{x}) \quad \rightarrow$
$\lambda(\overline{x}).t_1(\overline{x}) + \cdots + t_n(\overline{x}) = \lambda(\overline{x}).t_1(\overline{x}^{\pi}) + \cdots + t_n(\overline{x}^{\pi})$
If $+$ is an AC constant.

Fig. 3. The rules for testing the compatibility of the frozen part with the solved subproblems.

a variable a *weakly shared variable*. Note also that this occurs only when F has no value with head $+_i$. After applying **Merge**, P_j is to be solved again but this time the value of F will be a term with head $+_j$. Hence, no new frozen equation with variable F will be created by solving P_j. The key of the proof relies on the following lemma.

Lemma 4. *No rule of figures 2 and 3 creates new weakly shared variables.*

Theorem 3. *The following algorithm terminates and computes a DAG solved form for for unification of higher-order patterns modulo AC.*

1. *Apply as long as possible* **VA**,
2. *as long as possible do*
   (a) *apply as long as possible the rules of figure 2*
   (b) *apply the rules of figure 3 until a DAG solved form is obtained, or some pure subproblem is made unsolved by* **Merge**.

5 Conclusion

We have presented a unification algorithm for higher-order patterns modulo AC. This will have applications in functional programming, algebraic-functional programming and hopefully for testing the local confluence of higher-order rewrite systems in the presence of associative-commutative constants. Our result will

not extend as such to E-unification of higher-order patterns for an arbitrary theory E, since the algorithm heavily relies on the properties of AC. Yet, it will be interesting to apply similar methods to other well-known theories of interest. We have in mind the usual extensions of AC (like AC1, ACI,...), but also richer theories like Abelian groups or Boolean rings. This will require not only to design an elementary unification algorithm for these theories, but also to adapt the combination method to non-regular or collapsing equational theories.

References

1. Alexandre Boudet. *Unification dans les Mélanges de Théories équationnelles.* Thèse de doctorat, Université Paris-Sud, Orsay, France, February 1990.
2. Alexandre Boudet. Combining unification algorithms. *Journal of Symbolic Computation*, 16:597–626, 1993.
3. Alexandre Boudet. Competing for the AC-unification race. *Journal of Automated Reasoning*, 11:185–212, 1993.
4. Alexandre Boudet, Evelyne Contejean, and Hervé Devie. A new AC-unification algorithm with a new algorithm for solving diophantine equations. In *Proc. 5th IEEE Symp. Logic in Computer Science, Philadelphia*, pages 289–299. IEEE Computer Society Press, June 1990.
5. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. In Michel Bidoit and Max Dauchet, editors, *Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, Lille, France, April 1997. Springer-Verlag.
6. Val Breazu-Tannen. Combining algebra and higher-order types. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, July 1988.
7. R. Hindley and J. Seldin. *Introduction to Combinators and λ-calculus.* Cambridge University Press, 1986.
8. Gérard Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots \omega$. Thèse d'Etat, Univ. Paris 7, 1976.
9. 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.
10. 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.
11. 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.
12. T. Nipkow. Higher order critical pairs. In *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.
13. 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.
14. Jörg H. Siekmann. Unification theory. *Journal of Symbolic Computation*, 7(3 & 4), 1989. Special issue on unification, part one.