Order-sorted equational unification

Claude Kirchner

INRIA

UNITÉ DE RECHERCHE
INRIA-LORRAINE

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N°954

Programme 1

ORDER-SORTED EQUATIONAL UNIFICATION

Claude KIRCHNER

Décembre 1988

RR-0954

Order-Sorted Equational Unification*

Unification Equationnelle dans les Algèbres à types Ordonnés

Claude Kirchner†

LORIA & INRIA (Lorraine)
BP 239
54506 Vandœuvre-Les-Nancy Cedex
France
E-mail: ckirchner@crin.crin.fr

Novembre 1988

Programme 1

Abstract

Order-sorted equational unification is studied from an algebraic point of view. We show how order-sorted equational unification algorithms can be built when the order-sorted signature is regular (i.e. every term has a unique least sort) and the equational specification is sort-preserving (i.e. any A-equal terms have the same least sort). Under these conditions the transformations rules allowing to build unification algorithms in the unsorted framework can be extended to the order-sorted one. This allows us to generalize the known results to order-sorted equational unification, in particular when there exist overloaded symbols with different properties. An important application is order-sorted associative-commutative unification for which no direct algorithm was given until now.

Résumé

Nous étudions dans ce rapport l'unification équationelle dans les algèbres à types ordonnés. Nous montrons comment construire des algorithmes d'unification lorsque la signature de l'algèbre est régulière (i.e. quand chaque terme a un unique plus petit type) et lorsque la théorie équationelle conserve les types (i.e. tous termes A-égaux ont le même plus petit type). Sous ces conditions, les règles de transformations permettant de faire l'unification dans le cadre non typé sont étendues au cadre à types ordonnés. Cela nous permet de généraliser les résultats connus, en particulier lorsqu'il existe des opérateurs surchargés ayant des propriétés différentes. Une conséquence importante est la construction directe d'un algorithme d'unification associative-commutative dans le contexte des algèbres à types ordonnés.

*Presented at the fifth international conference on logic programming. Seattle 15-20 August 1988.
†Partly supported by the GRECO de programmation (CNRS).

1 Introduction

Following the growing interest for order-sorted algebras [7,25] either for application to the study of the mathematical and operational semantics of high level programming languages [6,3,24] or for theorem proving concern [4,27], the unification problem in order-sorted algebras has been investigated since 1983 [2]. One year later, C. Walther [29] studied the unification problem in the trivial specification, that is without axioms, giving results about the cardinality of complete set of unifiers when there is no overloaded operator. This work is extended in [26,28] to obtain a first classification of order-sorted unification problems without multiple function declarations. Concurrently, M.Schmidt-Schauss has addressed the problem of solving equations in a generalized order-sorted framework allowing axioms and sort declarations of terms [21,19]. His main results are that in an order-sorted specification with term declarations, syntactic unification is undecidable, and for well-suited order-sorted specifications, equational unification can be solved by first solving the problem without any use of the sort information and then by restricting the solutions that have been found to the right sorts. Using a model theoretic approach (with a different notion of model than the authors above), categorical tools and the previous idea, J. Meseguer, J. Goguen and G. Smolka [17] gave characterization theorems for the existence of complete set of A-unifiers that are singleton, finite or minimal. Moreover, they gave a sufficient decidable condition for when order-sorted unification can be related to unsorted unification.

The common characteristic of all the previous works on equational unification is that they are done by first solving the unsorted instance of the order-sorted equational unification problem. Our approach is quite different, since we give tools for solving directly the problem. The algebraic point of view presented here allows using the sort information as soon as possible during the unification process in order to discard failure quickly and to direct more efficiently the unification process. Moreover, it allows to reuse mutation operations of unsorted unification algorithms directly in the order-sorted framework, thus without loosing any sort information. A similar point of view can be applied to order-sorted matching: this is not developed here and more information is given in [14].

Any order-sorted unification algorithm can be described as the result of a simplification step followed by a solving step, as in the unsorted case [13,10]. The *simplification step* is a succession of decomposition, merging and mutation steps, transforming the initial unification problem into an equivalent disjunction of systems of fully decomposed equations of the form $x == t$ where x appears once in a variable part of an equation in a system. The *solving step* consists in finding a generating (or complete) set of the *finite* solutions of the simplified system.

The main differences between order-sorted equational unification and unsorted equational unification are mainly the following:

1. Solving simple equations, i.e. of the form $x == t$, where the variable x does not occur in the term t, is no more trivial and requires the knowledge of the whole signature. Furthermore, a minimal complete set of unifiers of such an equation may not be finite.

2. The simplification process (to be specific the mutation) is itself more complex than in the unsorted case if the theory is not sort-preserving or when there are overloaded operators with different properties. A typical example of such an overloading is the multiplication defined on quaternions, complex, reals and naturals, that is associative and commutative on all these structures, except on quaternions where it is only associative.

In this paper, our main contributions to the field, developed in the third section, are:

1. to extended the algebraic tools for solving equational unification problems in unsorted theories to the order-sorted framework, thus allowing to use the full power of strong typing at unification time,

2. to give equational unification algorithms in sort-preserving theories with possibly different properties of overloaded symbols

3. to get results concerning the classification of unification problem with respect to the signature in an algebraic style *a la* Herbrand or Martelli-Montanari.

The next section is devoted to the basis of the order-sorted logic needed in this work. In the last section we develop an example where the approach is applied to overloaded symbols with different properties. By lack of space, we omit all the proofs.

2 The basic framework

We present here the basis of the order-sorted framework in the style of [25] which also presents historical remarks and give an overview of the use of order-sorted equational logic for algebraic specifications that have been initiated by J. Goguen [5,7]. For a logic including sort declaration of terms, one may consult [19].

2.1 Order-sorted algebras

Let S be a set of **sort symbols** denoted $s_1, s_2, \ldots$ and F a set of **function symbols** denoted $f, g, \ldots$ with a fixed **arity** $|f|$, X a set of **variables** denoted $x, y, \ldots$, such that every variable has a sort s, often denoted $x : s$. We suppose given a partial order $\leq$ on the set of sort symbols: it can be defined as the least partial order containing **subsort declarations** of the form $s_1 < s_2$ where s_1 and s_2 are sort symbols. If two sorts s and s' are not comparable we write $s \bowtie s'$. A **function declaration** has the form $(f : s_1, \ldots, s_n \rightarrow s)$, where n is the arity of f and $s_1 \ldots, s_n, s$ are sort symbols. An **order-sorted signature** Σ is a 4-tuple of a sort set S_Σ, a partial order $\leq$ on S_Σ, a sort declaration and an operator declaration sets denoted respectively SD_Σ and OD_Σ. A signature is finite if all its components are finite. A set of Σ-variables is a set of variables whose sorts belong to S_Σ.

A Σ-**term** t of sort $s \in S_\Sigma$ is either a variable of sort s' with $s' \leq s$ or has the form $f(t_1, \ldots, t_n)$ where there is a declaration $f : s'_1, \ldots, s'_n \rightarrow s'$ in the set of operator declarations OD_Σ such that $s' \leq s$ and for $i \in [1..n]$, t_i is a Σ-term of sort s'_i. We may speak of terms instead of Σ-terms when the signature Σ is clear from the context. The set of Σ-terms built on a set of Σ-variables X is denoted $T(\Sigma, X)$. The set of variables occurring in a term t is denoted $Var(t)$.

A Σ-**substitution** σ is a mapping from a finite set of variables $D(\sigma)$ called its **domain** such that $\forall (x : s) \in D(\sigma), \sigma(x)$ is a term of sort s. The image $I(\sigma)$ is defined as usual: $I(\sigma) = \cup_{x \in D(\sigma)} Var(\sigma(x))$. Σ-substitutions are extended to Σ-terms classically. A substitution σ is denoted by its graph $\{(x_1 \leftarrow t_1), \ldots, (x_n \leftarrow t_n)\}$. $\sigma_{|W}$ is the restriction of the substitution σ to the subset W of X. If Φ is a set of substitutions then $\Phi_{|W} = \{\sigma_{|W} | \sigma \in \Phi\}$ is the set of elements of Φ restricted to W.

A Σ-algebra $\mathcal{A}$ consists of denotations $s^\mathcal{A}$ and $f^\mathcal{A}$ for the sort and function symbols in Σ, such that:

- $s^\mathcal{A}$ is a set,
- if $(s < s')$ is a subsort declaration in SD_Σ then $s^\mathcal{A} \subseteq s'^\mathcal{A}$,
- $C^\mathcal{A} = \cup_{s \in \Sigma} s^\mathcal{A}$ is the carrier of $\mathcal{A}$.
- $f^\mathcal{A}$ is a mapping $D_f^\mathcal{A} \rightarrow C^\mathcal{A}$ whose domain $D_f^\mathcal{A}$ is a subset of $(C^\mathcal{A})^{|f|}$.

- If $(f : s_1, \ldots, s_n \rightarrow s) \in \Sigma$ and if for $i \in [1..n]$ $a_i \in s_i^{\mathcal{A}}$, then $(a_1, \ldots, a_n) \in D_f^{\mathcal{A}}$ and $f^{\mathcal{A}}(a_1, \ldots, a_n) \in s^{\mathcal{A}}$.

As one would expect, the Σ-terms form a Σ-algebra. A **Σ-axiom** is a pair of Σ-terms denoted $s = t$. An **order-sorted equational specification** (Σ, A) consists of a signature Σ and a set of axioms A.

Validity of an axiom is defined as usual [25]. For an equational specification $\mathcal{S} = (\Sigma, A)$, we write $t =_{\mathbf{A}} t'$ if and only if t and t' are two Σ-terms and the axiom $t = t'$ is valid in any model of $\mathcal{S}$.

A signature Σ is **regular** when every Σ-term t has a least sort $ls(t)$. A specification is regular iff its signature is regular. An example of non-regular signature is $\{a :\rightarrow s_1, a :\rightarrow s_2\}$. All the examples of signatures given in the following are regular. One can prove that the regularity of *finite* signature is decidable and that $\emptyset$-unification in non-regular signature is infinitary while unification in regular signature is finite [19]. A regular specification (Σ, A) is **sort-preserving** if $ls(t) = ls(t')$ whenever $t =_A t'$.

We denote by $\leq_{\mathbf{A}}$ the subsumption preorder on $T(\Sigma, X)$ defined by: $t \leq_A t'$ iff $t' =_A \sigma(t)$ for a substitution σ called a **match** from t to t'. Composition of substitutions σ and ρ is denoted by $\sigma \circ \rho$, thus $\sigma \circ \rho(x) = \sigma(\rho(x))$. Given a subset V of X, we define $\sigma \leq_A \sigma'$ $[V]$ iff $\exists \sigma''$ $s.t.$ $\forall x \in V, \sigma'' \circ \sigma(x) =_A \sigma'(x)$. The qualification $[V]$ is omitted when $V = X$.

2.2 Unification

A detailed technical exposition of unsorted equationel unification is presented in [11,12]. A survey of already known results, essentially in unsorted cases, can be found in [22].

A **Σ-multiequation** is a multiset of Σ-terms, a **Σ-equation** is a multiequation containing exactly two terms, it is denoted $t == t'$. A **unificand** or **unification problem** in a specification (Σ, A) is either a Σ-multiequation or a set of Σ-multiequations $\{e_1, \ldots, e_n\}$ also called a **system** and denoted $(e_1 \wedge \ldots \wedge e_n)$, or a set of systems $\{S_1, \ldots, S_n\}$, also called a **disjunction of systems** and written as $S_1 \vee \ldots \vee S_n$. The set of variables occurring in an unificand U is denoted $Var(U)$. For simplicity we only keep in this paper equations since a multiequation can always be written in the form of a system of equations. For a specification $\mathcal{S} = (\Sigma, A)$, a **$\mathcal{S}$-solution** of the equation $t_1 == t_2$ is a Σ-substitution σ such that $\sigma(t_1) =_A \sigma(t_2)$. A substitution σ is $\mathcal{S}$-solution of a system if and only if σ is $\mathcal{S}$-solution of all the equations in the system. σ is $\mathcal{S}$-solution of a disjunction of systems if and only if σ is $\mathcal{S}$-solution of at least one of its systems. Solving a unificand U consists in finding the set $SU(U, \mathcal{S})$ of all its $\mathcal{S}$-solutions. But in fact it is sufficient to find a 'basis' of the set $\mathcal{S}$-solutions, called a **complete set of $\mathcal{S}$-solution** [18], denoted $CSU(U, \mathcal{S})$ and defined as follow: Φ is a **complete set of $\mathcal{S}$-solutions** of the unificand U away from the set of variables W such that $Var(U) \subseteq W$ iff:

1. $\forall \sigma \in \Phi, D(\sigma) \subseteq Var(U)$ and $I(\sigma) \cap W = \emptyset$
2. $\forall \sigma \in \Phi, \sigma \in SU(U, \mathcal{S})$.
3. $\forall \alpha \in SU(U, \mathcal{S}), \exists \sigma \in \Phi$ such that $\sigma \leq_A \alpha$ $[Var(U)]$.

Furthermore Φ is said **minimal** if it satisfies:

4 $\forall \sigma, \sigma' \in \Phi, \sigma \leq_A \sigma' [Var(U)] \Rightarrow \sigma = \sigma'$.

By definition, two unificands U_1 and U_2 are **$\mathcal{S}$-equivalent** iff they have the same set of $\mathcal{S}$-solutions. This is denoted $U_1 \rightleftharpoons_{\mathcal{S}} U_2$ or simply $U_1 \rightleftharpoons U_2$.

3 Simplification of a unification problem

We present the simplification of a unification problem with high-level transformation rules, thus separating as much as possible actions from control.

For a specification $\mathcal{S}$, a transformation Tr of a unificand U is called **correct** if no new solution is introduced:

$$SU(Tr(U), \mathcal{S}) \subseteq SU(U, \mathcal{S})$$

and **complete** when

$$SU(U, \mathcal{S}) \subseteq SU(Tr(U), \mathcal{S})_{|Var(U)}.$$

The last restriction on the variables of the transformed solution set is the same as in unsorted theories and is treated with the dependency relation in [12]. It is due to the fact that new variables may be introduced by the transformation Tr as it is the case for example when generalizing a unificand. In the case of order-sorted specifications this can also occur as in the following example:

Example 1 *If $s_1 < s_2$, $s_1 < s_3$ and $a :\rightarrow s_1$, then the equation $x : s_2 == x : s_3$ is transformed, as we will see later, into the system $\{x : s_2 == z : s_1, y : s_3 == z : s_1\}$ and $\alpha = \{(x \leftarrow a), (y \leftarrow a)\}$ is a solution of the equation but not of the system above.*

3.1 Axioms independent transformations

Our notations are the same as [13]. Note only that the system constructor is $\wedge$ and that the disjunction system constructor is $\vee$. ∇ is a system without solution that always exists when the signature contains a sort with at least two different constants.

For a set of axioms A, the set of *A-decomposable* symbols $F_d^{\mathcal{S}}$ is the largest subset of F such that: for any f and f' in $F_d^{\mathcal{S}}$, for any terms $t = f(t_1, ..., t_n)$, $t' = f'(t'_1, ..., t'_p)$:

1. $f \neq f' \Rightarrow SU(t == t', \mathcal{S}) = \emptyset$
2. $f = f' \Rightarrow (t == t' \rightleftharpoons_{\mathcal{S}} \{t_i == t'_i\}_{i=1,...,n})$.

The next transformations preserve the set of $\mathcal{S}$-solutions of the unificand considered:

Decomposition $\quad \dfrac{(f(t_1, \ldots, t_n) == f(t'_1, \ldots, t'_n))}{[(t_1 == t'_1) \wedge \ldots \wedge (t_n == t'_n)]}$ if $f, g \in F_d^{\mathcal{S}}$

Clash of symbols $\quad \dfrac{(f(t_1, \ldots, t_n) == g(t'_1, \ldots, t'_p))}{\nabla}$ if $f, g \in F_d^{\mathcal{S}}$ and $f \neq g$

Merging $\quad \dfrac{(x : s == t) \wedge (x : s == t')}{(x : s == t) \wedge (t == t')}$

Trivial equation $\quad \dfrac{(t == t) \wedge U}{U}$

Note that for all these rules, the sort information is not used. In fact, when $A = \emptyset$, rules for simplifying systems are the same as in the unsorted case, difficulties come later.

3.2 Mutation

Decomposition and merging yield the following kinds of multiequations:

1. $f(t_1, \ldots, t_n) == g(t'_1, \ldots, t'_p)$ with at least one of f or g not in F_d^S.
2. $x == t$.

The last kind of equations is said *fully decomposed.*

In case (1), it is necessary to find a correct and complete transformation that allows the decomposition-merging process to be continued. To do so, we introduce as in the unsorted case [10,12] the **mutation** transformation that, for a specification (Σ, A), takes a unificand U and returns a new unificand denoted $Mut_{(\Sigma,A)}(U)$.

Example 2 *If the symbol + is commutative (and only commutative) then the following transformation of the unification problem is correct and complete:*

commutative mutation $$\frac{(t_1 + t_2 == t_3 + t_4)}{[(t_1 == t_3 \wedge t_2 == t_4) \vee (t_1 == t_4 \wedge t_2 == t_3)]}$$

Note that, since + is only commutative, no sort information has to be used.

Mutation in order-sorted and unsorted theories are identical (i.e. the transformation is applied on terms as if they were unsorted, like in the example above) only for sort-preserving theories with no overloaded symbols with *different properties.* A simple example of such an overloading with different properties, given in an OBJ like syntax [6], is as follows:

Example 3

```
specification E is
  sort s1, s2
  subsort s1 < s2
  op a : -> s1
  op b : -> s2
  op + : s1 s1 -> s1 [associative-commutative]
  op + : s2 s2 -> s2 [commutative]
  var x,y,z : s2
end specification
```

When solving the equation $\mathtt{b} + \mathtt{y} == \mathtt{a} + \mathtt{z}$, there is no ambiguity on the properties of +, its rank should be (`s2 s2 -> s2`) and thus one can apply the mutation rule for commutativity only on this equation. The situation is similar to what happens for matching where the right-hand side of the equation tells the property that has to be used. By contrast, for the equation $\mathtt{x} + \mathtt{y} == \mathtt{a} + \mathtt{z}$ one *cannot* guess the property of + before instantiation of the variable `z`. The different possibilities (two in this specific case) have then to be tested.

Note that no checking is needed a posteriori to ensure that correctness is preserved since the properties of the highest operator (in this example `+:s2 s2 -> s2`) are included into the property of the ones under it (here `+:s1 s1 -> s1`).

For the example, we get:

$$\mathtt{Mut_E}(\mathtt{x} + \mathtt{y} == \mathtt{a} + \mathtt{z}) = (\mathtt{Mut_{AC}}(\mathtt{x} + \mathtt{y} == \mathtt{a} + \mathtt{z}) \vee \mathtt{Mut_C}(\mathtt{x} + \mathtt{y} == \mathtt{a} + \mathtt{z}))$$

where $Mut_{\mathtt{AC}}$ (resp. $Mut_{\mathtt{C}}$) denotes the unsorted mutation operation for the unsorted associative-commutative theory (resp. the commutative one).

This leads us to distinguish two cases:

1. The specification is not sort-preserving, in which case the mutation operation is specific to the order-sorted framework and has to be directly provided.

2. The specification is sort-preserving and a mutation operation is known for each subspecification relative to one sort. This is the subject of the remainder of this section.

In order to give an intuition of the technical stuff that is needed, let us first give a more complicated version of the previous example.

Example 4

```
specification F is
  sort s1, s2
  subsort s1 < s2
  op a, c : -> s1
  op b : -> s2
  op + : s1 s1 -> s1 [associative-commutative]
  op + : s2 s2 -> s2 [associative]
end specification
```

It is almost the same as Example 3 except that the operator + is now associative on **s2**. The mutation in **F** of the equation $\mathtt{e} = (\mathtt{a} + (\mathtt{x} : \mathtt{s1} + \mathtt{b}) == (\mathtt{c} + \mathtt{a}) + \mathtt{b})$ can not be done with $\mathtt{Mut}_{\mathtt{A}}(\mathtt{e})$ since it may not be complete: the solution $\{(\mathtt{x} \leftarrow \mathtt{c})\}$ may not be found since the *usual* mutation for the associative theory is flattening the terms (more generally, it uses information not restricted to the root operator) and thus is presupposing that all the + under the top one are also *only* associative, which is false in this example.

The way out consists to first generalize the terms in such a way that this kind of confusion can no more be done by the unsorted mutation. In the case of the example, **e** is generalized in the system $(\mathtt{a} + \mathtt{z}_1 : \mathtt{s2} == \mathtt{z}_2 : \mathtt{s1} + \mathtt{b}) \wedge (\mathtt{z}_1 == \mathtt{x} + \mathtt{b}) \wedge (\mathtt{z}_2 == \mathtt{c} + \mathtt{a})$ and the unsorted mutation for associativity can then safely applied on the first equation and the system further solved. The following paragraph is formalizing these transformations.

For a regular sort-preserving order-sorted specification S and a sort s, let A^s be the largest subset of A such that every axioms in A^s is of least sort s. For a symbol f, A^s_f denotes the subset of A^s with top symbol f. In the previous example, $\mathtt{A}^{\mathtt{s1}} = \{\mathtt{AC}(+)\}$ and $\mathtt{A}^{\mathtt{s2}} = \{\mathtt{C}(+)\}$. If the specification $S = (\Sigma, A)$ is regular and sort-preserving, there exists a partition $(A_i)_{i \in I}$ of A such that $\forall i, \exists s \in S_\Sigma$ such that $A_i = A^s$.

Any system S can always be generalized in such a way that its equations contain only terms of height zero or one built on a symbol f (i.e. are either constant or variable or of the form $f(a_1, \ldots . a_n)$ where the a_i are variables or constants). A system of height one built on the symbol f is denoted S^1_f and called f-top system of height one. Note that it is sufficient to precise the mutation transformation on such systems.

In a regular sort-preserving specification $S = (\Sigma, A)$, we call sorts of a multiequation $e = \{t_1, \ldots, t_n\}$ the set $so_e = ls(t_1) \wedge \ldots \wedge ls(t_n)$ where $s \wedge s'$ denotes the set of maximal sorts less than s and s'. If S is a f-top system of height one, so_S is the set $\wedge_{e \in S} so_e$.

We can now formulate the mutation transformation for order-sorted theories with overloaded operator with different properties:

general mutation $$\frac{Mut_S(S^1_f)}{\bigvee_{A^s_f \text{ with } s \in so_{S^1_f}} Mut_{A^s_f}(S)}$$

where $Mut_{A_f^s}$ denotes the unsorted mutation operation for the unsorted theory defined by the set of axioms A_f^s.

Lemma 1 *For a regular and sort-preserving specification, provided that the unsorted mutation operation are correct and complete, the* general mutation *transformation is correct and complete.*

4 Solving elementary equations

One of the main difference between unsorted and order-sorted equational unification is that complete set of $\mathcal{S}$-solutions for elementary equations, i.e. equation of the form $x == t$ where x in a variable and t a (possibly variable) term, are in general more complex in the second case. It comes from the fact that by instanciating variables, the terms x and t may have a common sort, smaller than the sort of the un-instanciated terms.

Example 5 [2] *Let be* $\mathtt{S} = \{\mathtt{List}, \mathtt{Nelist}\}$ *with* $\mathtt{NeList} < \mathtt{List}$ *and the overloaded operator* append *:*

$$\begin{aligned}
&\mathtt{append}: \mathtt{List} \times \mathtt{NeList} \to \mathtt{NeList}\\
&\mathtt{append}: \mathtt{NeList} \times \mathtt{List} \to \mathtt{NeList}\\
&\mathtt{append}: \mathtt{List} \times \mathtt{List} \to \mathtt{List}\\
&\mathtt{append}: \mathtt{NeList} \times \mathtt{NeList} \to \mathtt{NeList}
\end{aligned}$$

Then the equation $\mathtt{x} : \mathtt{NeList} == \mathtt{append}(\mathtt{y_1} : \mathtt{List}, \mathtt{y_2} : \mathtt{List})$ *has the following minimal complete set of* $\emptyset$*-solutions.*

$$\begin{aligned}
\Sigma = \{ \quad & (\mathtt{x} \leftarrow \mathtt{append}(\mathtt{z} : \mathtt{NeList}, \mathtt{z'} : \mathtt{List}), \mathtt{y_1} \leftarrow \mathtt{z}, \mathtt{y_2} \leftarrow \mathtt{z'})\\
& (\mathtt{x} \leftarrow \mathtt{append}(\mathtt{z'} : \mathtt{List}, \mathtt{z} : \mathtt{NeList}), \mathtt{y_1} \leftarrow \mathtt{z'}, \mathtt{y_2} \leftarrow \mathtt{z})\}
\end{aligned}$$

But it can be more tricky when axioms are added:

Example 6 *Consider the following sorts* $\mathtt{s_1} \leq \mathtt{s}, \mathtt{s"}$ *and* $\mathtt{s'_1} \leq \mathtt{s"}, \mathtt{s'}$ *the signature:*

$$\begin{aligned}
&\mathtt{a}: \quad \to \mathtt{s_1}\\
&\mathtt{b}: \quad \to \mathtt{s'_1}\\
&\mathtt{f}: \ \mathtt{s_1} \to \mathtt{s_1}\\
&\mathtt{f}: \ \mathtt{s'_1} \to \mathtt{s'_1}
\end{aligned}$$

and the axiom $\mathtt{a} = \mathtt{b}$. *Then the equation* $\mathtt{x} : \mathtt{s} == \mathtt{y} : \mathtt{s'}$ *has the following infinite minimal complete set of* $\mathcal{S}$*-unifiers*

$$\Sigma = \{(\mathtt{x} \leftarrow \mathtt{f}^{\mathtt{n}}(\mathtt{a})), (\mathtt{y} \leftarrow \mathtt{f}^{\mathtt{n}}(\mathtt{b})) | \mathtt{n} \geq 0\}$$

Let us first consider the simplest case:

Lemma 2 *The equation* $x : s == t$ *with* $ls(t) \leq s$ *has* $\{(x \leftarrow t)\}$ *as minimal complete set of* $\mathcal{S}$*-solution.*

These equations are said in **solved form.**

4.1 Solving equations $x : s == y : s'$ with $s \bowtie s'$

Definition 1 *An order-sorted equational specification S is said* **direct** *iff*

$$\forall t, t', t =_A t' \Rightarrow ls(t) \leq ls(t') \text{ or } ls(t') \leq ls(t)$$

Thus in direct order-sorted equational theories there is no axioms that collapse two terms of incomparable sorts, as in the previous example.

Lemma 3 *Let S be a direct order-sorted equational specification and $e = (x : s == y : s')$ an elementary equation where x and y are variables. Then*

$$\Sigma = \bigcup_{s" \in s \wedge s'} \{(x \leftarrow z : s", y \leftarrow z : s")\}.$$

is a complete set of S-solutions of e.

In other words the following transformation rule is correct and complete:

solving $x == y$: $$\frac{x : s == y : s'}{\bigvee_{s'' \in s \wedge s'} (x == z : s'') \wedge (x == z : s'')}$$

Note that the disjunction of system obtained with this transformation contains at most one system when the signature considered is **downward complete** that is such that for every two sort symbols s_1 and s_2, the set $\max\{s | s \leq s_1 \text{ and } s \leq s_2\}$ contains at most one element. It is thus a necessary condition for a direct specification to be unitary unifying i.e. such that every equation has a minimal complete set of S-solution of at most one element [23]. Of course one can give a similar condition for the specification to be finitary unifying.

Note that a particular instance of the rule above is of importance since it allows to cut down the search space.

clash of sorts $$\frac{x : s == y : s'}{\nabla} \quad \text{if } s \wedge s' = \emptyset$$

This rule can be used at simplification of systems time, so that incompatibilities are early detected.

Example 7 *In the specification* $(\mathtt{s_1} < \mathtt{s_3}, \mathtt{s_2} < \mathtt{s_3}, (\mathtt{f} : \mathtt{s_3}, \mathtt{s_3} \rightarrow \mathtt{s_3}))$ *the equation* $\mathtt{f}(\mathtt{u} : \mathtt{s_3}, \mathtt{f}(\mathtt{u} : \mathtt{s_3}, \alpha)) == \mathtt{f}(\mathtt{x} : \mathtt{s_1}, \mathtt{f}(\mathtt{y} : \mathtt{s_2}, \beta))$ *will be transformed by decomposition and merging into the system* $\{\mathtt{u} == \mathtt{x} : \mathtt{s_1} == \mathtt{y} : \mathtt{s_2}, \alpha == \beta\}$ *which has no solution by application of* clash of sorts. *But in the approach consisting to first solve the equation as it were unsorted, the clash of sort is not detected at this level and the problem $\alpha == \beta$ (which can be quite big) is first solved.*

4.2 Solving equations $x : s == t$ with $s \bowtie ls(t)$

For equations of the form $x : s == t$ where t is a non-variable term with $x \notin Var(t)$, there are two cases:

1. if $ls(t) \leq s$ then the minimal complete set of unifiers is $\{(x \leftarrow t)\}$ as stated above,
2. If $ls(t) \not\leq s$ then s and $ls(t)$ may have common subsorts that have to be identified with respect to the structure of t. This is very close to the so-called *intended parse* algorithm in [8,15]. We give now an equational formulation of the transformation needed to solve the problem.

$$\textsf{solving } x == t \quad \frac{x : s == f(t_1, \ldots, t_n)}{\bigvee_{\substack{f : s_1, \ldots, s_n \to s' \\ s' \leq s \\ s_1, \ldots, s_n \textit{ maximal}}} (\wedge_{i=1,\ldots,n} x_i : s_i == t_i) \wedge (x : s == f(x_1, \ldots, x_n))}$$

where the n-tuple of sorts $(s_1, \ldots, s_n)$ is said maximal iff it is maximal for the componentwise extension of the sort order.

Lemma 4 *The* solving $x == t$ *rule is correct and complete.*

As in [7], we say that a signature is **coregular** iff for every Σ-function symbol f and every sort symbol s in S_Σ, the set

$$\max\{(s_1, \ldots, s_n) | (f : s_1, \ldots, s_n \to s') \in \Sigma \text{ and } s' \leq s\}$$

has at most one element.

If the signature is coregular then, from the rule solving $x == t$, it is clear that the simplification of an equation $x == t$ gives a disjunction of systems containing none or only one system. It is thus a necessary condition for sort-preserving specification to be unitary unifying.

5 Solving fully decomposed systems

If there exists a mutation operation for the specification S considered, the decomposition merging mutation and solving transformation rules yield, when they terminate, a disjunction of systems whose equations e are in solved form that is $e = (x : s == t)$ with $ls(t) \leq s$. Such systems are said **fully decomposed**.

In order to solve fully decomposed systems, let us introduce the classical [16,9] ordering on fully decomposed equations:

$$e = (x == t) \prec e' = (x' == t') \Leftrightarrow x \in \mathrm{Var}(t')$$

Let us call **strict** those specifications $\mathcal{S}$ such that: if a fully decomposed system S has $\mathcal{S}$-solutions then $\prec^+$ is a strict ordering on S. We say that the system S is **cyclic** if $(S, \prec)$ has a cycle. **Simple theories** are such that there is no term A-equal to one of its proper subterm. Then, nice results of [1] are that a specification is simple if and only if it is strict and that any finite specification is simple. We also consider **strongly complete** theories [10], defined as the theories such that every equation $x : s == t$ (with the variable x possibly appearing in t) with $ls(t) \leq s$, has a complete set of $\mathcal{S}$-unifiers where $\{x\}$ is the domain of every substitution in it.

In unsorted theories, one can show that strict theories are strongly complete [1]. This result extend to the order-sorted framework without difficulties. This allows to describe in a fairly easy way the set of $\mathcal{S}$-solutions of a fully decomposed system of equations:

$$\textsf{cycle} \quad \frac{S}{\nabla} \quad \text{if } S \text{ is cyclic}$$

$$\textsf{solving} \quad \frac{CSU(\{x_1 : s_1 == t_1, \ldots, x_l : s_l == t_l\})}{\{(x_1 \to t_1) \circ \ldots \circ (x_l \to t_l)\}} \quad \text{if } i < j \Rightarrow e_j \not\preceq e_i \text{ and } ls(t_i) \leq s_i$$

Theorem 1 *Let S be a strict, sort-preserving and regular specification. Then, provided they terminate, the transformation rules we give above yield a complete set of unifier of any unificand.*

The problem of termination is of course dependant of the control on the inference rules and of the specification $\mathcal{S}$. But when there is no overloading with different properties or when no generalization is needed in **general mutation** (as in example 3), then since the order-sorted mutation is equivalent to the unsorted one, the termination of the simplification process will depend on the signature.

6 Conclusion

This algebraic study of order-sorted equational unification shows that provided that the signature is regular, which is a natural restriction, the tools already built for equational unsorted theories generalize nicely, allowing a precise study of the problem. In particular in the case of sort-preserving axioms, we have generalized the already known results on order-sorted equational unification when the symbols are overloaded with different properties. As a consequence, the tools built for studying combination of unification algorithms in an algebraic way [11,20] generalize to order-sorted equational unification. Another consequence of the results described here is to solve directly the problem of associative-commutative order-sorted unification.

Acknowledgments: I would like to thank Hélène Kirchner and Aristide Megrelis for many useful discussions.

References

[1] H-J. Bürckert, A. Herold, and M. Schmidt-Schauß. On equational theories, unification and decidability. In *Proceedings of the Second Conference on Rewriting Techniques and Applications*, Springer-Verlag, Bordeaux (France), May 1987.

[2] R.J. Cunningham and A.J.J. Dick. *Rewrite Systems on a Lattice of Types.* Technical Report, Imperial College, Department of Computing, 1983.

[3] K. Futatsugi, J. Goguen, J-P. Jouannaud, and J. Meseguer. Principles of OBJ-2. In B. Reid, editor, *Proceedings of 12th ACM Symposium on Principles of Programming Languages*, pages 52–66. Association for Computing Machinery, 1985.

[4] I. Gnaedig, C. Kirchner, and H. Kirchner. Equational completion in order-sorted algebras. In M. Dauchet and M. Nivat, editors, *Proceedings of the 13th Colloquium on Trees in Algebra and Programming*, pages 165–184, Springer-Verlag, Nancy (France), 1988.

[5] J. Goguen. *Order Sorted Algebra.* Technical Report, UCLA Computer Science Department, 1978. Semantics and Theory of Computation Report 14.

[6] J. Goguen, C. Kirchner, H. Kirchner, A. Megrelis, J. Meseguer, and T. Winkler. An introduction to OBJ-3. In J-P. Jouannaud and S. Kaplan, editors, *Proceedings of the First International Workshop on Conditional Term Rewriting Systems*, Springer-Verlag, Orsay (France), june 1988. Also as internal report CRIN: 88-R-001.

[7] J. Goguen and J. Meseguer. *Order-Sorted Algebra I: Partial and Overloaded Operations, Errors and Inheritance.* Technical Report, SRI International, Computer Science Lab, 1988. Given as lecture at Seminar on Types, Carnegie-Mellon University, June 1983.

[8] J.A. Goguen, J.P. Jouannaud, and J. Meseguer. Operational semantics for order-sorted algebra. In *Proceeding of the 12th International Colloquium on Automata, Languages and Programming*, pages 221–231, Nafplion (Greece), 1985.

[9] G. Huet. Résolution d'equations dans les langages d'ordre 1,2, ...,ω. Thèse d'état de l'Université de Paris 7, 1976.

[10] C. Kirchner. Computing unification algorithms. In *Proceeding of the first symposium on Logic In Computer Science, Boston (USA)*, pages 206–216, 1986.

[11] C. Kirchner. *From Unification in Combination of Equational to A New AC-Unification algorithm.* Technical Report 87-R-132, Centre de Recherche en Informatique de Nancy, 1987. To appear in the proc. of the CREAS.

[12] C. Kirchner. Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles. Thèse d'état de l'Université de Nancy I, 1985.

[13] C. Kirchner. Methods and tools for equational unification. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*, Austin (Texas), May 1987.

[14] C. Kirchner. Order sorted equational matching. Manuscript, 1986.

[15] C. Kirchner, H. Kirchner, and J. Meseguer. *Operational semantics of OBJ-3.* Technical Report 87-R-87, Centre de Recherche en Informatique de Nancy, 1987.

[16] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions On Programming Languages And Systems*, 4(2):258–282, 1982.

[17] J. Meseguer, J.A. Goguen, and G. Smolka. Order sorted unification. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*, Austin (Texas), May 1987.

[18] G. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[19] M. Schmidt-Schauss. *Computational Aspects of an Order-Sorted Logic with Term Declarations.* PhD thesis, Universität Kaiserslautern (West Germany), 1987.

[20] M. Schmidt-Schauss. *Unification in a combination of arbitrary disjoint equational theories.* Technical Report, universität Kaiserslautern, 1987.

[21] M. Schmidt-Schauss. Unification in many sorted equational theories. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction*, Springer Verlag, 1986.

[22] J. Siekmann. Unification theory. *Journal of Symbolic Computation*, to appear in 1988.

[23] J. Siekmann and P. Szabo. Universal unification. In R. Shostak, editor, *Proceedings 7th international Conference on Automated Deduction*, pages 1–42, Springer-Verlag, Napa Valley (California, USA), 1984.

[24] G. Smolka. *TEL (Version 0.9), Report and User Manual.* Technical Report SEKI-Report SR-87-11, Universitaet Kaiserslautern, West Germany, February 1988.

[25] G. Smolka, W. Nutt, J.A. Goguen, and J. Meseguer. Order sorted equational computation. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*, Austin (Texas), May 1987.

[26] C. Walther. Classification of many sorted unification problems. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction*, pages 525–537, Springer-Verlag, 1986.

[27] C. Walther. Many-sorted claculus based on resolution and paramodulation. In *Proceedings of 8th IJCAI, Karlsruhe*, pages 882–891, 1983.

[28] C. Walther. Many-sorted unification. *Journal of the Association for Computing Machinery*, 35(1):1–17, January 1988.

[29] C. Walther. Unification in many sorted theories. In T. O'Shea, editor, *Proceedings of the European Conference on Artificial Intelligence, Pisa, Italy*, pages 593–602, ECAI, 1984.

Contents

1 Introduction **2**

2 The basic framework **3**
2.1 Order-sorted algebras 3
2.2 Unification 4

3 Simplification of a unification problem **5**
3.1 Axioms independent transformations 5
3.2 Mutation 6

4 Solving elementary equations **8**
4.1 Solving equations $x : s == y : s'$ with $s \bowtie s'$ 9
4.2 Solving equations $x : s == t$ with $s \bowtie ls(t)$ 9

5 Solving fully decomposed systems **10**

6 Conclusion **11**

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique