UPMAIL Technical Report No. 150
November 27, 1997

Bag and Set Unification

Evgeny Dantsin*

Steklov Institute of Mathematics at St.Petersburg
Fontanka 27, St.Petersburg 191011
Russia

Andrei Voronkov†

Computing Science Department
Uppsala University
Box 311, S-751 05 Uppsala
Sweden

email: {dantsin,voronkov}@csd.uu.se

*Supported by grants from the Swedish Royal Academy of Sciences, the Swedish Institute and INTAS/RFBR.
†Supported by a TFR grant.

Abstract

Unification in logic programming deals with tree-like data represented by terms. We extend this unification to terms that represent finite sets and bags (finite multisets). An extended unification algorithm is presented as an algorithm solving equality constraints. This algorithm runs in nondeterministic polynomial time. Thus, the unifiability problem for bags, finite sets and trees is shown to be NP-complete. Our results can be also regarded as results about handling complex values in deductive databases. Namely, our algorithm provides a procedural semantics of logic query languages for bags, finite sets and trees.

Contents

Section 1

Introduction

Logic programming languages deal with tree-like data represented by terms. Applications require to handle other kinds of data such as finite sets or bags (finite multisets). For example, this problem arises in databases: relational query languages typically deals with tuples of atomic values and extension to more complex values is required in many applications. Various kinds of complex values in databases and logic programming have been considered in (Abiteboul & Grumbach 1988, Abiteboul & Grumbach 1991, Kuper 1990, Beeri, Naqvi, Schmueli & Tsur 1991, Vadaparty 1991, Shmueli, Tsur & Zaniolo 1992, Leone & Rullo 1993, Beeri & Kornatzky 1994, Abiteboul & Beeri 1995, Kanella kis, Kuper & Revesz 1995, Grumbach & Vianu 1995, Dovier, Omodeo, Pontelli & Rossi 1996, Dantsin & Voronkov 1997*a*, Voronkov 1992, Voronkov 1997).

In this paper we extend unification, the core mechanism of logic programming, to handling bags and sets. This extension implies changes in the standard logic programming semantics, namely the following steps should be done to use bags and finite sets.

1. Instead of the Herbrand universe represented by the set of ground terms, a more complex domain of values should be introduced.
2. Unification algorithm should be changed to capture equality on this domain.

Our paper follows this pattern. In Sections 2 we introduce *the domain* $\mathcal{D}$ that contains bags, finite sets and trees. We also define equality on the domain. This domain describes untyped values, for example one can construct a bag whose members are sets and trees.

In Section 3 we briefly sketch the semantics of logic programs with complex values. We consider the model-theoretic semantics and the procedural semantics given by *constraint SLD-resolution.*

Section 4 is the main section of this paper. It describes a nondeterministic unification algorithm for $\mathcal{D}$ formalized as solving of equality constraints over $\mathcal{D}$. In Appendix we prove that this algorithm is sound, complete and runs in nondeterministic polynomial time. As a consequence, we obtain NP-completeness of the unifiability problem for $\mathcal{D}$.

In Section 5 we consider the notion of a complete set of unifiers and prove that one can always find a complete set of unifiers having at most an exponential number of unifiers. We show that there exist bag unification and finite set unification problems for which any complete set of unifiers is exponential. We also prove that in some particular cases our algorithm is optimal, i.e. it computes a complete set of minimal unifiers.

Section 6 contains several examples of application of the unification algorithm.

In Section 7 we consider some possible extensions of our results. They include: (i) the typed version of $\mathcal{D}$ and (ii) colored bags or sets.

In Section 8 we briefly sketch some further research directions. Related works are considered throughout the paper and also in this section.

There are several aspects of handling complex values that are beyond the scope of our paper. We do not consider the introduction of the object structure on $\mathcal{D}$ and we do not discuss questions like object identity. We do not consider semantics of negation.

The main contribution of our paper is a presentation of a nondeterministic polynomial time unification algorithm over $\mathcal{D}$, its modification for other domains and related complexity results. The following novelties are introduced in our paper compared to other papers in the area:

1. A unification algorithm for a domain containing bags, finite sets and trees *together* is presented. Unification algorithms for finite sets have been discussed in the literature: (Kapur & Narendran 1986) investigated flat finite sets, (Dovier et al. 1996) investigated so-called colored sets). As far as we know, unification algorithms for bags have not been described previously[1].

2. The unifiability problem for $\mathcal{D}$ is NP-complete. As a consequence, we prove that the bag unifiability and the finite set unifiability problems are NP-complete. NP-completeness of the finite set unifiability problem for flat sets (i.e. sets whose members are non-set terms) is proved in (Kapur & Narendran 1986). NP-completeness for a domain containing hereditarily finite sets (i.e. the domain allowing for sets of sets of sets etc.) is claimed in (Dovier et al. 1996). However, their paper presents no proof of membership in NP[2].

We consider related work throughout the paper and also in Section 8.

[1]Incorporating bags in a logic programming language is stated as an open problem in (Dovier et al. 1996, page 34).

[2]We cite (Dovier et al. 1996, page 10): "we refer the interested reader to (Kapur & Narendran 1986) for a similar and easily adaptable proof." (Kapur & Narendran 1986) prove NP-completeness of set unifiability for flat sets, i.e. sets whose members are non-set terms. It is hardly possible to adapt their proof to hereditarily finite sets at all.

Section 2

Bags, sets and trees

There are several possibilities to define data models that deal with bags, finite sets and trees. We shall choose a particular data model in this section and consider some variations in Section 7. For brevity we shall say "set" instead of "finite set" in the context of this data model. "Bag" is a synonym of "finite multiset".

The domain of complex values used in this paper will be denoted by $\mathcal{D}$. The definition of $\mathcal{D}$ is parametrized by a set $\mathcal{F}$ of *function symbols*. *Constants* are function symbols of arity 0. Intuitively, constants represent atomic values, like integers or strings, and function symbols of arity ≥ 1 are *tree constructors* that are used to construct complex values from existing ones. As usual, symbols in $\mathcal{F}$ are assigned positive arities.

Definition 2.1 (domain $\mathcal{D}$)
Given a set $\mathcal{F}$ of function symbols, the *domain* $\mathcal{D}$ is defined inductively as follows.

1. Any constant in $\mathcal{F}$ belongs to $\mathcal{D}$. These constants are called *atomic values.*
2. If $v_1, \ldots, v_n \in \mathcal{D}$, where $n \geq 0$, then the bag consisting of $v_1, \ldots, v_n$ belongs to $\mathcal{D}$. This bag is denoted by $\{\!|v_1, \ldots, v_n|\!\}$ and called a *bag value.*
3. If $v_1, \ldots, v_n \in \mathcal{D}$, where $n \geq 0$, then the set $\{v_1, \ldots, v_n\}$ belongs to $\mathcal{D}$. This set is called a *set value.*
4. If $f \in \mathcal{F}$ has arity $n \geq 1$ and $v_1, \ldots, v_n \in \mathcal{D}$, then the expression $f(v_1, \ldots, v_n)$ belongs to $\mathcal{D}$. This expression represents the tree whose root has n children; the root is labelled by f and the children are $v_1, \ldots, v_n$. The expression $f(v_1, \ldots, v_n)$ is called a *tree value.*

As one can see from this definition, the domain $\mathcal{D}$ consists of untyped values. For example, the set $\{\{\!|1|\!\}, \{2\}, 3, f(4)\}$ contains a bag, a set, an atomic value and a tree.

Definition 2.2 (equality on $\mathcal{D}$)
Equality on $\mathcal{D}$ is defined inductively as follows.

1. Two atomic values are equal if they coincide.
2. Two bag values $\{\!|u_1, \ldots, u_m|\!\}$ and $\{\!|v_1, \ldots, v_n|\!\}$ are equal if $m = n$ and there is a permutation ρ of the sequence $1, \ldots, m$ such that u_i is equal to $v_{\rho(i)}$ for all i.

3. Two set values $\{u_1, \ldots, u_m\}$ and $\{v_1, \ldots, v_n\}$ are equal if each u_i is equal to some v_j, and vice versa, each v_j is equal to some u_i.

4. A tree value $f(u_1, \ldots, u_m)$ is equal to a tree value $g(v_1, \ldots, v_n)$ if f coincides with g and u_i is equal to v_i for all i.

5. No other equalities hold on $\mathcal{D}$. For example, no bag value is equal to a set value or a tree value.

In order to represent elements of $\mathcal{D}$ in the query language, we introduce the corresponding notion of a term. We assume that $\{\!|\,|\!\}$ and $\{\}$ are two constants foreign to $\mathcal{F}$. They represent the empty bag and the empty set respectively. Two binary function symbols $\{\!|\,|\,|\!\}$ and $\{\,|\,\}$ are assumed to be foreign to $\mathcal{F}$ too. They are used to construct bags and sets. Namely, if a term s represents an arbitrary element of $\mathcal{D}$ and a term t represents a bag then the term $\{\!|s\,|\,t|\!\}$ represents the bag formed by appending the element corresponding to s to the bag corresponding to t. Similarly, a term $\{s\,|\,t\}$ represents the set formed by adding the element represented by s to the set represented by t.

Definition 2.3 (terms and their sorts)
All terms will be divided into three sorts: sort of bags $\mathfrak{b}$, sort of sets $\mathfrak{s}$ and the universal sort $\mathfrak{u}$. We assume that there are three kinds of variables called *bag variables*, *set variables* and *universal variables*.

1. A bag variable is a term of sort $\mathfrak{b}$. A set variable is a term of sort $\mathfrak{s}$. A universal variable is a term of sort $\mathfrak{u}$.

2. The constant $\{\!|\,|\!\}$ is a term of sort $\mathfrak{b}$. The constant $\{\}$ is a term of sort $\mathfrak{s}$. Any constant of $\mathcal{F}$ is a term of sort $\mathfrak{u}$.

3. If s is an arbitrary term and t is a term of sort $\mathfrak{b}$ then $\{\!|s\,|\,t|\!\}$ is a term of sort $\mathfrak{b}$. If s is an arbitrary term and t is a term of sort $\mathfrak{s}$ then $\{s\,|\,t\}$ is a term of sort $\mathfrak{s}$. Any expression of the form $f(t_1, \ldots, t_n)$, where $f \in \mathcal{F}$ and $t_1, \ldots, t_n$ are arbitrary terms, is a term of sort $\mathfrak{u}$.

4. No other terms can be formed. For example, $\{x\,|\,f(a)\}$ is not a well-formed term since $f(a)$ is of sort $\mathfrak{u}$.

We also introduce the partial order $\sqsubseteq$ on the terms. For any terms s, t, we write $s \sqsubseteq t$ if either t is of sort $\mathfrak{u}$ or s and t have the same sort. Terms s and t are called *sort-incompatible* if neither $s \sqsubseteq t$ nor $t \sqsubseteq s$. It is easy to see that s and t are sort-incompatible if one of them is of sort $\mathfrak{b}$ and the other $\mathfrak{s}$.

Definition 2.4 (valuation)
A mapping ν from the set of all variables to $\mathcal{D}$ is called a *valuation* if ν maps each bag variable to a bag value and each set variable to a set value. Universal variables can be mapped to any values. We extend valuations from variables to terms as follows. Let ν be a valuation and r be a non-variable term. The value $\nu(r)$ is defined inductively.

1. $\nu(\{\!|\,|\!\})$ is the bag value $\{\!|\,|\!\}$.

2. If r is $\{\!|s\,|\,t|\!\}$ and $\nu(t)$ is a bag value $\{\!|v_1, \ldots, v_n|\!\}$, then $\nu(r)$ is the bag value $\{\!|\nu(s), v_1, \ldots, v_n|\!\}$.

3. $\nu(\{\})$ is the set value $\{\}$.

4. If r is $\{s \,|\, t\}$ and $\nu(t)$ is a set value $\{v_1, \ldots, v_n\}$, then $\nu(r)$ is the set value $\{\nu(s), v_1, \ldots, v_n\}$.

5. If r is $f(v_1, \ldots, v_n)$, where $n \geq 0$, then $\nu(r)$ is the tree or atomic value $f(\nu(v_1), \ldots, \nu(v_n))$.

As one can see from this definition, the value $\nu(r)$ for a ground term r remains the same for all ν. Hence, elements of $\mathcal{D}$ can be alternatively defined as equivalence classes of ground terms by the equality relation.

Definition 2.5 (equation, solution)
By an *equation* we mean any expression $s = t$, where s, t are terms. A *solution* to an equation is a valuation ν such that $\nu(s)$ is equal to $\nu(t)$.

Section 3

Complex values in logic programs

In this section we sketch several semantics of logic programs over $\mathcal{D}$. The definitions are rather standard and can be easily generalized to domains other than $\mathcal{D}$.

The notion of a *Horn clause* is defined as usual. We shall denote a Horn clause $A_1 \wedge \ldots \wedge A_n \supset A$ by

$$A \texttt{ :- } A_1, \ldots, A_n,$$

where $n \geq 0$. We assume that atoms A_i are not equalities. A *Herbrand model* $\mathfrak{M}$ is any model such that (i) the carrier set of $\mathfrak{M}$ is $\mathcal{D}$ and (ii) each ground term is interpreted in $\mathfrak{M}$ by itself. As usual, a Herbrand model can be identified with the set of ground atoms true in this model. Thus we can redefine a Herbrand model $\mathfrak{M}$ as a set of ground non-equality atoms (meaning the set of ground non-equality atoms true in $\mathfrak{M}$). For two Herbrand models $\mathfrak{M}_1$ and $\mathfrak{M}_2$, we write $\mathfrak{M}_1 \subseteq \mathfrak{M}_2$ if the set of all ground atoms true in $\mathfrak{M}_1$ is a subset of all ground atoms true in $\mathfrak{M}_2$. Obviously, there exists the model which is minimal with respect to $\subseteq$, namely the model in which all non-equality atoms are false. This model is denoted by $\mathcal{H}$.

A *logic program over* $\mathcal{D}$ is a finite set of Horn clauses. A *Herbrand model of a logic program* $\mathcal{P}$ *over* $\mathcal{D}$ is any model of $\mathcal{P}$ that is a Herbrand model. The following statement is a straightforward generalization of the standard facts of logic programming theory (Lloyd 1987, Apt 1990):

Theorem 3.1 Any logic program over $\mathcal{D}$ has the least Herbrand model.

As usual, the least Herbrand model can be constructed using a suitable fixpoint operator. However, in this paper we limit our attention to only some semantics, not defining all possible semantics of logic programs over $\mathcal{D}$.

Consider any Herbrand model $\mathfrak{M}$. Since any member of $\mathcal{D}$ is the value of a ground term, we can mean the corresponding ground terms when we speak about members of $\mathfrak{M}$.

Example 3.2 Here is an example of a logic program over $\mathcal{D}$. This program defines the binary predicate `non_empty_intersection`. The atom `non_empty_intersection`(u, v) is true if and only if u and v are sets having non-empty intersection. The program consists of one clause:

$$\texttt{non\_empty\_intersection}(\{x \mid y\}, \{x \mid z\}).$$

There are several ways of defining a procedural semantics of logic programs over $\mathcal{D}$. The standard way is to use a modification of SLD-resolution. Instead of defining SLD-resolution using substitutions, we shall define *constraint SLD-resolution* similar to (Maher 1992, Maher 1993).

Let φ be any formula whose free variables are $x_1, \ldots, x_n$. By $\exists\varphi$ we denote the formula $\exists x_1 \ldots \exists x_n \varphi$, and similar for $\forall\varphi$. Let V be a subset of $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ be the set of variables $\{x_1, \ldots, x_n\} \setminus V$. Then $\exists_{-V}\varphi$ denotes the formula $\exists y_1 \ldots \exists y_m \varphi$. Two formulas φ_1 and φ_2 are *equivalent with respect to a set of variables* V if $\mathcal{H} \models \exists_{-V}\varphi_1 \equiv \exists_{-V}\varphi_2$. Here and in the sequel, the symbol $\models$ denotes the truth of a formula in a model. The symbol $\equiv$ is used to denote the standard logic equivalence connective. The symbol $\perp$ denotes a formula that is false in all models.

Definition 3.3 (constraint)
A *constraint* is either $\perp$ or a conjunction of equations $s_1 = t_1 \wedge \ldots \wedge s_n = t_n$ where $n \geq 0$. A constraint $\mathcal{C}$ is *satisfiable* if $\mathcal{H} \models \exists\mathcal{C}$. Two constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ are *equivalent* if they are equivalent with respect to the set of all variables, i.e. if $\mathcal{H} \models \mathcal{C}_1 \equiv \mathcal{C}_2$. A *solution* to a constraint $\mathcal{C}$ is any valuation that makes $\mathcal{C}$ true in $\mathcal{H}$.

As one can see from this definition, constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent if and only if they have the same sets of solutions. Note that if a constraint $\mathcal{C}$ is satisfiable, then $\exists\mathcal{C}$ is true in *all* Herbrand models, since the interpretation of equality is the same in all Herbrand models.

Definition 3.4 (constraint goal)
A *goal* is any conjunction $A_1 \wedge \ldots \wedge A_n$ of atoms, where $n \geq 0$, denoted $A_1, \ldots, A_n$. The *empty goal*, denoted $\Box$, is the goal with $n = 0$. A *constraint goal* is a pair $G \cdot \mathcal{C}$, where G is a goal and $\mathcal{C}$ is a constraint.

Definition 3.5 (resolvent)
A constraint goal $G \cdot \mathcal{C}$ is a *resolvent* of a constraint goal $(A_1, \ldots, A_n) \cdot \mathcal{C}'$ with respect to a logic program $\mathcal{P}$ if the following conditions are satisfied:

1. some A_i has the form $P(s_1, \ldots, s_k)$;
2. a variant of some clause in $\mathcal{P}$ has the form $P(t_1, \ldots, t_k)$ `:-` $B_1, \ldots, B_m$ such that its variables are disjoint from the variables of $(A_1, \ldots, A_n) \cdot \mathcal{C}'$;
3. G is $A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n$;
4. $\mathcal{C}$ is $\mathcal{C}' \wedge s_1 = t_1 \wedge \ldots \wedge s_k = t_k$;
5. $\mathcal{C}$ is satisfiable.

Usually, we shall simply write "$G \cdot \mathcal{C}$ is a resolvent of $G' \cdot \mathcal{C}'$, denoted $G' \cdot \mathcal{C}' \leadsto G \cdot \mathcal{C}$, when $\mathcal{P}$ and $\mathcal{D}$ are clear from the context.

Definition 3.6 (constraint resolution)
Constraint resolution derivation is any sequence of resolvents

$$G_1 \cdot \mathcal{C}_1 \leadsto G_2 \cdot \mathcal{C}_2 \leadsto \ldots \leadsto G_n \cdot \mathcal{C}_n.$$

The following fundamental fact connects the operational semantics (constraint resolution) with the model-theoretical semantics.

Theorem 3.7 Let $\mathcal{P}$ be a logic program over $\mathcal{D}$, ν be a valuation and $G(x_1, \ldots, x_n)$ be a goal whose variables are $x_1, \ldots, x_n$. The following conditions are equivalent:

1. $G(\nu(x_1), \ldots, \nu(x_n))$ is true in the least Herbrand model of $\mathcal{P}$.

2. There is a derivation $G(x_1, \ldots, x_n) \cdot \quad \rightsquigarrow \ldots \rightsquigarrow \Box \cdot \mathcal{C}(x_1, \ldots, x_n, y_1, \ldots, y_m)$ such that the constraint $\mathcal{C}(\nu(x_1), \ldots, \nu(x_n), y_1, \ldots, y_m)$ is satisfiable.

The proof of this statement is a straightforward generalization of the standard completeness proofs in logic programming and constraint logic programming (Lloyd 1987, Apt 1990, Maher 1992, Maher 1993).

Section 4

Unification algorithm

In this section we introduce a unification algorithm for $\mathcal{D}$. The algorithm is presented as a nondeterministic constraint satisfaction algorithm whose work consists of repeated applications of *transformation rules*. Such an application nondeterministically replaces a constraint by another constraint. The algorithm preserves equivalence in the following sense. Suppose that $\mathcal{C}$ is a constraint and there are n nondeterministic choices for replacing $\mathcal{C}$ by $\mathcal{C}_1, \ldots, \mathcal{C}_n$ respectively. Then $\mathcal{C}$ is equivalent to $\mathcal{C}_1 \vee \ldots \vee \mathcal{C}_n$ with respect to all variables of $\mathcal{C}$.

4.1 Some definitions and notation

Notation for bags and sets. Like logic programming notation for lists, we write $\{\!|s_1, \ldots, s_n \,|\, t|\!\}$ and $\{\!|s_1, \ldots, s_n|\!\}$ for the terms $\{\!|s_1 \,|\, \ldots \{\!|s_n \,|\, t|\!\} \ldots |\!\}$ and $\{\!|s_1 \,|\, \ldots \{\!|s_n \,|\, \{\!||\!\}|\!\} \ldots |\!\}$ respectively, and similar for sets. We use letters x, y, z, u, v, w with or without indices to denote variables. Capital letters $X, Y, \ldots$ (possibly with indices) stand for sequences of variables: for example, $\{X, Y \,|\, u\}$ and $\{X, Y\}$ are abbreviations for $\{x_1, \ldots, x_k, y_1, \ldots, y_m \,|\, u\}$ and $\{x_1, \ldots, x_k, y_1, \ldots, y_m\}$ respectively. When we use notation $\{\!|X \,|\, s|\!\}$ and $\{X \,|\, s\}$, where s is a variable, we assume that X is non-empty. The *length* of a sequence X, i.e. the number of its members, is denoted by $|X|$. If X and Y are $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ respectively, $X = Y$ stands for $x_1 = y_1, \ \ldots, \ x_n = y_n$. Sometimes, we shall use capital letters joined by the set union $\cup$, for example $X \cup Y$. In this case, we mean by such an expression the sequence obtained by appending the sequence Y to the sequence X and removing duplicates. For example, if X is x, x, y and Y is y, y, z then $X \cup Y$ denotes the sequence x, y, z.

Systems of equations. In the description of the algorithm we sometimes regard constraints of the form $s_1 = t_1 \wedge \ldots \wedge s_m = t_m$ as systems of equations $\{s_1 = t_1, \ldots, s_m = t_m\}$. In this case $\perp$ means the system which, by definition, has no solution.

Reduction of a system using $x = y$. Let x and y be variables appearing in a system S. Consider the following transformation of S. First, remove $x = y$ and $y = x$ (if any) from S. Then do the following:

1. If x and y are the same variable, do nothing.
2. Otherwise, if x, y are sort-incompatible, transform S into $\perp$.
3. Otherwise, if $y \sqsubseteq x$, replace x by y in all equations in S and add $x = y$ to S.

4. Otherwise (in this case $x \sqsubseteq y$), replace y by x in all equations in S and add $y = x$ to S.

This transformation is called *reduction using* $x = y$. For example, if we have the system pictured below on the left, reduction of the system using $z_2 = z_3$ (assuming that z_2, z_3 are universal variables) yields the system on the right:

$x_1 = \{\!\vert y_1 \mid x_2 \vert\!\}$	$x_1 = \{\!\vert y_1 \mid x_2 \vert\!\}$
$x_1 = \{\!\vert z_1, y_1 \mid u \vert\!\}$	$x_1 = \{\!\vert z_1, y_1 \mid u \vert\!\}$
$y_1 = \{z_1 \mid y_2\}$	$y_1 = \{z_1 \mid y_2\}$
$y_2 = \{z_2 \mid y_2\}$	$y_2 = \{z_3 \mid y_2\}$
$y_2 = \{z_3 \mid y_3\}$	$y_2 = \{z_3 \mid y_3\}$
$y_3 = \{z_2, z_3 \mid y_1\}$	$y_3 = \{z_3, z_3 \mid y_1\}$
$z_1 = f(z_2, z_3)$	$z_1 = f(z_3, z_3)$
	$z_2 = z_3$

Isolated equations. Let S be a system containing an equation $x = t$. This equation is called *isolated* in S if (i) $t \sqsubseteq x$ and (ii) x has exactly one occurrence in S (namely, the occurrence in the left-hand side of $x = t$).

Note that if $x = y$ belongs to S and is not isolated in S, then reduction using $x = y$ changes $x = y$ to an isolated equation $x = y$ or $y = x$.

Simple equations. We define a *bag equation* as an equation $x = \{\!\vert Y \mid z \vert\!\}$, where x and z are bag variables and Y is a sequence of variables. Similarly, $x = \{Y \mid z\}$, where x and z are set variables and Y is a sequence of variables, is called a *set equation.* We define a *simple equation* as any of the following equations: (i) a bag equation, (ii) a set equation, (iii) an equation $x = f(x_1, \ldots, x_n)$, where $n \geq 0$ and $x, x_1, \ldots, x_n$ are variables.

Lemma 4.1 Any system S can be transformed in polynomial time to a system S' consisting of simple or isolated equations such that S and S' are equivalent with respect to all variables of S.

Proof. We shall only sketch the transformation leaving details to the reader. It consists of three parts.

1. *Get rid of non-variable terms in left sides of equations* by introducing new variables. An equation $s = t$ is replaced by two equations $x = s$ and $x = t$, where x is a new variable of the universal sort.

2. *Get rid of non-simple equations $x = t$, where t is not a variable*, by introducing new variables. For example, the equation $x = \{\{\}, f(a), b\}$ is replaced by five equations

$$x = \{y, z, v \mid y\},\ y = \{\},\ z = f(u),\ u = a,\ v = b,$$

where y, z, u, v and new variables, y is of the sort $\mathfrak{s}$ and z, u, v are of the sort $\mathfrak{u}$.

3. Now any non-simple equation is an equation between two variables $x = y$. If this equation is not isolated, apply reduction by $x = y$.

$\square$

$$
\begin{aligned}
x_1 &= \{\!\!|\, y_1 \mid x_2 \,|\!\!\} \\
x_1 &= \{\!\!|\, z_1, y_1 \mid u \,|\!\!\} \\
y_1 &= \{ z_1 \mid y_2 \} \\
y_2 &= \{ z_2 \mid y_2 \} \\
y_2 &= \{ z_3 \mid y_3 \} \\
y_3 &= \{ z_2, z_3 \mid y_1 \} \\
z_1 &= f(z_2, z_3)
\end{aligned}
$$

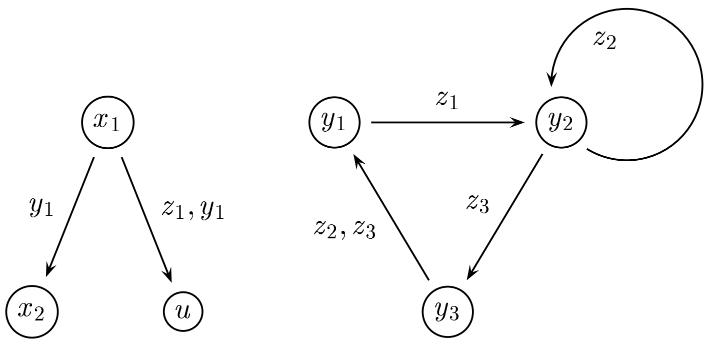


Figure 4.1: A system S and its graph $\mathcal{G}_S$

From now on we deal only with systems consisting of simple or isolated equations.

Graphs associated with systems. To illustrate the algorithm, we associate with any system S a directed graph denoted by $\mathcal{G}_S$ and called *the graph of* S. The nodes of this graph are all bag and set variables occurring in S. If S contains a bag equation $x = \{\!\!|\, Y \mid z \,|\!\!\}$ or a set equation $x = \{Y \mid z\}$, then the graph $\mathcal{G}_S$ contains an edge from x to z labelled by Y. An example of a system and its graph is given in Figure 4.1.

Final variables. We call a bag variable x *final in* S if the graph $\mathcal{G}_S$ contains no edge coming from x. We call a set variable x *final in* S if whenever $\mathcal{G}_S$ contains a path from x to another node y, it also contains a path from y to x. For example, in the system in Figure 4.1 there are two final bag variables, x_2 and u; and three final set variables, y_1, y_2 and y_3.

Bag and set cycles. We define a *bag cycle* as a bag equation $x = \{\!\!|\, Y \mid x \,|\!\!\}$ or a sequence of bag equations

$$x_1 = \{\!\!|\, Y_1 \mid x_2 \,|\!\!\},\ x_2 = \{\!\!|\, Y_2 \mid x_3 \,|\!\!\},\ \ldots,\ x_m = \{\!\!|\, Y_m \mid x_1 \,|\!\!\},$$

where $m \geq 2$ and $x_1, \ldots, x_m$ are pairwise different variables. A *set cycle* is defined similarly. For example, the system in Figure 4.1 contains exactly two cycles, namely the set cycle

$$y_1 = \{z_1 \mid y_2\},\ y_2 = \{z_3 \mid y_3\},\ y_3 = \{z_2, z_3 \mid y_1\}.$$

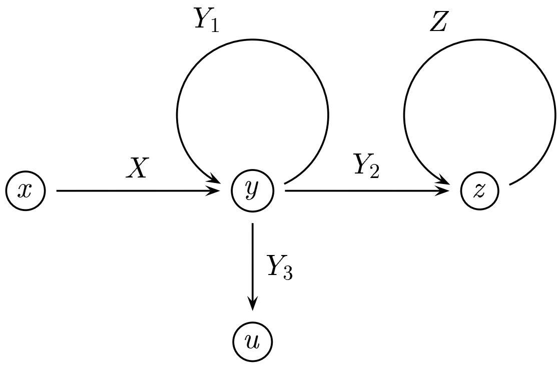
and the set cycle $y_2 = \{z_2 \mid y_2\}$.

Bag and set extensions. Assume that a system S contains a sequence of bag equations

$$y_1 = \{\!\!|\, X_1 \mid y_2 \,|\!\!\},\ y_2 = \{\!\!|\, X_2 \mid y_3 \,|\!\!\},\ \ldots,\ y_m = \{\!\!|\, X_m \mid y_{m+1} \,|\!\!\}$$

where $m \geq 1$ and y_{m+1} is a final variable. We say that the bag equation

$$y_1 = \{\!\!|\, X_1, \ldots, X_m \mid y_{m+1} \,|\!\!\}$$



$$\begin{aligned}
x &= \{X, Y_2 \mid z\}\\
x &= \{X, Y_2, Z \mid z\}\\
x &= \{X, Y_1, Y_2 \mid z\}\\
x &= \{X, Y_1, Y_2, Z \mid z\}\\
x &= \{X, Y_3 \mid u\}\\
x &= \{X, Y_1, Y_3 \mid u\}
\end{aligned}$$

Figure 4.2: Extensions

is an *extension* of the equation $y_1 = \{\!| X_1 \mid y_2 |\!\}$.

Similarly, if S contains a sequence of set equations

$$y_1 = \{X_1 \mid y_2\},\ y_2 = \{X_2 \mid y_3\}, \quad \dots, \quad y_m = \{X_m \mid y_{m+1}\},$$

where $m \geq 1$ and y_{m+1} is a final variable, then an *extension* of the set equation $y_1 = \{X_1 \mid y_2\}$ is defined as the equation

$$y_1 = \{X_1 \cup \ldots \cup X_m \mid y_{m+1}\}.$$

Figure 4.2 shows examples of set extensions: the equation $x = \{X \mid y\}$ has six extensions (in the system that consists of the equations corresponding to the edges of the given graph).

Note that if a bag equation has no extension then S has a bag cycle. Every set equation has an extension.

Reduction of a system by using $X = Y$. Let X and Y be sequences $x_1, \dots, x_n$ and $y_1, \dots, y_n$ of variables respectively. Informally, reduction using $X = Y$ is successive reductions using $x_1 = y_1, \dots, x_n = y_n$. More precisely, denote the system S by S_0 and denote the system $\{X = Y\}$ by E_0. Define S_i and E_i for $1 \leq i \leq n$ as follows. Let $u = v$ be any equation in E_{i-1}. Then S_i is obtained from S_{i-1} by reduction using $u = v$ and E_i is obtained from E_{i-1} by reduction using $u = v$ and removing $u = v$ or $v = u$. *Reduction using* $X = Y$ is defined as the transformation replacing S by S_n. For example, consider the system in Figure 4.1. Let X and Y be y_2, y_3 and y_1, y_1 respectively. Then reduction using $X = Y$ is illustrated in Table 4.1.

Correspondences between sequences. Let X and Y be sequences of variables. We define *correspondences* between X and Y, namely correspondences of two kinds called *bag correspondences* and *set correspondences*. A *bag correspondence* between X and Y is defined as an equivalence relation on $X \cup Y$ such that every equivalence class R satisfies the following condition:

> The total number of occurrences of the variables of R in the sequence X is equal to the total number of occurrences of the variables of R in the sequence Y.

E_0	S_0
	$x_1 = \{\!\{y_1 \mid x_2\}\!\}$
	$x_1 = \{\!\{z_1, y_1 \mid u\}\!\}$
$y_2 = y_1$	$y_1 = \{z_1 \mid y_2\}$
$y_3 = y_1$	$y_2 = \{z_2 \mid y_2\}$
	$y_2 = \{z_3 \mid y_3\}$
	$y_3 = \{z_2, z_3 \mid y_1\}$
	$z_1 = f(z_2, z_3)$

E_1	S_1
	$x_1 = \{\!\{y_2 \mid x_2\}\!\}$
	$x_1 = \{\!\{z_1, y_2 \mid u\}\!\}$
	$y_2 = \{z_1 \mid y_2\}$
	$y_2 = \{z_2 \mid y_2\}$
$y_3 = y_2$	$y_2 = \{z_3 \mid y_3\}$
	$y_3 = \{z_2, z_3 \mid y_2\}$
	$z_1 = f(z_2, z_3)$
	$y_1 = y_2$

E_2	S_2
	$x_1 = \{\!\{y_3 \mid x_2\}\!\}$
	$x_1 = \{\!\{z_1, y_3 \mid u\}\!\}$
	$y_3 = \{z_1 \mid y_3\}$
	$y_3 = \{z_2 \mid y_3\}$
	$y_3 = \{z_3 \mid y_3\}$
	$y_3 = \{z_2, z_3 \mid y_3\}$
	$z_1 = f(z_2, z_3)$
	$y_1 = y_2$
	$y_2 = y_3$

Table 4.1: Reduction by several equations

For example, let X and Y be x, x, y, z and x, y, y, u respectively. Then there are three bag correspondences between X and Y. The first consists of two equivalence classes $\{x, y\}$ and $\{u, z\}$, the second consists of $\{x, u\}$ and $\{y, z\}$, and the third consists of one class $\{x, y, z, u\}$. Obviously, there exists a bag correspondence between X and Y if and only if X and Y have the same length.

A bag correspondence is called *minimal* if no equivalence class can be split into proper disjoint subsets such that the resulting equivalence relation remains a bag correspondence. In the above example, the first and second correspondences are minimal and the third one is not minimal.

A *set correspondence* between X and Y is an equivalence relation on $X \cup Y$ such that for every equivalence class R, we have

R contains at least one variable of X and at least one variable of Y.

Like the case of bag correspondences, a set correspondence is called *minimal* if no equivalence class can be split into proper disjoint subsets such that the resulting equivalence relation remains a set correspondence. In this case, it means that each equivalence class contains either exactly one variable of X or exactly one variable of Y.

For example, let X and Y be x, x, y, z and x, u, v, w respectively. Then the equivalence relation consisting of classes $\{x, u\}$, $\{y, v\}$ and $\{z, w\}$ is a minimal set correspondence between X and Y. The relation consisting of $\{x, y, u\}$ and $\{z, v, w\}$ is a set correspondence but it is not minimal.

Let E be any set of equations $x_i = y_j$ where x_i is in X and y_j is in Y. By $\sim_E$ we denote is the smallest equivalence relation on $X \cup Y$ containing all pairs (x, y) such that $(x = y) \in E$. We say that E is a (minimal) bag or set correspondence between X and Y if such is $\sim_E$. For example, the set $\{x = u, y = v, z = w\}$ is a minimal set correspondence between X and Y that denote x, x, y, z and x, u, v, w respectively.

4.2 Rules

Rule 1 (Tree Decomposition)
This rule can be applied to a system S if S contains two distinct equations $z = f(x_1, \ldots, x_n)$ and $z = f(y_1, \ldots, y_n)$, where $f \in \mathcal{F}$ and $n \geq 0$. Remove the equation $z = f(x_1, \ldots, x_n)$ from S and reduce S using $x_1 = y_1, \ldots, x_n = y_n$. (This rule is close to the term decomposition rule of (Martelli & Montanari 1982)).

Rule 2 (Bag Decomposition)
The rule can be applied to a system S if the system contains bag equations $z = s$ and $z = t$, where s and t are different. If at least one of $z = s$ and $z = t$ has no extension then transform S into $\perp$ (the system contains a bag cycle). Otherwise, choose any extensions of $z = s$ and $z = t$, remove $z = s$ and $z = t$ from S, and consider two possible cases.

1. *The extensions have the forms* $z = \{\!|X \mid u|\!\}$ *and* $z = \{\!|Y \mid u|\!\}$. If $|X| \neq |Y|$ then transform S into $\perp$. Otherwise, add to S the equation $z = \{\!|Y \mid u|\!\}$. Nondeterministically generate a minimal bag correspondence E between X and Y and reduce S using E.

2. *The extensions have the forms* $z = \{\!|X \mid u|\!\}$ *and* $z = \{\!|Y \mid v|\!\}$, *where the variables* u *and* v *are different.* Nondeterministically divide X into disjoint parts X_1 and X_2 (one of them may be empty). Similarly, nondeterministically divide Y into Y_1 and Y_2. The parts X_1 and Y_1 are required to be of the same length, i.e. $|X_1| = |Y_1|$. (Informal comment: X_1 and Y_1 are intended to coincide as bags, X_2 and Y_2 are intended to have no common elements). Then S is transformed as follows.

   (a) Add the equation $z = \{\!|X_2, Y_1, Y_2 \mid w|\!\}$, where w is a new bag variable.

   (b) If X_2 is non-empty, add the equation $v = \{\!|X_2 \mid w|\!\}$. Otherwise, reduce S using $v = w$.

   (c) If Y_2 is non-empty, add the equation $u = \{\!|Y_2 \mid w|\!\}$. Otherwise, reduce S using $u = w$.

   (d) If X_1 and Y_1 are non-empty, nondeterministically generate a minimal bag correspondence E between X_1 and Y_1 and reduce S using E.

We illustrate Bag Decomposition by the following example. Assume that S consists of the equations

$$\begin{array}{ll} z = \{\!|x_1 \mid y_1|\!\} & z = \{\!|x_3, x_3 \mid y_4|\!\} \\ y_1 = \{\!|x_1, x_2 \mid y_2|\!\} & y_4 = \{\!|x_4, x_5 \mid y_5|\!\} \\ y_1 = \{\!|x_3 \mid y_3|\!\} & \end{array}$$

Bag Decomposition can transform this system in several ways (depending on nondeterministic choices in the application). One of the possible results is obtained as follows. The application starts with choosing extensions of the equations $z = \{\!|x_1 \mid y_1|\!\}$ and $z = \{\!|x_3, x_3 \mid y_4|\!\}$. Choose the extensions

$$\begin{array}{l} z = \{\!|x_1, x_1, x_2 \mid y_2|\!\} \\ z = \{\!|x_3, x_3, x_4, x_5 \mid y_5|\!\} \end{array}$$

respectively. Then case 2 of the rule can be applied. Choose x_1, x_1 as X_1 and x_3, x_3 as Y_1. This choice leads to the following system:

$$\begin{array}{ll} z = \{\!|x_2, x_3, x_3, x_4, x_5 \mid w|\!\} & y_1 = \{\!|x_3, x_2 \mid y_2|\!\} \\ y_5 = \{\!|x_2 \mid w|\!\} & y_1 = \{\!|x_3 \mid y_3|\!\} \\ y_2 = \{\!|x_4, x_5 \mid w|\!\} & y_4 = \{\!|x_4, x_5 \mid y_5|\!\} \\ & x_1 = x_3 \end{array}$$

Rule 3 (Set Decomposition)
The rule can be applied to a system S if the system contains set equations $z = s$ and $z = t$, where s and t are different. If z is a final variable, do the following. Suppose that s is $\{X \mid u\}$ and t is

$\{Y \mid v\}$. Replace $z = s$ and $z = t$ by the equation $z = \{X \cup Y \mid z\}$ and reduce the system using $z = u, z = v$. Otherwise, choose any extensions of $z = s$ and $z = t$, remove $z = s$ and $z = t$ from S, and consider two possible cases.

1. *The extensions have the forms $z = \{X \mid u\}$ and $z = \{Y \mid u\}$.* Nondeterministically divide X into disjoint parts X_1 and X_2. Similarly, divide Y into Y_1 and Y_2. If one of the parts X_1 and Y_1 is empty then the other is required to be empty too. (Informally: X_1 and Y_1 coincide as sets, X_2 and Y_2 have no common members). Consider two cases.
   (a) *Both X_1 and Y_1 are empty.* Add the equation $u = \{X_2 \cup Y_2 \mid u\}$ and reduce S using $z = u$.
   (b) *Both X_1 and Y_1 are non-empty.* Nondeterministically generate a minimal set correspondence E between X_1 and Y_1. Then transform S as follows.
      i. Add the equation $z = \{Y_1 \mid u\}$.
      ii. If at least one of X_2 and Y_2 is non-empty, add the equation $u = \{X_2, Y_2 \mid u\}$.
      iii. Reduce S using E.
2. *The extensions have the forms $z = \{X \mid u\}$ and $z = \{Y \mid v\}$, where the variables u and v are different.* Nondeterministically divide X into disjoint parts X_1 and X_2 and divide Y into Y_1 and Y_2. If one of the parts X_1 and Y_1 is empty then the other is required to be empty too. Consider two cases.
   (a) *Both X_1 and Y_1 are empty.* Add the equations

$$
\begin{aligned}
z &= \{X_2 \cup Y_2 \mid w\} \\
u &= \{Y_2 \mid w\} \\
v &= \{X_2 \mid w\}
\end{aligned}
$$

   where w is a new variable.
   (b) *Both X_1 and Y_1 are non-empty.* Transform S as follows.
      i. Add equation $z = \{X_2 \cup Y_1 \cup Y_2 \mid w\}$, where w is a new set variable.
      ii. If X_2 is non-empty, add the equation $v = \{X_2 \mid w\}$. Otherwise reduce S using $v = w$.
      iii. If Y_2 is non-empty, add the equation $u = \{Y_2 \mid w\}$. Otherwise reduce S using $u = w$.
      iv. Nondeterministically generate a minimal set correspondence E between X_1 and Y_1 and reduce S using E.

Set Decomposition is illustrated by the following example. Let S consist of the equations

$$
\begin{array}{ll}
z = \{x_1 \mid y_1\} & z = \{x_3 \mid y_2\} \\
y_1 = \{x_1, x_2, x_3 \mid y_4\} & y_2 = \{x_3 \mid y_3\} \\
 & y_2 = \{x_4, x_5 \mid y_3\}
\end{array}
$$

Choose extensions of the equations with z in their left-hand sides. Let

$$
\begin{aligned}
z &= \{x_1, x_2, x_3 \mid y_4\} \\
z &= \{x_3 \mid y_3\}
\end{aligned}
$$

be these extensions. These extensions satisfy case 2 of Set Decomposition. Choose $x_1x_2x_3$ as X_1 and x_3 as Y_1. According to case 2b we obtain

$$
\begin{array}{ll}
z = \{x_3 \mid w\} & y_2 = \{x_3 \mid w\} \\
y_1 = \{x_3, x_3, x_3 \mid w\} & y_2 = \{x_4, x_5 \mid w\} \\
y_3 = w & x_1 = x_3 \\
y_4 = w & x_2 = x_3
\end{array}
$$

Now we can apply Set Decomposition, case 1 to the two equations containing y_2. Choose x_3 as X_1 and x_4 as Y_1. We obtain

$$
\begin{array}{ll}
z = \{x_4 \mid w\} & y_2 = \{x_4 \mid w\} \\
y_1 = \{x_4, x_4, x_4 \mid w\} & w = \{x_5 \mid w\} \\
y_3 = w & x_1 = x_4 \\
y_4 = w & x_2 = x_4 \\
x_3 = x_4 &
\end{array}
$$

Rule 4 (Function Failure)
If the system S contains equations $x = f(x_1, \ldots, x_m)$ and $x = g(y_1, \ldots, y_n)$ where $m, n \geq 0$ and f and g are distinct function symbols, transform S into $\perp$.

Rule 5 (Type Failure)
If the system S contains an equation $x = t$ such that $t \not\sqsubseteq x$, transform S into $\perp$.

Definition 4.2 (solved form)
A system S of equations is said to be *in solved form* if no rule is applicable to S.

In particular, $\perp$ is in solved form.

It is easy to see that a system S is in solved form if and only if (i) it does not contain two different equations $x = s$ and $x = t$ and (ii) for any equation $x = t$ in S we have $t \sqsubseteq x$.

Definition 4.3 (transformation algorithm)
The transformation algorithm nondeterministically chooses Rules 1–5 and applies them to an input system until no rule is applicable.

Thus, the transformation algorithm is a nondeterministic algorithm that transforms any input system into a system in solved form.

Variable dependency graph. We introduce one more graph associated with a system S. The *variable dependency graph* of S is the graph whose nodes are variables occurring in S and whose edges are defined as follows. There is an edge coming from x to y if at least one of the following conditions is satisfied:

1. S contains an equation $x = f(y_1, \ldots, y_n)$, where $f \in \mathcal{F}$ and y is one of $y_1, \ldots, y_n$, ;
2. S contains an equation $x = \{\!|y_1, \ldots, y_n \mid z|\!\}$ such that y is one of $y_1, \ldots, y_n, z$;
3. S contains an equation $x = \{y_1, \ldots, y_n \mid z\}$ such that y is one of $y_1, \ldots, y_n$.

Definition 4.4 (occur-check algorithm)
The occur-check algorithm is applied to a system S in solved form. If the variable dependency graph of S has a cycle, the algorithm transforms S into $\perp$. Otherwise, the algorithm does not change S.

Definition 4.5 (unification algorithm)
The unification algorithm is the composition of the transformation algorithm and the occur-check algorithm.

The following three theorems assert main properties of the above three algorithms. The proofs of the first two theorems are given in Appendix A.

Theorem 4.6 (transformation algorithm)

1. The transformation algorithm runs in nondeterministic polynomial time.
2. Let $S_1, \ldots, S_n$ be all nondeterministic results of the application of the transformation algorithm to a system S. Then
   (a) If a valuation ν is a solution to S, then there is a solution ν' to some S_i such that ν and ν' coincide on all variables of S.
   (b) If a valuation ν' is a solution to some S_i, then it is also a solution to S.

Note that this theorem can be reformulated in terms of $S, S_1, \ldots, S_n$ in the following way. Let $y_1, \ldots, y_m$ be all variables of $S_1, \ldots, S_n$ that do not occur in S. Then S is equivalent to $\exists y_1 \ldots \exists y_m (S_1 \vee \ldots \vee S_n)$.

Theorem 4.7 (occur-check algorithm)

1. The occur-check algorithm runs in polynomial time.
2. A system S in solved form has a solution if and only if the application of the occur-check algorithm to S does not yield $\perp$.

The following theorem obviously follows from the previous two.

Theorem 4.8 (unification algorithm)

1. The unification algorithm runs in nondeterministic polynomial time.
2. Let $S_1, \ldots, S_n$ be all nondeterministic results of the application of the unification algorithm to a system S. Then
   (a) If a valuation ν is a solution to S, then there is a solution ν' to some S_i such that ν and ν' coincide on all variables of S.
   (b) If a valuation ν' is a solution to some S_i, then it is also a solution to S.

Theorem 4.9 The problem of solvability of constraints over $\mathcal{D}$ is NP-complete.

Proof. NP-hardness follows from (Kapur & Narendran 1986). NP-completeness is a straightforward consequence of the soundness and completeness of our algorithm. □

We shall prove similar results for other domains in Section 7. Application of the algorithm is illustrated by some examples in Section 6. Our algorithm can be optimized in several ways, but such optimizations are beyond the scope of our paper.

The set unification algorithm presented in this paper is slightly more sophisticated than the one of (Dovier et al. 1996). The unification algorithm of (Dovier et al. 1996) does not establish NP-completeness of the set unifiability problem: the algorithm is nondeterministic and the termination proof for any nondeterministically chosen branch uses a lexicographic well-ordering and may give branches of exponential depth. Section 5.2 contains a more detailed comparison of these algorithms.

Section 5

Unifiers

Usually, unification problems are formulated in terms of finding *unifiers*, i.e. substitutions that make two terms equal. In this chapter, we make an analysis of bag and set unification in terms of *complete sets of unifiers*. We assume basic knowledge of substitutions. We shall modify standard definitions of substitutions to $\mathcal{D}$.

Substitutions. Substitutions will be denoted by σ, θ. The term obtained from a term t by the application of a substitution θ is denoted by $t\theta$. The *empty substitution* is denoted by ε. The definition of substitutions is standard, with one additional restriction: we require $x\theta \sqsubseteq x$ for every variable x. This means that if x is a bag (respectively set) variable, then $x\theta$ is a bag (respectively set) term. We say that a substitution θ_1 is *more general* than a substitution θ_2 with respect to a set of variables V, denoted $\theta_1 \leq_V \theta_2$, if there is a substitution θ' such that $\mathcal{H} \models \forall(v\theta_1\theta' = v\theta_2)$ for every variable $v \in V$. We shall simply write $\theta_1 \leq \theta_2$ when the set of variables V is clear from the context. Concrete substitutions will be denoted by $[x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n]$.

Unifiers of terms. A substitution σ is called a *unifier* of terms s and t if $\mathcal{H} \models \forall(s\sigma = t\sigma)$. Let V be the set of all variables of s, t. A unifier σ of s and t is called *minimal* if for any unifier σ' of s and t such that $\sigma' \leq_V \sigma$, we have $\sigma \leq_V \sigma'$ too. A set Σ of substitutions is called a *complete set of (minimal) unifiers* of s and t if (i) every substitution in Σ is a (minimal) unifier of s and t; (ii) for every unifier σ' of s and t there exists $\sigma \in \Sigma$ such that $\sigma \leq_V \sigma'$.

Unifiers for systems of equations. A substitution σ is a *unifier* for a system of equations $\{s_1 = t_1, \dots, s_n = t_n\}$ if this substitution is a unifier of s_i and t_i for every i. The notions of a minimal unifier, and a (minimal) set of complete unifiers are defined similar to the corresponding notions for terms.

Complete sets of unifiers can be used in the operational semantics of logic programming with complex values instead of constraint SLD-resolution in the following way. We modify SLD-resolution to use unifiers from a complete set instead of the most general unifier, for details see (Gallier & Raatz 1989). Complete sets of unifiers provide an alternative to constraints when complete sets of unifiers have a good structure (for example, when they are finite and have a small number of members). In this section we shall investigate the structure of complete sets of unifiers.

It is well known (Baader & Siekmann 1994) that if a system S has a finite complete set of minimal unifiers consisting of n members then we have:

1. Any complete set of unifiers has at least n members.

2. Any complete set of minimal unifiers has precisely n members.

5.1 Unifiers for $\mathcal{D}$ computed by the algorithm

Lemma 5.1 Let S be a system of equations in solved form different from $\perp$ such that the variable dependency graph of S has no cycle. Then S has a complete set of unifiers consisting of one substitution.

Proof. We shall use induction on the number of equations of S. If S has no equations, then the empty substitution forms a complete set of unifiers.

Suppose that S has more than one equation. Since the variable dependency graph of S is cycle-free, there are two possibilities considered below. We show the corresponding unifiers for both cases leaving the straightforward proof to the reader.

1. S can be represented as $\{x = t\} \cup S'$ such that x does not occur in t and S' is a proper subset of S. By the induction hypothesis, there exists a complete set of unifiers for S' consisting of one substitution $\sigma = [x_1 \leftarrow s_1, \ldots, x_m \leftarrow s_m]$. The unifier for S is the substitution

$$[x \leftarrow t\sigma, x_1 \leftarrow s_1, \ldots, x_m \leftarrow s_m].$$

2. S can be represented as $E \cup S'$ such that S' is a proper subset of S and E is a set cycle

$$z_1 = \{Y_1 \,|\, z_2\}, \;\; \ldots, \;\; z_n = \{Y_n \,|\, z_1\},$$

where $z_1, \ldots, z_n$ are final set variables. By the induction hypothesis, there exists a complete set of unifiers for S' consisting of one substitution $\sigma = [x_1 \leftarrow s_1, \ldots, x_m \leftarrow s_m]$. The unifier for S is the substitution

$$[z_1 \leftarrow \{Y_1\sigma \cup \ldots \cup Y_n\sigma \,|\, w\}, \ldots, z_n \leftarrow \{Y_1\sigma \cup \ldots \cup Y_n\sigma \,|\, w\}, x_1 \leftarrow s_1, \ldots, x_m \leftarrow s_m],$$

where w is a new set variable.

$\square$

Definition 5.2 (resulting unifiers)
Let S be a system of equations. Consider all nondeterministic results of the application of the unification algorithm to S. Each of them is either $\perp$ or a system in solved form. By Lemma 5.1, such a system has a complete set of unifiers consisting of a single substitution (its construction is described in the proof of Lemma 5.1). Let σ be any of these substitutions. It follows from Theorem 4.8 that the restriction of σ to the variables of S is a unifier for S. We call such a unifier a *resulting unifier* for S.

Theorem 5.3 For any system S, the set of all resulting unifiers for S is a finite complete set of unifiers and its cardinality is at most exponential in the size of S.

Proof. The completeness of the set of resulting unifiers is a consequence of the completeness part of Theorem 4.8 on the unification algorithm. The rest follows from the running time part of the same theorem and Lemma 5.1. Since the unification algorithm runs in nondeterministic polynomial time, the number of all possible results is at most exponential in the input size. Every result gives at most one resulting unifier, therefore the number of resulting unifiers does not exceed an exponent. □

Example 5.4 We give an example when the algorithm computes a non-minimal unifier. Consider the system consisting of four equations

$$z_1 = f(x),\ z_1 = f(y),\ z_2 = \{\!|x, u \,|\, w|\!\},\ z_2 = \{\!|y, v \,|\, w|\!\}.$$

If we start with Tree Decomposition using the first two equations, we obtain

$$z_1 = f(y),\ x = y,\ z_2 = \{\!|y, u \,|\, w|\!\},\ z_2 = \{\!|y, v \,|\, w|\!\}.$$

Now we have to apply Bag Decomposition. The only minimal bag correspondence between y, u and y, v is $u = v$, so we obtain

$$z_1 = f(y),\ x = y,\ z_2 = \{\!|y, v \,|\, w|\!\},\ u = v.$$

Thus, we obtain the only unifier

$$[z_1 \leftarrow f(y), x \leftarrow y, z_2 \leftarrow \{\!|y, v \,|\, w|\!\}, u \leftarrow v].$$

If we would start with Bag Decomposition first, we can select the minimal bag correspondence $x = v, y = u$ which would have resulted in

$$z_1 = f(v),\ z_1 = f(u),\ z_2 = \{\!|v, u \,|\, w|\!\},\ x = v,\ y = u.$$

After Tree Decomposition we obtain the unifier

$$[z_1 \leftarrow f(u), x \leftarrow u, z_2 \leftarrow \{\!|u, u \,|\, w|\!\}, x \leftarrow u, y \leftarrow u, v \leftarrow u],$$

which is not minimal.

This example also shows that resulting unifiers may depend on the order of the application of inference rules. Now we show examples (both for bags and sets) when every complete set has at least an exponential number of unifiers.

Example 5.5 Consider the equation $\{\!|x_1, \ldots, x_n|\!\} = \{\!|y_1, \ldots, y_n|\!\}$, where $x_1, \ldots, x_n, y_1, \ldots, y_n$ are pairwise different variables. For every permutation ρ of the set $\{1, \ldots, n\}$, define the substitution σ_ρ as $[x_1 \leftarrow y_{\rho(1)}, \ldots, x_n \leftarrow y_{\rho(n)}]$. One can prove that any minimal unifier is equivalent to one of σ_ρ. It is also easy to see that for two different permutations ρ_1, ρ_2 the substitutions $\sigma_{\rho_1}, \sigma_{\rho_2}$ are incomparable with respect to $\leq_V$, where V is $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$. Hence, the set of all substitutions σ_ρ is a complete set of minimal unifiers. This set has $n!$ members. Hence, any complete set of unifiers for this equation has at least $n!$ members.

Example 5.6 Consider the equation $\{x_1, x_2\} = \{y_1, \ldots, y_n\}$, where $n > 1$ and $x_1, x_2, y_1, \ldots, y_n$ are pairwise different variables. For every non-empty proper subset R of the set $\{1, \ldots, n\}$, define the substitution σ_R such that

$$\sigma_R(y_i) = \begin{cases} x_1, & \text{if } y_i \in R \\ x_2, & \text{if } y_i \notin R \end{cases}$$

One can prove that any minimal unifier is equivalent to one of σ_R. It is also easy to see for two different subsets R_1, R_2 the substitutions $\sigma_{R_1}, \sigma_{R_2}$ are incomparable with respect to $\leq_V$, where V is $\{x_1, x_2, y_1, \ldots, y_n\}$. Hence, the set of all substitutions σ_R is a complete set of minimal unifiers. This set has $2^n - 2$ members. Hence, any complete set of unifiers for this equation has at least $2^n - 2$ members.

5.2 Comparison with Dovier et.al.

Although our algorithm has much in common with the one described in (Dovier et al. 1996), it has some essential differences. First, we show that our algorithm is "optimal", at least for a certain kind of equations.

Lemma 5.7 Consider an equation $\{x_1, \ldots, x_n\} = \{y_1, \ldots, y_m\}$, where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are universal variables, not necessarily distinct. Then the set of resulting unifiers for this equation is a complete set of minimal unifiers.

Proof. Denote the set of resulting unifiers by Ξ. By Theorem 5.3, the set Ξ is complete. In order to show that every unifier in Ξ is minimal, it is enough to prove that any two different substitutions in Ξ are incomparable with respect to $\leq$.

Let $\sigma_1, \sigma_2 \in \Xi$ be two different substitutions. According to the algorithm, σ_1, σ_2 are most general solutions for two different minimal set correspondences E_1, E_2 between $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$. Since E_1, E_2 are different and minimal, there are variables z, u such that $z\sigma_1$ coincides with $u\sigma_1$, but $z\sigma_2$ and $u\sigma_2$ are different. Then $\sigma_2 \not\leq \sigma_1$. Likewise, $\sigma_1 \not\leq \sigma_2$. □

Lemma 5.8 Consider an equation $\{\!|x_1, \ldots, x_n|\!\} = \{\!|y_1, \ldots, y_n|\!\}$, where $x_1, \ldots, x_n, y_1, \ldots, y_n$ are universal variables, not necessarily distinct. Then the unification algorithm returns a complete set of minimal unifiers.

Proof. Similar to Lemma 5.7. □

In some case it is possible to calculate the cardinality of a complete set of minimal unifiers. We shall consider some simple set unification problems and compare the numbers of unifiers computed by our algorithm and the algorithm of (Dovier et al. 1996).

Lemma 5.9 Let X and Y be sequences consisting of m and n variables respectively, all pairwise different. Then there exists exactly $S_{m,n}$ different minimal set correspondences between X and Y, where $S_{m,n}$ is defined for $m, n \geq 1$ as follows:

$$S_{m,n} = \begin{cases} 1, & \text{if } m = 1; \\ 1, & \text{if } n = 1; \\ \sum_{k=1}^{n-1} \binom{n}{k} \cdot S_{m-1,n-k} + n \cdot \sum_{k=1}^{m-2} \binom{m-1}{k} \cdot S_{m-k-1,n-1} & \text{if } m, n \geq 2. \end{cases}$$

Proof. The case when $m = 1$ or $n = 1$ is obvious.

Suppose $m, n \geq 2$. Take an arbitrary element of X, say x, and consider any minimal set correspondence between X and Y. There are two possibilities:

1. *The variable x occurs in an equivalence class that contains no other variable of X.* In this case the equivalence class of x can be any proper subset of Y. If the subset of Y has k members, we have $S_{m-1,n-k}$ possibilities for a minimal set correspondence between the rest of the variables. This gives $\sum_{k=1}^{n-1} \binom{n}{k} \cdot S_{m-1,n-k}$ possibilities.

2. *The variable x occurs in an equivalence class that contains some other variables of X.* In this case the equivalence class of x contains exactly one variable of Y. By a similar consideration one can easily check that there are $\sum_{k=1}^{m-2} n \cdot \binom{m-1}{k} \cdot S_{m-k-1,n-1}$ possibilities for this case.

□

Combining Lemmas 5.7 and 5.9, we obtain the following.

Lemma 5.10 Let X and Y be sequences consisting of m and n variables respectively, all pairwise different. Then the unification algorithm computes exactly $S_{m,n}$ resulting unifiers for the equation $\{X\} = \{Y\}$.

The following statement can be proved similar to Lemma 5.9.

Lemma 5.11 Let X and Y be sequences consisting of m and n variables respectively, all pairwise different. Then the number $D_{m,n}$ of different set correspondences between X and Y can be calculated as

$$D_{m,n} = \begin{cases} 1, & \text{if } m = 1; \\ 1, & \text{if } n = 1; \\ 1 + \sum_{k=0}^{m-2} \sum_{l=1}^{n-1} \binom{m-1}{k} \cdot \binom{n}{l} \cdot D_{m-k-1,n-l} & \text{if } m, n \geq 2. \end{cases}$$

The algorithm of (Dovier et al. 1996) on the input $\{x_1, \ldots, x_m\} = \{y_1, \ldots, y_n\}$ returns $D_{m,n}$ unifiers that are most general unifiers for all set correspondences between $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$.

The numbers $S_{m,n}$ and $D_{m,n}$ for $m, n \leq 10$ are given in Figure 5.1[1]. As one can see from the tables, the numbers are large for already small values of m and n, which shows the limited practical possibility of computing a complete set of unifiers.

[1]These numbers have been calculated by a program written in C using the package `gmp` for arbitrarily large integers.

$S_{m,n}$									
n \ m	2	3	4	5	6	7	8	9	10
2	2								
3	6	15							
4	14	48	184						
5	30	165	680	2,945					
6	62	558	2,664	13,080	63,756				
7	126	1,827	11,032	59,605	320,292	1,748,803			
8	254	5,820	46,904	281,440	1,663,248	9,791,824	58,746,304		
9	510	18,177	200,232	1,379,745	8,906,544	56,499,849	361,679,040	2,361,347,073	
10	1,022	56,010	849,160	6,970,400	49,148,100	336,192,220	2,290,065,200	15,803,119,440	111,310,111,900

$D_{m,n}$									
n \ m	2	3	4	5	6	7	8	9	10
2	3								
3	7	25							
4	15	79	339						
5	31	241	1,351	6,721					
6	63	727	5,235	31,831	179,643				
7	127	2,185	20,119	147,841	979,567	6,166,105			
8	255	6,559	77,379	682,951	5,264,595	37,852,039	262,308,819		
9	511	19,681	298,951	3,160,321	28,187,431	229,876,081	1,784,037,031	1,347,127,4401	
10	1,023	59,047	1,161,555	14,703,031	151,254,123	1,392,595,807	12,036,511,635	100,285,059,751	818,288,740,923

Figure 5.1: Numbers $S_{m,n}$ and $D_{m,n}$ for $2 \leq m, n \leq 10$

Section 6

Examples

We illustrate the above unification algorithm by simple examples.

Example 6.1 Consider the following problem: find bags u and v such that the addition of 0 to u and the addition of 1 to v yield the same bag. The corresponding system S consists of one equation $\{\!|0 \mid u|\!\} = \{\!|1 \mid v|\!\}$. The example is illustrated in Figure 6.1.

First, we transform S into the system of simple equations shown in part (a) of the figure. The only rule that we can apply is Bag Decomposition, case 2. We have X and Y consisting of x_0 and x_1 respectively. Consider the nondeterministic division such that X_1 and Y_1 are empty and X_2 and Y_2 are x_0 and x_1 respectively. Then we obtain the system in solved form shown in part (b) of the figure. Its variable dependency graph shown in part (c) is cycle-free. The corresponding unifier θ can be obtained as explained in the proof of Lemma 5.1:

$$\theta = [z \leftarrow \{\!|0, 1 \mid w|\!\},\ v \leftarrow \{\!|0 \mid w|\!\},\ u \leftarrow \{\!|1 \mid w|\!\},\ x_0 \leftarrow 0,\ x_1 \leftarrow 1].$$

If we restrict this unifier to the initial set of variables, we obtain a resulting unifier

$$\theta = [v \leftarrow \{\!|0 \mid w|\!\},\ u \leftarrow \{\!|1 \mid w|\!\}].$$

Another possibility for a nondeterministic choice is when X_1 is x_0, Y_1 is x_1 and X_2, Y_2 are empty. After reductions, including reduction by $x_0 = x_1$ we obtain a system containing $x_0 = 0$ and $x_0 = 1$. Tree Decomposition reduces this system to $\bot$. Hence, our algorithm gives no other solutions.

Example 6.2 Consider the logic program defining the relation `non_empty_intersection` that holds on sets having non-empty intersection. The program is defined by one fact

$$\texttt{non\_empty\_intersection}(\{x \mid u\}, \{x \mid v\}).$$

Suppose that the query to this program is

$$\texttt{non\_empty\_intersection}(\{\texttt{database}, \texttt{security}, \texttt{integrity}\},\ \{\texttt{security}, \texttt{CIA}, \texttt{KGB}\}).$$

Constraint SLD-resolution gives the following system of equations:

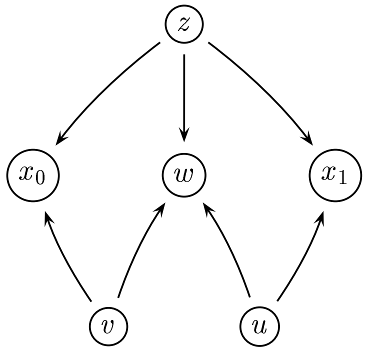
$$\{\texttt{database}, \texttt{security}, \texttt{integrity}\} = \{x \mid u\}$$
$$\{\texttt{security}, \texttt{CIA}, \texttt{KGB}\} = \{x \mid v\}.$$

$$\begin{aligned} z &= \{\!|x_0 \,|\, u|\!\} \\ z &= \{\!|x_1 \,|\, v|\!\} \\ x_0 &= 0 \\ x_1 &= 1 \end{aligned}$$

(a)

$$\begin{aligned} z &= \{\!|x_0, x_1 \,|\, w|\!\} \\ v &= \{\!|x_0 \,|\, w|\!\} \\ u &= \{\!|x_1 \,|\, w|\!\} \\ x_0 &= 0 \\ x_1 &= 1 \end{aligned}$$

(b)



(c)

Figure 6.1:

Transformation into a system of simple equations gives the following system:

$$\begin{array}{ll} y = \{x \,|\, u\} & z = \{x \,|\, v\} \\ y = \{y_1, y_2, y_3 \,|\, w\} & z = \{z_1, z_2, z_3 \,|\, w\} \\ y_1 = \texttt{database} & z_1 = \texttt{security} \\ y_2 = \texttt{security} & z_2 = \texttt{CIA} \\ y_3 = \texttt{integrity} & z_3 = \texttt{KGB} \\ w = \{\} & \end{array}$$

Since there are two equations with y in their left-hand sides, one can apply Set Decomposition (case 2), where X stands for x and Y stands for y_1, y_2, y_3. The sequence Y can be divided into parts Y_1 and Y_2 and we consider the case when Y_1 consists of one variable y_2 and Y_2 consists of two variables y_1, y_3. Then X_1 must be x and X_2 must be empty. We obtain the following system:

$$\begin{array}{ll} y = \{y_1, y_2, y_3 \,|\, w_0\} & z = \{y_2 \,|\, v\} \\ u = \{y_1, y_3 \,|\, w_0\} & z = \{z_1, z_2, z_3 \,|\, w_0\} \\ y_1 = \texttt{database} & z_1 = \texttt{security} \\ y_2 = \texttt{security} & z_2 = \texttt{CIA} \\ y_3 = \texttt{integrity} & z_3 = \texttt{KGB} \\ w_0 = \{\} & x = y_2 \\ w = w_0. & \end{array}$$

Now we can only apply the same rule (case 2 again) once more. Consider the case when z_1, z_2, z_3 is divided into z_1 and z_2, z_3. Then we obtain the following system:

$$\begin{array}{ll}
y = \{y_1, z_1, y_3 \mid w_1\} & z = \{z_1, z_2, z_3 \mid w_1\} \\
u = \{y_1, y_3 \mid w_1\} & v = \{z_2, z_3 \mid w_1\} \\
y_1 = \texttt{database} & z_1 = \texttt{security} \\
y_2 = z_1 & z_2 = \texttt{CIA} \\
y_3 = \texttt{integrity} & z_3 = \texttt{KGB} \\
w_1 = \{\} & x = z_1 \\
w = w_1 & w_0 = w_1.
\end{array}$$

This system is in solved form and its variable dependency graph is cycle-free. The corresponding unifier (restricted to the variables of the initial system) is

$$[x \leftarrow \texttt{security},\ u \leftarrow \{\texttt{database}, \texttt{integrity}\},\ v \leftarrow \{\texttt{CIA}, \texttt{KGB}\}].$$

Example 6.3 Let us redefine the `non_empty_intersection` predicate for bags instead of sets:

$$\texttt{non\_empty\_intersection}(\{\!|x \mid u|\!\}, \{\!|x \mid v|\!\}).$$

Consider the following query (find bags x_0 and x_1 such that the addition of 0 to x_0 and the addition of 1 to x_1 give bags with non-empty intersection).

$$\texttt{non\_empty\_intersection}(\{\!|0 \mid x_0|\!\}, \{\!|1 \mid x_1|\!\}).$$

Constraint SLD-resolution yields the following set of equations:

$$\begin{array}{l}
\{\!|0 \mid x_0|\!\} = \{\!|x \mid u|\!\} \\
\{\!|1 \mid x_1|\!\} = \{\!|x \mid v|\!\}.
\end{array}$$

Transformation into a system of simple equations gives

$$\begin{array}{ll}
z_0 = \{\!|x \mid u|\!\} & z_1 = \{\!|x \mid v|\!\} \\
z_0 = \{\!|y_0 \mid x_0|\!\} & z_1 = \{\!|y_1 \mid x_1|\!\} \\
y_0 = 0 & y_1 = 1.
\end{array}$$

Choose two equations with the same variable in their left-hand sides. Let these equations be $z_0 = \{\!|x \mid u|\!\}$ and $z_0 = \{\!|y_0 \mid x_0|\!\}$. They satisfy case 2 of Bag Decomposition. Nondeterministically choose x as X_2 and y_0 as Y_2 (the sequences X_1 and Y_1 are empty). Replace these two equations by three equations according to the rule. Handle the equations with z_1 in a similar way. We obtain the following system:

$$\begin{array}{ll}
z_0 = \{\!|x, y_0 \mid w_0|\!\} & z_1 = \{\!|x, y_1 \mid w_1|\!\} \\
x_0 = \{\!|x \mid w_0|\!\} & x_1 = \{\!|x \mid w_1|\!\} \\
u = \{\!|y_0 \mid w_0|\!\} & v = \{\!|y_1 \mid w_1|\!\} \\
y_0 = 0 & y_1 = 1.
\end{array}$$

The corresponding unifier restricted to the variables of the initial system is

$$[x_0 \leftarrow \{\!|x \mid w_0|\!\},\ x_1 \leftarrow \{\!|x \mid w_1|\!\}].$$

Consider another possibility for the split of X and Y. Let X_1 be x and Y_1 be y_0. Application of case 2 gives

$$\begin{array}{ll} z_0 = \{\!|y_0 \,|\, w_0|\!\} & z_1 = \{\!|y_0 \,|\, v|\!\} \\ x_0 = w_0 & z_1 = \{\!|y_1 \,|\, x_1|\!\} \\ y_0 = 0 & y_1 = 1 \\ u = w_0 & x = y_0. \end{array}$$

Applying Bag Decomposition again with y_0 as X_2 and y_1 as Y_2, we obtain

$$\begin{array}{ll} z_0 = \{\!|y_0 \,|\, w_0|\!\} & z_1 = \{\!|y_0, y_1 \,|\, w_1|\!\} \\ x_0 = w_0 & \\ y_0 = 0 & y_1 = 1 \\ u = w_0 & x = y_0 \\ x_1 = \{\!|y_0 \,|\, w_1|\!\} & v = \{\!|y_1 \,|\, w_1|\!\}. \end{array}$$

This system gives us the unifier $[x_1 \leftarrow \{\!|0 \,|\, w_1|\!\}]$.

Section 7

Other domains for bags and finite sets

In this section we consider other possibilities for representing bags and finite sets in logic programming and deductive databases, present corresponding changes in our algorithm and prove NP-completeness for other domains.

In our exposition, the universe of bags, finite sets and trees was sorted, but in a very weak way. We had two aims which resulted in the three-sorted signature:

1. To have untyped bags, finite sets and trees, for example the same bag can contain bag, finite set and tree members at the same time;

2. To have ordinary bags and finite sets (see the definition of $\mathcal{D}$). In particular, our sets satisfy *extensionality*: two sets are equal if and only if they have the same elements.

By dropping the first requirement, we obtain a *typed universe* described below in Section 7.2. By dropping the second requirement, we obtain a *universe of colored bags and colored sets* described below in Section 7.3. We show how to modify our algorithm to deal with these two universes.

7.1 Minimal domains for which unifiability is NP-complete

In order to show that NP-completeness holds for these domains, we first present two NP-hardness proofs for "very small" domains. We call a tree value *flat* if it has the form $f(a_1, \ldots, a_n)$, where all a_i are atomic value.

Lemma 7.1 The unifiability problem is NP-hard already for the subdomain of $\mathcal{D}$ consisting of sets of flat terms.

Proof. The proof is similar to the one given in (Kapur & Narendran 1986).

We shall use reduction of the 3-SAT problem which can be described as follows. A *literal* is either $x = \mathfrak{t}$ or $x = \mathfrak{f}$. A *3-clause* is a multiset consisting of three literals. An *instance* of the 3-SAT problem is any finite multiset M of 3-clauses. The goal is to find the assignment of $\mathfrak{t}, \mathfrak{f}$ to variables in M so that every clause contains either $\mathfrak{t} = \mathfrak{t}$ or $\mathfrak{f} = \mathfrak{f}$.

In order to encode 3-SAT, we shall represent any 3-clause C of the form $v_1 = c_1, v_2 = c_2, v_3 = c_3$ by the term $f(v_1, c_1, v_2, c_2, v_3, c_3)$, denoted t_C. Denote by T the set of all ground terms having one of the following forms:

$$
\begin{array}{l}
f(\mathfrak{t}, \mathfrak{t}, \_, \_, \_, \_) \\
f(\mathfrak{f}, \mathfrak{f}, \_, \_, \_, \_) \\
f(\_, \_, \mathfrak{t}, \mathfrak{t}, \_, \_) \\
f(\_, \_, \mathfrak{f}, \mathfrak{f}, \_, \_) \\
f(\_, \_, \_, \_, \mathfrak{t}, \mathfrak{t}) \\
f(\_, \_, \_, \_, \mathfrak{f}, \mathfrak{f})
\end{array}
$$

where each occurrence of $\_$ denotes either $\mathfrak{t}$ or $\mathfrak{f}$ (there are 56 such terms). It is easy to see that the assignment of $\mathfrak{t}, \mathfrak{f}$ to variables of a clause C makes C true if and only if $t_C \in T$.

Let S be any instance of the 3-SAT problem. Consider the set unification problem

$$\{t_C \mid C \in S\} \cup T = T.$$

It is easy to check that solutions to this equation are precisely all valuations that evaluate all clauses in S to $\mathfrak{t}$. Note that this construction works both for $\mathcal{D}$ and the subdomain of $\mathcal{D}$ consisting of sets of flat terms. $\Box$

Note that this subdomain of $\mathcal{D}$ is in some sense a minimal domain for which unifiability is NP-hard. Indeed, unifiability for tree terms is (almost) linear (Paterson & Wegman 1978, Martelli & Montanari 1982) and unifiability for terms consisting of atomic values can be made in polynomial time (we leave details to the reader).

Lemma 7.2 The unifiability problem is NP-hard already for the subdomain of $\mathcal{D}$ consisting of bags of flat terms.

Proof. We shall slightly modify the proof of Lemma 7.1. Let S be any instance set of the 3-SAT problem having k members. The proof of Lemma 7.1 does not work immediately for sets replaced by bags since the bags on the left and on the right of the equation would differ in length. So we shall add some terms to both sides to make the same number of members.

Consider the bag unification problem $s = t$ of the following form. The term t is a bag consisting of k copies of T. The term s is the bag consisting of all terms t_C for $C \in S$ plus $55k$ terms of the form $f(\_, \_, \_, \_, \_, \_)$, where all occurrences of $\_$ denote pairwise different variables. The rest of the proof is similar. $\Box$

Again, this subdomain is a minimal domain for which unifiability is NP-hard.

7.2 A typed universe

We shall present a typed universe for complex values containing bags and finite sets following (Abiteboul & Beeri 1995). The domain of (Abiteboul & Beeri 1995) does not contain bags but we add them in a natural way.

We assume to have a set of *domain names* $\widehat{D_1}, \widehat{D_2}, \ldots$ and an infinite set of *attributes* $A_1, A_2, \ldots$ The domain names are associated with *domains* $D_1, D_2, \ldots$, i.e. non-empty sets whose elements are called *atomic values*.

Complex values are constructed from atomic ones using constructors. A type is associated with each value. We shall denote the set containing all values by $\mathcal{T}$.

Definition 7.3 *Types* and *values* for $\mathcal{T}$ are defined as follows:

1. If $\widehat{D}$ is a domain name, then $\widehat{D}$ is an *atomic* type. For each $a \in D$, the element a is a value of this type.

2. If $T_1, \ldots, T_n$, where $n \geq 0$, are types and $A_1, \ldots, A_n$ are distinct attributes, then $[A_1 : T_1, \ldots, A_n : T_n]$ is a *tuple* type. If $v_1, \ldots, v_n$ are values of types $T_1, \ldots, T_n$, respectively, then $[A_1 : v_1, \ldots, A_n : v_n]$ is a value of the type.

3. If T is a type, then $\{\!|T|\!\}$ is a bag type. Any finite multiset of values of type T is a value of type $\{\!|T|\!\}$.

4. If T is a type, then $\{T\}$ is a set type. Any finite set of values of type T is a value of type $\{T\}$.

The notion of *term* for $\mathcal{T}$ is defined similar to the one for $\mathcal{D}$ with the following changes.

1. Since the atomic domains are typed, the corresponding constant of the language are also typed.

2. We have *tuple terms* $[A_1 : t_1, \ldots, A_n : t_n]$ made of terms $t_1, \ldots, t_n$ with the natural interpretation.

3. Bag and set constructors satisfy the natural restrictions on the types of their arguments.

We show how to modify our unification algorithm to $\mathcal{T}$. To this end, we define an *embedding* of $\mathcal{T}$ into $\mathcal{D}$. We first define a mapping from terms of $\mathcal{T}$ to terms of $\mathcal{D}$.

We assume that constants of $\mathcal{T}$ are also constants in $\mathcal{D}$ and that symbols $A_1, A_2, \ldots$ denoting attributes in $\mathcal{T}$ are constants in $\mathcal{D}$. Let f be any binary function symbol in $\mathcal{F}$. For any term t of $\mathcal{T}$ we define its image $t_{\mathcal{D}}$ which is a term of $\mathcal{D}$, in the following way:

1. If c is any constant, then $c_{\mathcal{D}}$ is c.

2. $[A_1 : t_1, \ldots, A_n : t_n]_{\mathcal{D}}$ is $\{\!|f(A_1, t_{1\mathcal{D}}), \ldots, f(A_n, t_{n\mathcal{D}})|\!\}$.

3. $\{\!|s \,|\, t|\!\}_{\mathcal{D}}$ is $\{\!|s_{\mathcal{D}} \,|\, t_{\mathcal{D}}|\!\}$.

4. $\{s \,|\, t\}_{\mathcal{D}}$ is $\{s_{\mathcal{D}} \,|\, t_{\mathcal{D}}\}$.

We define the *unification algorithm for $s = t$ and $\mathcal{T}$* as the unification algorithm for $s_{\mathcal{D}} = t_{\mathcal{D}}$ and $\mathcal{D}$ without occur check. We leave it to the reader to see that the algorithm is sound, correct, and respects the types of terms. It is also easy to see that, because of the type structure, occur check is never needed.

In order to modify the algorithm to work directly on terms of $\mathcal{T}$, we should make only one change for tuple terms. When we have an equation $[A_1 : x_1, \ldots, A_n : x_n] = [A_1 : y_1, \ldots, A_n : y_n]$, we behave similar to Tree Decomposition: we remove this equation and make reduction using $x_1 = y_1, \ldots, x_n = y_n$.

Theorem 7.4 The following statements hold for unifiability for the domain $\mathcal{T}$.

1. The unifiability problem is NP-complete.

2. There always exists a complete set of minimal unifiers whose cardinality is at most exponential.

3. There are examples where the cardinality of every complete set of unifiers is exponential.

The same results holds for the subdomain of $\mathcal{T}$ obtained by removing all bags and bag types (and similar for sets and set types).

Proof. From the above informal presentation of unification algorithm for $\mathcal{T}$ it follows that the unifiability problem is in NP.

In order to show that the problem is NP-hard (also in absence of bags or sets), we describe how to modify Lemmas 7.1 and 7.2 to also hold for $\mathcal{T}$. Consider only Lemma 7.1. We introduce the atomic domain $D = \{\mathsf{t}, \mathsf{f}\}$ and consider the following type:

$$\{[A_1 : \widehat{D}, \dots, A_6 : \widehat{D}]\},$$

where A_1–A_6 are any distinct attributes. By inspecting the proof of Lemma 7.1 one can see that the sets used in this proof are of this type. This proves part 1 of the theorem.

It is easy to check that the resulting unifiers constructed as in Lemma 5.1 form a complete set of unifiers for $\mathcal{T}$. This proves part 2. Examples 5.5 and 5.6 also hold for $\mathcal{T}$. This proves part 3. □

7.3 A colored universe

In this section we consider a domain for bags, finite sets and trees that is completely type-free. This domain extends to bags the domain of *colored sets* introduced in (Dovier et al. 1996).

Having a set constructor and other function symbol in a completely type-free language causes the following problem. Since the language is type-free, we have to admit terms of the form $\{s \,|\, t\}$, where t is an arbitrary term, not necessarily denoting a set. Consider any ground term $\{s_1, \dots, s_n \,|\, t\}$, where t does not denote a set. In (Dovier et al. 1996) such a term t is called a *color* and the corresponding set value is called a *colored set* with elements $s_1, \dots, s_n$ and color t. Two colored sets are equal when they have the same colors and same elements. Similar to defining colored sets, we may also define colored bags. In this case the corresponding part of Definition 2.1 of $\mathcal{D}$ should be replaced by

2. If $v_1, \dots, v_n \in \mathcal{D}$, where $n \geq 1$, and t is either $\{\!|\,|\!\}$ or any non-bag value then $\{\!| v_1, \dots, v_n \,|\, t |\!\}$ belongs to $\mathcal{D}$ and is called a *(colored) bag value.*

3. If $v_1, \dots, v_n \in \mathcal{D}$, where $n \geq 1$, and t is either $\{\}$ or any non-set value then $\{v_1, \dots, v_n \,|\, t\}$ belongs to $\mathcal{D}$ and is called a *(colored) set value.*

A part of Definition 2.2 of equality should also be replaced by

2. Two colored bag values $\{\!| u_1, \dots, u_m \,|\, s |\!\}$ and $\{\!| v_1, \dots, v_n \,|\, t |\!\}$ are equal if $m = n$, s is equal to t, and there is a permutation ρ of the set $\{1, \dots, m\}$ such that u_i is equal to $v_{\rho(i)}$ for all i.

3. Two colored set values $\{u_1, \dots, u_m \,|\, s\}$ and $\{v_1, \dots, v_n \,|\, t\}$ are equal if s is equal to t, each u_i is equal to some v_j, and vice versa, each v_j is equal to some u_i, and s is equal to t.

To modify the unification algorithm for colored bags and sets, no serious changes are needed.

Section 8

Conclusion, related work and future research

We have already considered some related works, especially (Dovier et al. 1996), throughout the paper. In this section we also briefly mention other related work.

Our results can be used to implement logic programs and deductive databases working with bags, finite sets and trees. Another application of our results is the characterization of complexity for nonrecursive Horn clause logic programs over bags and/or finite sets and/or trees. In (Dantsin & Voronkov 1997*b*) we proved the following result. *If equation solving over a domain is in NP, then the complexity of nonrecursive Horn clause logic programs over this domain is in NEXPTIME* (see (Dantsin & Voronkov 1997*b*) for precise definitions). This, together with another result of (Dantsin & Voronkov 1997*b*) proves that the complexity of nonrecursive logic programs over bags and/or sets and/or trees is NEXPTIME-complete.

It is interesting to consider constraint logic programming over bags and finite sets which uses more powerful primitives than just the bag and set constructors. For examples, what is the complexity of constraint satisfaction when we also have primitives like $\cup$ or $\subseteq$? An algorithm for extended constraints is given in (Stolzenburg 1996) but only for a limited special case of flat finite sets and without the complexity analysis of constraint satisfiability. Set constraints have recently received a considerable attention in connection with program verification, but mostly for infinite sets and constraints that are less relevant to databases or logic programs, see e.g. (Aiken 1992, Bachmair, Ganzinger & Waldmann 1993, McAllister, Givan, Witty & Kozen 1996, Charatonic & Podelski 1997) and the survey (Pacholski & Podelski 1997)).

Results on AC- and ACI-unification are relevant to this problem but not immediately applicable, see (Stickel 1981, Fortenbacher 1987, Lincoln & Christian 1989, Boudet, Contejean & Marcé 1996).

Also, it is interesting to consider a suitable representation of graphs (up to isomorphism) and the complexity of unification over graphs.

Appendix A

Proofs

A.1 Proof of Theorem 4.6

A.1.1 Proof of the termination part

We prove that the transformation algorithm terminates in a polynomial number of nondeterministic steps. In order to prove it, we define several functions on systems of isolated or simple equations. Let S be such a system. Then

$Bag(S)$ is the number of different final bag variables in S;
$Set(S)$ is the number of different final set variables in S;
$NI(S)$ is the number of different non-isolated equations in S;
$Cycle(S)$ is the number of different equations of the form $x = \{Y \mid x\}$ in S;
$rank(S) = 2Bag(S) + 2Set(S) + NI(S) - Cycle(S)$

By the *rank of* S we mean the number $rank(S)$. The ranks of the system $\perp$ is by definition 0. Note that the rank of every system is non-negative because every equation $x = \{Y \mid x\}$ is non-isolated. Clearly, termination in nondeterministic polynomial time will be proved if we show that every application of a transformation rule decreases the rank.

Lemma A.1 (reduction using $X = Y$)
Reduction using $X = Y$ does not increase the rank.

Proof. The reduction can decrease $Bag(S)$ and $Set(S)$ but cannot increase them. Every reduction using $x = y$ can add one equation $x = y$ to S but this equation is isolated in the resulting system, so $NI(S)$ does not increase. $Cycle(S)$ can increase but cannot decrease (except the case when S is transformed into $\perp$). □

Lemma A.2 (tree decomposition)
Tree decomposition decreases the rank.

Proof. Tree decomposition removes one non-isolated equation and thereby decreases $NI(S)$. The other items remain the same. Further reduction does not increase the rank by Lemma A.1. □

Lemma A.3 (bag decomposition)
The bag decomposition rule decreases the rank.

Proof. Consider case 1 and case 2 of the rule. Case 1 either transforms the system to $\perp$ or replaces two non-isolated equations $z = s$ and $z = t$ by one equation $z = \{\!|Y \mid u|\!\}$. Thereby *NI*(S) decreases by at least 1, while the other items do not change. In case 2 the rank changes as follows:

1. Two non-isolated equations $z = s$ and $z = t$ are replaced by $z = \{\!|X_2, Y_1, Y_2 \mid w|\!\}$. The rank decreases by at least 1.

2. A new final variable w appears. The rank increases by 2.

3. If $v = \{\!|X_2 \mid w|\!\}$ is added then the variable v becomes non-final and the rank decreases by at least 1. Otherwise, the variable v disappears and the rank decreases by 2.

4. Similarly, if $u = \{\!|Y_2 \mid w|\!\}$ is added then the variable u becomes non-final and the rank decreases by at least 1. Otherwise, the variable u disappears and the rank decreases by 2.

Thus, summing up, we obtain that case 2 decreases the rank by at least 1. □

Lemma A.4 (set decomposition)
The set decomposition rule decreases the rank.

Proof. If the variable z is final then the rule replaces two non-isolated equations $z = s$ and $z = t$ by one equation of the form $z = \{Y \mid z\}$, so the rank decreases by at least 2. Otherwise, consider the following possible cases.

Case 1a The non-isolated equations $z = s$ and $z = t$ are replaced by the equation $u = \{X_2 \cup Y_2 \mid u\}$. The rank decreases by at least 2.

Case 1b Two non-isolated equations $z = s$ and $z = t$ are replaced by $z = \{Y_1 \mid u\}$. This decreases the rank by at least 1. If the equation $u = \{X_2, Y_2 \mid u\}$ is added, the rank remains the same because both *NI*(S) and *Cycle*(S) increase by 1.

Case 2a Two non-isolated equations $z = s$ and $z = t$ are replaced by three equations. Both variables u and v become non-final and a new final variable w appears. The first decreases the rank by at most 1. The second decreases the rank by 2.

Case 2b In this case we have

1. Two non-isolated equations $z = s$ and $z = t$ are replaced by $z = \{X_2 \cup Y_1 \cup Y_2 \mid w\}$. The rank decreases by at least 1.
2. A new final variable w appears. The rank increases by 2.
3. If $v = \{X_2 \mid w\}$ is added then the variable v becomes non-final and the rank decreases by at least 1. Otherwise, the variable v disappears and the rank decreases by 2.
4. Similarly, if $u = \{Y_2 \mid w\}$ is added then the variable u becomes non-final and the rank decreases by at least 1. Otherwise, the variable u disappears and the rank decreases by 2.

Summing up these subcases, we obtain that case 2b decreases the rank by at least 1.

□

To complete the proof of the termination part, it remains to note that both other rules, function failure and type failure, merely transforms the system into $\perp$.

A.1.2 Proof of the soundness and completeness part

In order to prove that the transformation algorithm is sound and complete, we shall show that each rule has the following property: if $S_1, \ldots, S_k$ are all nondeterministic results of the application of the rule to a system S then

$$\mathcal{H} \models S \equiv \exists y_1 \ldots \exists y_l (S_1 \vee \ldots \vee S_k) \tag{A.1}$$

where $y_1, \ldots, y_l$ are all variables of $S_1, \ldots, S_k$ not occurring in S.

In the sequel, when we say that a formula φ is true, we mean $\mathcal{H} \models \varphi$. Likewise, formulas φ_1 and φ_2 are equivalent if $\mathcal{H} \models \varphi_1 \equiv \varphi_2$.

Lemma A.5 (reduction using $X = Y$)
Let S be a system and S' be obtained from S by reduction using $x_1 = y_1, \ldots, x_n = y_n$. Suppose that

$$\mathcal{H} \models S \supset (x_1 = y_1 \wedge \ldots \wedge x_n = y_n).$$

Then S and S' are equivalent.

Proof. Induction by n and the definition of the reduction. □

Lemma A.6 (tree decomposition)
Let S be a system and S' be obtained from S by tree decomposition. Then S and S' are equivalent.

Proof. Let the rule be applied to $z = f(x_1, \ldots, x_n)$ and $z = f(y_1, \ldots, y_n)$. Denote by S^- the system $S \setminus \{x = f(x_1, \ldots, x_n)\}$. We have $\mathcal{H} \models S \supset x = f(x_1, \ldots, x_n) \wedge x = f(y_1, \ldots, y_n)$. Then $\mathcal{H} \models S \supset x_1 = y_1 \wedge \ldots \wedge x_n = y_n$. Hence, $\mathcal{H} \models S \equiv S^- \wedge x_1 = y_1 \wedge \ldots \wedge x_n = y_n$. Note that reduction using $x_1 = y_1, \ldots, x_n = y_n$ transforms $S^- \cup \{x_1 = y_1, \ldots, x_n = y_n\}$ into S'. By Lemma A.5 the latter systems are equivalent. Hence, S and S' are equivalent. □

Lemma A.7 (bag and set extensions)
Let $z = s$ be a bag or set equation in a system S. Let $z = s'$ be any extension of $z = s$. Then S is equivalent to the system obtained from S by replacement of $z = s$ by $z = s'$.

Proof. For bags it is a straightforward consequence of the following fact: formulas $x_1 = \{\!|Y_1 \,|\, x_2|\!\} \wedge x_2 = \{\!|Y_2 \,|\, x_3|\!\}$ and $x_1 = \{\!|Y_1, Y_2 \,|\, x_3|\!\} \wedge x_2 = \{\!|Y_2 \,|\, x_3|\!\}$ are equivalent. For sets the proof is the same. □

Lemma A.8 (bag and set equalities)
An equation $\{\!|X|\!\} = \{\!|Y|\!\}$ is equivalent to the disjunction $\bigvee E$ of all minimal bag correspondences E between X and Y. Similarly, $\{X\} = \{Y\}$ is equivalent to the disjunction of all minimal set correspondences between X and Y.

Proof. It is not hard to argue that $\{X\} = \{Y\}$ is equivalent to the disjunction of all bag correspondences between X and Y. Then note that for every bag correspondence E there exists a minimal set correspondence E' such that $E \supset E'$. Therefore, non-minimal bag correspondences can be removed from the disjunction. The proof for sets is similar. □

Lemma A.9 (bag and set inequalities)
Let $X \neq Y$ denote the conjunction of all inequalities $x_i \neq y_j$ where x_i is in X and y_j is in Y. Then the implication

$$(X \neq Y) \supset (z = \{\!|X \mid u|\!\}) \wedge (z = \{\!|Y \mid v|\!\})$$

is equivalent to

$$(X \neq Y) \supset \exists w(z = \{\!|X, Y \mid w|\!\}) \wedge (u = \{\!|Y \mid w|\!\}) \wedge (v = \{\!|X \mid w|\!\}).$$

A similar equivalence holds for sets too.

Proof. Straightforward. □

Lemma A.10 (bag decomposition)
The bag decomposition rule has property (A.1).

Proof. Let the rule be applied to $z = s$ and $z = t$. If any of these equations has no extension then the system S has a bag cycle. In this case S has no solution and, obviously, (A.1) holds. Assume that $z = s$ and $z = t$ have extensions and consider two possible cases. Denote by S^- the system $S \setminus \{z = s, z = t\}$.

Case 1. There are extensions $z = \{\!|X \mid u|\!\}$ and $z = \{\!|Y \mid u|\!\}$. If $|X| \neq |Y|$, then (A.1) trivially holds since S has no solution. Assume that $|X| = |Y|$. Using Lemma A.7, we obtain that S is equivalent to

$$S^- \wedge (z = \{\!|Y \mid u|\!\}) \wedge (\{\!|X|\!\} = \{\!|Y|\!\}).$$

By Lemma A.8 the equation $\{\!|X|\!\} = \{\!|Y|\!\}$ can be replaced by the disjunction $\bigvee E$ of all minimal bag correspondences E between X and Y. Note that each system S_i in (A.1) is the system

$$S^- \wedge (z = \{\!|Y \mid u|\!\}) \wedge E$$

reduced by E. Therefore, using Lemma A.5, we obtain (A.1).

Case 2. There are extensions $z = \{\!|X \mid u|\!\}$ and $z = \{\!|Y \mid v|\!\}$. Let X be divided into disjoint parts X_1 and X_2 in an arbitrary way. Similarly, let Y be divided into Y_1 and Y_2 such that $|X_1| = |Y_1|$. Consider the conjunction

$$(\{\!|X_1|\!\} = \{\!|Y_1|\!\}) \wedge (X_2 \neq Y_2)$$

where the first member is absent if X_1 and Y_1 are empty, and the second member is absent if X_2 or Y_2 is empty. Denote by D the disjunction of all such conjunctions. Clearly, D is true for any X and Y. Therefore S is equivalent to $S \wedge D$. Using this equivalence and Lemmas A.7 and A.9, we obtain that S is equivalent to

$$S^- \wedge \bigvee \exists w C(X_1, X_2, Y_1, Y_2).$$

Here the disjunction is taken over all partitions of X and Y into X_1, X_2 and Y_1, Y_2, and the formula $C(X_1, X_2, Y_1, Y_2)$ denotes the conjunction

$$z = \{\!|X_2, Y_1, Y_2 \mid w|\!\} \;\wedge\; v = \{\!|X_2 \mid w|\!\} \;\wedge\; u = \{\!|Y_2 \mid w|\!\} \;\wedge\; \{\!|X_1|\!\} = \{\!|Y_1|\!\}.$$

Now, using Lemma A.8, we replace bag equations $\{\!|X_1|\!\} = \{\!|Y_1|\!\}$ by disjunctions of minimal bag correspondences between X_1 and Y_1. Applying Lemma A.5, we obtain (A.1). □

Lemma A.11 (set decomposition)
Set decomposition has property (A.1).

Proof. This proof is quite similar to the proof of Lemma A.10. The main difference is in processing set cycles. To tackle it, we use the following equivalence

$$(u = \{X \mid v\} \ \wedge \ v = \{Y \mid u\}) \ \equiv \ (u = v \ \wedge \ u = \{\!|X \cup Y \mid u|\!\}).$$

Finally, we note that property (A.1) obviously holds for the function-failure and type-failure rules.

□

A.2 Proof of Theorem 4.7

Since the occur-check algorithm is in fact an algorithm that checks whether a graph has a cycle, the running time part is obvious. If the graph has no cycle, then the existence of a solution follows from Lemma 5.1 on the existence of a unifier. If the graph has a cycle, it is not hard to argue that no solution exists.

Bibliography

Abiteboul, S. & Beeri, C. (1995), 'The power of languages for the manipulation of complex values', *VLDB Journal* **4**, 727–794.

Abiteboul, S. & Grumbach, S. (1988), Col: A logic-based language for complex objects, *in* J. Schmidt, S. Ceri & M. Missikoff, eds, 'Advances in Database Technology - EDBT'88. Proceedings of the International Conference on Extending Database Technology', Vol. 303 of *Lecture Notes in Computer Science*, Springer Verlag, Venice, Italy, pp. 271–293.

Abiteboul, S. & Grumbach, S. (1991), 'A rule-based language with functions and sets', *ACM Transactions on Database Systems* **16**(1), 1–30.

Aiken, A. (1992), Set constraints: Results, applications and future directions, *in* 'Proceedings of the Workshop on Principles and Practice of Constraint Programming', Vol. 874 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 326–335.

Apt, K. (1990), Logic programming, *in* J. Van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, Elsevier Science, Amsterdam, chapter 10, pp. 493–574.

Baader, F. & Siekmann, J. (1994), Unification theory, *in* D. Gabbay, C. Hogger & J. Robinson, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming', Oxford University Press.

Bachmair, L., Ganzinger, H. & Waldmann, U. (1993), Set constraints are the monadic class, *in* 'Proc. IEEE Conference on Logic in Computer Science (LICS)', IEEE Computer Society Press, pp. 75–83.

Beeri, C. & Kornatzky, Y. (1994), 'A logical query language for hypermedia systems', *Information Sciences* **77**, 1–38.

Beeri, C., Naqvi, S., Schmueli, O. & Tsur, S. (1991), 'Set constructors in a logic database language', *Journal of Logic Programming* **10**, 181–232.

Boudet, A., Contejean, E. & Marcé, C. (1996), AC-complete unification and its application to theorem proving, *in* H. Ganzinger, ed., '7th International Conference on Rewriting Techniques and Applications', Vol. 1103 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 18–32.

Charatonic, W. & Podelski, A. (1997), Set constraints with intersection, *in* G. Winskel, ed., 'Proc. IEEE Conference on Logic in Computer Science (LICS)', IEEE Computer Society Press, pp. 362–372.

Dantsin, E. & Voronkov, A. (1997*a*), Complexity of query answering in logic databases with complex values, *in* S. Adian & A. Nerode, eds, 'Logical Foundations of Computer Science. 4th International Symposium, LFCS'97', Vol. 1234 of *Lecture Notes in Computer Science*, Yaroslavl, Russia, pp. 56–66.

Dantsin, E. & Voronkov, A. (1997*b*), Complexity of query answering in logic databases with complex data, UPMAIL Technical Report 149, Uppsala University, Computing Science Department.

Dovier, A., Omodeo, E., Pontelli, E. & Rossi, G. (1996), '{log}: A language for programming in logic with finite sets', *Journal of Logic Programming* **28**(1), 1–44.

Fortenbacher, A. (1987), 'An algebraic approach to unification under associativity and commutativity', *Journal of Symbolic Computations* **3**(3), 217–229.

Gallier, J. & Raatz, S. (1989), 'Extending SLD-resolution to equational Horn clauses using E-unification', *Journal of Logic Programming* **6**(3), 3–44.

Grumbach, S. & Vianu, V. (1995), 'Tractable query languages for complex object databases', *Journal of Computer and System Sciences* **51**(2), 149–167.

Kanella kis, P., Kuper, G. & Revesz, P. (1995), 'Constraint query languages', *Journal of Computer and System Sciences* **51**, 26–52.

Kapur, D. & Narendran, P. (1986), NP-completeness of the set unification and matching problems, *in* J. Siekmann, ed., 'Proc. 8th CADE', Vol. 230 of *Lecture Notes in Computer Science*, pp. 489–495.

Kuper, G. (1990), 'Logic programming with sets', *Journal of Computer and System Sciences* **41**, 44–64.

Leone, N. & Rullo, P. (1993), 'Ordered logic programming with sets', *Journal of Logic and Computation* **3**(6), 621–642.

Lincoln, P. & Christian, J. (1989), 'Adventures in associative-commutative unification (a summary)', *Journal of Symbolic Computations* **8**(1/2), 217–240.

Lloyd, J. (1987), *Foundations of Logic Programming (2nd edition)*, Springer Verlag.

Maher, M. (1992), A CLP view of logic programming, *in* 'Proc. Conf. on Algebraic and Logic Programming', Vol. 632 of *Lecture Notes in Computer Science*, pp. 364–383.

Maher, M. (1993), A logic programming view of CLP, *in* 'International Conference on Logic Programming', pp. 737–753.

Martelli, A. & Montanari, U. (1982), 'An efficient unification algorithm', *ACM Transactions on Programming Languages and Systems* **4**(2), 258–282.

McAllister, D., Givan, R., Witty, C. & Kozen, D. (1996), Tarskian set constraints, *in* 'Proc. IEEE Conference on Logic in Computer Science (LICS)', IEEE Computer Society Press, New Brunswick, New Jersey, pp. 138–147.

Pacholski, L. & Podelski, A. (1997), Set constraints: a pearl in research on constraints, *in* G. Smolka, ed., 'Proceedings of the Third International Conference on Constraint Programming', number 1330 *in* 'Lecture Notes in Computer Science'.

Paterson, M. & Wegman, M. (1978), 'Linear unification', *Journal of Computer and System Sciences* **16**, 158–167.

Shmueli, O., Tsur, S. & Zaniolo, C. (1992), 'Compilation of set terms in the logic data language (LDL)', *Journal of Logic Programming* **12**(1), 89–119.

Stickel, M. (1981), 'A complete unification algorithm for associative-commutative functions', *Journal of the Association for Computing Machinery* **28**(3), 423–434.

Stolzenburg, F. (1996), Membership-constraints and complexity in logic programming with sets, *in* F. Baader & K. Schulz, eds, 'Frontiers in Combining Systems', Kluwer Academic, pp. 285–302.

Vadaparty, K. (1991), On the power of rule-based languages with sets, *in* 'ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', pp. 26–36.

Voronkov, A. (1992), Logic programming with bounded quantifiers, *in* A. Voronkov, ed., 'Logic Programming', Vol. 592 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, pp. 486–514.

Voronkov, A. (1997), Logic programming with bounded quantifiers revisited, UPMAIL Technical Report 114, Uppsala University, Computing Science Department.