In D. Kapur (ed.), *Lecture Notes in Computer Science: 11^{th} International Conference on Automated Deduction.*
New York: Springer-Verlag, 1992.

Sorted Unification Using Set Constraints

Tomás E. Uribe

Dept. of Computer Science and Beckman Institute, University of Illinois,
405 North Mathews Avenue, Urbana, Illinois, 61801, USA

Abstract. This paper describes a new representation for sortal constraints and a unification algorithm for the corresponding constrained terms. Variables range over sets of terms described by *systems of set constraints* that can express limited inter-variable dependencies. These sets of terms are more general than regular tree languages, but are still closed under intersection. The new unification algorithm shows sorted unification to be decidable for a broad class of sorted signatures, which we call *semi-linear*, and, more generally, for sort theories with a least Herbrand model that can be represented using the new constraints. A finite representation of a complete set of well-sorted unifiers can always be found, even in those cases where this set is infinite.

1 Introduction

Sorts are widely used to add more "structure" to first order logic, improving the efficiency of deductive systems [Cohn, 1989]. Increasingly, sorted logic is being viewed as an instance of constraint logic [Comon and Delor, 1991, Frisch, 1991]. This is the approach adopted in this paper: Instead of defining "well-sorted" terms and substitutions, we define a class of monadic constraints where variables range over sets of ground terms described by *systems of set constraints*. An algorithm that finds the solutions to a set of equations between constrained terms is then presented. The sets of ground terms over which variables can range are more general than regular tree languages, but are still closed under intersection.

Nearly all specifications of order-sorted logics impose restrictions on the sorted signature in order to ensure the decidability of unification. Implicitly, these restrictions have been such that sorts are at most as general as regular tree languages. Schmidt-Schauß [1989] considered the specification of sorted signatures through the more general mechanism of *term declarations*, but left open the problem of unification under *linear* signatures, where no dependencies between variables in the term declarations can be enforced. Linear signatures also correspond to regular tree languages, so that our algorithm shows unification to be decidable not only for these but a more general class of signatures, which we call *semi-linear*, where limited inter-variable dependencies in the term declarations are allowed. In general, if we are given a sort theory whose least Herbrand model can be represented with the new constraints, then sorted unification under that theory can be carried out (see Section 3).

Our constraints are closely related to the extended *membership constraints* presented by Comon and Delor [1991] in the case where *tree automata with equality tests* [Bogaert and Tison, 1992] are used. Instead of referring to the normalization and determinization of these automata, we present alternate algorithms based on transforming systems of set constraints. The results in this paper are theoretical in nature, and the problem of finding efficient implementations for the algorithms presented here is left open. Such implementations could increase the overall efficiency of deductive systems by expanding the range of sort theories that the unifier can reason about, and by allowing, in some cases, the finite representation of otherwise infinite sets of answers.

This paper summarizes a portion of [Uribe, 1991], where all complete proofs can be found. Section 2 presents the preliminary definitions and introduces the new constraints, together with a solved form for them. Section 3 briefly describes the relationship among these constraints, sort theories and sorted signatures. Section 4 presents an algorithm to rewrite constraints into solved form. Section 5 presents the actual unification algorithm, and Section 6 discusses related work.

2 Set Expressions and Constraints

We are given a countably infinite set of variables $\mathcal{V}$ and a ranked set of function symbols $\mathcal{F}$, from which terms are constructed in the usual way.[1] For any syntactic object P, $Var(P)$ denotes the set of variables occurring in P. A term t is *ground* if $Var(t) = \emptyset$. A substitution θ is written as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where each variable x_i is mapped to t_i, $x_i \neq x_j$ for $i \neq j$, and $t_i \neq x_i$ for all i. The term t_i is denoted by $x_i\theta$. The set of all variables x such that $x \neq x\theta$ is denoted by $Dom(\theta)$, and the set of all terms t such that $t = x\theta$ for $x \in Dom(\theta)$ is denoted by $Rng(\theta)$. Given substitutions θ and Ω, the substitution obtained by composing Ω onto θ is denoted by $\theta \cdot \Omega$; for all $x \in \mathcal{V}$, $x(\theta \cdot \Omega) = (x\theta)\Omega$. A substitution θ is *idempotent* if $\theta = \theta \cdot \theta$. Given a set of equations between terms $S = \{s_1 = t_1, \ldots, s_n = t_n\}$, a substitution θ is a *unifier* for S if $s_1\theta = t_i\theta$ for all i, $1 \leq i \leq n$. A substitution θ is *more general* than a substitution ρ if $\rho = \theta \cdot \gamma$ for some substitution γ.

We now define *systems of set constraints* (SSC's). These are based on the normal form for collections of set equations used in [Heintze and Jaffar, 1990], from which most of the notation in this section is in fact borrowed. However, new definitions and algorithms have to be used: Even though the syntax is simpler, we include limited inter-variable dependencies that allow SSC's to describe non-regular tree languages. The basic building blocks for SSC's are *set expressions*:

Definition 1 (Set Expressions). A *set expression* is (i) a term, (ii) one of the symbols "$\top$" and "$\bot$", or (iii) $(t \sqcup s)$ or $(t \sqcap s)$, where s and t are set expressions.

We do *not* allow "$\top$", "$\bot$", or any set expression that contains "$\sqcap$" or "$\sqcup$" to be nested inside a function symbol. We treat "$\sqcup$" and "$\sqcap$" as the idempotent, associative and commutative operators they will represent, so that a set expression of the form $(e_1 \sqcap (e_2 \sqcap \ldots (e_{n-1} \sqcap e_n) \ldots))$ can be written as $e_1 \sqcap \ldots \sqcap e_n$. A set expression is *atomic* if it is "$\bot$", "$\top$", or a term. A set expression is in *disjunctive normal form* (DNF) if it is of the form $e_1 \sqcup \ldots \sqcup e_m$, where each e_i is of the form $a_1 \sqcap \ldots \sqcap a_n$ for atomic set expressions $a_1, \ldots, a_n$. The result of rewriting a set expression exp into DNF is denoted by $DNF(exp)$.

[1] $\mathcal{F}$ should contain at least one constant symbol, so that the *Herbrand Universe*—the set of all ground terms—is not empty.

Definition 2 (Set Equations and Constraints). A *set equation* is a pair of set expressions $\{E_1, E_2\}$, written as $E_1 \equiv E_2$. A *set constraint* is an *ordered* set equation $X \equiv E$, where X is a variable and E is a set expression, written as $\langle X \equiv E\rangle$. The variable X is said to be *constrained* by E.

Set constraints will be written between brackets, as in $\langle X \equiv (Y \sqcap Z) \sqcup f(a)\rangle$.

Definition 3 (SSC's). A *system of set constraints* (SSC) is a finite set C of set constraints, $\{\langle Y_1 \equiv E_1\rangle, \ldots, \langle Y_n \equiv E_n\rangle\}$, such that $Y_i \neq Y_j$ if $i \neq j$. The set of variables $\{Y_1, \ldots, Y_n\}$ is the *domain* of C, denoted by $Dom(C)$, and the set of set expressions $\{E_1, \ldots, E_n\}$ is the *range* of C, denoted by $Rng(C)$. For each $Y_i \in Dom(C)$, the set expression E_i is denoted by $C(Y_i)$.

For our purposes, we will assume that $Var(Rng(C)) = Dom(C)$ for all SSC's C.

Definition 4 (Interpretations). An *interpretation* I defined on a finite set of variables $V \subset \mathcal{V}$ is a mapping from the variables in V into sets of ground terms. This mapping is extended to set expressions as follows:

1. $I(\top)$ is the Herbrand Universe; $I(\bot)$ is the empty set, $\emptyset$.
2. If X is a variable in V, then $I(X)$ is given by the original interpretation. If X is a variable not in V, then $I(X) \stackrel{\text{def}}{=} \emptyset$.
3. $I(e_1 \sqcap e_2) \stackrel{\text{def}}{=} I(e_1) \cap I(e_2)$; $I(e_1 \sqcup e_2) \stackrel{\text{def}}{=} I(e_1) \cup I(e_2)$.
4. Finally, if t is an arbitrary non-variable term, $I(t) \stackrel{\text{def}}{=} \{t\theta \,|\, \theta$ is a substitution with $X\theta \in I(X)$ for all $X \in Var(t)\}$. That is, $I(t)$ contains all those terms that can be obtained by replacing all occurrences of each variable X in t by an element of $I(X)$. Note that $I(t) = \emptyset$ if $I(X) = \emptyset$ for some $X \in Var(t)$.

The last condition ensures that equal variables within $f(t_1, \ldots, t_n)$ correspond to the same term, and is the main difference between these set constraints and Heintze and Jaffar's [1990] normal forms.

Two interpretations I and I' are *equivalent* over a set of variables V, written as $I =_V I'$, if for all $X \in V$, $I(X) = I'(X)$. A partial order on interpretations can be defined in the usual way, where $I \leq I'$ if for all $X \in \mathcal{V}$, $I(X) \subseteq I'(X)$.

Definition 5 (Models). Given an SSC C, an interpretation I is a *model* of C if $I(X) = I(C(X))$ for all $X \in Dom(C)$.

Two SSC's C and C' are *equivalent* over a set of variables V if for any model I of C, there exists a model I' of C' such that $I =_V I'$, and vice-versa. In this case, we write $C \Longleftrightarrow_V C'$. Given an SSC C and a set equation $E_1 \equiv E_2$, we write $C \models E_1 \equiv E_2$ if for any model I of C, $I(E_1) = I(E_2)$.

In general, SSC's need not have unique models over their constrained variables (though they always have a model). However, there is a simple sufficient condition under which an SSC has a unique model:

Definition 6 (Cycles). A *cycle* in an SSC C is a sequence of variables in $Dom(C)$, $X_1, \ldots, X_n$, such that $X_1 = X_n$ and for each i, $1 \leq i < n$, $C(X_i)$ contains an un-nested occurrence of X_{i+1}.[2]

[2] A variable is *un-nested* in a set expression if it is not a subterm of any other term.

Intuitively, a cycle means that a variable is defined directly in terms of itself, with no mediating function symbol. For example, the SSC $\{\langle X \equiv X \sqcap (a \sqcup b)\rangle\}$ has the simplest possible kind of cycle, while $\{\langle X \equiv f(X)\rangle\}$ does *not* contain a cycle.

Proposition 7. *Let C be an SSC with no cycles. Then for any two models I and I' of C, $I =_{Var(C)} I'$. Therefore, the model I of C such that $I(X) = \emptyset$ for all $X \notin Dom(C)$ is the* least *model of C.*

Definition 8 (Constrained Expressions). For any SSC C with no cycles, we denote the least model of C by M_C. A set expression E *constrained* by C, written as E/C, is said to stand for the set of terms $M_C(E)$. If $t \in M_C(E)$, we say that t is an *instance* of E/C.

All the SSC's we consider will have no cycles, so M_C will always be well-defined. Note that if $C_1 \Longleftrightarrow_V C_2$ and $Var(E) \subseteq V$, then $M_{C_1}(E) = M_{C_2}(E)$.

Given an arbitrary SSC C and $X_1, X_2 \in Dom(C)$, the question of whether the intersection of $M_C(X_1)$ and $M_C(X_2)$ is empty is undecidable. This follows from the undecidability of sorted unification for general sorted signatures with term declarations [Schmidt-Schauß, 1989], given the equivalence between SSC's and sorted signatures described in Section 3. This is why we will consider only *shallow* SSC's, where the only nested terms in $Rng(C)$ are variables:

Definition 9 (Shallow SSC's). An SSC C is *shallow* if the only terms in $Rng(C)$ are variables, constants, or of the form $f(V_1, \ldots, V_n)$ for an n-ary function f and variables $V_1, \ldots, V_n$, $n \geq 1$.

Definition 10 (Solved Form). An SSC C is in *solved form* if C has no cycles and for all $X \in Dom(C)$, $C(X)$ is "$\perp$", "$\top$", or $e_1 \sqcup \ldots \sqcup e_m$, $m \geq 1$, where $e_i \neq e_j$ for all $i \neq j$ and each e_i is a variable, a constant, or $f(X_1, \ldots, X_n)$ for variables $X_1, \ldots, X_n$, $n \geq 1$.

Note that all SSC's in solved form are shallow. As we show in Section 4, any shallow SSC C can be rewritten into an SSC in solved form, C', such that $C \Longleftrightarrow_{Dom(C)} C'$.

Example 1. $C = \{\langle Y \equiv X\rangle, \langle X \equiv f(X,X) \sqcup a\rangle\}$ is a shallow SSC in solved form; $M_C(Y) = M_C(X)$ is the infinite set of terms $\{t_0, t_1, t_2, \ldots\}$, where $t_0 = a$ and $t_n = f(t_{n-1}, t_{n-1})$, for $n \geq 1$. This is not a regular tree language. The set of ground instances of the constrained term $g(Y)/C$ is

$$M_C(g(Y)) = \{g(a),\ g(f(a,a)),\ g(f(f(a,a), f(a,a))),\ \ldots\}.$$

3 Sorted Signatures, Sort Theories and Constraints

Sorted logics can be specified using a *sorted signature* that describes well-sorted and ill-sorted terms [Schmidt-Schauß, 1989], or through the use of monadic constraints that refer to a background *sort theory* [Frisch, 1991].[3] For the purposes of this paper, sorted signatures can be described in terms of corresponding sort theories: A *sort theory* is a finite set of first order sentences containing only unary *sort predicates*.

[3] See [Cohn, 1989] for an overview of other dimensions among which sorted logics can vary. The above are the only two of interest to us now.

If equational theories are not included, a sorted signature Σ is equivalent to a sort theory $\mathcal{A}_\Sigma$ that can be written as a collection of Horn clauses. $\mathcal{A}_\Sigma$ is the signature's *relativization*, as defined by Schmidt-Schauß [1989]. For each sort symbol S in Σ, $\mathcal{A}_\Sigma$ contains a sort predicate with the same name. The sets of ground terms of a sort S as given by Σ correspond to the interpretation of S in the least Herbrand model of $\mathcal{A}_\Sigma$; that is, a ground term g is said to be of sort S in Σ if, and only if, $\mathcal{A}_\Sigma \models S(g)$. In both formulations, we write $x{:}S$ to indicate that x is a variable of sort S.

An *elementary* sorted signature[4] is equivalent to a set of Horn clauses of the form $S_1(x) \rightarrow S_2(x)$ for each subsort declaration $S_1 \sqsubseteq S_2$, and of the form $S_1(x_1) \wedge \ldots \wedge S_n(x_n) \rightarrow S_0(f(x_1, \ldots, x_n))$, where $x_i \neq x_j$, for each function declaration $f(S_1 \times \ldots \times S_n){:}S_0$. As first pointed out by Comon [1989], these signatures are also equivalent to finite frontier-to-root tree automata: The sets of ground terms that belong to each sort are regular tree languages. This is not always the case if we allow arbitrary *term declarations* as presented by Schmidt-Shauß [1989]. A term declaration $S_0[t]$ is equivalent to a Horn clause of the form $S_1(x_1) \wedge \ldots \wedge S_n(x_n) \rightarrow S_0(t)$, where $x_1, \ldots, x_n$ are the different variables in t and $S_1, \ldots, S_n$ are their respective sorts.

In the general case, sorted unification under term declarations is undecidable. Usually, only elementary signatures are considered, and their *regularity*—that every term have a unique least sort— is often required to ensure that there is always a finite number of maximally general unifiers.[5] These two restrictions are not needed if SSC's can be used: The infinite sets of unifiers can be represented finitely. We do make some general assumptions about sort theories (and hence signatures): The subsort relation should be a partial order, and for each sort S, there should be a ground term g of sort S, which ensures that sorts are not empty. Under these conditions, unification under a sorted signature is operationally equivalent to unification under the corresponding sort theory: The well-sorted substitutions and the generality ordering between them are the same [Uribe, 1991].

Definition 11. Let Σ be a sorted signature. An SSCC_Σ is *equivalent* to Σ if for each sort S in Σ there is a variable $\Upsilon_S \in Dom(C_\Sigma)$ such that the set of ground terms of sort S is equal to $M_{C_\Sigma}(\Upsilon_S)$.

If Σ can be rewritten as a Horn clause sort theory, then the model M_{C_Σ} corresponds to the least Herbrand model $\mathcal{M}$ of Σ, in the sense that $M_{C_\Sigma}(\Upsilon_S)$ is equal to the interpretation of the predicate S under $\mathcal{M}$.

If an SSC C_Σ is equivalent to a sorted signature Σ, then a sorted term t can be translated into a constrained term t'/C as follows: Let t' be the result of replacing each variable $x{:}S$ in t by a new variable N_x, and let C be the union of C_Σ and the set constraints $\{\langle N_x \equiv \Upsilon_S\rangle \mid x{:}S \in t\}$. The set of "well-sorted ground instances" of t will be exactly equal to $M_C(t')$. A unification problem given Σ can be rewritten as the problem of finding the solutions to a set of equations between terms constrained by SSC's, by translating in this way the terms in the original equation set. Thus,

[4] One with only subsort and function declarations, in Schmidt-Schauß's terms; these are often simply called "sorted signatures."

[5] Regularity is in general an undecidable property of signatures with term declarations [Schmidt-Schauß, 1989]. A consequence of the results presented here is that it is decidable for semi-linear signatures.

the algorithms of the following sections show that sorted unification under a class of signatures is decidable if these signatures can be represented as equivalent shallow SSC's.

Definition 12 (Semi-linear). A term t is *linear* if each variable in t occurs exactly once in t. A term t is *semi-linear* if each occurrence of a given variable in t has the same prefix.[6] A sorted signature is (semi-) linear if t is (semi-) linear for all term declarations $S[t]$. An SSC C is (semi-) linear if all terms appearing in $Rng(C)$ are (semi-) linear.

If Σ is a finite *semi-linear* sorted signature (or a corresponding sort theory) such that the subsort relation is a partial order, then one can easily obtain an SSC C_Σ in solved form that is equivalent to Σ in the sense of Definition 11 [Uribe, 1991].

Declarations (Σ)	**Axioms** ($\mathcal{A}_\Sigma$)
$\text{EVEN} \sqsubseteq \text{INT}$	$\forall x\ Even(x) \rightarrow Int(x)$
$\text{ODD} \sqsubseteq \text{INT}$	$\forall x\ Odd(x) \rightarrow Int(x)$
$\text{SQUARE} \sqsubseteq \text{INT}$	$\forall x\ Square(x) \rightarrow Int(x)$
$\text{EVEN}[0]$	$Even(0)$
$\text{EVEN}[s(x{:}\text{ODD})]$	$\forall x\ Odd(x) \rightarrow Even(s(x))$
$\text{ODD}[s(x{:}\text{EVEN})]$	$\forall x\ Even(x) \rightarrow Odd(s(x))$
$\text{EVEN}[sum(x{:}\text{EVEN}, y{:}\text{EVEN})]$	$\forall x \forall y\ Even(x) \wedge Even(y) \rightarrow Even(sum(x,y))$
$\text{SQUARE}[mult(x{:}\text{INT}, x{:}\text{INT})]$	$\forall x\ Int(x) \rightarrow Square(mult(x,x))$

Table 1. Sorted signature declarations and corresponding sort theory axioms.

Example 2. Table 1 shows a set of subsort and term declarations Σ together with a corresponding sort theory. Note that Σ is semi-linear: The only term declaration with repeated variables is $\text{SQUARE}[mult(x{:}\text{INT}, x{:}\text{INT})]$. An SSC in solved form equivalent to Σ is:

$$C_\Sigma = \left\{ \begin{array}{cc} \langle \Upsilon_{int} \equiv \Upsilon_{even} \sqcup \Upsilon_{odd} \sqcup s(\Upsilon_{int}) \rangle, & \langle \Upsilon_{square} \equiv mult(\Upsilon_{int}, \Upsilon_{int}) \rangle, \\ \langle \Upsilon_{even} \equiv 0 \sqcup s(\Upsilon_{odd}) \sqcup sum(X_e, Y_e) \rangle, & \langle \Upsilon_{odd} \equiv s(\Upsilon_{even}) \rangle, \\ \langle X_e \equiv \Upsilon_{even} \rangle, & \langle Y_e \equiv \Upsilon_{even} \rangle \end{array} \right\}$$

A linear SSC with no cycles is equivalent to a term grammar: $M_C(X)$ is a regular tree language for each $X \in Dom(C)$. Linear signatures can be expressed as equivalent linear SSC's. Any linear SSC with no cycles can be rewritten into an SSC in solved form, equivalent to C over $Dom(C)$, that is also linear. In this case, simpler versions of the algorithms presented here can be used.

4 Simplifying Constraints

A crucial element in most constraint-solving procedures is the ability to rewrite constraints into a solved form. In a constraint logic framework, the only operational requirement is that we be able to check whether a conjunction of constraints has

[6] For example, $f(g(x), g(x), y)$ is semi-linear, but $f(h(x, g(x)), y, h(x, g(x)))$ is not.

a (non-empty) solution (see [Jaffar and Lassez, 1987]). Rewriting SSC's into solved form (see Definition 10) is part of the process by which we do this.

We now present an algorithm that rewrites a shallow SSC into an equivalent one in solved form. This is an extension of the corresponding simplification procedure presented by Heintze and Jaffar [1990]. The main difficulty appears when simplifying subexpressions of the form $f(X_1, \ldots, X_n) \sqcap f(Y_1, \ldots, Y_n)$. Since we are expressing inter-variable dependencies within terms, we cannot do a simple pairwise matching of variables; instead, a form of unification must be used.

We begin with a number of transformations on an SSC C. Each replaces a set expression E in a set constraint $\langle X \equiv E \sqcup exp \rangle \in C$ by a new set expression and possibly adds new set constraints over new variables:[7]

S1: Replace $f(X_1, \ldots, X_n) \sqcap g(Y_1, \ldots, Y_m)$ by $\perp$ if $f \neq g$, for $m, n \geq 0$.

S2: Replace $(exp \sqcap \perp)$ by $\perp$, $(exp \sqcap \top)$ by exp, $(exp \sqcup \perp)$ by exp, and $(exp \sqcup \top)$ by $\top$, where exp is any set expression. Replace $(t \sqcap t)$ by t, and $(t \sqcup t)$ by t, for any term t.

S3: Replace $V \sqcap exp$ by $DNF(C(V) \sqcap exp)$, where $V \in Dom(C)$ and exp contains no un-nested occurrences of V.

S4: Replace $f(X_1, \ldots, X_n) \sqcap f(Y_1, \ldots, Y_n)$ by $f(Z_1, \ldots, Z_n)$, provided that:
(a) $Z_i = Z_j$ if, and only if, $X_i = X_j$ or $Y_i = Y_j$, $1 \leq i, j \leq n$, and
(b) $C \models Z_i \equiv \bigcap_{V \in S} V$, for each i, $1 \leq i \leq n$, where

$$S = \{X_j \mid Z_j = Z_i, 1 \leq j \leq n\} \cup \{Y_j \mid Z_j = Z_i, 1 \leq j \leq n\}.$$

In order to satisfy (b), new set constraints of the form $\langle Z_i \equiv V_1 \sqcap \ldots \sqcap V_n \rangle$ can be added to C, where Z_i is a new variable and each $V_i \in Dom(C)$.

Proposition 13. *If C' is obtained by applying any of the above transformations to an* SSC *C, then $C \Longleftrightarrow_{Dom(C)} C'$.*

To ensure termination, the actual simplification algorithm needs to keep track of the variables introduced by transformation S4. Some of these new variables will represent the intersection of variables in $Dom(C)$, while others will be "renamings" of these, needed to avoid introducing new inter-variable dependencies. To do this, we maintain a set that defines a function from variables into subsets of $Dom(C)$, indicating when a variable stands for the intersection of the variables in a subset of $Dom(C)$; Int is a set of pairs of the form $\langle V, \{V_1, \ldots, V_n\}\rangle$, where $Int(V) = S$ if $\langle V, S\rangle \in Int$ and $\{V\}$ otherwise. We also need to maintain Rep, a one-to-one mapping from subsets of $Dom(C)$ into the unique variable that represents its intersection (if such a variable exists); $Rep(S) = V$ if $\langle S, V\rangle \in Rep$.[8] We now define the following variant of transformation S4, as applied to an SSC C and the sets Int and Rep:

[7] In this and the following algorithms we assume that all set expressions in $Rng(C)$ are in disjunctive normal form. In practice, rewriting set expressions into DNF can be delayed until strictly necessary, to increase efficiency.

[8] This is an extension of the labeling scheme used by Heintze and Jaffar [1990], where variables are subscripted with the set of variables whose intersection they represent.

Input: A shallow SSC C with no cycles.
Output: An SSC in solved form.
Let $Int = \emptyset$, and let $Rep = \{\langle \{X\}, X\rangle \mid X \in Dom(C)\}$.
Exhaustively apply transformation S3.
WHILE any of transformations S1, S2 or S4′ apply, DO:
    Apply one of transformations S1, S2, or S4′.
    IF transformation S4′ is applied adding new elements to Rep,
        THEN exhaustively apply transformation S3.

Fig. 1. Simplification algorithm

S4′:
Consider an occurrence of $f(X_1, \ldots, X_n) \sqcap f(Y_1, \ldots, Y_n)$.
Let $t_1 = f(X_1, \ldots, X_n)$ and $t_2 = f(Y_1, \ldots, Y_n)$.
Let π be a renaming substitution such that $Var(t_2\pi)$ is a set of new variables.
Let θ be a most general unifier for $\{t_1 = t_2\pi\}$ such that $Var(Rng(\theta)) \subseteq Var(t_2\pi)$.
FOR EACH variable V in $Var(f(X_1, \ldots, X_n)\theta)$, DO:
- Let $S_V := \bigcup_{X \in \{X_i \mid X_i\theta = V\} \cup \{Y_i \mid Y_i\pi\theta = V\}} Int(X)$.
- If there exists a variable Z such that $Rep(S_V) = Z$, add the set constraint $\langle V \equiv Z\rangle$; otherwise, add the set constraint $\langle V \equiv V_1 \sqcap \ldots \sqcap V_m\rangle$, where $\{V_1, \ldots, V_m\} = S_V$, and let $Rep := Rep \cup \langle S_V, V\rangle$.
- Let $Int := Int \cup \{\langle V, S_V\rangle\}$.

Replace $t_1 \sqcap t_2$ by $t_1\theta$.

Proposition 14. *Given that $Int(V) = \{X_1, \ldots, X_n\}$ only if $C \models V \equiv X_1 \sqcap \ldots \sqcap X_n$ and that $Rep(\{X_1, \ldots, X_n\}) = V$ only if $C \models V \equiv X_1 \sqcap \ldots \sqcap X_n$, then transformation* S4′ *is equivalent to transformation* S4*. Furthermore, the above hypotheses will still hold for C' and the new Int and Rep after transformation* S4′ *is applied.*

The simplification algorithm can now be obtained by initializing Int and Rep and applying transformations S1–S3 and S4′ as shown in Figure 1. Transformations S1–S3 do not change Int and Rep. If the input is a shallow SSC with no cycles, then the simplification algorithm of Figure 1 terminates, and the output SSC is in solved form. Note that if C has no cycles then transformation S3 can only be applied a finite number of times to C; and if C contains no cycles, then the resulting C' after applying any of transformations S1–S4 contains no cycles either.

We can show that at any point during the simplification algorithm, if $Int(X) = \{Y_1, \ldots, Y_n\}$ or $Rep(\{Y_1, \ldots, Y_n\}) = X$, then $C \models X \equiv Y_1 \sqcap \ldots \sqcap Y_n$. Together with Propositions 13 and 14, this means that if C is the input SSC and C' is the output SSC (which is in solved form), then $M_C(X) = M_{C'}(X)$ for all $X \in Dom(C)$.

Recall that we allow $M_C(X) = \emptyset$ for a variable X constrained by an SSC C. Since we will use SSC's to represent non-empty solutions to sets of equations between constrained terms, we will need to be able to identify these *empty variables* once solved form is obtained.

Input: An SSC C in solved form. Let all variables in $Dom(C)$ be unmarked. While any of the following applies for an unmarked variable $X \in Dom(C)$, do:

- If $\langle X \equiv \top \sqcup exp \rangle \in C$, mark X.
- If $\langle X \equiv k \sqcup exp \rangle \in C$ for some constant k, mark X.
- If $\langle X \equiv Y \sqcup exp \rangle \in C$ and Y is a marked variable, mark X.
- If $\langle X \equiv f(Y_1, \ldots, Y_n) \rangle \in C$ and each Y_i is a marked variable, mark X.

At the end, a variable $X \in Dom(C)$ is unmarked if, and only if, $M_C(X) = \emptyset$.

Fig. 2. Identifying empty variables

Figure 2 presents an algorithm that identifies all the empty variables in a solved form SSC C, again adapted from the corresponding algorithm given by Heintze and Jaffar [1990]. At the end of the algorithm, $X \in Dom(C)$ is empty in C if, and only if, X is unmarked.[9]

5 The Unification Algorithm

Definition 15 (Constrained Equation Sets). A *constrained equation set* is a finite set S of equations between terms together with an SSC C such that $Var(S) \subseteq Dom(C)$, and is written as S/C. A substitution θ *solves* an equation set S/C if (i) for all equations $t_1 = t_2$ in S, $t_1\theta = t_2\theta$, and (ii) $X\theta \in M_C(X)$ for all $X \in Dom(\theta)$. Equivalently, we say that θ is a *solution* of S/C.

The unification problem is to find all the substitutions (if any) that solve a constrained equation set S/C. The algorithm presented here is based on a well-known approach to sorted unification. Given S/C, an unsorted most general unifier ψ for S is computed (ignoring C), and is then *weakened*: Constraints are found for the variables in $Rng(\psi)$ such that that for all $X \in Dom(\psi)$, $X\psi$ ranges over a subset of $M_C(X)$. Furthermore, the new constraints should represent all the solutions to the original constrained equation set S/C. Section 5.1 presents a procedure that will do this, by solving sets of *membership constraints*. The actual unification algorithm is then presented in Section 5.2.

5.1 Solving Membership Constraints

Given a substitution $\{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ and an SSC in solved form C, where $\{X_1, \ldots, X_n\} \subseteq Dom(C)$ and $Var(\{t_i\}) \cap Dom(C) = \emptyset$, we want to find all the substitutions ρ such that $t_i\rho \in M_C(X_i)$ for all i, $1 \leq i \leq n$. This process is equivalent to weakening an unsorted substitution so as to make it "well-sorted." The problem can be formulated in a more general way, as follows:

[9] Once empty variables have been identified, C can be rewritten into an equivalent SSC, C', such that that no empty variables appear in $Rng(C')$; this algorithm is straightforward (see [Uribe, 1991]), so it is not presented here.

Definition 16 (Membership Constraints). Given an SSC C, a *membership constraint* is an expression $t \Subset E$, for a term t and a set expression E. A substitution ρ *solves* a set of membership constraints $\{t_1 \Subset E_1, \ldots, t_n \Subset E_n\}$ if for all i, $1 \le i \le n$, $t_i\rho \in M_C(E_i)$.[10]

The following algorithm finds the solutions to a set of membership constraints given a *solved form* SSC C and *shallow* set expressions E_i. The solutions are represented using substitutions θ_j and SSC's C_j such that $M_{C_j}(t_i\theta_j) \subseteq M_C(E_i)$ for each membership constraint $t_i \Subset E_i$. In general, there is no single pair $\langle\theta, C'\rangle$ that can capture all possible solutions; however, we will always be able to find a *finite* set of such pairs such that every solution is an instance of one of them.

We assume that C contains no occurrences of "$\perp$": After using the algorithm in Section 4 to identify empty variables, it is simple to obtain an SSC equivalent to C such that for all empty variables X, $C(X) = \perp$ and X appears nowhere else in $Rng(C)$. If any of the variables X for an initial membership constraint $t \Subset X$ is empty in C, then there are no (non-empty) solutions. Otherwise, all the set constraints of the form $\langle X \equiv \perp\rangle$ can be removed from C.

The algorithm maintains a set $\mathcal{S}$ of *partial solutions*, each of which is a pair $\langle\theta_i, U_i\rangle$ for a set of membership constraints U_i and a substitution θ_i. Part 1 of the algorithm applies transformations to a set $\mathcal{S}$ of such pairs. Each transformation acts upon a pair $\langle\theta, U\rangle$ in $\mathcal{S}$. The substitution θ is not changed unless otherwise noted:

W1: $U = U_o \cup \{f(t_1,\ldots,t_n) \Subset g(X_1,\ldots,X_m) \sqcup exp\} \Longrightarrow U_o \cup \{f(t_1,\ldots,t_n) \Subset exp\}$ where exp is not empty, $f \ne g$, $m, n \ge 0$. Otherwise, remove $\langle\theta, U\rangle$ from $\mathcal{S}$.
W2: $U = U_o \cup \{f(t_1,\ldots,t_n) \Subset X \sqcup exp\} \Longrightarrow U_o \cup \{f(t_1,\ldots,t_n) \Subset C(X) \sqcup exp\}$.
W3: Consider a pair $\langle\theta, U\rangle$ where $U = U_o \cup \{f(t_1,\ldots,t_n) \Subset f(X_1,\ldots,X_n)\}$, for $n \ge 2$.

Let Ω be an idempotent most general unifier for the equation set
$\{t_i = t_j \mid X_i = X_j\}$.
If this unification fails, remove $\langle\theta, U\rangle$ from $\mathcal{S}$. Otherwise:
Let U'_o be the result of applying Ω to
the left hand side of each membership constraint in U_o.
Replace U by $U'_o \cup \{t_1\Omega \Subset X_1, \ldots, t_n\Omega \Subset X_n\}$ and replace θ by $\theta \cdot \Omega$.

W4: If $U = U_o \cup \{f(t_1,\ldots,t_n) \Subset e_1 \sqcup \ldots \sqcup e_m\}$ for $n > 1, m > 1$, where each e_i is of the form $f(Y_1,\ldots,Y_n)$, replace the pair $\langle\theta, U\rangle \in \mathcal{S}$ by the pairs $\langle\theta, U_o \cup \{f(t_1,\ldots,t_n) \Subset e_1\}\rangle, \ldots, \langle\theta, U_o \cup \{f(t_1,\ldots,t_n) \Subset e_m\}\rangle$.
W5: $U = U_o \cup \{f(t) \Subset f(X_1) \sqcup \ldots \sqcup f(X_n)\} \Longrightarrow U_o \cup \{t \Subset X_1 \sqcup \ldots \sqcup X_n\}$.
W6: $U = U_o \cup \{c \Subset e_1 \sqcup \ldots \sqcup e_n\} \Longrightarrow U_o$ for a constant c, if $e_i = c$ for some i, $1 \le i \le n$; if c is a constant and $e_i \ne c$ for all i, remove $\langle\theta, U\rangle$ from $\mathcal{S}$.
W7: $U = U_o \cup \{t \Subset \top \sqcup exp\} \Longrightarrow U_o \cup \{X_1 \Subset \top, \ldots, X_n \Subset \top\}$, where t is any non-variable term and $\{X_1, \ldots X_n\} = Var(t)$.

Figure 3 shows the weakening algorithm, which always terminates with a finite number of solutions. All the elements of each set of membership constraints U_i at

[10] In Comon and Delor's [1991] notation, a set of membership constraints would be the conjunction $t_1 \in S_1 \wedge \ldots \wedge t_n \in S_n$, where each S_i is the tree language $M_C(E_i)$.

Input: A set of membership constraints $U = \{t_i \in X_i\}$ and an SSC C in solved form, where $Var(\{t_i\})$ and $Var(C)$ are disjoint.
Output: A set of pairs $\langle \theta_i, C_i \rangle$, for SSC's C_i and substitutions θ_i.

Part 1: Let $\mathcal{S} = \{\langle\{\}, U\rangle\}$. Exhaustively apply transformations W1–W7 to $\mathcal{S}$.

Part 2: For each $\langle \theta_i, U_i \rangle \in \mathcal{S}$ do:
Let C_i be the union of C and the set constraints
$\langle X \equiv exp_1 \sqcap \ldots \sqcap exp_n \rangle$, for each $X \in Var(U)$, where
$\{X \in exp_1, \ldots, X \in exp_n\}$ are the membership constraints for X in U_i.

Return $\{\langle \theta_i, C_i \rangle\}$.

Fig. 3. Weakening algorithm

the end of Part 1 are of the form $X \in exp$, and the SSC C_i constructed from U_i in Part 2 is a shallow SSC with no cycles.

We must note that the substitution component of the partial solutions can be replaced by a set of equations to which transformations analogous to those in a normal unification algorithm are applied. The "variable isolation" transformation would also instantiate the terms in the left hand side of the membership constraints, while a new version of transformation W3 simply adds the appropriate equations to this set. No additional control is necessary to ensure the termination of this extended set of transformations—see [Uribe, 1991]— so that the algorithm of Figure 3 is a particular version of that more general one.

We now present the main steps in a proof of the correctness and completeness of the weakening algorithm:

Definition 17 (Solving Partial Solutions). A substitution ρ *solves* a partial solution $\langle \theta, U \rangle$ if (i) $t_i\rho \in M_C(E_i)$ for all membership constraints $t_i \in E_i$ in U, and (ii) for all variables X, $X\rho = (X\theta)\rho$.

It is simple, though tedious, to show that a substitution ρ solves an element of a set of partial solutions S if, and only if, ρ solves an element of the set S' resulting from applying any of transformations W1–W7 to S. Thus, a substitution ρ solves the set of membership constraints input to the algorithm of Figure 3 if, and only if, ρ solves some partial solution $\langle \theta_i, U_i \rangle$ in $\mathcal{S}$ at the end of Part 1. It is also easy to verify that each C_i obtained in Part 2 is a shallow SSC with no cycles. The following theorem puts everything together:

Theorem 18. *A substitution ρ solves the set $\{t_1 \in X_i, \ldots, t_n \in X_n\}$ input to the algorithm of Figure 3 if, and only if, for some output pair $\langle \theta_i, C_i \rangle$, $X\rho \in M_{C_i}(X\theta_i)$ for all $X \in Var(\{t_1, \ldots, t_n\})$.*

That is, all possible solutions to the input are represented by the output, and all the solutions represented by the output are correct: $\bigcup_i M_{C_i}(t\theta_i)$ is exactly equal

Input: A constrained equation set S/C, where C is in solved form.
Output: A set of pairs $\langle \theta_i, C_i \rangle$ where θ_i is a substitution and C_i is an SSC in solved form.

Step 1: If S is not unifiable, return $\emptyset$. Otherwise: Let Γ be an idempotent most general unifier for S. Let Θ be obtained from Γ by renaming $Var(Rng(\Gamma))$ into new variables, so that $Var(Rng(\Theta))$ and $Var(C)$ are disjoint.
$\Theta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$, where $X_i \in Var(S)$.
Step 2: Let $U = \{t_i \Subset X_i\}$, and let $\mathcal{T} = \{\langle \theta_i, C_i \rangle\}$ be the result of solving the set of membership constraints U given C, using the algorithm of Figure 3.
Step 3: Rewrite the SSC's in $\mathcal{T}$ into solved form, using the algorithm of Figure 1. If $M_{C_i}(X) = \emptyset$ for some $X \in Var(Rng(\Theta))$, remove $\langle \theta_i, C_i \rangle$ from $\mathcal{T}$.
Step 4: For each pair in $\mathcal{T}$, replace θ_i by $\theta'_i = \Theta \cdot \theta_i$ and return the new $\mathcal{T}$.

Fig. 4. Unification algorithm for constrained equation sets

to the set of all the instances of t that can belong to $M_C(exp)$ for each input membership constraint $t \Subset exp$. Note that the SSC's in the output are not necessarily in solved form —but they are always shallow. Therefore, the simplification algorithm of Section 4 and the algorithm that identifies empty variables (in Figure 2) have to be used to check whether the constrained variables output by the weakening algorithm are empty. This is included in the unification algorithm of the following section, which integrates all the algorithms presented so far.

5.2 Unifying Constrained Terms

Given a constrained equation set S/C, we want to characterize all the substitutions ρ such that $t_1\rho = t_2\rho$ for all equations $t_1 = t_2 \in S$ and $X\rho \in M_C(X)$ for all $X \in Var(S)$. Figure 4 describes the unification algorithm that does this.

Since we assume that C is in solved form and $Var(S) \subseteq Dom(C)$, the input for the weakening algorithm in Step 2 is in the required form. Thus, the solving procedure terminates, and the SSC's C_i at the end of Step 2 are all shallow. Therefore, Step 3 also terminates. The algorithm is *correct* and *complete*, as follows:

Theorem 19. *A substitution ρ solves the constrained equation set S/C input to the unification algorithm of Figure 4 if, and only if, for some pair $\langle \theta'_i, C_i \rangle$ in the output, $X\rho \in M_{C_i}(X\theta'_i)$ for all $X \in Var(S)$.*

Note that if $\emptyset$ is returned, then the input equation set has no solutions.

The following very simple example is presented by Frisch [1991] to show that a finite set of terms can have an infinite number of incomparable maximally general well-sorted unifiers. Given the sort theory

$$\Sigma = \{\forall x \; \mathrm{T}(i(x)) \rightarrow \mathrm{T}(i(s(x))), \; \mathrm{T}(i(a))\},$$

the two terms $z{:}\mathrm{T}$ and $i(s(y))$ have an infinite number of most general well-sorted unifiers, none of which is more general (with respect to Σ) than the others:

$$\left\{ \begin{array}{cc} \{y \mapsto a,\ z{:}\mathrm{T} \mapsto i(s(a))\}\,, & \{y \mapsto s(a),\ z{:}\mathrm{T} \mapsto i(s(s(a)))\}\,,\ \ldots \\ \{y \mapsto s * n(a),\ z{:}\mathrm{T} \mapsto i(s(s * n(a)))\}\,, & \ldots \end{array} \right\}$$

The following SSC in solved form represents the least model of Σ:

$$C_\Sigma = \{\, \langle \Upsilon_{Top} \equiv \top \rangle, \langle \Upsilon_T \equiv i(V) \rangle, \langle V \equiv a \sqcup s(V) \rangle \,\}$$

Unifying $i(s(y))$ and z under the constraint $C_\Sigma \cup \{\langle y \equiv \Upsilon_{Top} \rangle,\ \langle z \equiv \Upsilon_T \rangle\}$ results in $\Theta = \{y \mapsto N, z \mapsto i(s(N))\}$, where the new constraint is $C_\Sigma \cup \{\langle N \equiv V \rangle\}$. Thus, an infinite number of unifiers is represented using a single constrained substitution.

6 Related Work

Constraint logic has been extensively studied, particularly the case of constraint logic programming (CLP), where the unification procedure is replaced by one that determines the solvability of constraints over a given structure [Jaffar and Lassez, 1987]. Burckert [1990] presents a general resolution principle for constrained clauses. Chen and Hsiang [1991] propose a method for finitely representing, and unifying, structured infinite sets of terms, and show how these *recurrence domains* can be incorporated into the CLP framework. These domains are neither more nor less general than the class of tree languages generated by shallow SSC's.

The *substitutional framework* for sorted deduction presented by Frisch [1991] provides the theoretical basis for building hybrid deductive systems where monadic constraints are imposed on variables, and where a constrained variable is treated as schematic for the set of ground terms known to satisfy the constraint. If a sort theory has a least Herbrand model that can be represented using SSC's, it follows from Frisch's [1991] results that the correctness and completeness of a variety of deductive systems are preserved if SSC's are incorporated as constraints and the ordinary unification procedure is replaced by the unification algorithm for constrained terms presented here [Uribe, 1991].

Shallow SSC's are the "grammar-like" counterpart of a subclass of the *tree automata with equality tests* presented by Bogaert and Tison [1992]. These extended tree automata are used by Comon and Delor [1991] to generalize a simplification algorithm for arbitrary equational formulae with membership constraints; this algorithm was initially presented using regular tree languages as constraints. Our unification algorithm can be seen as a instance of this procedure, using a different representation for constraints and where negation is not allowed. The languages generated by SSC's correspond to Bogaert and Tison's [1992] $Rec_=$ class (i.e. are recognized by tree automata whose tests contain no inequalities). Comon and Delor point out that their basic algorithm can be generalized to any class of tree languages that has certain closure and decidability properties.

Similarly, Cohn and Frisch [1992] present a method for solving sorted unification problems under an arbitrary sort theory, provided an oracle that answers certain questions about the theory.[11] The unification algorithm we present can be seen as

[11] The properties for classes of languages identified by Comon and Delor [1991] ensure that these questions can be effectively answered for the corresponding sort theories.

an effective version of this procedure for those cases where the sort theory has a least model that can be described using a shallow SSC.

Algorithms for solving general systems of set constraints (that do not express inter-variable dependencies) are presented in [Heintze and Jaffar, 1991] and [Aiken and Wimmers, 1991]. Heintze and Jaffar [1990] present an algorithm that approximates the least model of a logic program by ignoring inter-variable dependencies. The approximation is represented as a term grammar—that is, as a linear, shallow SSC. Therefore, this approximation algorithm can be used to "pre-process" a sort theory, in those cases where the approximation is exact, so that the algorithms we present here (in a more specialized version) can be used.

The problem of finding similar approximation algorithms that express inter-variable dependencies is left open; so is that of finding efficient implementations of the algorithms presented here. Future prospects also include extending the results to allow for more expressive set constraints, particularly those that contain inequations as well as equations between sets.

Acknowledgements

Thanks to Anthony Cohn, Alan Frisch, Peter Haddawy, David Page, Christian Prehofer and Richard Scherl, with whom I had useful discussions about this work, as well as the anonymous referees for their comments. Special thanks to David Page, Richard Scherl and the Universidad de los Andes Department of Mathematics for their assistance in the completion of this paper. This research was partially supported by NASA under grant number NAG 1-613.

References

[Aiken and Wimmers, 1991] Alexander Aiken and Edward L. Wimmers. Solving systems of set constraints. Research report 8257 (#75451), IBM Almaden Research Center, 1991.

[Bogaert and Tison, 1992] Bruno Bogaert and Sophie Tison. Equality and inequality constraints on direct subterms in tree automata. In *Proc. of the* 9^{th} *Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171, Cachan, France, February 1992. Springer-Verlag.

[Bürckert, 1990] Hans-Jürgen Bürckert. A resolution principle for clauses with constraints. In Mark E. Stickel, editor, *Proc. of the* 10^{th} *International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 178–192, Kaiserslautern, W. Germany, July 1990. Springer-Verlag.

[Chen and Hsiang, 1991] Hong Chen and Jieh Hsiang. Recurrence domains: Their unification and application to logic programming. Technical report, State University of New York at Stony Brook, Stony Brook, NY, July 1991. Short version appears as *Logic Programming with Recurrence Domains*, proc. ICALP'91.

[Cohn and Frisch, 1992] Anthony G. Cohn and Alan M. Frisch. An abstract view of sorted unification. In Deepak Kapur, editor, *Proc. of the* 11^{th} *International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 178–192, Saratoga Springs, NY, June 1992. Springer-Verlag.

[Cohn, 1989] Anthony G. Cohn. Taxonomic reasoning with many-sorted logics. *Artificial Intelligence Review*, 3:89–128, 1989.

[Comon and Delor, 1991] Hubert Comon and Catherine Delor. Equational formulae with membership constraints. Rapport de Recherche 649, LRI, Université de Paris Sud, Orsay, France, March 1991. Revised version of *Equational Formulas in Order-Sorted Algebras*, proc. ICALP 90.

[Comon, 1989] Hubert Comon. Inductive proofs by specifications transformation. In *Proc. of the Third International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 76–91, Chapel Hill, April 1989. Springer-Verlag.

[Frisch, 1991] Alan M. Frisch. The substitutional framework for sorted deduction: Fundamental results on hybrid reasoning. *Artificial Intelligence*, 49:161–198, 1991.

[Heintze and Jaffar, 1990] Nevin Heintze and Joxan Jaffar. A finite presentation theorem for approximating logic programs. Research Report RC 16089 (#71415), IBM T.J. Watson Research Center, September 1990. A preliminary version appears in *Proc. of the* 17^{th} *Annual ACM Symposium on Principles of Programming Languages*, January 1990.

[Heintze and Jaffar, 1991] Nevin Heintze and Joxan Jaffar. A decision procedure for a class of herbrand set constraints. Technical Report CMU-CS-91-110, Carnegie Mellon University, February 1991. Abstract appears in *Proc. of the* 5^{th} *Annual IEEE Symposium on Logic in Computer Science*, June 1990.

[Jaffar and Lassez, 1987] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proc. of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, January 1987.

[Schmidt-Schauß, 1989] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, volume 395 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, New York, 1989.

[Uribe, 1991] Tomás E. Uribe. Sorted unification and the solution of semi-linear membership constraints. Technical Report UIUCDCS-R-91-1720, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, December 1991.