TLP **6** (6): 645–701, 2006. © 2006 Cambridge University Press
doi:10.1017/S1471068406002730 Printed in the United Kingdom

Set unification

AGOSTINO DOVIER*
Dip. di Matematica e Informatica, Università di Udine, Via delle Scienze 206, 33100 Udine, Italy
(*e-mail:* dovier@dimi.uniud.it)

ENRICO PONTELLI†
Department of Computer Science, New Mexico State University, Box 30001, MSC CS, Las Cruces, NM 88003, USA
(*e-mail:* epontell@cs.nmsu.edu)

GIANFRANCO ROSSI‡
Dip. di Matematica, Università di Parma, Via M. D'Azeglio, 85/A, 43100 Parma, Italy
(*e-mail:* gianfranco.rossi@unipr.it)

submitted 22 January 2003; revised 11 November 2004; accepted 20 July 2005

Abstract

The unification problem in algebras capable of describing *sets* has been tackled, directly or indirectly, by many researchers and it finds important applications in various research areas, e.g. deductive databases, theorem proving, static analysis, rapid software prototyping. The various solutions proposed are spread across a large literature. In this paper we provide a uniform presentation of unification of sets, formalizing it at the level of set theory. We address the problem of deciding existence of solutions at an abstract level. This provides also the ability to classify different types of set unification problems. Unification algorithms are uniformly proposed to solve the unification problem in each of such classes. The algorithms presented are partly drawn from the literature – and properly revisited and analyzed – and partly novel proposals. In particular, we present a new goal-driven algorithm for general $ACI1$ unification and a new simpler algorithm for general $(Ab)(C\ell)$ unification.

KEYWORDS: Unification theory, set theory, $ACI1$ unification

1 Introduction

Sets are familiar mathematical objects, and they are often used as an high-level abstraction to represent complex data structures, whenever the order and repetitions of elements are immaterial. A key operation when dealing with set data structures is

* A. Dovier is partially supported by MIUR project 2005015491, and by GNCS 2005 project on constraints and their applications.
† E. Pontelli is partially supported by NSF Grants CNS-0220590, CNS-0454066, and HRD-0420407 and by MIUR project 2005015491.
‡ G. Rossi is partially supported by by MIUR project 2005015491, and by GNCS 2005 project on constraints and their applications.

comparing two sets. According to the traditional *extensionality axiom* (Kunen 1980), two sets are equal if and only if they contain the same elements. The problem of set equality is usually formally addressed within first-order logic. In this context, a set is represented by a first-order term, called a *set term*, built from symbols of a suitable alphabet, using selected function symbols as set constructors. Since, in general, variables can occur within a set term in place of either individuals or sets, solving equations between set terms amounts to solving a *set unification* or a *set matching* problem. Intuitively, the set unification problem is the problem of computing (or simply testing the existence of) an assignment of values to the variables occurring in two set terms which makes them denote the same set. Set matching can be seen as a special case of set unification, where variables are allowed to occur in only one of the two set terms which are compared. Set unification can be thought of as an instance of E-unification (Siekmann 1989), i.e., unification modulo an equational theory E, where E describes the (semantic) properties of the interpreted symbols used to represent sets.

Two main approaches for representing sets as terms have been presented in the literature. The *union-based representation* makes use of the union operator ($\cup$) to construct sets, while the *list-like representation* builds sets using an *element insertion* constructor (typically denoted by $\{\cdot \mid \cdot\}$). The list-like representation has been frequently used in the context of logic languages embedding sets. It is used for instance in Kuper (1990), Jayaraman (1992), in Beeri *et al.* (1991) – where $\{\cdot \mid \cdot\}$ is called **scons** – in the language $\{\log\}$ (Dovier *et al.* 1996), and in the Gödel language (Hill and Lloyd 1994). In various papers dealing with computable set theory, $\{\cdot \mid \cdot\}$ is used and called `with` (Cantone *et al.* 2001).

The union-based representation, on the contrary, has been often used when dealing with the problem of set unification on its own (Büttner 1986; Livesey and Siekmann 1976), where set unification is dealt with as an *Associative-Commutative-Idempotent (ACI) unification* problem, i.e., unification in presence of operators satisfying the *Associativity*, *Commutativity*, and *Idempotence* properties. In (Legeard and Legros 1991) sets are represented using the union-based approach; however, since set operations are evaluated only when applied to *ground sets*, set unification is not required at all.

The computational complexity properties of the set unification and set matching problems have been investigated by Kapur and Narendran (1986, 1992), who established that these decision problems are NP-complete. Complexity of the set unification/matching operation, however, depends on which forms of set terms (e.g., flat or nested sets, with zero, one, or more set variables) are allowed. The form of set terms in turn is influenced by the set constructors used to build them. Thus, different complexity results can be obtained for different classes of set terms.

In this paper we present a uniform survey of the problem of unification in presence of sets, across different set representations and different admissible classes of set terms. We provide a uniform presentation of a number of different approaches and compare them. Unification algorithms for each considered unification problem are presented and analyzed. These algorithms are either drawn from the literature or they represent *novel solutions* proposed by the authors. In particular a goal-driven

algorithm for general ACI unification is proposed, together with a new algorithm (with a simple termination proof) for general $(Ab)(C\ell)$ unification.

1.1 Application domains of set unification

Various forms of set unification have been proposed by many authors, in different application frameworks:

Declarative programming languages with sets: Various declarative programming languages relying on sets as *first-class objects* have been proposed, which provide different forms of set unification. Most of these languages are instances of the *Constraint Logic Programming* paradigm (Dovier *et al.* 1996; Dovier *et al.* 2000; Yakhno and Petrov 2000) or of the *Functional-Logic* paradigm (Jayaraman 1992; Arenas-Sánchez and Rodríguez-Artalejo 2001). The specification language Z (Spivey 1992) makes use of sets as data abstraction; attempts have been made to produce executable versions of Z, such as the ZAP compiler (Grieskamp 1999) (whose implementation, however, does not embed a set unification algorithm).

Deductive databases: Various proposals have been put forward for embedding sets as primitive data structures in deductive database languages, providing set unification or set matching as a built-in mechanism for set manipulation (Liu 1998; Abiteboul and Grumbach. 1991; Naqvi and Tsur 1989; Shmueli *et al.* 1992; Lim and Ng 1997; Kifer and Lausen 1989). In these frameworks, it is common to deal with sets involving unions of variables.

AI and Automated deduction: Set abstraction and operations have been shown to be fundamental in various subfields of Artificial Intelligence. They have been used as tools for the description of linguistic theories in Natural Language Processing (Manandhar 1994). In particular, unification based grammars augmented with set descriptions (Rounds 1988; Pollard and Moshier 1990) require set unification. Set unification has been used in discovery procedures for determining categorial grammars from linguistic data (Marciniec 1997). Set data structures have also been used in pattern matching and pattern directed invocation in various AI languages (Livesey and Siekmann 1976). Proposals dealing with computable properties and algorithmic manipulation of set structures have appeared also in the area of automated deduction, e.g., to reduce the length of proofs (Policriti and Schwartz 1997).

Program analysis and Security: Codish and Lagoon (2000) described an application of elementary $ACI1$ unification to the problem of *sharing analysis* of logic programs. Wang et al. show how a system based on $CLP(\mathcal{SET})$ (hence, on set unification) can be used to model access control (Wang *et al.* 2004).

1.2 Unification algorithms

The problem of solving set unification has been mostly tackled in the form of ACI unification, and unification algorithms, returning the set of all the solutions to a given problem, have been proposed. The first work proposing a viable solution to ACI unification is Livesey and Siekmann (1976). This work mostly deals with AC unification – by reducing it to the solution of Diophantine equations – and

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

only in the end it suggests a solution of the ACI problem, by replacing arithmetic equations with Boolean equations. Direct solutions of the ACI problem have been proposed by Büttner (1986) and Baader and Büttner (1988). More recently, Baader and Schulz (1996) provided a general methodology allowing the unification with constants algorithms proposed for ACI to be extended to general $ACI1$ unification algorithms.

In recent years, a number of efforts have emerged that propose set unification algorithms for the list-like representation of sets, hence for a different equational theory (called $(Ab)(C\ell)$ in Dovier *et al.* (1996)). A first proposal in this direction is the algorithm sketched by Jayaraman and Plaisted (1989). A more general and complete algorithm is the one in Dovier *et al.* (1996). The problem of set unification in this context has been tackled by different authors (Arenas-Sánchez and Dovier 1997; Dovier *et al.* 1998; Stolzenburg 1996; Stolzenburg 1999; Dantsin and Voronkov 1999). In particular, the algorithms presented in Arenas-Sánchez and Dovier (1997) and Stolzenburg (1999) provide solutions which are optimal, in terms of number of unifiers, for large classes of unification problems. The algorithms in Arenas-Sánchez and Dovier (1997) and Dantsin and Voronkov (1999) ensure polynomial time complexity in each non-deterministic branch of the computation.

Various authors have considered simplified versions of the $(Ab)(C\ell)$ problem obtained by imposing restrictions on the form of the set terms. In particular, various works have been proposed to study the simpler cases of matching (Shmueli *et al.* 1992) and unification of *Bound Simple* set terms, i.e., bound set terms of the form $\{s_1, \ldots, s_n\}$, where each s_i is either a constant or a variable (Arni *et al.* 1992; Arni *et al.* 1996; Greco 1996). A parallel algorithm for such restricted $(Ab)(C\ell)$ unification has been presented in Lim and Ng (1997). Set matching is also discussed in Kapur and Narendran (1986).

All these algorithms, however, have been developed in separate contexts, without considering any relationship among them. They have never been formally compared and related. A contribution of this paper is to provide a uniform presentation of the problem, covering most of its different instances, and surveying the different solutions developed.

1.3 Overall structure of the paper

The paper is organized as follows. In Section 2 we define the universe of sets we are dealing with, along with a suitable abstract syntax for representing them and a syntactical classification of the set unification problems. In Section 3 we present a number of examples of unification problems which provide motivations for set unification. In Section 4 we discuss the complexity of the set unification decision problem for each syntactic class of set terms listed in Section 2. In Section 5 we introduce the basic notions and notation concerning E-unification and the equational theories used in E-unification with sets. In Section 6 we describe the problem of $ACI1$ *unification with constants* and its impact on set unification. In Section 7 we extend the discussion to the $(Ab)(C\ell)$ *unification* problem, i.e., the problem of set unification in presence of set terms based on the element insertion

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

constructor $\{\cdot \mid \cdot\}$, and we present a new algorithm for this case. In Section 8 we tackle the most general problem of unification of terms containing both $ACI1$ and free (uninterpreted) function symbols. A new general $ACI1$ unification algorithm is presented. Some related topics are discussed in Section 9, and concluding remarks are presented in Section 10. In Appendix A the proofs of the main results of the paper are reported.

2 Sets and the set unification problem

In this section we characterize the universe of sets we deal with, and we discuss some well-known operations on sets. Finally, we formally introduce the set unification problem.

2.1 A universe of sets

A set is an arbitrary, unordered, collection of elements. Typically, a set is specified either *intensionally*, by means of a property that characterizes membership to the set, or *extensionally*, by explicit enumeration of all its elements. In this paper we restrict our attention to extensional sets. For instance, $\{a, b, c\}$ is the (extensional) set which contains exactly the elements a, b, and c. We denote mathematically the fact: "a belongs to the set $\{a, b, c\}$" using the membership relation: $a \in \{a, b, c\}$. We assume the *extensionality* axiom (Kunen 1980) that states that two sets are equal if and only if they contain the same elements. Thus, $\{a, b, c\}$ is the unique set containing exactly a, b, and c. $\{a, c, b\}$, $\{b, a, c\}$, etc. are alternative ways to describe the same set. A particular set is the empty set $\emptyset$, that contains no elements. A set containing only one element is said to be a *singleton*. If s is a set, then we will denote with $|s|$ its cardinality.

A set is *finite* if it contains a finite number of elements.[1] For instance $\emptyset, \{\emptyset\}$, and $\{a, b, c\}$ are finite sets. However, this definition does not remove all possible cases leading to infinity. The singleton set $\{\mathbb{N}\}$ is a finite set, but its unique element $\mathbb{N}$ is an infinite set. A set is said to be *hereditarily finite* if it is finite and all its elements are hereditarily finite. This definition leaves still a further possibility for infinity. Let us consider the sets x and y that satisfy the equations $x = \{\emptyset, y\}, y = \{x\}$. They are hereditarily finite, but they hide an infinite descending chain $x \ni y \ni x \ni y \ni \cdots$. These sets, where the membership relation is allowed to be not well-founded, are called *non well-founded sets* (or *hypersets*) (Aczel 1988; Barwise and Moss 1996). Hypersets are very important in some areas, such as concurrency theory, but they are not accepted in traditional set theory, where sets are expected to be well-founded.

Let us focus on *hereditarily finite and well-founded sets*. We can consider two approaches to set theory:

- *pure sets*, in which the only entity that does not contain elements is the empty set $\emptyset$, and

[1] A precise, formal, characterization of the notion of finiteness is outside the scope of this work. For a theoretical analysis of this topic see Tarski (1924).

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

- *sets with individuals*, in which there exists a collection $\mathscr{U}$ of individuals, where each element of $\mathscr{U}$ is not a set and does not contain elements. Since the elements of $\mathscr{U}$ are not sets, we also have that $\emptyset \notin \mathscr{U}$.

In the second approach, the extensionality axiom has to be revised for the elements of $\mathscr{U}$, since

(*i*) two individuals are different even if they contain the same elements (namely, none), and
(*ii*) all the elements in $\mathscr{U}$ are different from $\emptyset$.

In this paper we will focus on the approach based on sets with individuals, as it generalizes the pure sets approach (by taking $\mathscr{U} = \emptyset$).

Let us introduce the universe of sets we are interested in (see also Cantone *et al.* (2001, p. 88)). As usual, the subset relation $x \subseteq y$ denotes the formula $\forall z\,(z \in x \rightarrow z \in y)$. If s is a set, with $\wp_{\text{fin}}(s) = \{x \,:\, x \subseteq s \wedge x \text{ is finite}\}$ we denote the set of all its finite subsets.

Definition 1
The *Universe* $\mathbb{HIF}^{\mathscr{U}}$ *of hereditarily finite sets based on* $\mathscr{U}$ is obtained as follows:

$$\begin{cases} \mathbb{HIF}_0^{\mathscr{U}} & \stackrel{\text{def}}{=} & \wp_{\text{fin}}(\mathscr{U}) \\ \mathbb{HIF}_{i+1}^{\mathscr{U}} & \stackrel{\text{def}}{=} & \mathbb{HIF}_i^{\mathscr{U}} \cup \wp_{\text{fin}}\left(\mathbb{HIF}_i^{\mathscr{U}}\right) \\ \mathbb{HIF}^{\mathscr{U}} & \stackrel{\text{def}}{=} & \bigcup_{i \in \mathbb{N}} \mathbb{HIF}_i^{\mathscr{U}} \end{cases}$$

The sets in $\mathbb{HIF}_0^{\mathscr{U}}$ contain the finite subsets of the set of individuals: these particular sets are called *flat* sets. The sets introduced in $\mathbb{HIF}_i^{\mathscr{U}}$, with $i > 0$, may contain elements that are sets themselves. We refer to such sets as *nested sets*. For instance, if $\mathscr{U} = \{a, b, c\}$, then $\mathbb{HIF}_0^{\mathscr{U}}$ consists of the flat sets:

$$\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}$$

Some nested sets are the following:

$$\{\emptyset\},\ \{\{a\}\},\ \{\emptyset, \{\{a\}, b\}, \{\{\{c\}\}\}\}$$

2.2 Abstract set terms

So far we have represented sets by exploiting the usual intuitive notation based on braces and commas. In order to deal with sets as primitive data objects in a first-order language, however, we need to precisely represent them as first-order terms of the language. For this reason, one or more function symbols are selected to be used as *set constructors*. Set constructors will allow complex sets to be built from simpler ones. Many different approaches are possible. The two approaches that, to our knowledge, have received more attention in the literature are the following.

1. *Union-based representation.* This solution is based on the use of the *union* constructor $\cup$ and, possibly, the *singleton* constructor $\{\cdot\}$. $s \cup t$ represents the set which contains the elements of the sets s and t, that is,

$$s \cup t = \{x \,:\, x \in s \vee x \in t\},$$

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

while $\{t\}$ represents the set containing the single element t. With this approach, the finite set $\{t_0, \ldots, t_n\}$ is represented by a union of singletons: $\{t_0\} \cup \cdots \cup \{t_n\}$, where $t_0, \ldots, t_n$ are either sets or individuals. The empty set is represented by a distinguished constant $\emptyset$.

2. *List-like representation.* An alternative representation of sets is based on the *element insertion* constructor $\{\cdot \mid \cdot\}$. $\{t \mid s\}$ represents the set obtained by adding the element t (either a set or an individual) to the set s if it is not yet in s, that is

$$\{t \mid s\} = \{x \ : \ x \in s \vee x = t\}.$$

The empty set is represented by a distinguished constant $\emptyset$. Thus, the finite set $\{t_0, \ldots, t_n\}$ is represented by a sequence of element insertions:

$$\{t_0 \mid \{\cdots \{t_n \mid \emptyset\} \cdots\}\}$$

where $t_0, \ldots, t_n$ are either sets or individuals.

As far as the syntactic representation of the individuals (i.e., the elements of $\mathscr{U}$) is concerned, we can represent them either

- as constant symbols different from $\emptyset$ (*simple individual terms*) or
- as terms of the form $f(t_1, \ldots, t_n)$, $n > 0$, f different from $\cup$ and $\{\cdot \mid \cdot\}$, and $t_1, \ldots, t_n$ terms (*general individual terms*).

Both the union-based and the list-like representations allow the elements of the sets to be either individual terms or other set terms. Individual and set terms can be nested at any level.

Let us observe that the element insertion constructor $\{\cdot \mid \cdot\}$ can be represented using $\cup$, i.e., $\{s \mid t\} \stackrel{\mathscr{def}}{=} \{s\} \cup t$. However, in Dovier *et al.* (2000) it is proved that, without singleton sets, the two symbols are not mutually definable, unless we allow the use of complex formulae involving universal quantifiers. Observe moreover that the $\cup$ symbol allows one to define set inclusion: $x \subseteq y$ is equivalent to $x \cup y = y$.

Furthermore, let us observe that the definition of $\cup$, being based on membership, makes sense on sets, not on individuals. For instance, the union of two individuals a and b would be a memberless object. There is no way of stating that a is equal or different from $a \cup b$ without introducing new, non-standard, axiomatizations. For this reason, we assume that the $\cup$ constructor is used only on sets. Similar considerations apply to the second argument of the $\{\cdot \mid \cdot\}$ operator.

For the sake of simplicity, in the rest of the work we will make use of a simpler and more intuitive abstract syntax to denote sets, disregarding the concrete representation used to encode them as terms in the language at hand.

Definition 2

An *abstract set term* is a term of the form

$$\{X_1, \ldots, X_m, a_1, \ldots, a_n, s_1, \ldots, s_p\} \cup Y_1 \cup \cdots \cup Y_q \quad m, n, p, q \geqslant 0$$

where X_i, Y_i are variables, a_i are individual terms, and s_i are abstract set terms distinct from variables. The Y_i variables are called *the set variables* of the abstract

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

set term. In particular,

- when $m = n = p = q = 0$, the term is simply written as $\emptyset$.
- when $m = n = p = 0$ and $q = 1$, the term is the set variable Y_1.

The *size* $||s||$ of an abstract set term s is the number of occurrences of symbols in s.

As a notational convention, we will usually use a, b, c, possibly subscripted, to denote individual terms, and r, s, t, possibly subscripted, to denote (abstract) set terms or individual terms. Variables are denoted by identifiers with capital letters.

When $q \leqslant 1$, the abstract set term can be rendered concretely using both representations described above. For example, $\{X_1, X_2, a, b, c\} \cup Y_1$ can be seen as a shorthand for both the concrete terms $\{X_1 \mid \{X_2 \mid \{a \mid \{b \mid \{c \mid Y_1\}\}\}\}\}$ and $\{X_1\} \cup \{X_2\} \cup \{a\} \cup \{b\} \cup \{c\} \cup Y_1$. Conversely, when $q > 1$, the $\cup$ constructor is required; thus, only the union-based representation is feasible.

When clear from the context we will omit the word "abstract", referring to abstract set terms simply as set terms.

Set terms may contain variables, both as individuals (the variables X_i's) and as sets (the variables Y_j's). A set term containing variables denotes a possibly infinite collections of sets. For instance, the term $\{a, X, b\}$ denotes all sets containing two individuals, a and b, and possibly a third unknown element X. If X takes the value a or b then the set will have only two elements. Otherwise, e.g., $X = c$, the set will contain three elements. Note that the set terms $\{a, a, b\}$, $\{a, b, a\}$, $\{b, a, a, b\}$, etc. are accepted notations for the same set, i.e., the (unique) set containing exactly a and b. Note also that variables in set terms could be implicitly forced to assume set values using the fact that the $\cup$ constructor requires two set arguments. Thus, for instance, the variable Y in the set term $\{a, b\} \cup Y$ can take only set values. Set terms are called *non-ground* (*ground*) if they do (do not) contain variables. Finally, note that general individual terms can be non-ground. For instance, $f(X, Y)$ is a non-ground term, but the fact that the outermost symbol is not a set constructor ensures that it is an individual.

Example 1
The following are abstract set terms.

- $\{1, 2, 3\}$ $(m = 0, n = 3, p = 0, q = 0)$
- $\{\emptyset\}$ $(m = 0, n = 0, p = 1, q = 0)$
- $\{X_1, X_2, a, b, c, d\} \cup Y$ $(m = 2, n = 4, p = 0, q = 1)$
- $Y_1 \cup Y_2$ $(m = 0, n = 0, p = 0, q = 2)$
- $\{X, a, b, c, \{1, 2, 3\}, \{\emptyset\}\}$ $(m = 1, n = 3, p = 2, q = 0)$
- $\{X_1, X_2, a, f(\{a, \emptyset\}), \emptyset\} \cup Y_1$ $(m = 2, n = 2, p = 1, q = 1)$

2.3 Set equivalence and set unification

The most natural decision test regarding set terms is testing whether they represent the same set or, in the case of non-groundness, testing whether there exists an assignment for the variables that forces the two terms to represent the same set.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

Definition 3
Given two terms s and t, $s = t$ is said to be an *equation*. A conjunction $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$ of equations is said to be a *system of equations*. Systems of equations are also commonly viewed as sets of equations.

If $X_1, \ldots, X_n$ are the variables occurring in a system of equations C, we denote with $\vec{\exists} C$ the formula $\exists X_1 \cdots \exists X_n C$. The existence of an assignment for the variables in s and t that forces the two terms to represent the same set will be denoted by $\mathbb{HF} \models \vec{\exists} s = t$, formally defined below.

Before defining the interpretation of ground abstract set terms in $\mathbb{HF}$, we first show how individual terms (syntax) can be related to individuals (semantics). Let us assume that $\mathcal{U}$ is an infinite set of individuals. Simple individual terms denote distinct elements of $\mathcal{U}$. For the sake of simplicity, in our examples, the individual terms $a, b, c, \ldots$ will be interpreted as the corresponding individuals $a, b, c, \ldots$ of $\mathcal{U}$ – we use the so-called *unique name assumption*. General individual terms $f(s_1, \ldots, s_m)$ and $g(t_1, \ldots, t_n)$, with f different from g, denote distinct elements of $\mathcal{U}$, different from all the individuals associated to the simple individual terms. Each function symbol f of arity n is interpreted as a one-to-one function $f^{\mathbb{HF}}$ from $\mathbb{HF}$ to $\mathcal{U}$.

Definition 4
If $s \equiv \{a_1, \ldots, a_n, s_1, \ldots, s_p\}$ is a ground set term, then its interpretation in $\mathbb{HF}$, denoted by $s^{\mathbb{HF}}$, is the following set:

- if $n = 0$ and $p = 0$ then $s^{\mathbb{HF}}$ is the empty set
- otherwise, $s^{\mathbb{HF}}$ is the set containing exactly the elements $a_1^{\mathbb{HF}}, \ldots, a_n^{\mathbb{HF}}$ and $s_1^{\mathbb{HF}}, \ldots, s_p^{\mathbb{HF}}$, where
  - — if a_i is a simple individual term, then $a_i^{\mathbb{HF}}$ is simply the corresponding individual.
  - — if a_i is of the form $f(t_1, \ldots, t_n)$ then $a_i^{\mathbb{HF}}$ is the individual associated to $f^{\mathbb{HF}}(t_1^{\mathbb{HF}}, \ldots, t_n^{\mathbb{HF}})$.

If s and t are two ground set terms, then $\mathbb{HF} \models (s = t)$ if and only if $s^{\mathbb{HF}}$ is the same set as $t^{\mathbb{HF}}$.
If s and t are two set terms, and $X_1, \ldots, X_n$ are all variables in s and t, then $\mathbb{HF} \models \vec{\exists} s = t$ if and only if there exists an assignment σ of ground set terms to $X_1, \ldots, X_n$ such that $\mathbb{HF} \models (s = t)\sigma$.

Definition 5
If s and t are two set terms, the *set unification decision (SUD)* problem is the problem of checking whether $\mathbb{HF} \models \vec{\exists}(s = t)$. If s and t are ground, the problem is also called *set equivalence*.

Definition 6
If s and t are two set terms and $X_1, \ldots, X_n$ are the variables occurring in them, the *set unification solution (SUS)* problem is the problem of finding an assignment σ of sets and/or individual terms to the variables $X_1, \ldots, X_n$, such that $\mathbb{HF} \models (s = t)\sigma$.

We give a more standard and complete definition of the unification problem in Section 5. Note that, if two general individuals have the same outermost symbols

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

but the ordered list of arguments is different, then they denote two distinct individuals (e.g., $f(a,b)$ and $f(b,a)$). However, if the two individual terms contain set terms as their arguments, in order to decide whether the individuals are or not the same, one needs to compare the sets denoted by the involved set terms. For example, the general individual terms $f(\{a,b\},c)$ and $f(\{b,b,a\},c)$ denote the same individual since $\{a,b\}$ denotes the same set as $\{b,b,a\}$.

From a computational point of view, the complexity of the SUD problem depends on the syntactic form of the two set terms s and t. As a matter of fact, while the set equivalence test of ground set terms denoting flat sets, such as $\{a,b,c\}$ and $\{b,c,a\}$, is rather easy, when the SUD problem deals with nested set terms involving variables it becomes NP-complete (see Section 4.4). Thus, to classify the set unification problem, we subdivide set terms in different syntactic classes.

Definition 7
For $m \geqslant 0, n \geqslant 0, p \geqslant 0, q \geqslant 0$, the class $\mathsf{set}(m,n,p,q)$ is the collection of abstract set terms of the form:

$$\{X_1,\ldots,X_{m'},a_1,\ldots,a_{n'},s_1,\ldots,s_{p'}\} \cup Y_1 \cup \cdots \cup Y_{q'}$$

where $0 \leqslant m' \leqslant m$, $0 \leqslant n' \leqslant n$, $0 \leqslant p' \leqslant p$, $0 \leqslant q' \leqslant q$, and $s_i \in \mathsf{set}(m,n,p,q)$.

Observe that $\emptyset \in \mathsf{set}(m,n,p,q)$ for all m,n,p,q. Furthermore, $\mathsf{set}(m_1,n_1,p_1,q_1) \subseteq \mathsf{set}(m_2,n_2,p_2,q_2)$ if $m_1 \leqslant m_2$ and $n_1 \leqslant n_2$ and $p_1 \leqslant p_2$, and $q_1 \leqslant q_2$. Interesting special cases can be obtained by setting some of these parameters to 0:

$\mathsf{ground} = \bigcup_{n\geqslant 0,p\geqslant 0} \mathsf{set}(0,n,p,0)$**:** the collection of set terms of the form $\{a_1,\ldots,a_n,s_1,\ldots,s_p\}$, with a_i simple individual terms and s_i ground set terms.

$\mathsf{gflat}(q) = \bigcup_{n\geqslant 0} \mathsf{set}(0,n,0,q)$**:** the collection of set terms of the form $\{a_1,\ldots,a_n\} \cup Y_1 \cup \cdots \cup Y_{q'}$, with a_i simple individual terms and Y_i variables ranging over $\mathsf{gflat}(q)$ sets (i.e., sets denoted by $\mathsf{gflat}(q)$ set terms) $(0 \leqslant q' \leqslant q)$.

$\mathsf{flat}(q) = \bigcup_{m\geqslant 0,n\geqslant 0} \mathsf{set}(m,n,0,q)$**:** the collection of set terms of the form $\{X_1,\ldots,X_m,a_1,\ldots,a_n\} \cup Y_1 \cup \cdots \cup Y_{q'}$, with a_i and X_i denoting simple individual terms, and Y_i ranging over $\mathsf{flat}(q)$ sets $(0 \leqslant q' \leqslant q)$.

$\mathsf{nested}(q) = \bigcup_{m\geqslant 0,n\geqslant 0,p\geqslant 0} \mathsf{set}(m,n,p,q)$**:** the collection of set terms of the form $\{X_1,\ldots,X_m,t_1,\ldots,t_n,s_1,\ldots,s_p\} \cup Y_1 \cup \cdots \cup Y_{q'}$, with t_i general individual terms, s_i non-variable $\mathsf{nested}(q)$ set terms, X_i ranging over general individuals or $\mathsf{nested}(q)$ sets, and Y_i ranging over $\mathsf{nested}(q)$ sets $(0 \leqslant q' \leqslant q)$.

$\mathsf{gflat}(q)$ and $\mathsf{flat}(q)$ denote flat sets only, while ground and $\mathsf{nested}(q)$ account for nested sets. For the same q, we have that $\mathsf{gflat}(q) \subseteq \mathsf{flat}(q) \subseteq \mathsf{nested}(q)$. Moreover, ground is included in $\mathsf{nested}(q)$ (namely, in $\mathsf{nested}(0)$), but it is not included in the other classes, since ground accounts also for nested sets. Actually, these classes could be further subdivided into finer subclasses. For instance, we could further distinguish between ground and non-ground nested set terms, with simple or general individuals. However, the four classes we identified above turn out to be sufficient for our analysis.

Observe that, in the concrete representation of sets, the union constructor is not required whenever $q \leqslant 1$. For these sets we can use the list-like representation, based

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

on the element insertion constructor $\{\cdot \mid \cdot\}$. For $q > 1$, instead, we need the union constructor, and possibly the singleton constructor. For instance, the abstract set term $X \cup \{Y\} \cup Z$ can be immediately encoded using the union-based representation while it has no corresponding encoding in the list-like representation. These observations will play an important role when we will address the various unification problems.

3 Examples

This section presents a series of instances of the set unification problem. This allows us to give an intuitive idea of the expressive power of the different frameworks considered in the rest of the paper.

Chords: This is the problem of determining whether two sets of musical notes denote the same chord (a chord is a *set* of at least three notes, i.e., order and repetitions do not matter). We can encode the problem as a set unification problem between two (flat) ground set terms:

$$\{c, e, g, b\flat\} = \{g, g, e, b\flat, c, e\}$$

where c, e, g, $b\flat$ are constants representing musical notes (i.e., individuals of the language).

Courses covering: This is the problem of verifying whether two teachers are covering three courses in their current course assignment. The problem can be encoded as a set unification problem between a gflat(q) set term, composed only of variables, for the teachers, and a (flat) ground set term for the courses:

$$Teacher_1 \cup\ Teacher_2 = \{course_1, course_2, course_3\}$$

Note that, in this case, variables range over unions of elements (i.e., subsets of $course_1 \cup course_2 \cup course_3$) rather than simply over individuals. Thus we cannot use the list-like representation for its concrete rendering.

Graph Coloring: Let us consider the graph-coloring problem consisting of the undirected graph

$$\langle\, \{X_1, X_2, X_3, X_4\}\,,\ \{\{X_1, X_2\}, \{X_2, X_3\}, \{X_3, X_4\}, \{X_4, X_1\}\}\, \rangle$$

and a set of colors

$$\{red, green, blue\}$$

This problem can be easily encoded as a single equation between two nested(q) ($q \geqslant 1$) set terms in the following way:

$$\{\{X_1, X_2\}, \{X_2, X_3\}, \{X_3, X_4\}, \{X_4, X_1\}\} \cup R = \{\{red, green\}, \{red, blue\}, \{green, blue\}\}$$

The right-hand side set is used to encode the set of all viable unordered pairs of colors, and it can be a ground set term.

The solution of this equation (see Definition 6) provides a solution of the corresponding graph-coloring problem. A possible solution (actually, the first

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

one returned by the $CLP(\mathcal{SET})$ interpreter (Dovier *et al.* 2000)) is:

$$X_1 = red, X_2 = green, X_3 = red, X_4 = blue, R = \{\{green, blue\}\}$$

Solutions that make use of only two colors are also computed, such as $X_1 = X_3 = red$, $X_2 = X_4 = green$ and $R = \{\{red, blue\}, \{green, blue\}\}$.

Finite State Automata: Let us consider a deterministic finite state automata on the alphabet $\{0, 1\}$, containing the set of states $Q = \{q_0, \ldots, q_{n-1}, q_n\}$, where q_0 is the initial state and q_n is the unique final state. $Q_1 = \{q_0, \ldots, q_{n-1}\}$ denotes the set of non-final states of the automata. We would like to "learn" the structure of the automata by looking at positive and negative examples of strings that should be either accepted or rejected. This problem can be encoded as follows. The set of transitions D is represented by a nested(q) ($q \geqslant 0$) set term whose elements are triples (*source*, *symbol*, *destination*) (where $(\cdot, \cdot, \cdot)$ is a ternary free function symbol used to build the triples):

$$D = \{(q_0, 0, X_{0,0}), (q_0, 1, X_{0,1}), \ldots, (q_n, 0, X_{n,0}), (q_n, 1, X_{n,1})\}$$

Observe that the destination of each transition is, at this point, unknown. If $a_0 \cdots a_k$ is a string of length $k + 1$ that should be accepted, then we need to add an equation:

$$\{(q_0, a_0, Y_1), (Y_1, a_1, Y_2), \ldots, (Y_k, a_k, q_n)\} \cup D = D$$

that forces the transitions $(q_0, a_0, Y_1), (Y_1, a_1, Y_2), \ldots, (Y_k, a_k, q_n)$ to belong to D. Note that the left-hand side of the equation is a nested(q) set term ($q \geqslant 1$). Therefore, we can use the concrete list-like representation to encode it, based on the element insertion constructor $\{\cdot \mid \cdot\}$, as well as the union-based representation. If $b_0 \cdots b_h$ is a string that should not be accepted, then we need to add the equations:

$$\{(q_0, b_0, Y_1), (Y_1, b_1, Y_2), \ldots, (Y_h, b_h, Y_{h+1})\} \cup D = D \ , \ \{Y_{h+1}\} \cup Q_1 = Q_1$$

that force the state Y_{h+1} resulting from the execution to be in Q_1, and hence not a final state.

For example, if we want a four-state automata that accepts the strings 000 and 001 and rejects the strings 011 and 10, then we write the system of equations:

$$\begin{aligned}
&Q = \{q_0, q_1, q_2, q_3\}, Q_1 = \{q_0, q_1, q_2\},\\
&D = \{(q_0, 0, X_{00}), (q_0, 1, X_{01}), (q_1, 0, X_{10}), (q_1, 1, X_{11}),\\
&\qquad (q_2, 0, X_{20}), (q_2, 1, X_{21}), (q_3, 0, X_{30}), (q_3, 1, X_{31}) \ \},\\
&\{W_3\} \cup Q_1 = Q_1, \{K_2\} \cup Q_1 = Q_1,\\
&\{(q_0, 0, W_1), (W_1, 1, W_2), (W_2, 1, W_3)\} \cup D = D,\\
&\{(q_0, 1, K_1), (K_1, 0, K_2)\} \cup D = D,\\
&\{(q_0, 0, Y_1), (Y_1, 0, Y_2), (Y_2, 0, q_3)\} \cup D = D,\\
&\{(q_0, 0, Z_1), (Z_1, 0, Z_2), (Z_2, 1, q_3)\} \cup D = D
\end{aligned}$$

A possible solution (the first one returned by the $CLP(\mathcal{SET})$ interpreter) is (see also Figure 1):

$$\begin{aligned}
D \ = \ &\{(q_0, 0, q_1), (q_0, 1, q_2), (q_1, 0, q_3), (q_1, 1, q_2),\\
&(q_2, 0, q_0), (q_2, 1, q_0), (q_3, 0, q_3), (q_3, 1, q_3)\}
\end{aligned}$$

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

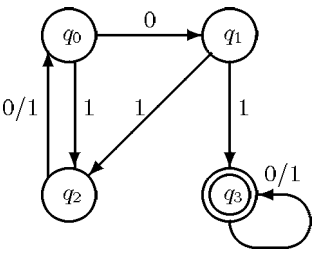


Fig. 1. The DFA computed from examples and counterexamples

Paths and Subgraphs: Let us represent an undirected graph G as the set of all its edges, where each edge is represented by the set of its two constituting nodes. Let us consider the problem of computing all the subgraphs of G with nodes $\{c_1, \ldots, c_n\}$ such that each subgraph contains at least one path between two given nodes, e.g., c_1, c_n. This problem can be immediately encoded as a set unification problem. In fact, all the subgraphs of G are given by the solutions for G_1 of the equation

$$G_1 \cup G_2 = G \tag{1}$$

The subgraphs containing the required path can be obtained by adding the equations:

$$G_3 = G_1 \cup \{\{c_1, c_1\}, \ldots, \{c_n, c_n\}\}, \quad G_3 = \{\{c_1, X_1\}, \{X_1, X_2\}, \ldots, \{X_{n-1}, c_n\}\} \cup G_3$$

Observe that G_1 is temporarily extended to the new graph G_3 by introducing artificial loops, thus allowing us to recognize paths of length less than n. Also, observe that the equation (1) cannot be rendered concretely using the list-like representation, since its left-hand side set term involves more than one variable ranging over set terms (i.e., it belongs to the nested(q) class, $q \geqslant 2$).

4 The set unification decision problem and its complexity

In this section, we discuss the complexity of the Set Unification Decision problem for each one of the syntactic classes of set terms listed in Section 2.

4.1 *SUD for the* ground *class*

The set equivalence test for two ground abstract set terms s and t can be solved in worst-case time $O(||s|| + ||t||)$ (see Definition 2). The proof is based on a tree representation of a well-founded set and on the existence of a fast algorithm for proving graph bisimulation. We first focus on the *pure* case (without individuals).

We can use a tree $G = \langle N, E \rangle$, rooted in $v \in N$, where N is the set of nodes and E is the set of edges of G, to represent a pure set. Edges represent memberships, namely $\langle m, n \rangle$ means that m has n as an element, and the nodes in the tree denote all the sets that contribute to the construction of the set. A node without outgoing edges represents the empty set $\emptyset$. It is possible to write a procedure that translates a ground set term denoting a pure set into a tree in linear time. An example showing

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

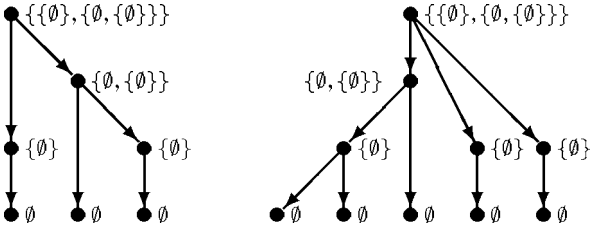


Fig. 2. Two bisimilar trees obtained from $\{\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}$ and $\{\{\{\emptyset,\emptyset\},\emptyset\},\{\emptyset\},\{\emptyset\}\}$

two trees obtained in this way is shown in Figure 2. From the figure it is possible to observe the sets associated to the subtrees.

According to Aczel (1988), a *bisimulation* between a graph $G_1 = \langle N_1, E_1\rangle$ and a graph $G_2 = \langle N_2, E_2\rangle$ is a relation $b \subseteq N_1 \times N_2$ such that:

1. $\forall u \in N_1\ \exists v \in N_2$ such that $(u\ b\ v)$ and $\forall v \in N_2\ \exists u \in N_1$ such that $(u\ b\ v)$
2. $(u_1\ b\ u_2) \wedge \langle u_1, v_1\rangle \in E_1 \Rightarrow \exists v_2 \in N_2(\ (v_1\ b\ v_2)\ \wedge\ \langle u_2, v_2\rangle \in E_2\ )$
3. $(u_1\ b\ u_2) \wedge \langle u_2, v_2\rangle \in E_2 \Rightarrow \exists v_1 \in N_1(\ (v_1\ b\ v_2)\ \wedge\ \langle u_1, v_1\rangle \in E_1\ )$.

We can use the notion of bisimulation on trees. Specifically, given a tree G_1, rooted in node v_1, and a tree G_2, rooted in node v_2, G_1 is *bisimilar* to G_2 if and only if there exists a bisimulation b between G_1 and G_2 such that $v_1\ b\ v_2$. It is simple to verify whether the two trees of Figure 2 are bisimilar. Observe that conditions 2. and 3. resemble the extensionality axiom (Section 2.1) – in fact, pure sets are equal if and only if their graph representations are bisimilar (Aczel 1988). In Dovier *et al.* (2004) it is proved that bisimilarity between *acyclic* and rooted graphs can be tested in linear time. This result is based on an algorithm that guesses an initial partition of the nodes – in particular, all leaves are initially placed in the same class – and refines it using a suitable computation strategy.

As far as sets with individuals are concerned, the situation is similar. Let us assume that $a_1, \ldots, a_m$ are the individuals occurring in the two terms. One can obtain the two trees as in the previous case, but adding a label to each node: 0 for a set node and i if the node contains the individual a_i. Then one can run the same graph algorithm as in the previous case with a single change at the beginning: the leaf nodes are split into different classes according to their labels.

Remark 1

In the procedure described above, for ground sets with individuals, we need to partition leaf nodes according to their labels (individual names). A similar problem will emerge in other procedures presented in the paper, where constants symbols and variables must be ordered. If we assume that the input is given as a string and the set of constant/variable symbols used is known in advance, then we can order them in linear time using radix sort. If we assume that the input terms are represented by trees using *structure sharing* (namely, there are no multiple occurrences of nodes representing the same constant/label), we have an implicit ordering of constants given by their memory locations. If, otherwise, the input is simply a string or a graph without structure sharing, we first need to provide the ordering of the symbols used, which requires time $O((||s|| + ||t||)\log(||s|| + ||t||))$.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

4.2 SUD for the gflat(q) class

Let q be fixed and consider two gflat(q) set terms to be tested: $s = \{a_1, \ldots, a_n\} \cup Y_1 \cdots \cup Y_{q'}$ and $t = \{b_1, \ldots, b_{n'}\} \cup Z_1 \cup \cdots \cup Z_{q''}$ ($q' \leqslant q$ and $q'' \leqslant q$). Let:

$$\begin{array}{lcllcl} V_1 & = & vars(s) \setminus vars(t) & C_1 & = & consts(s) \setminus consts(t) \\ V_2 & = & vars(t) \setminus vars(s) & C_2 & = & consts(t) \setminus consts(s) \\ V_3 & = & vars(s) \cap vars(t) & C_3 & = & consts(s) \cap consts(t) \end{array} \quad (2)$$

where $vars(\alpha)$ and $consts(\alpha)$ denote the set of variables and the set of simple individual terms occurring in the term α, respectively (see Remark 1 for a comment on the time required to determine these sets).

If $q' = q'' = 0$ (i.e., $vars(s) = vars(t) = \emptyset$), then we are in the ground case studied in the previous section.

If q' and q'' are both greater than 0 (i.e., $vars(s) \neq \emptyset$ and $vars(t) \neq \emptyset$), then s and t are always unifiable: a solution can be obtained by assigning the set $\{a_1, \ldots, a_n, b_1 \ldots, b_{n'}\}$ to all the variables in $vars(s) \cup vars(t)$.

If exactly one between q' and q'' is 0, then we have that:

- if $q' = 0$, then the problem admits a solution if and only if $C_2 = \emptyset$;
- if $q'' = 0$, then the problem admits a solution if and only if $C_1 = \emptyset$.

Thus, to solve the SUD problem for gflat(q) set terms we simply need to compute the sets C_i and V_i, a task that can be accomplished in time $O(||s|| + ||t||)$. The considerations made in Remark 1 apply to this case as well.

4.3 SUD for the flat(q) class

Let q be fixed and consider two flat(q) set terms to be tested:

$$s = \{X_1, \ldots, X_m, a_1, \ldots, a_n\} \cup Y_1 \cup \cdots \cup Y_{q'}$$

and

$$t = \{W_1, \ldots, W_{m'}, b_1, \ldots, b_{n'}\} \cup Z_1 \cup \cdots \cup Z_{q''}$$

($q' \leqslant q$ and $q'' \leqslant q$), and let V_i and C_i be the sets defined in formula (2).

If $m = m' = 0$ we are in the case gflat(q) studied before. If q' and q'' are both greater than 0, then a trivial solution always exists, as in the gflat(q) case.

If $q' = q'' = 0$, then we can observe that a necessary condition for the existence of a solution is that:

$$|V_1| + |V_2| + |V_3| \geqslant |C_1| + |C_2|, |V_1| + |V_3| \geqslant |C_2|, |V_2| + |V_3| \geqslant |C_1| \quad (3)$$

Condition (3) is also sufficient. If (3) holds, then we will be able to construct a solution by assigning a different value from C_2 to each variable in V_1, a different value from C_1 to each variable in V_2, and by assigning all remaining elements of C_1 and C_2 (if any) to the variables in V_3. Condition (3) guarantees that there are enough variables in V_3. If some variables are not assigned by this algorithm, then the solution can be easily completed. For example, when $|V_1| > |C_2|$, we can complete the solution by assigning any value from C_2 or C_3 to the remaining variables of V_1.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

Table 1. *Complexity of the SUD problem $s = t$ and E-theory used to solve the SUS problem*

	ground	gflat(q), $q > 0$	flat(q), $q = 0, 1$
SUD Complexity	$O(\|\|s\|\| + \|\|t\|\|)$	$O(\|\|s\|\| + \|\|t\|\|)$	$O(\|\|s\|\| + \|\|t\|\|)$
Theory	$(Ab)(C\ell)$	$ACI1$ with constants	$(Ab)(C\ell)$

	flat(q), $q > 1$	nested(q), $q = 0, 1$	nested(q), $q > 1$
SUD Complexity	$O(\|\|s\|\| + \|\|t\|\|)$	NP	NP
Theory	gen. $ACI1$	$(Ab)(C\ell)$	gen. $ACI1$

If exactly one of q' or q'' is 0 (without loss of generality, let us assume $q'' = 0$), then we can determine V_i and C_i as in the previous cases, but without considering the variables Y_i, Z_i. The problem admits a solution if and only if $|V_2| + |V_3| \geqslant |C_1|$.

Thus, the SUD problem for flat(q) set terms can be reduced to the problem of computing the sets V_i and C_i. This can be done in time $O(||s|| + ||t||)$ (again, see Remark 1). As discussed more extensively in Section 7.4, the class of problems flat(0) has been studied in Arni *et al.* (1992) and Greco (1996), where these set terms are called *Bound Simple set terms.*

4.4 SUD for the nested(q) *class*

The set unification test for nested sets with non-ground elements (i.e., with general individuals) has been proved to be NP-hard in Kapur and Narendran (1986) even for the simple case of nested(q) with $q = 0$. We report here the NP-hardness proof from Dovier *et al.* (1996). Let us consider an instance of 3SAT, e.g.:

$$(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee \neg X_3)$$

Checking its satisfiability is equivalent to testing set unification of the two following nested(0) set terms:

$$\{\{X_1, Y_1\}, \{X_2, Y_2\}, \{X_3, Y_3\}, \{X_1, Y_2, X_3, \emptyset\}, \{Y_1, X_2, Y_3, \emptyset\}\} \text{ and } \{\{\emptyset, \{\emptyset\}\}\}$$

where we interpret $\emptyset$ as false and $\{\emptyset\}$ as true.

To prove that the test is in NP, instead, one needs to prove that, when it is satisfiable, there exists a witness for this situation that can be verified in polynomial time. Proofs for this result are rather complex and they can be found in (Kapur and Narendran 1992; Omodeo and Policriti 1995).

4.5 Summary of results for the SUD problem

Table 1 summarizes the complexity of the SUD problem for the different classes of set terms we have introduced. The *Theory* row will be explained in the next sections.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

4.6 Equations vs. systems

We have defined the SUD and SUS problems on a single equation. The notions can be extended to deal with systems of equations as well: in this case we need to check whether all the equations in the system are simultaneously unifiable.

In the ground case nothing changes: each equation is analyzed independently. For gflat(q) we know from Kapur and Narendran (1992) and Hermann and Kolaitis (1997) that the $ACI1$ with constants unification problem for systems of equations can be reduced to propositional Horn satisfiability and, thus, it is in P. In Section 6 we prove the equivalence of this problem with the gflat(q) unification problem.

As far as the flat(q) class is concerned, we can adapt the reduction of the 3SAT problem done for the nested(q) class, using the constant 1 instead of the set $\{\emptyset\}$. The instance of 3SAT is mapped to the system of equations:

$$\{X_1, Y_1\} = \{\emptyset, 1\}, \{X_2, Y_2\} = \{\emptyset, 1\}, \{X_3, Y_3\} = \{\emptyset, 1\},$$
$$\{X_1, Y_2, X_3, \emptyset\} = \{\emptyset, 1\}, \{Y_1, X_2, Y_3, \emptyset\} = \{\emptyset, 1\}$$

where all equations involve only flat(q) set terms. Thus, while the flat(q) SUD problem for a single equation requires linear time, the same problem for systems of equations is NP-complete.

Regarding the nested(q) class, each system of equations $\{s_1 = t_1, \ldots, s_n = t_n\}$ can be polynomially reduced to an equisatisfiable equation as follows:

$$\{(\underline{1}, s_1), \ldots, (\underline{n}, s_n)\} = \{(\underline{1}, t_1), \ldots, (\underline{n}, t_n)\}$$

where $\underline{n}$ is a polynomial encoding of the natural number n (e.g., $\underline{0} = \emptyset$, $\underline{n+1} = \{\underline{n}\}$) and (x, y) is an encoding of the ordered pair (e.g., $(x, y) = \{\{x\}, \{x, y\}\}$). Thus, the complexity of the problem on systems of equations is the same as for a single equation.

5 E-unification

E-unification is concerned with solving term equations modulo an equational theory E. Set unification can be seen as an instance of the E-unification problem, where the underlying equational theory contains the identities that capture the properties of set terms, i.e., the fact that the ordering and repetitions of elements in a set are immaterial. Different approaches have been considered to encode sets. Accordingly, different choices of E should be considered to describe their basic properties.

We assume the reader to be familiar with the notions of equational theory, E-unification, E-unifier and related topics (Siekmann 1989; Baader and Snyder 2001). In this section we introduce a few basic notations concerning E-unification and set unification which will be useful in the rest of the paper.

A *signature* Σ consists of a set of function symbols. Terms built from Σ and from a denumerable set $\mathcal{V}$ of variables are called Σ-*terms*. $\mathcal{T}(\Sigma, \mathcal{V})$ is the set of all the Σ-terms – and it is called the *term algebra*. Given a sequence of terms $t_1, \ldots, t_n$, $vars(t_1, \ldots, t_n)$ denotes the set of variables occurring in the terms. *vars* is naturally extended to equations and sets of equations.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

A *substitution* $\sigma : \mathcal{V} \longrightarrow \mathcal{T}(\Sigma, \mathcal{V})$ is represented by the notation $[X_1/t_1, \ldots, X_n/t_n]$, where $dom(\sigma) = \{X_1, \ldots, X_n\}$ (the *domain* of σ) and, for each $i = 1, \ldots, n$, $\sigma(X_i) = t_i$. A substitution is uniquely extended to a function over $\mathcal{T}(\Sigma, \mathcal{V})$ using structural induction. The application of a substitution σ to a term t will be denoted by $t\sigma$ (or, equivalently, by $\sigma(t)$).

An *equational theory* is a finite collection of identities E, where each identity is written as $s \approx t$, and s, t are terms belonging to $\mathcal{T}(\Sigma, \mathcal{V})$. The relation $=_E$ is the *least congruence relation* on the term algebra $\mathcal{T}(\Sigma, \mathcal{V})$, which contains E and it is closed under substitution (Baader and Snyder 2001). Function symbols not occurring in E are said to be *free*.

We describe now the properties of the function symbols that we use as the set constructors. The properties that the $\cup$ constructor should have in a set theory can be described by the following identities:

$$\begin{array}{lrcll} (A) & (X \cup Y) \cup Z & \approx & X \cup (Y \cup Z) & (\textit{Associativity}) \\ (C) & X \cup Y & \approx & Y \cup X & (\textit{Commutativity}) \\ (I) & X \cup X & \approx & X & (\textit{Idempotence}) \end{array}$$

Moreover, the constant symbol $\emptyset$, used to denote the empty set, is the identity element for the $\cup$ operator. This is stated by:

$$(1) \quad X \cup \emptyset \approx X \quad (\textit{Identity})$$

Let E_{ACI1} be the equational theory consisting of identities (A), (C), (I), and (1).

The $\{\cdot \mid \cdot\}$ constructor, instead, should exhibit the properties described by the following identities:

$$\begin{array}{lrcll} (Ab) & \{X \mid \{X \mid Z\}\} & \approx & \{X \mid Z\} & (\textit{Absorption}) \\ (C\ell) & \{X \mid \{Y \mid Z\}\} & \approx & \{Y \mid \{X \mid Z\}\} & (\textit{Commutativity on the left}) \end{array}$$

A substitution σ is an *E-unifier* (or, simply, a *unifier* when the context is clear) of two terms s, t if $s\sigma =_E t\sigma$ – i.e., $s\sigma$ and $t\sigma$ belong to the same E-congruence class.

An *E-unification problem* over Σ is a system of equations $\mathcal{E} = \{s_1 = t_1, \ldots, s_n = t_n\}$ between Σ-terms. A substitution μ which is an E-unifier of all the equations in $\mathcal{E}$ is said to be an E-unifier (or an *E-solution*) of $\mathcal{E}$. The set of all the E-unifiers of $\mathcal{E}$ is denoted by $\mathcal{U}_E(\mathcal{E})$.

Let E be an equational theory and $\mathcal{W}$ a set of variables ($\mathcal{W} \subseteq \mathcal{V}$). $\mathcal{U}_E(\mathcal{E})$ can be sorted with respect to the pre-order $\leqslant_E^{\mathcal{W}}$: given two substitutions σ_1, σ_2:

$$\sigma_1 \leqslant_E^{\mathcal{W}} \sigma_2 \quad \text{iff} \quad \begin{array}{l} \text{there exists a substitution } \lambda \text{ such that} \\ \sigma_2(X) =_E (\sigma_1 \circ \lambda)(X) \text{ for all } X \text{ in } \mathcal{W}. \end{array}$$

In this case we say that σ_1 is *more general modulo E on* $\mathcal{W}$ than σ_2. If $\sigma_1 \leqslant_E^{\mathcal{W}} \sigma_2$ and $\sigma_2 \leqslant_E^{\mathcal{W}} \sigma_1$, then we say that $\sigma_1 =_E^{\mathcal{W}} \sigma_2$. Whenever $\mathcal{W}$ is omitted from $\leqslant_E^{\mathcal{W}}$, then $\mathcal{W}$ is implicitly assumed to be $vars(\mathcal{E})$.

While traditional syntactic unification problems between Herbrand terms admit at most one most general unifier (mgu), E-unification problems may not have a single most general unifier. In this context, the role of the most general unifier is taken on by a minimal complete set of unifiers. A *complete set of E-unifiers* for an

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

E-unification problem $\mathscr{E}$ is a set $\mathscr{C}$ of E-unifiers (i.e., a subset of $\mathscr{U}_E(\mathscr{E})$) that satisfies the additional condition:

- for each E-unifier σ there exists an element θ in $\mathscr{C}$ such that $\theta \leqslant_E \sigma$.

A complete set of E-unifiers $\mathscr{C}$ is called a *minimal complete set of E-unifiers* if it fulfills the *minimality* condition:

- for any pair μ_1, μ_2 in $\mathscr{C}$, if $\mu_1 \leqslant_E \mu_2$, then $\mu_1 = \mu_2$.

A substitution σ in a minimal complete set of E-unifiers $\mathscr{C}$ is called a *maximal general E-unifier*. When $\mathscr{C}$ is a singleton set $\{\sigma\}$ we say that σ is the *most general E-unifier*. If one minimal set of E-unifiers can be obtained from another one by variable renaming and vice versa, the two sets are equivalent and only one of them needs to be computed.

A special form of systems of equations, called the *solved form*, plays an important role in the definition of unification algorithms. An equation e of the form $X = t$ is said to be in *solved form* with respect to a system $\mathscr{E}$ if X does not occur neither in t nor elsewhere in $\mathscr{E}$. In this case, X is said to be a *solved* variable in $\mathscr{E}$. A system $\mathscr{E}$ is said to be in solved form if, for all e in $\mathscr{E}$, e is in solved form with respect to $\mathscr{E}$. From a system in solved form $\{X_1 = t_1, \ldots, X_n = t_n\}$, it is simple to derive the most general E-unifier $[X_1/t_1, \ldots, X_n/t_n]$.

E-unification problems can be classified according to whether their signature Σ contains free elements (i.e., function symbols that do not occur in E). In particular, it is possible to distinguish between:

- *elementary unification*, where the terms to be unified are built only using the symbols appearing in the considered equational theory;
- *unification with constants*, where the terms to be unified are built using symbols in the equational theory and additional free constants;
- *general unification*, where the terms to be unified are arbitrary terms containing function symbols which are either free or present in the equational theory.

The unification problem studied in the next section falls in the class of unification with constants. The remaining sections consider general unification problems.

The SUD problem studied in Section 4 is an abstract case of the *E-unifiability problem* (namely, deciding whether or not an E-unifier exists). In the next sections we deal with the SUS problem, i.e., the problem of determining a *complete set* of E-unifiers of an equation $s = t$ or of a system of equations $\mathscr{E}$.

6 $ACI1$ with constants

According to the classification presented in Section 2 the simplest non-ground set terms we deal with are those belonging to the $\mathsf{gflat}(q)$ class. In this section we show that the SUS problem for this class can be solved by using the solution to the $ACI1$ with constants unification problem.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

6.1 Language and semantics

Let $\Sigma = \{\emptyset, \cup, c_1, c_2, \ldots\}$ be a signature composed of the binary function symbol $\cup$, the constant symbol $\emptyset$, and an *arbitrary* number (possibly infinite) of free constant symbols $c_1, c_2, \ldots$

Definition 8
An *ACI*1 *with constants term* is either a variable, a constant, or a Σ-term of the form $s_1 \cup s_2$, where s_1 and s_2 are *ACI*1 with constants terms.

The properties of the function symbols $\cup$ and $\emptyset$ are described by the identities (A), (C), (I), and (1) introduced in Section 5. Thanks to the associativity property (A), *ACI*1 with constants terms can be always written as strings of the form $\alpha_1 \cup \cdots \cup \alpha_m$ where α_i is either a variable, $\emptyset$, or a constant term c_i. Moreover, using $(C), (I)$, and (1) we can restrict our attention to terms without duplications of sub-terms and without $\emptyset$ as a sub-term (unless the whole term is $\emptyset$).

Flat set terms with variable elements (i.e., $\mathsf{flat}(q)$ set terms) are not expressible in this language. Indeed the language does not allow us to distinguish individuals from sets. Variables in a set term are always interpreted as set variables. Furthermore, nested set terms are not expressible in this language (Dovier *et al.* 2000).

6.2 Which kind of set unification

The *ACI*1 with constants language allows us to describe the set unification problem for $\mathsf{gflat}(q)$ set terms. The SUS problem for this class can be solved using the solution to the corresponding *ACI*1 with constants unification problem (defined below). As an example, let us consider the $\mathsf{gflat}(q)$ unification problem:

$$\{a, b\} \cup Y_1 \cup Y_2 = \{a, b, c, d\}$$

The solutions for this problem are those mapping Y_1 and Y_2 to subsets of $\{a, b, c, d\}$ such that c and d are in the image of Y_1 or Y_2. For instance, $[Y_1/\{a, c\}, Y_2/\{a, b, d\}]$ is a solution. Let us consider now the related *ACI*1 with constants unification problem:

$$a \cup b \cup Y_1 \cup Y_2 = a \cup b \cup c \cup d$$

In this case, a, b, c, d are not interpreted as set elements. However, thanks to the properties of the $\cup$ operator, the solutions for this problem are closely related to those for the $\mathsf{gflat}(q)$ unification problem. The solutions for the *ACI*1 with constants unification problem are those mapping Y_1 and Y_2 to unions of elements of $\{a, b, c, d\}$ such that c and d are in the image of Y_1 or Y_2. For instance, $[Y_1/a \cup c, Y_2/a \cup b \cup d]$.

We formalize this idea by defining a function $(\cdot)^*$ that translates $\mathsf{gflat}(q)$ set terms into *ACI*1 with constants terms as follows:

$$(\{a_1, \ldots, a_n\} \cup Y_1 \cup \cdots \cup Y_q)^* \stackrel{\mathscr{D}ef}{=} a_1 \cup \cdots \cup a_n \cup Y_1 \cup \cdots \cup Y_q$$

$(\cdot)^*$ admits an inverse function. The function can also be extended to substitutions: $\sigma^*(X) = (\sigma(X))^*$.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

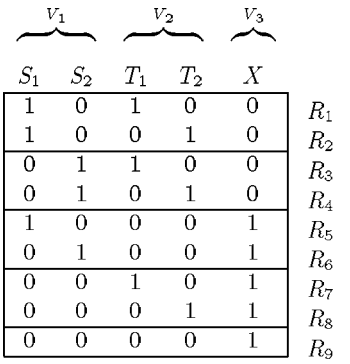


Fig. 3. The ACI-matrix for the problem $S_1 \cup S_2 \cup X = T_1 \cup T_2 \cup X$

Lemma 1
σ is a solution of the gflat(q) SUS problem $s = t$ if and only if σ^* is a $ACI1$ unifier of $s^* = t^*$.

For the proof, see Appendix A.

Example 2
The following are set terms and set unification problems which are allowed in $ACI1$ with constants:

- $X_1 \cup X_2 \cup X_3 = X_4 \cup X_1$
- $a \cup b \cup X_1 \cup X_2 = c \cup X_3$ – that is $(\{a, b\} \cup X_1 \cup X_2)^* = (\{c\} \cup X_3)^*$
- the first problem of Section 3 (the *Chords* problem) can be encoded as the $ACI1$ with constants problem $c \cup e \cup g \cup b\flat = g \cup g \cup e \cup b\flat \cup c \cup e$.

6.3 Unification algorithm

A general algorithm capable of computing a minimal complete set of $ACI1$-unifiers for $ACI1$ with constants unification problems has been presented in Baader and Büttner (1988).

Given two Σ-terms s and t the algorithm computes a complete set S of $ACI1$-unifiers for $s = t$. Without loss of generality, we assume that if only one of the terms is ground, then it is t. The set S can be extracted from a schema of Boolean ACI-matrices. Each column of the matrix is associated to a variable in $s = t$. Each row, instead, is associated to new variables that will enter in the solutions. The matrix is composed of identity matrices, by matrices of 0 with exactly one column set to 1, and by 0 matrices.

Example 3
Let us consider the problem:

$$S_1 \cup S_2 \cup X = T_1 \cup T_2 \cup X$$

The sets $V_1, V_2, V_3, C_1, C_2, C_3$ are computed as in formula (2) of Section 4.2: $V_1 = \{S_1, S_2\}$, $V_2 = \{T_1, T_2\}$, $V_3 = \{X\}$, and $C_1 = C_2 = C_3 = \emptyset$. Since the given problem does not involve constants, the matrix is unique (see Figure 3). $R_1, \ldots, R_9$ are new

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

variables that allow to compactly represent the unique mgu:

$$\begin{bmatrix} S_1 & / & R_1 \cup R_2 \cup R_5, \\ S_2 & / & R_3 \cup R_4 \cup R_6, \\ T_1 & / & R_1 \cup R_3 \cup R_7, \\ T_2 & / & R_2 \cup R_4 \cup R_8, \\ X & / & R_5 \cup R_6 \cup R_7 \cup R_8 \cup R_9 \end{bmatrix}$$

The two 1's in a row state that the two variables should have a part in common in each solution. For instance, in the first row it is stated that R_1 is a part of S_1 and of T_1 (in other words, $R_1 = S_1 \cap T_1$).

When the problem involves constants, the matrices have also rows for C_1, C_2, C_3. In this case several matrices are non-deterministically generated. Each of them describes a solution; their union covers the whole solution space.

Example 4
Let us consider the problem:

$$X_1 \cup X_2 \cup X_3 = a \cup b$$

where $V_1 = \{X_1, X_2, X_3\}, C_2 = \{a, b\}$, $V_2 = V_3 = C_1 = C_3 = \emptyset$. There are 49 ACI-matrices for this problem. Two of them are:

V_1			
X_1	X_2	X_3	
0	1	1	a
1	0	1	b $\}$ C_2

V_1			
X_1	X_2	X_3	
0	0	1	a
0	1	0	b $\}$ C_2

yielding the unifiers: $[X_1/b, X_2/a, X_3/a \cup b]$, $[X_1/\emptyset, X_2/b, X_3/a]$.

The number of ACI-matrices to be computed for a given $ACI1$ unification problem is $(2^{|V_2|}-1+|V_3|)^{|C_1|}(2^{|V_1|}-1+|V_3|)^{|C_2|}(2^{|V_1|}+2^{|V_2|}-1)^{|C_3|}$ which is $O(2^{(||s||+||t||)^2})$ (Baader and Büttner 1988).

The detection of a solution of a unification problem (i.e., solving the SUS problem) clearly implies solving the related decision problem. Thus, the complexity of finding a solution can be no better than the complexity of solving the corresponding decision problem. In this case, both the problems can be solved in linear time (with the assumption in Remark 1). This can be achieved as follows. First verify that the decision problem $s = t$ has a positive answer; this can be done in linear time thanks to the results in Lemma 1 and Section 4.2. If the test succeeds, then a solution can be constructed by assigning to each variable X in $s = t$ a term composed of the union of all the constants present in $s = t$. For further details the reader is referred to Baader and Büttner (1988).

6.4 Discussion

A simpler unification problem – called the *elementary* $ACI1$ unification problem – has been considered in the literature. This problem involves terms which are

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

constructed using only variables, the constant $\emptyset$, and the binary constructor $\cup$ (i.e., a subcase of $\mathsf{gflat}(q)$ with $n = 0$ and $q \geqslant 0$). This problem is simpler in the sense that the decision problem has always a positive answer, i.e., each unification problem $s = t$ has a solution. Therefore, the complexity of finding an arbitrary solution is $O(1)$. Furthermore, each elementary $ACI1$ unification problem admits a single most general unifier. In Appendix B we show a variant of the ACI-matrices for this simplified problem.

As a final remark, Hermann and Kolaitis (1997) and Kapur and Narendran (1992) show how the result presented in this section can be extended to provide a polynomial time solution to systems of $ACI1$ with constants unification problems.

7 General $(Ab)(C\ell)$ unification

Set terms involving variable elements and/or nested sets are not expressible in the language of $ACI1$ with constants (see Section 6.2). The proposal we describe in this section is intended to enlarge the domain of discourse to the more general class of $\mathsf{nested}(q)$ set terms with $q \leqslant 1$. As already observed at the end of Section 2.3, in this case we can rely on the element insertion operator $\{\cdot \mid \cdot\}$ as the set constructor for the concrete implementation of sets. This choice allows the presence of at most one set variable in each set term, while $ACI1$ with constants does not place any restriction on the number of set variables which can occur in each set term. On the other hand, it allows us to represent *nested* sets – which is not possible using $ACI1$ with constants unification. Moreover, it allows sets to be viewed and manipulated in a fashion similar to *lists*. As a matter of fact, this approach has been adopted in a number of logic and functional-logic programming languages (e.g., $CLP(\mathcal{SET})$ (Dovier and Rossi 1993; Dovier *et al.* 2000), SEL (Jayaraman 1992), SETA (Arenas-Sánchez and Rodríguez-Artalejo 2001)).

The unification algorithm we propose here is similar to the one presented in Dovier *et al.* (1996) – but with a considerably simpler termination proof. The underlying equational theory contains the two identities (Ab) and $(C\ell)$ shown in Section 5, stating the fundamental properties of the set constructor $\{\cdot \mid \cdot\}$.

7.1 Language and semantics

Σ is a signature containing the binary function symbol $\{\cdot \mid \cdot\}$, the empty set constant symbol $\emptyset$, and an arbitrary number (possibly infinite) of free function symbols with arbitrary arities.

Definition 9
An $(Ab)(C\ell)$ *set term* is either a variable, or the constant $\emptyset$, or a Σ-term of the form $\{t \mid s\}$, where t is a Σ-term and s is an $(Ab)(C\ell)$ set term. An *individual term* is either a variable or a Σ-term of the form $f(t_1, \ldots, t_n)$ with $f \not\equiv \{\cdot \mid \cdot\}, f \not\equiv \emptyset$ and $t_1, \ldots, t_n$ are Σ-terms (if $n = 0$ it is a constant term).

The function symbol $\{\cdot \mid \cdot\}$ has the properties described by the identities (Ab) and $(C\ell)$ introduced in Section 5. Hence, set terms denote hereditarily finite sets based

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

```
AbCl_unify(E_in) :
   E_s := ∅; E_ns := E_in; E_aux := ∅;
   E := ⟨E_s, E_ns, E_aux⟩;
   while E_ns ≠ ∅ or E_aux ≠ ∅ do
      if E_aux ≠ ∅ then e := pop(E_aux)
      else select arbitrarily an equation e from E_ns and remove it;
      AbCl_unify_actions(E,e);
   AbCl_unify_final(E)
```

Fig. 4. General $(Ab)(C\ell)$ Unification Algorithm (main)

on $\mathcal{U}$, while individual terms denote arbitrary elements of the universe $\mathcal{U}$. As a notational convenience $\{s_1 \mid \{s_2 \mid \cdots \{s_n \mid t\} \cdots\}\}$ will be written as $\{s_1, \ldots, s_n \mid t\}$ or simply as $\{s_1, \ldots, s_n\}$ when t is $\emptyset$.

7.2 Which kind of set unification

The $(Ab)(C\ell)$ language allows us to describe the SUD and SUS problems for nested(1) set terms, i.e., arbitrary nested sets with at most one set variable per set term. In particular, the language allows us to deal with all classes of set terms that are included in nested(1), namely ground, gflat(1), and flat(1).

Example 5
The following are set terms and set unification problems which are allowed in $(Ab)(C\ell)$:

- $\{X, \{Y\}\} = \{Z, \emptyset\}$
- $\{\{X_1, a\} \mid Y_1\} = \{X_3 \mid Y_2\}$ (i.e., in abstract syntax – cf. Section 2 – $\{\{X_1, a\}\} \cup Y_1 = \{X_3\} \cup Y_2$)
- the *Graph coloring* problem of Section 3 can be encoded as an $(Ab)(C\ell)$ problem:

$$\{\{X_1, X_2\}, \{X_2, X_3\}, \{X_3, X_4\}, \{X_4, X_1\} \mid R\} = \{\{c_1, c_2\}, \{c_1, c_3\}, \{c_2, c_3\}\}$$

On the other hand, the problem $A \cup B \cup C = \{a\} \cup D$ cannot be expressed in this framework.

7.3 Unification algorithm

The algorithm consists of three parts. The first part (AbCl_unify – see Figure 4) chooses one equation at a time using a semi-deterministic strategy. The second part (AbCl_unify_actions – see Figure 5) performs the rewriting of the selected equation. A final processing of *membership equations*, i.e., equations of the form $X = \{t_0, \ldots, t_n \mid X\}$ with $X \notin vars(t_0, \ldots, t_n)$, (AbCl_unify_final – see Figure 5) constitutes the third and final part of the algorithm.

The system $\mathcal{E}$ is split into three parts: $\mathcal{E}_{\mathtt{s}}$ is the solved form part (initially set to empty), $\mathcal{E}_{\mathtt{ns}}$ is a system of equations (initially set to the input system $\mathcal{E}_{\mathtt{in}}$), and $\mathcal{E}_{\mathtt{aux}}$ is a system of equations dealt with as a stack. For $\mathcal{E}_{\mathtt{aux}}$ we assume the existence of a push operation that puts an equation on the top of the stack and of a pop

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

`AbCl_unify_actions(`$\mathscr{E}, e$`):`
`  case` e `of`

(1)	$X = X$	$\mapsto$	$\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}}$
(2)	$t = X$ t is not a variable	$\mapsto$	$\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}} \wedge (X = t)$
(3)	$X = f(t_1, \ldots, t_n)$ $f \not\equiv \{\cdot \mid \cdot\}$ and X occurs in $f(t_1, \ldots, t_n)$	$\mapsto$	`fail`
(4)	$X = \{t_0, \ldots, t_n \mid t\}$ $t \not\equiv \{\cdots\}$ and X occurs in t $(X \not\equiv t)$, or X occurs in $t_0, \ldots, t_n$	$\mapsto$	`fail`
(5)	$X = t$ X does not occur in t	$\mapsto$	$\mathscr{E}_{\mathtt{s}} := \mathscr{E}_{\mathtt{s}}[X/t] \wedge (X = t)$; $\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}}[X/t]; \mathscr{E}_{\mathtt{aux}} := \mathscr{E}_{\mathtt{aux}}[X/t]$
(6)	$f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$ $f \not\equiv g$	$\mapsto$	`fail`
(7)	$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ $f \not\equiv \{\cdot \mid \cdot\}$	$\mapsto$	$\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}} \wedge (s_1 = t_1 \wedge \ldots \wedge s_n = t_n)$
(8)	$\{t \mid s\} = \{t' \mid s'\}$	$\mapsto$	`AbCl_step(`$\mathscr{E}, \{t \mid s\} = \{t' \mid s'\}$`)`

`AbCl_step(`$\mathscr{E}, \{t \mid s\} = \{t' \mid s'\}$`) :`
`  if tail(`s`) and tail(`s'`) are not the same variable then choose one among:`
(i) $\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}} \wedge (t = t')$;`push(`$s = s'$, $\mathscr{E}_{\mathtt{aux}}$`)`
(ii) $\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}} \wedge (t = t')$;`push(`$\{t \mid s\} = s'$, $\mathscr{E}_{\mathtt{aux}}$`)`
(iii) $\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}} \wedge (t = t')$;`push(`$s = \{t' \mid s'\}$, $\mathscr{E}_{\mathtt{aux}}$`)`
(iv) `push(`$s = \{t' \mid N\}$, $\mathscr{E}_{\mathtt{aux}}$`);push(`$\{t \mid N\} = s'$, $\mathscr{E}_{\mathtt{aux}}$`)`
N new variable
`  else let` $\{t \mid s\} \equiv \{t_0, \ldots, t_m \mid X\}$ `and` $\{t' \mid s'\} \equiv \{t'_0, \ldots, t'_n \mid X\}$, X `variable;`
`  select arbitrarily` i `in` $\{0, \ldots, n\}$`; choose one among:`
(i) $\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}} \wedge (t_0 = t'_i)$;`push(`$\{t_1, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_{i-1}, t'_{i+1}, \ldots, t'_n \mid X\}, \mathscr{E}_{\mathtt{aux}}$`)`
(ii) $\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}} \wedge (t_0 = t'_i)$;`push(`$\{t_0, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_{i-1}, t'_{i+1}, \ldots, t'_n \mid X\}, \mathscr{E}_{\mathtt{aux}}$`)`
(iii) $\mathscr{E}_{\mathtt{ns}} := \mathscr{E}_{\mathtt{ns}} \wedge (t_0 = t'_i)$;`push(`$\{t_1, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_n \mid X\}, \mathscr{E}_{\mathtt{aux}}$`)`
(iv) `push(`$X = \{t_0 \mid X\}, \mathscr{E}_{\mathtt{aux}}$`);push(`$\{t_1, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_n \mid X\}, \mathscr{E}_{\mathtt{aux}}$`)`

`AbCl_unify_final(`$\mathscr{E}$`) :`
`  Repeatedly perform any of the following actions;`
`  if neither applies then stop with success;`

(1)	$X = \{t^0_0, \ldots, t^0_{n_0} \mid X\} \wedge \ldots \wedge X = \{t^k_0, \ldots, t^k_{n_k} \mid X\} \wedge \mathscr{E}$ $k > 0$, the number of all membership equations involving X X does not occur in $t^0_0, \ldots, t^0_{n_0}, \ldots, t^k_0, \ldots, t^k_{n_k}$	$\mapsto$	$X = \{t^0_0, \ldots, t^0_{n_0}, \ldots, t^k_0, \ldots, t^k_{n_k} \mid N\} \wedge \mathscr{E}[X/\{t^0_0, \ldots, t^0_{n_0}, \ldots, t^k_0, \ldots, t^k_{n_k} \mid N\}]$
(2)	$X = t \wedge \mathscr{E}$ X occurs in t	$\mapsto$	`fail`

Fig. 5. General $(Ab)(C\ell)$ unification rewriting rules

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

operation that returns and removes the equation on the top of the stack. Given a system of equations $\mathscr{E}_{\mathtt{in}}$, the algorithm non-deterministically returns either `fail` or a collection of systems in solved form.

In the algorithm we make use of the function `tail`, defined as follows:

$$\mathtt{tail}(t) \quad = \quad \begin{cases} t & \text{if } t \text{ is a variable or a term } f(t_1,\ldots,t_n),\ f \not\equiv \{\cdot\,|\,\cdot\} \\ \mathtt{tail}(t_2) & \text{if } t \text{ is } \{t_1\,|\,t_2\} \end{cases}$$

For instance, if $s = \{a, b\}$, namely $s = \{a\,|\,\{b\,|\,\emptyset\}\}$, then $\mathtt{tail}(s) = \emptyset$.

The core of the unification algorithm (Figure 5) is very similar in structure to the traditional unification algorithms for standard Herbrand terms (Martelli and Montanari 1982). In particular, rule (1) is also known as the *Trivial* rule, rule (2) as the *Orient* rule, rules (3) and (4) are the *Occurs Check* rules, rule (5) is known as the *Variable Elimination* rule, rule (6) as the *Symbol Clash* rule, and rule (7) as the *Term Decomposition* rule (Jouannaud and Kirchner 1991). The main difference is represented by the presence of rule (8), whose aim is the reduction of equations between two set terms. Reduction of this kind of equations is performed by the procedure `AbCl_step` (see Figure 5) that implements the two identities (Ab) and ($C\ell$). (Ab) and ($C\ell$) are equivalent, for terms denoting sets, to the following axiom (Dovier *et al.* 1998):

$$(E_k^s) \quad \forall Y_1 Y_2 V_1 V_2 \quad \left(\begin{array}{r} \{Y_1\,|\,V_1\} = \{Y_2\,|\,V_2\} \ \leftrightarrow \\ (Y_1 = Y_2 \wedge V_1 = V_2) \vee \\ (Y_1 = Y_2 \wedge V_1 = \{Y_2\,|\,V_2\}) \vee \\ (Y_1 = Y_2 \wedge \{Y_1\,|\,V_1\} = V_2) \vee \\ \exists K\,(V_1 = \{Y_2\,|\,K\} \wedge V_2 = \{Y_1\,|\,K\}) \end{array} \right)$$

which can be easily converted into a rewriting rule to be used in the unification algorithm. (E_k^s) is in a sense a "syntactic version" of the extensionality axiom, which allows the extensionality property to be expressed in terms of only equations, without having to resort to any membership, universal quantifiers, or inclusion operation. (E_k^s) allows also to account for equations of the form

$$\{t_0,\ldots,t_m\,|\,X\} = \{t'_0,\ldots,t'_n\,|\,X\},$$

where the two sides are set terms with the same variable as tail element. Unfortunately, a blind application of the rewriting rule obtained from (E_k^s) would lead to non-termination in this situation. This is the reason why this case has been isolated and dealt with as special in the algorithm (within the procedure `AbCl_step`), actually splitting the rewriting rule obtained from (E_k^s) into two distinct rules.

A call to `AbCl_step` introduces equations in the stack $\mathscr{E}_{\mathtt{aux}}$ that are immediately processed. This generates a deterministic sequence of actions. We refer to the sequence of actions performed until the stack becomes empty as the *global effect* of `AbCl_step`.

Membership equations, i.e., equations of the form $X = \{t_0,\ldots,t_n\,|\,X\}$, with $X \notin vars(t_0,\ldots,t_n)$, are not dealt with by any rule of `AbCl_unify_actions`. This kind of equations turns out to be satisfiable for any X containing $t_0,\ldots,t_n$ since duplicates are immaterial in a set thanks to (Ab) and ($C\ell$) (this justifies the name

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

membership equations). These equations are processed at the end of `AbCl_unify` by the procedure `AbCl_unify_final`. Also, observe that the occur-check test performed by the standard unification algorithm is modified accordingly, so as to distinguish this special case from others (rules (3) and (4)).

Correctness and completeness of the algorithm presented in this paper derive immediately from the similar algorithm of Dovier *et al.* (1996). The termination proof for this algorithm, however, turns out to be simpler than that in Dovier *et al.* (1996), since here we rely on a more deterministic strategy, and we provide a separate treatment of membership equations. Basically, in the algorithm of this paper we avoid the repeated application of the rewriting rule:

$$X = \{t_0, \ldots, t_n \mid X\} \mapsto X = \{t_0, \ldots, t_n \mid N\}$$

that increases the number of variables in the algorithms in Dovier *et al.* (1996, 2000). This change allows the number of variables in the system to be kept under control. The simpler termination proof can be found in Appendix A.

Example 6
Let us consider the unification problem

$$\{X_1, X_2, X_3\} = \{a, b, c\}$$

(i.e., $\{X_1 \mid \{X_2 \mid \{X_3 \mid \emptyset\}\}\} = \{a \mid \{b \mid \{c \mid \emptyset\}\}\}$). The algorithm `AbCl_unify` returns the following six independent solutions that constitute the minimal complete set of E-unifiers for the given unification problem:

$$\begin{array}{ll} X_1 = a, X_2 = b, X_3 = c & X_1 = c, X_2 = a, X_3 = b \\ X_1 = a, X_2 = c, X_3 = b & X_1 = b, X_2 = c, X_3 = a \\ X_1 = b, X_2 = a, X_3 = c & X_1 = c, X_2 = b, X_3 = a \end{array}$$

In general, the algorithm `AbCl_unify` may open a large – though finite – number of alternatives, possibly leading to redundant solutions. Arenas-Sánchez and Dovier (1997) and Stolzenburg (1999) show how to improve the algorithm to minimize the number of redundant unifiers.

7.4 Discussion

The problem of finding solutions we tackle here extends the satisfiability problem for set unification (i.e., the SUD problem), shown to be NP-complete (c.f. Section 4.4). To be precise, we mean that there exists an algorithm on a non-deterministic machine that can also find the answer (the correct class is FNP). Omodeo and Policriti (1995) propose a methodology to guess a solution of a conjunction of literals built using variables, the constant symbol $\emptyset$, the function symbol $\{\cdot \mid \cdot\}$ and the predicate symbols $=, \in, \cup, \cap$, and $\setminus$. The unification problem is the particular case where only positive literals based on the equality predicate $=$ are used. A *guess* is represented by a graph containing a number of nodes polynomially bounded by the number of variables in the original problem. Verification of whether a guess is a solution of the problem can be done in polynomial time. Omodeo and Policriti (1995) also show how this technique can be extended to the general problem with

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

free function symbols – the one we deal with in this paper. A non-deterministic algorithm based on a "guess-and-verify" technique has also been proposed in Kapur and Narendran (1986).

The algorithm presented here, as well as those in Dovier *et al.* (1996) and Arenas-Sánchez and Dovier (1997), have the common drawback that, due to the explicit application of substitutions during the solving process they have a computational complexity which falls outside of the FNP class. Nevertheless, it is possible to encode this algorithm using well-known techniques – such as multi-equations or graphs with structure sharing (Martelli and Montanari 1982; Paterson and Wegman 1978) – that allow us to maintain a polynomial time complexity along each non-deterministic branch of the computation. For instance, in Aliffi *et al.* (1999) a goal driven algorithm in FNP for non-well-founded and hybrid sets has been presented. In that paper it is also shown how to use the algorithm for well-founded sets, to solve the problem dealt with in this section. A similar result is presented in Dantsin and Voronkov (1999). A detailed discussion of such kinds of enhancements, however, is outside the scope of this paper.

As far as the size of the computed complete set of unifiers is concerned, we can observe that the algorithm opens, for each level of nesting, a number of alternatives equivalent to the number of solutions returned by the global effect of `AbCl_step`. This number is no greater than the size of the minimal complete set of $(Ab)(C\ell)$-unifiers for the problem:

$$\{X_1, \ldots, X_h \mid M\} = \{X_{h+1}, \ldots, X_n \mid N\}$$

This value has a rough upper bound equal to $O(2^{n \lg n})$ (Arenas-Sánchez and Dovier 1997). Since this process can be repeated once for each nesting, a rough upper bound to the number of solutions is $O(2^{n^2 \lg n})$.

Various authors have considered simplified versions of the $(Ab)(C\ell)$ problem obtained by imposing restrictions on the form of the set terms. Most notable is the use of sets in the context of relational and deductive databases (Liu 1998; Abiteboul and Grumbach. 1991; Naqvi and Tsur 1989; Lim and Ng 1997). Typical restrictions which have been considered are flat and completely specified set terms, i.e., elements either of the $\mathsf{gflat}(q)$ or $\mathsf{flat}(0)$ classes. Specialized algorithms have been provided for some of these cases. In particular, various works have been proposed to study the simpler case of matching and unification of *Bound Simple* set terms (Greco 1996), i.e., elements of $\mathsf{flat}(0)$. These restrictions are sufficient to make the task of deciding unifiability between set terms very simple – as also discussed in Section 4.

Let us illustrate the results in the simple case of matching (Arni *et al.* 1996) (the approach has been generalized to sequential unification in Greco (1996) and to parallel unification in Lim and Ng (1997)). In the case of matching, the two set terms s and t to be unified can be written as:

$$s \equiv \{c_1, \ldots, c_r, X_1, \ldots, X_h\} \qquad t \equiv \{b_1, \ldots, b_k, c_1, \ldots, c_r\}$$

where, according to our notation (see Section 4.2), $b_i \in C_2$, $c_i \in C_3$, and $X_i \in V_1$ ($C_1 = \emptyset$ otherwise the problem has no solutions). The two terms unify iff $h \geqslant k$ (see

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

Section 4.3). From Arni *et al.* (1996) we know that the number of solutions is

$$\sum_{i=0}^{k}(-1)^i \binom{k}{i} (k+r-i)^h$$

The set of substitutions representing the correct solutions of the matching problem $s = t$ can be obtained by:

- computing all the h-multisets of $\{b_1,\ldots,b_k,c_1,\ldots,c_r\}$ that contain all the elements of the set $\{b_1,\ldots,b_k\}$
- computing all the distinct permutations of each multiset.

An algorithm based on this approach is optimal, in the sense that it computes exactly a complete and minimal set of unifiers, with a complexity that is linear in the size of such set of unifiers.

8 General $ACI1$ unification

The unification problem considered in Section 6 is capable of dealing with flat set terms containing an arbitrary number of set variables. On the other hand, the unification problem of Section 7 allows unification between possibly nested set terms with at most one set variable per set term. The goal of this section is to provide a solution to unification problems which do not fall in any of the two above categories, namely, unification problems in presence of set terms which can be nested at any depth and which may contain an arbitrary number of set variables. We will refer to this kind of problems as *general* $ACI1$ *unification problems.*

We propose a novel solution that combines the algorithms of Sections 6 and 7 developed for solving $ACI1$ unification with constants and general $(Ab)(C\ell)$ unification. The result is a new goal-driven algorithm for general $ACI1$ unification.

8.1 Language and semantics

We consider a language whose signature Σ contains the constant $\emptyset$, the binary function symbol $\cup$, and a (possibly infinite) collection of free function symbols with arbitrary arities.

Definition 10
An $ACI1$ *set term* is either a variable, or the constant $\emptyset$, or a Σ-term of the form $t \cup s$, where t and s are Σ-terms. An *individual term* is either a variable or a Σ-term of the form $f(t_1,\ldots,t_n)$ with $f \not\equiv \cup$ and $f \not\equiv \emptyset$ and $t_1,\ldots,t_n$ are Σ-terms (if $n = 0$ it is a constant term).

The function symbols $\cup$ and $\emptyset$ have the properties described by the identities (A), (C), (I) and (1) introduced in Section 5. Hence, set terms denote hereditarily finite sets based on $\mathscr{U}$, while individual terms denote arbitrary elements of the universe $\mathscr{U}$.

In the rest of the discussion we will assume the existence of at least one function symbol $f \in \Sigma$ of arity greater than zero – note that if such symbol does not exist,

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

then we are in the case discussed in Section 6. Intuitively, terms based on such symbol will be used to encode singleton sets. Without loss of generality we assume to use the unary function symbol $\{\cdot\}$ to represent singleton sets (more generally, if the chosen symbol f is of arity n, $n \geqslant 0$, we could assume that the term $f(s, \emptyset, \ldots, \emptyset)$ is used to denote the singleton set containing the element s). In this way, it will be possible, for instance, to distinguish the individual element a from the set containing a (i.e., $\{a\}$). Moreover, as a notational convenience, we will denote the term $\{s_1\} \cup \cdots \cup \{s_n\}$ with $\{s_1, \ldots, s_n\}$.

8.2 Which kind of set unification

The general $ACI1$ language allows us to describe the SUD and SUS problems for any abstract set terms in $\mathsf{set}(m, n, p, q)$. In particular, the cases $\mathsf{flat}(q)$ and $\mathsf{nested}(q)$ with $q \geqslant 2$ are handled in this framework (and not in any of the previous ones).

Example 7
The following are set terms and set unification problems which are allowed in general $ACI1$:

- $\{\{A, B\} \cup C \cup D\} \cup E \cup F = \{\{X, 1\}\} \cup E \cup G$
- $\{\{g(a)\} \cup X\} \cup Z = \{b\} \cup T \cup S$

8.3 Unification algorithm

In this section, we propose a novel algorithm to directly solve the general $ACI1$ unification problem. The algorithm is composed of a main procedure (`general_aci`) and a rewriting function (`aci_step`), which deals with equations between set terms (see Figure 6).

The structure of the main procedure is very similar to the structure of standard unification algorithms for the Herbrand case. The algorithm maintains two separate collections of equations, $\mathscr{E}_{\mathtt{s}}$ and $\mathscr{E}_{\mathtt{ns}}$: the first collects the equations in solved form while the second contains the equations that require further processing. As in the case of $(Ab)(C\ell)$ unification, the main changes with respect to standard Herbrand unification are concerned with the two rules dealing with set terms (i.e., terms containing occurrences of $\cup$ at the outermost level):

- rule (5) which is aimed at dealing with equations of the form $X = \cdots \cup X$ which are satisfiable in the case of $ACI1$ theory, whereas they were not satisfiable if the $\cup$ symbol would be uninterpreted;
- rule (8) which is used to solve equations between two set terms.

We will use the notation $\bar{s}$ to denote the list of terms $s_1, \ldots, s_n$, and $\bar{s} = \bar{t}$ to denote $s_1 = t_1, \ldots, s_n = t_n$.

`aci_step` receives as input the equation between set terms to be solved and non-deterministically produces as result two systems of equations (corresponding to the $\mathscr{E}_{\mathtt{s}}$ and $\mathscr{E}_{\mathtt{ns}}$ of the main unification procedure) and a substitution. `aci_step` performs its task in four successive steps, as shown in Figure 6. *Term Propagation* is the only

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

```
general_aci(𝒮) :
   𝒮_s := ∅; 𝒮_ns := 𝒮 (i.e., the initial system of equations);
   while 𝒮_ns ≠ ∅ do
      select arbitrarily an equation e from 𝒮_ns and remove it;
      case e of
```

(1)	$X = X$	$\mapsto$	$\mathscr{E}_{\mathrm{ns}} := \mathscr{E}_{\mathrm{ns}}$
(2)	$t = X$ t is not a variable	$\mapsto$	$\mathscr{E}_{\mathrm{ns}} := \mathscr{E}_{\mathrm{ns}} \wedge (X = t)$
(3)	$X = t$ t can be re-ordered as $f_1(\bar{s}_1) \cup \cdots \cup f_n(\bar{s}_n) \cup V_1 \cup \cdots \cup V_m$ $n \geqslant 0$, $f_i \not\equiv \cup$, $m \geqslant 0$, and $X \in vars(\bar{s}_1, \ldots, \bar{s}_n)$	$\mapsto$	`fail`
(4)	$X = t$ X does not occur in t	$\mapsto$	$\mathscr{E}_{\mathrm{s}} := \mathscr{E}_{\mathrm{s}}[X/t] \wedge (X = t); \mathscr{E}_{\mathrm{ns}} := \mathscr{E}_{\mathrm{ns}}[X/t]$
(5)	$X = t$ t can be re-ordered as $t' \cup X \cup \cdots \cup X$, $t' = f_1(\bar{s}_1) \cup \cdots \cup f_n(\bar{s}_n) \cup V_1 \cup \cdots \cup V_m$, $f_i \not\equiv \cup$, $m \geqslant 0$, $X \notin vars(t')$	$\mapsto$	$\mathscr{E}_{\mathrm{ns}} := \mathscr{E}_{\mathrm{ns}} \wedge (X = t' \cup N)$ N new variable
(6)	$f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$ $f \not\equiv g$	$\mapsto$	`fail`
(7)	$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ $f \not\equiv \cup$	$\mapsto$	$\mathscr{E}_{\mathrm{ns}} := \mathscr{E}_{\mathrm{ns}} \wedge (s_1 = t_1 \wedge \ldots \wedge s_n = t_n)$
(8)	$s_1 \cup s_2 = t_1 \cup t_2$	$\mapsto$	Let $\langle \mathscr{E}'_1, \mathscr{E}'_2, \theta \rangle$ be a result of `aci_step`$(s_1 \cup s_2 = t_1 \cup t_2)$; $\mathscr{E}_{\mathrm{s}} := \mathscr{E}_{\mathrm{s}}\theta \wedge \mathscr{E}'_1; \mathscr{E}_{\mathrm{ns}} := \mathscr{E}_{\mathrm{ns}}\theta \wedge \mathscr{E}'_2$

```
aci_step(e):
   𝒮^n := Normalization(e);
   ⟨𝒮_1^ACI, 𝒮_2^ACI⟩ := Elementary_ACI1_Solution(𝒮^n) ;
   ⟨𝒮_1, 𝒮_2⟩ := Term_Propagation(𝒮_1^ACI, 𝒮_2^ACI) ;
   ρ := Variables_Removal(𝒮_1, 𝒮_2) ;
   return ⟨𝒮_1, 𝒮_2ρ, ρ⟩
```

Fig. 6. General $ACI1$ unification procedure and the function `aci_step`

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

(don't know) non-deterministic step of the whole algorithm. Both *Term Propagation* and *Variables Removal* can lead to a failure for some of the non-deterministic choices performed within *Term Propagation*. Let us analyze these steps in more detail.

Normalization:

input: A system consisting of the single equation

$$f_1(\bar{l}_1) \cup \cdots \cup f_{k_1}(\bar{l}_{k_1}) \cup L_1 \cup \cdots \cup L_{k_2} = g_1(\bar{r}_1) \cup \cdots \cup g_{h_1}(\bar{r}_{h_1}) \cup R_1 \cup \cdots \cup R_{h_2} \quad (4)$$

where L_i, R_j $(0 \leqslant i \leqslant k_2,\ 0 \leqslant j \leqslant h_2)$ are variables and f_i, g_j $(0 \leqslant i \leqslant k_1,\ 0 \leqslant j \leqslant h_1)$ are function symbols different from $\cup$.

output: A system

$$\begin{aligned} \mathscr{E}^n \quad = \quad & N_1^L = f_1(\bar{l}_1) \wedge \cdots \wedge N_{k_1}^L = f_{k_1}(\bar{l}_{k_1}) \wedge \\ & N_1^R = g_1(\bar{r}_1) \wedge \cdots \wedge N_{h_1}^R = g_{h_1}(\bar{r}_{h_1}) \wedge \\ & N_1^L \cup \cdots \cup N_{k_1}^L \cup L_1 \cup \cdots \cup L_{k_2} = N_1^R \cup \cdots \cup N_{h_1}^R \cup R_1 \cup \cdots \cup R_{h_2} \end{aligned}$$

where N_i^L and N_j^R are new distinct variables.

This step, following the idea used in Lincoln and Christian (1989) and Baader and Schulz (1996), performs a normalization of the problem $\mathscr{E}$ into the problem $\mathscr{E}^n$ – producing an equation between set terms that contains only variables.

Elementary $ACI1$ Solution:

input: The system $\mathscr{E}^n$ produced by the *Normalization* step;

output: A pair of systems $\mathscr{E}_1^{ACI}$ and $\mathscr{E}_2^{ACI}$ obtained by solving the elementary $ACI1$ unification problem

$$N_1^L \cup \cdots \cup N_{k_1}^L \cup L_1 \cup \cdots \cup L_{k_2} = N_1^R \cup \cdots \cup N_{h_1}^R \cup R_1 \cup \cdots \cup R_{h_2} \quad (5)$$

of $\mathscr{E}^n$. This problem can be directly solved by using the techniques seen in Section 6 (see also Example 3). The result of the computation is a collection of equations of the form $V = A_{a_1,b_1} \cup A_{a_2,b_2} \cup \ldots$ where V is a variable occurring in the two terms to be unified and $A_{a_1,b_1}, A_{a_2,b_2}, \ldots$ are new variables generated by the unification algorithm. The solved form equations associated to L_j and R_i form the set $\mathscr{E}_1^{ACI}$. $\mathscr{E}_2^{ACI}$ is composed of the equations concerning the variables N_j^L and N_i^R. These variables are immediately replaced by the terms they have been set equal to during the *Normalization* step.

Term Propagation:

input: The pair of systems $\mathscr{E}_1^{ACI}$ and $\mathscr{E}_2^{ACI}$ produced in the previous step;

output: A pair of systems $\mathscr{E}_1$ and $\mathscr{E}_2$.

The equations in $\mathscr{E}_2^{ACI}$ can be simplified using the semantic properties of $\emptyset$ and $\cup$. As a matter of fact, the equations in $\mathscr{E}_2^{ACI}$ can be immediately satisfied by binding each $A_{i,j}$ appearing in the right-hand side of an equation either to $\emptyset$ or to a term which unifies with the left-hand side of the equation. Observe, however, that each $A_{i,j}$ can occur in the right-hand side of more than one equation; thus, it should receive a consistent binding in order to satisfy $\mathscr{E}_2^{ACI}$.

More precisely, a substitution λ describing the solution of the equations in $\mathscr{E}_2^{ACI}$ can be build as follows. Let us assume that an ordering has been fixed on the equations in $\mathscr{E}_2^{ACI}$ and on the variables $A_{i,j}$. Thus, for each $A_{i,j}$ occurring in $\mathscr{E}_2^{ACI}$

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

we can identify an equation $e_{A_{i,j}}$ which contains the "first" occurrence of such variable in its right-hand side. If $f(\bar{s})$ is the left-hand side of such equation, then $\lambda(A_{i,j})$ is *non-deterministically* defined to be either

- $\lambda(A_{i,j}) = \emptyset$ or
- $\lambda(A_{i,j}) = f(\bar{s})$.

As soon as the value of $\lambda(A_{i,j})$ has been determined, the substitution is immediately applied to $\mathscr{E}_2^{ACI}$. Once all the $A_{i,j}$ occurring in $\mathscr{E}_2^{ACI}$ have been processed, the system is reduced to a collection of equations of the form:

$$f(\bar{s}) = f_1(\bar{s}_1) \cup \cdots \cup f_h(\bar{s}_h)$$

with $h \geqslant 1$ (without loss of generality, we may assume that all the occurrences of $\emptyset$ in the union have been removed, as well as repetitions of the same term). The above result also relies on the assumption that at least one $A_{i,j}$ per equation is assigned a term different from $\emptyset$.

If some of the f_i is different from f for some equation, then another guess for λ must be chosen; if no choice leading to the satisfaction of this condition can be made, then the system does not admit solutions. Otherwise, let the output system $\mathscr{E}_2$ consist of all equations of the form:

$$\bar{s} = \bar{s}_1 \wedge \ldots \wedge \bar{s} = \bar{s}_h$$

for each equation in $\mathscr{E}_2^{ACI}$.

The other output system, $\mathscr{E}_1$, is obtained by applying λ to the input system $\mathscr{E}_1^{ACI}$, with the usual elimination of $\emptyset$ and repetitions in the unions. Thus,

$$\begin{aligned} \mathscr{E}_1 \quad = \quad & \textstyle\bigwedge_{1 \leqslant j \leqslant k_2} L_j = \bigcup_{i=1}^{h_1} \lambda(A_{i,k_1+j}) \cup \bigcup_{i=h_1+1}^{h_1+h_2} A_{i,k_1+j} \wedge \\ & \textstyle\bigwedge_{1 \leqslant i \leqslant h_2} R_i = \bigcup_{j=1}^{k_1} \lambda(A_{h_1+i,j}) \cup \bigcup_{j=k_1+1}^{k_1+k_2} A_{h_1+i,j} \end{aligned}$$

Variables Removal:

input: The pair $\mathscr{E}_1$ and $\mathscr{E}_2$ computed in the previous step;

output: The substitution ρ.

From $\mathscr{E}_1$ we can directly produce a substitution which allows all variables L_j and R_i to be removed. More precisely, this is obtained as follows. Let ρ_{L_j} and ρ_{R_i} denote the substitutions that respectively replace L_j $(1 \leqslant j \leqslant k_1)$ and R_i $(1 \leqslant i \leqslant h_1)$. In order to guarantee that $\mathscr{E}_1 \cup \mathscr{E}_2$ admits solutions we need to make sure that no cyclic conditions occur.

Let us define the relation $\Rightarrow$ as follows:

$$X \Rightarrow Y \quad \text{iff} \quad Y \in vars(X\rho_X)$$

and let us denote with $\Rightarrow^*$ the transitive closure of $\Rightarrow$.

A necessary condition for the solvability of the set of equations $\mathscr{E}_1$ is that

$$(\forall X \in \{L_1, \ldots, L_{k_1}, R_1, \ldots, R_{h_1}\})(X \not\Rightarrow^* X).$$

If this test is satisfied, then we can construct a global substitution

$$\rho = \rho_{L_1} \circ \cdots \circ \rho_{L_{k_1}} \circ \rho_{R_1} \circ \cdots \circ \rho_{R_{h_1}}$$

which allows all variables $\{L_1, \ldots, L_{k_1}, R_1, \ldots, R_{h_1}\}$ to be removed.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

A detailed description of the algorithms for the *Elementary ACI*1 *Solution* step and the *Term Propagation* step is reported in Appendix B.

Example 8
Let us consider the unification problem:

$$\{\{a\}\} \cup \{b\} \cup X = \{\{W\}\} \cup Y \cup Z$$

The *Normalization* step leads to the system

$$\mathscr{E}^n \equiv N_1^L = \{\{a\}\} \wedge N_2^L = \{b\} \wedge N_1^R = \{\{W\}\} \wedge N_1^L \cup N_2^L \cup X = N_1^R \cup Y \cup Z$$

The equation $N_1^L \cup N_2^L \cup X = N_1^R \cup Y \cup Z$ can be solved and its solution applied to the rest of the system (*Elementary ACI*1 *Solution* step), leading to:

$$\begin{array}{lcl} \mathscr{E}_1^{ACI} & \equiv & X = A_{1,3} \cup A_{2,3} \cup A_{3,3} \wedge \\ & & Y = A_{2,1} \cup A_{2,2} \cup A_{2,3} \wedge \\ & & Z = A_{3,1} \cup A_{3,2} \cup A_{3,3} \end{array} \qquad \begin{array}{lcl} \mathscr{E}_2^{ACI} & \equiv & \{\{a\}\} = A_{1,1} \cup A_{2,1} \cup A_{3,1} \wedge \\ & & \{b\} = A_{1,2} \cup A_{2,2} \cup A_{3,2} \wedge \\ & & \{\{W\}\} = A_{1,1} \cup A_{1,2} \cup A_{1,3} \end{array}$$

A possible substitution λ produced by the *Term Propagation* step is the following:

$A_{1,1}$	$A_{2,1}$	$A_{3,1}$	$A_{1,2}$	$A_{2,2}$	$A_{3,2}$	$A_{1,3}$
$\{\{a\}\}$	$\{\{a\}\}$	$\emptyset$	$\emptyset$	$\{b\}$	$\emptyset$	$\emptyset$

This produces the systems

$$\mathscr{E}_2 \equiv \{W\} = \{a\} \qquad \mathscr{E}_1 \equiv X = A_{2,3} \cup A_{3,3} \wedge\ Y = \{\{a\}\} \cup \{b\} \cup A_{2,3} \wedge Z = A_{3,3}$$

and the substitution $\rho = [X/A_{2,3} \cup A_{3,3},\ Y/\{\{a\}\} \cup \{b\} \cup A_{2,3},\ Z/A_{3,3}]$. From $\mathscr{E}_2$ it is then computed $[W/a]$.

8.4 Results for the general *ACI*1 unification algorithm

8.4.1 Soundness and Completeness

The soundness and completeness results can be derived as follows.

Lemma 2
Given an equation e of the form

$$f_1(\bar{l}_1) \cup \cdots \cup f_{k_1}(\bar{l}_{k_1}) \cup L_1 \cup \cdots \cup L_{k_2} = g_1(\bar{r}_1) \cup \cdots \cup g_{h_1}(\bar{r}_{h_1}) \cup R_1 \cup \cdots \cup R_{h_2}$$

let $\langle \mathscr{E}_1^i, \mathscr{E}_2^i, \rho^i \rangle$, for $i = 1, \ldots, k$, be the collection of all the distinct solutions non-deterministically produced by the call `aci_step`(e). Then:

- if σ is a unifier of $\mathscr{E}_1^i \cup \mathscr{E}_2^i$ then σ is a unifier of e and $\sigma \leqslant \rho^i$
- if σ is a unifier of e then there exists $1 \leqslant i \leqslant k$ and a substitution γ such that $\sigma \cup \gamma$ is a unifier of $\mathscr{E}_1^i \cup \mathscr{E}_2^i$.

For the proof, see Appendix A.

Theorem 1
The unification procedure `general_aci` is correct and complete with respect to the general *ACI*1 theory.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

Proof

Immediate from the above Lemma 2 concerning the auxiliary function `aci_step`, and from the classical results regarding Herbrand unification for the remaining rules. □

8.4.2 *Termination of* `general_aci`

The development of a termination proof for general unification algorithms for theories obtained using some or all of the (A), (C), (I), and (1) axioms is a well-known challenging task (Baader and Schulz 1996). Fages (1987) proposed a termination proof for general AC unification. The complexity measure developed by Fages to prove termination, however, is not applicable to our problem – mainly due to the need, in our algorithm, to introduce new variables to handle cases such as $X = Y \cup X$, that are unsatisfiable in AC but admit solutions in $ACI1$.

The detailed termination proof (Theorem 3) is reported in Appendix A. We give here the main ideas behind that proof. First of all, `aci_step` replaces an equation between two sets with equations between members of the sets, thus with equations of a "lower level". The process cannot enter in a loop thanks to the occur-check test which avoids the possibility of generating infinitely-nested sets. To formalize this idea we define the notion of *p-level* (Def. 11). Terms can be naturally viewed as trees. We use two kinds of edges in these trees, edges connecting $\cup$-nodes to their children and edges linking all other types of nodes. We show how the unification algorithm operates on this tree representation of terms, and we determine some properties related to cycles involving edges of the second type (+1-edges). Finally, we define a complexity measure built from the notion of p-level of the terms occurring in the system of equations. We show that this measure is well-ordered and that any given sequence of applications of rules either decreases it, or an occur-check failure is detected.

8.5 Discussion

A non-deterministic algorithm for general ACI is presented by Kapur and Narendran (1992) that can be adapted to general $ACI1$. Another algorithm for general $ACI1$ unification can be obtained as an instance of the general technique of Baader and Schulz (1996) for combining unification algorithms. Combining unification procedures for different unification problems has been a major topic of investigation for years (Siekmann 1984). Various proposals have been put forward to allow combination of unification procedures under different conditions on the equational theories (Yellick 1985; Herold 1986; Tiden 1986; Kirchner 1989; Schmidt-Schauß 1989). and Schulz (1996) proposed a general technique for combining unification procedures over disjoint theories under very simple restrictions, i.e., *constants restriction*. In the context of general $ACI1$ unification, we need to combine two theories: the theory $ACI1$ for $\emptyset$ and $\cup$, and the empty equational theory for all the other function symbols. The technique proposed by Baader and Schulz can thus be used to integrate the

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

unification procedure for $ACI1$ with constants and a standard Herbrand unification algorithm to obtain a unification procedure for general $ACI1$.

Let us briefly compare these two proposals with the unification algorithm for general $ACI1$ presented in this paper. All three unification procedures start with a *Normalization* step (implicit in Kapur and Narendran (1992)). New variables are introduced for subterms. As an example, the problem

$$\{X\} \cup \{Y\} \quad = \quad \{a\} \cup \{b\} \tag{6}$$

is rewritten as

$$N_1^L \cup N_2^L = N_1^R \cup N_2^R, N_1^L = \{X\}, N_2^L = \{Y\}, N_1^R = \{a\}, N_2^R = \{b\} \tag{7}$$

All three procedures introduce *don't know* non-determinism. In particular, Baader and Schulz (1996) introduces non-determinism in steps 3 and 4, where

- step 3 computes an arbitrary partition of the variables in independent sets (all the variables in the same component of the partition will be aliased to each other in the final solution);
- step 4 imposes an arbitrary order over the elements of the previously computed partition.

In the formula 7, for instance, there are 6 variables. Therefore, there are $\sum_{i=1}^{6} \left\{ {6 \atop i} \right\} = 203$ possible partitions of the set of variables[2], and $6! = 720$ possible strict orderings among the 6 variables. Actually, the problem (6) has only two independent solutions $X = a, Y = b$ and $X = b, Y = a$ that suggests the need of only 2 non-deterministic choices. The high number of choices in Baader and Schulz (1996) derives from the generality of the combination procedure (which is not specifically tied to the problem of set unification). On the other hand, it is unclear whether the instantiation of that framework to the problem at hand would actually reduce the number of alternatives compared to the algorithm we propose in this paper.

The unification procedure presented in Kapur and Narendran (1992) is rather different. It performs a series of non-deterministic guesses for the variables to find ground substitutions. It has two main practical drawbacks. The first is that the number of choices does not depend on the structure of the problem but rather on the signature. The second drawback is that the algorithm always returns ground substitutions. The number of ground substitutions of a general ACI problem can be infinite. Let us consider, for instance, the problem

$$\{\emptyset\} \cup Y = Y \tag{8}$$

$Y = \{\emptyset\}$, $Y = \{\emptyset\} \cup \{\{\emptyset\}\}$, $Y = \{\emptyset\} \cup \{\{\emptyset\}\} \cup \{\{\{\emptyset\}\}\}, \ldots$ are all the ground solutions for (8). However, a unique most general unifiers, $Y = \{\emptyset\} \cup N$ is sufficient to finitely describe all solutions (this is exactly what our algorithm returns). Even for problems where only ground unifiers are present, our algorithm has the advantage of using

[2] $\left\{ {n \atop i} \right\}$ is the number of partitions of n elements into i classes, known as *Stirling number of the second type* (Graham *et al.* 1994).

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

the symbols in the problem to drive the construction of the solution, instead of performing a blind enumeration based on the language signature.

As for the non-determinism introduced by our algorithm, first observe that the *Normalization* step allows us to call the elementary $ACI1$ unification step with terms containing only variables. In this case it is known that the unification problem admits a unique mgu. So, we are not exploiting the possibility of the $ACI1$ with constants unification algorithm to return non-deterministically all the mgus and we perform that choice later. The rationale behind this is that the non-variable terms $s_1, \ldots, s_\ell, t_1, \ldots, t_r$ in an equation $X_1 \cup \cdots \cup X_m \cup s_1 \cup \cdots \cup s_\ell = Y_1 \cup \cdots \cup Y_n \cup t_1 \cup \cdots \cup t_r$ can be compound terms. We do not know (yet) if some of them can be unified, and thus we cannot consider them as equal or different constants when calling the $ACI1$ with constants algorithm. Possible optimizations of our algorithm include the use of $ACI1$ with constants in those cases where a simple preprocessing allows us to quickly determine what individuals in the equations are equal or distinct. If V_1, V_2, V_3 are the set of variables in the elementary $ACI1$ unification problem as defined in Section 4.2, then the Boolean ACI matrix (Baader and Büttner 1988) is of size $(|V_1|+|V_2|+|V_3|)(|V_1||V_2|+|V_1||V_3|+|V_2||V_3|+|V_3|)$ and the new variables introduced are $|V_1||V_2|+|V_1||V_3|+|V_2||V_3|+|V_3|$. Our *elementary ACI matrix* (see Appendix B) introduces the same number of variables, but its size is $(|V_1|+|V_3|)(|V_2||V_3|)$. For instance, if $|V_1| = |V_2| = |V_3| = v$ we need space $4v^2$ against space $9v^3 + 3v^2$.

All non-deterministic choices are performed in the *Term Propagation* step. If k is the number of variables introduced by the matrix, this would potentially open 2^k non-deterministic choices. However, using the auxiliary Boolean matrix (see Appendix B) we do not try all these choices, since for each column and each row of the matrix for *Term Propagation* there must be at least one variable which is different from $\emptyset$. This decreases the number of choices. In the case of the system of equations (7) we have only 8 non-deterministic choices instead of the 2^4 expected (and the 203×720 of the naive application of the Baader-Schulz procedure).

As far as the difference in non-determinism between the general $ACI1$ and the general $(Ab)(C\ell)$ unification is concerned, we can observe that the $ACI1$ algorithm opens, for each level of nesting, a number of alternatives equivalent to the resolution of an $ACI1$ with constants problem; this leads to $O(2^{n^2})$ solutions (see Section 6.3). Since this process can be repeated once for each nesting, a rough upper bound to the number of solutions is $O(2^{n^3})$. Observe that this number of solutions is greater than those computed by the $(Ab)(C\ell)$, namely $O(2^{n^2 \log n})$. This fact suggests that the general $ACI1$ unification should be used only when the problem is really not expressible using the general $(Ab)(C\ell)$ unification and the full range of solutions is required.

9 Related work

Most of the related proposals have already been discussed throughout the paper. In this section we provide a brief overview of other related contributions.

Boolean unification. Boolean unification is a very powerful framework that allows one, in particular, to mimic the $ACI1$ with constants unification problems. The

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

richer language of Boolean unification, however, allows the various solutions of a given $ACI1$ problem to be encoded in a very compact way, as a single complex solution – instead of using multiple ACI-matrices as in Section 6. A fundamental work in this area is Martin and Nipkow (1989), which surveys both the *Boole's method* and the *Löwenheim's method*. The former has been originally described in Büttner and Simonis (1987) while the second has been initially described in Martin and Nipkow (1988). All these approaches deal with Boolean unification with constants, where the signature Σ contains a possibly infinite collection of constants, which intuitively represent the *elements* of the universe $\mathscr{U}$. The class of terms allowed in this framework extends the one considered in this paper by allowing a variety of different operators to be used in the construction of sets, such as intersection $\cap$ and complementation $\bar{(\cdot)}$.

The complexity of the decision problem of Boolean unification has been studied in Baader (1988). In the elementary case, i.e., without constants, the problem is NP-complete, while in the case with constants the problem becomes PSPACE-complete. However, if the input is of the form admitted by $ACI1$ unification, the test between two ground terms can be performed in linear time. The computation of the unifier for a given Boolean unification problem $s = t$ is based on the fact that μ is a unifier of $s = t$ if and only if μ is a unifier of $s \triangle t = \emptyset$, where $\triangle$ is a function symbol which is interpreted as the symmetric difference. Thus, to solve a unification problem, it is sufficient to solve a *matching problem*. The work in Büttner and Simonis (1987) shows that a *unique* most general unifier is sufficient to cover all the solutions. The generality of this scheme and the power of this unification procedure are balanced by the complexity of the answers produced – sets built using $\triangle$ are arguably more complex and less intuitive than those constructed using $\cup$.

Computable Set Theory. The work on Computable Set Theory (Cantone *et al.* 2001) has been mainly developed at the New York University, with the objective of enhancing the expressive power of inference engines for automated theorem provers, and for the implementation of the imperative set-based programming language SETL (Schwartz *et al.* 1986). The general problem is to identify computable classes of formulae of suitable sub-theories of Zermelo-Fraenkel set theory. In this context, the set unification problem is seen as a special case of the satisfiability problem for the $\exists^*\forall$-class of formulae. As a matter of fact, thanks to the extensionality axiom, testing whether two terms s and t with variables $X_1, \ldots, X_n$ are unifiable is equivalent to testing whether the following holds:

$$\mathbb{HF} \models \exists X_1 \cdots \exists X_n \forall Z (Z \in s \leftrightarrow Z \in t).$$

Unification algorithms can be obtained by instantiating the general (and complex) techniques for testing satisfiability of $\exists^*\forall$-formulas (Dovier *et al.*).

Set constraints. Set constraints (Kozen 1998; Aiken 1994) are conjunctions of literals of the form $e_1 \subseteq e_2$ where e_1 and e_2 are *set expressions*, constructed using variables, constant and function symbols, and the union, intersection, and complement of set expressions. Set expressions denote sets of Herbrand terms. An

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

expression identifies a subset of the Herbrand universe. A unification problem of the type $s = t$ can be expressed in this framework as the constraint $s \subseteq t \wedge t \subseteq s$. The framework is sufficiently powerful to solve $ACI1$ unification problems with constants; nevertheless, the peculiar interpretation given to terms in the language is such to prevent to encode large classes of set unification problems. In particular, to represent nested sets in set constraints we need to make use of a distinguished functional symbol $\{\cdot\}$ (as described also in Section 8.1); on the other hand, using the set constraint interpretation of expressions, the two expressions $\{\{s,t\}\}$ and $\{\{s\},\{t\}\}$ would be mapped to the same set.

Alternative representations of sets. Other syntactic representations of sets are also feasible. For instance a set of n elements can be represented by $\{\}_n(t_0,\ldots,t_n)$, where $\{\}_n$ is a function symbol of arity n. This solution requires the introduction of an infinite signature, with a different set constructor for each possible finite set cardinality. This approach has been adopted, for example, in (Shmueli *et al.* 1992). In order to use this solution it is necessary to introduce a complex infinite equational theory, capable of specifying the unifiability of set terms with different main functors – as in the case $\{\}_3(X,Y,Z) = \{\}_2(a,b)$.

This representation scheme allows one to express only set terms with a known upper bound on their cardinality. Namely, $|\{\}_n(t_1,\ldots,t_n)| \leqslant n$.

10 Conclusions

In this paper we have presented a survey of the problem of solving unification in the context of algebras for sets. We have abstractly defined the set unification problem and developed the corresponding equational theories, starting from the simpler case of $ACI1$ with constants and proceeding to the most comprehensive case of general $ACI1$ unification. We have presented decision and unification procedures for the different classes of unification problems and analyzed their complexity. Complexity results, as well as the suitable equational theory for a given set unification problem, are summarized in Table 1. The algorithms presented are either drawn from the literature or are brand new algorithms developed by the authors.

We believe this work fills a gap in the literature on this topic, by providing a uniform and complete presentation of this problem, and by presenting a comparative study of the different solutions proposed.

Acknowledgments

We thank the anonymous referees that helped us to improve the quality of presentation of the paper. The research presented in this paper has benefited from discussions with A. Formisano, E. G. Omodeo, C. Piazza, A. Policriti, and D. Ranjan, all of whom we would like to thank.

References

Abiteboul, S. and Grumbach., S. 1991. A Rule-Based Language with Functions and Sets. *ACM Trans. on Database Systems 16,* 1, 1–30.

Aczel, P. 1988. *Non-well-founded sets.* CSLI Lecture Notes, vol. 14. Stanford University Press.

Aiken, A. 1994. Set constraints: Results, Applications, and Future Directions. In *Principles and Practice of Constraint Programming*, A. Borning, Ed. Lecture Notes in Computer Science. Springer Verlag, 326–335.

Aliffi, D., Dovier, A., and Rossi, G. 1999. From Set to Hyperset Unification. *Journal of Functional and Logic Programming 1999,* 10, 1–48.

Arenas-Sánchez, P. and Dovier, A. 1997. A Minimality Study for Set Unification. *Journal of Functional and Logic Programming 1997,* 7, 1–49.

Arenas-Sánchez, P. and Rodríguez-Artalejo, M. 2001. A General Framework for Lazy Functional Logic, Programming with Algebraic Polymorphic Types. *Theory and Practice of Logic Programming 2,* 1, 185–245.

Arni, N., Greco, S., and Saccà, D. 1992. Set-term matching in logic programming. In *Database Theory – ICDT'92, 4th International Conference, Berlin, Germany, October 14-16, 1992, Proceedings*, J. Biskup and R. Hull, Eds. Lecture Notes in Computer Science, vol. 646. Springer, 436–449.

Arni, N., Greco, S., and Saccà, D. 1996. Matching of Bounded Set Terms in the Logic Language LDL++. *Journal of Logic Programming 27,* 1, 73–87.

Baader, F. 1998. On the Complexity of Boolean Unification. *Information Processing Letters 67,* 4, 215–220.

Baader, F. and Büttner, W. 1988. Unification in commutative and idempotent monoids. *Theoretical Computer Science 56*, 345–352.

Baader, F. and Schulz, K. U. 1996. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation 21*, 211–243.

Baader, F. and Snyder, W. 2001. Unification Theory. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Elsevier, Amsterdam, Chapter 8, 446–533.

Barwise, J. and Moss, L. 1996. *Vicious Circles. On the Mathematics of non-well-founded phenomena.* CSLI Lecture Notes, vol. 60. Stanford University Press.

Beeri, C., Naqvi, S., Shmueli, O., and Tsur., S. 1991. Set Constructors in a Logic Database Language. *Journal of Logic Programming 10,* 3, 181–232.

Büttner, W. 1986. Unification in the Data Structure Sets. In *Proc. of the Eight International Conference on Automated Deduction*, J. K. Siekmann, Ed. Vol. 230. Springer-Verlag, Berlin, 470–488.

Büttner, W. and Simonis, H. 1987. Embedding Boolean Expressions into Logic Programming. *Journal of Symbolic Computation 4*, 191–205.

Cantone, D., Omodeo, E. G., and Policriti, A. 2001. *Set Theory for Computing. From Decision Procedures to Declarative Programming with Sets.* Monographs in Computer Science. Springer-Verlag, Berlin.

Codish, M. and Lagoon, V. 2000. Type Dependencies for Logic Programs using ACI-Unification. *Theoretical Computer Science 238,* 1–2, 131–159.

Dantsin, E. and Voronkov, A. 1999. A Nondeterministic Polynomial-Time Unification Algorithm for Bags, Sets, and Trees. In *Proc. of Foundations of Software Science and Computation Structure. Second International Conference, FoSSaCS'99*, W. Thomas, Ed. Lecture Notes in Computer Science, vol. 1578. Springer-Verlag, Berlin, 180–196.

Dovier, A., Formisano, A., and Omodeo, E. Decidability results for sets with atoms. *ACM Transaction on Computational Logic* 7(2): 269–301, 2006.

Dovier, A., Omodeo, E. G., Pontelli, E., and Rossi, G. 1996. {log}: A Language for Programming in Logic with Finite Sets. *Journal of Logic Programming 28,* 1, 1–44.

Dovier, A., Piazza, C., and Policriti, A. 2000. Comparing Expressiveness of Set Constructor Symbols. In *Frontier of Combining Systems*, H. Kirchner and C. Ringeissen, Eds. Lecture Notes in Computer Science, vol. 1794. Springer-Verlag, Berlin, 275–289.

Dovier, A., Piazza, C., and Policriti, A. 2004. An efficient algorithm for computing bisimulation equivalence . *Theoretical Computer Science 311,* 1–3, 221–256.

Dovier, A., Piazza, C., Pontelli, E., and Rossi, G. 2000. Sets and Constraint Logic Programming. *ACM Transactions on Programming Languages and Systems 22,* 5, 861–931.

Dovier, A., Policriti, A., and Rossi, G. 1998. A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms. *Fundamenta Informaticae 36,* 2/3, 201–234.

Dovier, A. and Rossi, G. 1993. Embedding Extensional Finite Sets in CLP. In *Proc. of Int'l Logic Programming Symposium, ILPS'93*, D. Miller, Ed. The MIT Press, Cambridge, Mass., 540–556. Vancouver, BC, Canada.

Fages, F. 1987. Associative-Commutative Unification. *Journal of Symbolic Computation 3*, 257–275.

Graham, R. L., Knuth, D. E., and Patashnik, O. 1994. *Concrete Mathematics*. Addison-Wesley.

Greco, S. 1996. Optimal Unification of Bound Simple Set Terms. In *Proc. of Conference on Information and Knowledge Management*. ACM Press, 326–336.

Grieskamp, W. 1999. A set-based calculus and its implementation. Ph.D. thesis, Technical University of Berlin.

Hermann, M. and Kolaitis, P. 1997. On the Complexity of Unification and Disunification in Commutative Idempotent Semigroups. In *Principles and Practice of Constraint Programming – CP97, Third International Conference, Linz, Austria, October 29 – November 1, 1997, Proceedings*, G. Smolka, Ed. Lecture Notes in Computer Science, vol. 1330. Springer-Verlag, Berlin, 282–296.

Herold, A. 1986. Combination of Unification Algorithms. In *Proc. of 8th International Conference on Automated Deduction*, J. Siekmann, Ed. Lecture Notes in Computer Science, vol. 230. Springer-Verlag, Berlin, 450–469.

Hill, P. M. and Lloyd, J. W. 1994. *The Gödel Programming Language*. The MIT Press, Cambridge, Mass.

Jayaraman, B. 1992. Implementation of Subset-Equational Programs. *Journal of Logic Programming 12,* 4, 299–324.

Jayaraman, B. and Plaisted, D. A. 1989. Programming with Equations, Subsets and Relations. In *Proceedings of NACLP89*, E. Lusk and R. Overbeek, Eds. The MIT Press, Cambridge, Mass., 1051–1068. Cleveland.

Jouannaud, J. P. and Kirchner, C. 1991. Solving equations in abstract algebras: A rule-based survey of unification. In *Computational Logic: Essays in Honor of Alan Robinson*, J. L. Lassez and G. Plotkin, Eds. MIT Press.

Kapur, D. and Narendran, P. 1986. NP-completeness of the set unification and matching problems. In *8th International Conference on Automated Deduction*, J. H. Siekmann, Ed. Lecture Notes in Computer Science, vol. 230. Springer-Verlag, Berlin, 489–495.

Kapur, D. and Narendran, P. 1992. Complexity of Unification Problems with Associative-Commutative Operators. *Journal of Automated Reasoning 9*, 261–288.

Kifer, M. and Lausen, G. 1989. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In *International Conference on Management of Data and Symposium on Principles of Database Systems*. ACM Press, 134–146.

Kirchner, C. 1989. *From Unification in Combination of Equational Theories to a New AC-Unification Algorithm*. Resolution of Equations in Algebraic Structures, vol. 2. Academic Press.

Kozen, D. 1998. Set Constraints and Logic Programming. *Information and Computation 142,* 1, 2–25.

Kunen, K. 1980. *Set Theory. An Introduction to Independence Proofs*. Studies in Logic. North Holland, Amsterdam.

Kuper, G. M. 1990. Logic Programming with Sets. *Journal of Computer and System Science 41,* 1, 66–75.

Legeard, B. and Legros, E. 1991. Short Overview of the CLPS System. In *Symposium on Progr. Languages Implementation and Logic Programming*. Springer Verlag, 431–433.

Lim, S.-J. and Ng, Y.-K. 1997. Design and Analysis of Parallel Set-Term Unification. In *Proc. of Computing and Combinatorics, Third Annual International Conference*, T. Jiang and D. T. Lee, Eds. Lecture Notes in Computer Science, vol. 1276. Springer-Verlag, Berlin, 321–330.

Lincoln, P. and Christian, J. 1989. Adventures in Associative-Commutative Unification. *Journal of Symbolic Computation 8,* 1/2, 217–240.

Liu, M. 1998. Relationlog: a Typed Extension to Datalog with Sets and Tuples. *Journal of Logic Programming 36,* 3, 271–299.

Livesey, M. and Siekmann, J. 1976. Unification of Sets and Multisets. Technical report, Institut fur Informatik I, Universitat Karlsruhe.

Manandhar, S. 1994. An Attributive Logic of Set Descriptions and Set Operations. In *32nd Annual Meeting of the Association of Computational Linguistics*. ACL, 255–262.

Marciniec, J. 1997. Infinite Set Unification with Application to Categorial Grammar. *Studia Logica 58*, 339–355.

Martelli, A. and Montanari, U. 1982. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems 4*, 258–282.

Martin, U. and Nipkow, T. 1988. Unification in Boolean Rings. *Journal of Automated Reasoning 4,* 4, 381–396.

Martin, U. and Nipkow, T. 1989. Boolean Unification – The Story So Far. *Journal of Symbolic Computation 7,* 3/4, 275–293.

Naqvi, S. and Tsur, S. 1989. *A Logical Language for Data and Knowledge Bases*. Computer Science Press.

Omodeo, E. G. and Policriti, A. 1995. Solvable set/hyperset contexts: I. Some decision procedures for the pure, finite case. *Communications on Pure and Applied Mathematics 48,* 9–10, 1123–1155. Special Issue in honor of J.T. Schwartz.

Paterson, M. S. and Wegman, M. N. 1978. Linear Unification. *Journal of Computer and System Sciences 16*, 158–167.

Policriti, A. and Schwartz, J. T. 1997. *T*-Theorem Proving I. *Journal of Symbolic Computation 20,* 3, 315–342.

Pollard, C. J. and Moshier, M. D. 1990. Unifying partial description of sets. In *Information, Language and Cognition*, P. Hanson, Ed. University of British Columbia Press, Vancouver, BC, 285–322.

Rounds, W. C. 1988. Set values for unification based grammar formalisms and logic programming. Research Report CSLI-88-129, Center for the Study of Language Language and Information, Stanford, CA.

Schmidt-Schauss, M. 1989. Unification in a Combination of Arbitrary Disjoint Equational Theories. *Journal of Symbolic Computation 8,* 1/2, 51–99.

Schwartz, J., Dewar, R., Dubinsky, E., and Schonberg, E. 1986. *Programming with Sets: an Introduction to SETL*. Springer-Verlag, Berlin.

Shmueli, O., Tsur, S., and Zaniolo, C. 1992. Compilation of Set Terms in the Logic Data Language (LDL). *Journal of Logic Programming 12,* 1/2, 89–119.

Siekmann, J. K. 1984. Universal Unification. In *Proc. of 7th International Conference on Automated Deduction*, R. E. Shostak, Ed. Lecture Notes in Computer Science, vol. 170. Springer-Verlag, Berlin, 1–42.

Siekmann, J. K. 1989. Unification Theory. *Journal of Symbolic Computation 7,* 3/4, 207–274.

Spivey, J. 1992. *The Z Notation: a Reference Manual*. Prentice Hall.

Stolzenburg, F. 1996. Membership-Constraint and Complexity in Logic Programming with Sets. In *First Int'l Workshop on Frontier of Combining Systems*, F. Baader and K. Schulz, Eds. Kluwer Academic Publishers, 285–302.

Stolzenburg, F. 1999. An Algorithm for General Set Unification and Its Complexity. *Journal of Automated Reasoning 22,* 1, 45–63.

Tarski, A. 1924. Sur les ensembles fini. *Fundamenta Mathematicae VI*, 45–95.

Tiden, E. 1986. Unification in Combinations of Collapse-free Theories with Disjoint Sets of Function Symbols. In *Proceedings of the International Conference on Automated Deduction*, J. Siekmann, Ed. Lecture Notes in Computer Science, vol. 230. Springer-Verlag, Berlin, 431–449.

Wang, L., Wijesekera, D., and Jajodia, S. 2004. A logic-based framework for attribute based access control. In *Formal Methods in Security Engineering: From Specification to Code*. ACM, 45–55.

Yakhno, T. and Petrov, E. 2000. Extensional Set Library for ECLiPSe. In *Perspectives of System Informatics, Third International Andrei Ershov Memorial Conference, PSI'99*, D. Bjørner, M. Broy, and A. V. Zamulin, Eds. Lecture Notes in Computer Science, vol. 1755. Springer, 434–444.

Yellick, K. 1985. Combining Unification Algorithms for Confined Equational Theories. In *Proceedings of the Conference on Rewriting Techniques and Applications*, J.-P. Jouannaud, Ed. Lecture Notes in Computer Science, vol. 202. Springer-Verlag, Berlin.

Appendix A Proofs

A.1 Termination of `AbCl_unify`

To prove the following theorem, we will use the notions of solved variable and solved equation. Given a system $\mathscr{E}$ an equation in $\mathscr{E}$ is *solved* if it is of the form $X = t$ and X does not occur neither in t nor elsewhere in $\mathscr{E}$. If X is the r.h.s. of a solved equation then it is a solved variable. Moreover, *size* is the function returning the number of occurrences of constant and functional symbols in a term ($size(X) = 0, size(f(t_1, \ldots, t_n)) = 1 + \sum_{i=1}^{n} size(t_i)$).

Theorem 2 (`AbCl_unify` *termination*)
For any Herbrand system $\mathscr{E}$, and for any possible sequence of non-deterministic choices, `AbCl_unify`($\mathscr{E}$) terminates.

Proof
To start, do not consider the final call to `AbCl_unify_final`. We associate the complexity pair $\langle A, B \rangle$ to a system $\mathscr{E}$, where:

- A is the number of non-solved variables in $\mathscr{E}$
- let $p = \max\{size(\ell) : \ell = r \text{ in } \mathscr{E}\}$. For $i = 0, \ldots, p$, let $\eta(i)$ be the number of non-solved equations $\ell = r$ in $\mathscr{E}$ s.t. $size(\ell) = i$. Then B is the

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

list: $[\eta(p), \eta(p-1), \ldots, \eta(0)]$. We define the ordering among lists as follows:

$$\begin{aligned} x <_{list} y \quad \text{iff} \quad & (length(x) < length(y)) \; or \\ & (length(x) = length(y) \; and \; head(x) < head(y)) \; or \\ & (length(x) = length(y), head(x) = head(y), and \; tail(x) <_{list} tail(y)) \end{aligned}$$

where *length*, *head*, and *tail* are three functions on lists returning the length of the list, its first element, and the list deprived of its first element, respectively.

The ordering between two complexity pairs is the lexicographic ordering in which usual $<$ is used for the integer numbers of the first argument and $<_{list}$ for the second. It is immediate to prove that this ordering is well-founded.

We show that each non-failing call to `AbCl_unify_actions` causes the decreasing of the complexity. Well-foundedness of the ordering implies termination. By case analysis, we note that:

- rules 1, 2, and 7 cannot increase A, while B always decreases
- rule 5 decreases A
- rule 8 is more complicated to analyze, since it calls `AbCl_step`($\mathscr{E}, \{t \mid s\} = \{t' \mid s'\}$). In this case, equations are added on the part of the system dealt as a stack, driving the following rule applications. These sequences of rule applications always allow us to empty the stack. We consider these operations as a unique step that removes $\{t \mid s\} = \{t' \mid s'\}$ and introduces other equations in the system. Four cases must be distinguished:
  1. `tail`(s) and `tail`(s') are not variables: in this case A cannot increase and B decreases, since the equation is replaced by a certain number of equations between the elements of the two sets and between their tails, but all of fewer (leftmost) size;
  2. exactly one of them is a variable. Assume `tail`(s) is a variable: a substitution for it is computed and applied: A decreases. The situation when `tail`(s') is a variable is perfectly symmetrical.
  3. `tail`(s) and `tail`(s') are the different variables X and Y, respectively. One of the following cases happens:
     (a) a substitution $X = \{\ldots \mid Y\}$ is computed,
     (b) a substitution $Y = \{\ldots \mid X\}$ is computed,
     (c) a substitution $X = \{\ldots \mid N\}$ and $Y = \{\ldots \mid N\}$ (N a new variable, the same for the two equations) is computed.

     In all the three cases the application of the substitution causes A to decrease.
  4. `tail`(s) and `tail`(s') are the same variable X. In this case one equation $X = \{\ldots \mid X\}$ is added to $\mathscr{E}$ together with a certain number of equations between elements of the two sets $\{t \mid s\}$ and $\{t' \mid s'\}$. All these equations have (leftmost) size smaller than $\{t \mid s\}$.

To conclude the proof, let us observe that the termination of `AbCl_unify_final` is evident. For any variable X occurring in a equation $X = \{\ldots \mid X\}$ we perform at most one rewriting and application of substitution. X occurs elsewhere in the system only as l.h.s. Equations in solved form remain in solved form and do not fire any new action. □

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

A.2 Correspondence between $ACI1$ with constants and $\mathsf{gflat}(q)$ unification

Lemma 1. *σ is a solution of the SUS problem $s = t$ if and only if σ^* is a $ACI1$ unifier of $s^* = t^*$.*

Proof
Without loss of generality, we assume that symbols in s and t are sorted, so as they are of the form

$$\begin{aligned} s &= \underbrace{\{a_1,\ldots,a_m\}}_{C_1} \cup \underbrace{\{b_1,\ldots,b_n\}}_{C_3} \cup \underbrace{Y_1 \cup \cdots \cup Y_p}_{V_1} \cup \underbrace{W_1 \cup \cdots \cup W_q}_{V_3} \\ t &= \underbrace{\{d_1,\ldots,d_{m'}\}}_{C_2} \cup \underbrace{\{b_1,\ldots,b_n\}}_{C_3} \cup \underbrace{Z_1 \cup \cdots \cup Z_{p'}}_{V_2} \cup \underbrace{W_1 \cup \cdots \cup W_q}_{V_3} \end{aligned}$$

where C_i and V_i are determined according to formula (2) – Section 4.2. The corresponding $(s)^*$ and $(t)^*$ are:

$$\begin{aligned} s &= \underbrace{a_1 \cup \cdots \cup a_m}_{C_1} \cup \underbrace{b_1 \cup \cdots \cup b_n}_{C_3} \cup \underbrace{Y_1 \cup \cdots \cup Y_p}_{V_1} \cup \underbrace{W_1 \cup \cdots \cup W_q}_{V_3} \\ t &= \underbrace{d_1 \cup \cdots \cup d_{m'}}_{C_2} \cup \underbrace{b_1 \cup \cdots \cup b_n}_{C_3} \cup \underbrace{Z_1 \cup \cdots \cup Z_{p'}}_{V_2} \cup \underbrace{W_1 \cup \cdots \cup W_q}_{V_3} \end{aligned}$$

σ is a solution of $s = t$ if and only if

- for each a_i in C_1 there is X in $V_2 \cup V_3$ such that $\sigma(X) = \{a_i, \cdots\}$ and
- for each b_j in C_2 there is X in $V_2 \cup V_3$ such that $\sigma(X) = \{b_j, \cdots\}$ and
- each variable in $V_1 \cup V_2 \cup V_3$ is mapped on a set of constants in $C_1 \cup C_2 \cup C_3$ plus, possibly, other constants.

μ is a solution of $s = t$ if and only if

- for each a_i in C_1 there is X in $V_2 \cup V_3$ such that $\mu(X) = a_i \cup \cdots$ and
- for each b_j in C_2 there is X in $V_2 \cup V_3$ such that $\mu(X) = b_j \cup \cdots$ and
- each variable in $V_1 \cup V_2 \cup V_3$ is mapped on a union of constants in $C_1 \cup C_2 \cup C_3$ plus, possibly, other constants.

Clearly, $\mu = \sigma^*$. □

A.3 Soundness and completeness of general $ACI1$ unification algorithm

Lemma 2. *Given an equation e of the form*

$$f_1(\bar{l}_1) \cup \cdots \cup f_{k_1}(\bar{l}_{k_1}) \cup L_1 \cup \cdots \cup L_{k_2} = g_1(\bar{r}_1) \cup \cdots \cup g_{h_1}(\bar{r}_{h_1}) \cup R_1 \cup \cdots \cup R_{h_2}$$

let $\langle \mathscr{E}^i_1, \mathscr{E}^i_2, \rho^i \rangle$, for $i = 1,\ldots,k$, be the collection of all the distinct solutions non-deterministically produced by the call `aci_step`(e). *Then:*

- *if σ is a unifier of $\mathscr{E}^i_1 \cup \mathscr{E}^i_2$ then σ is a unifier of e and $\sigma \leqslant \rho^i$*
- *if σ is a unifier of e then there exists $1 \leqslant i \leqslant k$ and a substitution γ such that $\sigma \cup \gamma$ is a unifier of $\mathscr{E}^i_1 \cup \mathscr{E}^i_2$.*

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

Proof
Let us prove the lemma by showing that the conditions hold at each step of the construction of each solution.

- For the *Normalization* step, it is trivial to show that σ is a unifier for e if and only if $\sigma \cup \gamma$ is a unifier for $\mathscr{E}^n$, where γ possibly binds the new variables N_i^L, N_j^R. In this case k is equal to 1. The substitution γ is $[N_i^L/f_i(\bar{l}_i)\sigma \mid 1 \leqslant i \leqslant k_1] \cup [N_j^R/g_j(\bar{r}_j)\sigma \mid 1 \leqslant j \leqslant h_1]$.
- For the *Elementary ACI*1 *Solution* step the result follows from the results for elementary $ACI1$ unification (Baader and Büttner 1988). In this case we have that σ is a unifier of $\mathscr{E}^n$ if and only if $\sigma \cup \gamma$ is a unifier for $\mathscr{E}^{ACI}$, where $dom(\gamma) \subseteq \{A_{i,j} \mid 1 \leqslant i \leqslant h_1 + h_2 \wedge 1 \leqslant j \leqslant k_1 + k_2\}$.
- Let us consider the *Term Propagation* step. We prove that $\sigma' = \sigma \cup \gamma$ is a unifier for $\mathscr{E}^{ACI}$ (where $dom(\sigma) \cap dom(\gamma) = \emptyset$ and $dom(\gamma) = \{A_{i,j} \mid 1 \leqslant i \leqslant h_1 + h_2 \wedge 1 \leqslant j \leqslant k_1 + k_2\}$) iff $\sigma\gamma'$ is a unifier for $\mathscr{E}_1 \cup \mathscr{E}_2$ (where γ' is the restriction of γ to $\{A_{i,j} \mid h_1 + 1 \leqslant i \leqslant h_1 + h_2 \wedge k_1 + 1 \leqslant j \leqslant k_1 + k_2\}$).
  Let $\sigma' = \sigma \cup \gamma$ be a unifier for $\mathscr{E}^{ACI}$ and let us consider the equations e in $\mathscr{E}_2^{ACI}$ in the same arbitrary order used to build $\mathscr{E}_2$. Such equations have the form $f(\bar{s}) = \bigcup A_i \cup \bigcup B_j$. If σ' is a unifier for $\mathscr{E}^{ACI}$, then (from the ACI properties and Clark's Equational Theory) each $A_i\gamma$ and $B_j\gamma$ must be either $\emptyset$ or $f(\bar{s})\sigma$; furthermore, at least one of A_i, B_j must be assigned $f(\bar{s})\sigma$. Let I and J be the collection of indices for which respectively A_i and B_j receive $f(\bar{s})\sigma$ in σ'. We can use I and J to select a certain $\mathscr{E}_1 \cup \mathscr{E}_2$ – the one in which the λ is constructed by taking $\lambda(A_i) = f(\bar{s})$ ($\lambda(B_j) = f(\bar{s})$) for $i \in I$ ($j \in J$), and $\emptyset$ for the remaining variables in the equation. The process can be repeated for the remaining equations, until all the variables have received an assignment in λ. The consistency of σ' guarantees that this construction will provide a consistent $\mathscr{E}_2$. It is straightforward to observe that σ' is a unifier for $\mathscr{E}_2$. Observe also that $\sigma' \leqslant \lambda$, i.e., $\sigma' = \lambda \circ \theta$. This last fact, together with the fact that σ is a unifier for $\mathscr{E}_1^{ACI}$, is sufficient to conclude that σ' is a unifier for $\mathscr{E}_1$.
  Vice versa, let σ' be a unifier for a certain $\mathscr{E}_1 \cup \mathscr{E}_2$ produced by the algorithm. Since the construction was possible, then there is a substitution λ which has been used to convert $\mathscr{E}_2^{ACI}$ into $\mathscr{E}_2$. If σ' is a solution of the equations $\bar{s} = \bar{s}_1, \ldots, \bar{s} = \bar{s}_h$ present in $\mathscr{E}_2$, then σ' is also a unifier for the equation $f(\bar{s}) = (\bigcup A_i \cup \bigcup B_j)\lambda$ which produced such elements of $\mathscr{E}_2$. Thus, $\sigma' \cup [A/A\lambda \circ \sigma' \mid A \in dom(\lambda)]$ is a unifier for $\mathscr{E}_2^{ACI}$. The result for $\mathscr{E}_1^{ACI}$ is obvious.
- Correctness for the *Variables Removal* step follows from the fact that we are not interested in solutions over infinite terms. □

*A.4 Termination of general ACI*1 *unification algorithm*

Definition 11
Let $\mathscr{E}$ be a set of equations, and let us consider a function $lev : vars(\mathscr{E}) \longrightarrow \mathbb{N}$. This function can be extended over elements of $T(\Sigma, \mathscr{V})$ as follows:

$$\begin{aligned} lev(f(t_0, \ldots, t_n)) &= 1 + \max\{lev(t_0), \ldots, lev(t_n)\} \quad f \in \Sigma, f \not\equiv \cup \\ lev(s \cup t) &= \max\{lev(s), lev(t)\} \end{aligned}$$

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

The function *lev* is said to be a *partial p-level* if it satisfies the condition:

(*) $lev(\ell), lev(r) \leqslant p$, for any equation $\ell = r$ in $\mathscr{E}$.

Any partial p-level *lev* is said to be a *(complete) p-level* if it satisfies also the condition:

(**) $lev(\ell) = lev(r)$ for any equation $\ell = r$ in $\mathscr{E}$.

Lemma 3
Let us consider a system of equations $\mathscr{E}$, and let p be the number of occurrences of elements of Σ in $\mathscr{E}$; then, exactly one of the following conditions holds:

- there exists a complete p-level for $\mathscr{E}$
- for any natural number q, there are no complete q-levels for $\mathscr{E}$.

Proof
Given the system $\mathscr{E}$, it is possible to obtain, by adding a suitable number of new variables, an equivalent system $\mathscr{E}'$ in *flat form*, i.e., each equation in $\mathscr{E}$ is in one of the following forms:

1. $X = Y$
2. $X = f(Y_1, \ldots, Y_n)$, $f \in \Sigma$ and $f \notin \cup$
3. $X = Y_1 \cup Y_2$

Observe that at most p equations of type (2) can appear in $\mathscr{E}'$.

The goal is to map $\mathscr{E}'$ to a set of linear integer constraint systems. Each possible complete p-level for $\mathscr{E}'$ (and thus for $\mathscr{E}$) is a solution of at least one of such systems of constraints. Vice versa, each solution of one of these systems can be used to generate a complete q-level for $\mathscr{E}'$, for a suitable q. Such mapping is realized as follows: for each (term) variable X in $\mathscr{E}'$ we introduce a corresponding (integer) variable x; then we add equations and disequations according to the following rules:

$$\begin{aligned} X = X &\mapsto \textit{if } X \textit{ does not occur elsewhere, then add } x = 0 \\ X = Y &\mapsto x = y \\ X = f(Y_1, \ldots, Y_n), n > 0 &\mapsto \textstyle\bigvee_{i=1}^{n} (x = y_i + 1 \wedge \bigwedge_{j=1, j \neq i}^{n} y_j \leqslant y_i) \\ X = a &\mapsto x = 1 \\ X = Y_1 \cup Y_2 &\mapsto (x = y_1 \wedge y_2 \leqslant y_1) \vee (x = y_2 \wedge y_1 \leqslant y_2) \end{aligned}$$

Through simplifications (e.g., distributivity) it is possible to obtain a disjunction of systems $S_1 \vee \cdots \vee S_k$, where each system S_i contains only equations of the form:

$$x = y \qquad x = y + 1 \qquad x = 0 \qquad x = 1 \qquad x \leqslant y$$

Furthermore, in each system there can be at most p occurrences of equations of the type $x = y + 1$ and $x = 1$. Our aim is to show that, if one of the systems S_i is satisfiable, then there will be one solution σ of S_i such that for each variable x we have $\sigma(x) \leqslant p$.

Each system S_i can be further simplified using the following observations:

- All equations of the form $x = 1$ can be eliminated and replaced with the equations $w = 0$ and $x = w + 1$, where w is a new variable. Note that the total number of equations $x = y + 1$ is still at most p even after this simplification.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

$$
\begin{array}{rcl}
y_2 & = & 0 \\
y_2 & \leqslant & x \\
y_2 & \leqslant & y_1 \\
x & = & y_1 + 1 \\
y_1 & = & y_2 + 1 \\
y_3 & \leqslant & x
\end{array}
$$

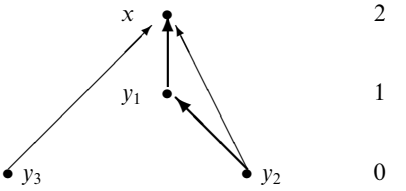


Fig. A 1. A simplified integer system, the corresponding graph, and a solution

- The equations of the form $x = y$ induce an equivalence relation on the variables. We can remove all these equations and replace each occurrence of each variable in S_i with a selected representative from its equivalence class.

For each S_i we can construct a labeled graph $G_{S_i} = \langle vars(S_i), E_i \rangle$ as follows (see example in Figure A 1; thick lines are used for $+1$ edges and thin lines for 0 edges):

- for each equation of the form $x = y + 1$ in S_i, generate an edge (y, x) with label $+1$
- for each equation of the form $x \leqslant y$ in S_i, generate an edge (x, y) with label 0, unless there is already an edge (x, y) with label $+1$.

If G_{S_i} contains a cycle with at least one edge labeled $+1$, then S_i will not admit solutions. Cycles in G_{S_i} composed only of edges of type 0 denote an implicit equality between the nodes in the cycle – thus we can collapse the cyclic component. These observations allow us to focus only on directed acyclic graphs.

A solution of S_i can be described as a labeling of the nodes of the graph. A *consistent* labeling σ of the nodes of the graph representing a solution should fulfill the following conditions:

- if (x, y) is an edge of type $+1$, then $\sigma(y) = \sigma(x) + 1$
- if (x, y) is an edge of type 0, then $\sigma(x) \leqslant \sigma(y)$
- if there is an equation $x = 0$ in S_i, then $\sigma(x) = 0$.

We claim that if the G_{S_i} admits a labeling with the above properties, then G_{S_i} also admits a labeling ϕ of the nodes such that for each node X we have $\phi(X) \leqslant r$, where r is the number of $+1$ edges in G_{S_i} – in particular $r \leqslant p$.

Let us develop a proof by lexicographical induction over the measure $\langle A, B \rangle$, where A is the number of $+1$ edges and B is the number of 0 edges in the graph.

$\langle 0, 0 \rangle$ In this case the graph is composed only of disconnected nodes, and the original system S_i contains only equations of the form $x = 0$; the solution σ such that $\sigma(x) = 0$ for each node x is a consistent 0-labeling.

$\langle m, n \rangle$ Let x be an arbitrary node of G_{S_i} with no outgoing edges, $(v_1, x), \ldots, (v_h, x)$ incoming edges of type $+1$, and $(w_1, x), \ldots, (w_k, x)$ incoming edges of type 0. With no loss of generality we assume $h + k \geqslant 1$. Let us distinguish the following cases:

1. $h = 0$: consider the graph G'_{S_i} obtained by removing node x and all its incoming edges (all of type 0). The measure for the graph G'_{S_i} is $\langle m, n - k \rangle$. By inductive hypothesis, there is a consistent m-labeling σ of G'_{S_i}. σ

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

can be extended to a consistent m-labeling of G_{S_i} by assigning $\sigma(x) = \max\{\sigma(w_1), \ldots, \sigma(w_k)\}$.

2. $h > 1$ and $k \geqslant 0$: in each consistent labeling of G_{S_i} we must have that $\sigma(v_1) = \cdots = \sigma(v_h) = \sigma(x) - 1$. Let us consider the graph G'_{S_i} obtained by collapsing nodes $v_1, \ldots, v_h$ into a single node v. The measure of G'_{S_i} is $\langle m - h + 1, n\rangle$; thus, by inductive hypothesis, it is possible to determine a consistent $(m-k+1)$-labeling σ of G'_{S_i}. σ can be extended into a consistent $(m-k+1)$-labeling of G_{S_i} by defining $\sigma(v_1) = \cdots = \sigma(v_h) = \sigma(v)$. By definition σ is also a consistent m-labeling of the graph.
3. $k = 0$ and $h = 1$: consider the graph G'_{S_i} obtained by removing X and its incoming edge. The measure of G'_{S_i} is $\langle m-1, n\rangle$, thus, by inductive hypothesis, there is a consistent $(m-1)$-labeling σ of such graph. This labeling can be extended to a consistent m-labeling of G_{S_i} by defining $\sigma(x) = \sigma(v_1) + 1$.
4. $k > 0$ and $h = 1$: in each consistent labeling of G_{S_i} we must have that:
   (a) $\sigma(v_1) = \sigma(x) - 1$
   (b) $\sigma(w_i) \leqslant \sigma(x)$ for $i = 1, \ldots, k$, thus $\sigma(w_i) = \sigma(x)$ or $\sigma(w_i) \leqslant \sigma(v_1)$

   Let us consider the following class of simplified graphs: we arbitrarily partition $\{w_1, \ldots, w_k\}$ into two subsets B_1, B_2 and we consider the graph obtained by:
   - removing all edges (w_i, x)
   - collapsing all nodes in $B_1 \cup \{x\}$
   - adding the edges (w_i, v_1) for each $w_i \in B_2$
   - if $B_1 = \emptyset$, then the node x and the edge (v_1, x) are removed.

   The two properties (4a) and (4b) guarantee that each consistent labeling of G_{S_i} is a consistent labeling of at least one of the simplified graphs, and each consistent labeling of a simplified graph can be extended (see below) to a consistent labeling of G_{S_i}. Since we are under the assumption that G_{S_i} admits consistent labelings, at least one of the simplified graphs admits consistent labelings. The measure of each simplified graph is $\langle m, n - |B_1|\rangle$ if $B_1 \neq \emptyset$, $\langle m-1, n\rangle$ otherwise. By inductive hypothesis we can build a consistent m-labeling (or $(m-1)$-labeling in the last case) σ for such graph. If $B_1 \neq \emptyset$, then σ can be extended to a consistent m-labeling of G_{S_i} by defining $\sigma(w_i) = \sigma(x)$ for each $w_i \in B_1$. Otherwise, a consistent m-labeling of G_{S_i} is obtained by defining $\sigma(x) = \sigma(v_1) + 1$. □

The notion of p-level has a direct interpretation on a graph-encoding of the system of equations. The unification algorithm itself can be mapped on a collection of graph manipulation operations. The mapping of the unification algorithm on graphs allows us to intuitively demonstrate that for each intermediate system of equations during the unification process it is possible to determine a partial p-level (where p is the number of occurrences of elements of Σ in the initial system).

Given the initial system $\mathscr{E}_0$ we define the directed labeled graph G_0 as follows:

- G_0 contains a node for each occurrence of a function symbol in $\mathscr{E}_0$; without loss of generality, we assume that each occurrence of a constant c has been

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

replaced with a term $c(B)$, where B is a fixed variable, and c is a new unary function symbol.

- G_0 contains a node for each variable in $\mathscr{E}_0$.
- For each term $f(t_1,\ldots,t_n)$ (f different from $\cup$) in $\mathscr{E}_0$, if μ is the node created for the specific occurrence of f, and v_i is the node created for the main functor of t_i (or for the variable t_i), then the edge (μ, v_i) with label $+1$ is added to G_0.
- let t be a term $t_1 \cup \cdots \cup t_n$ such that: $n > 1$, the main functor of each t_i is different from $\cup$, and either
  - — the term t is the left-hand side or the right-hand side of an equation in $\mathscr{E}_0$; or
  - — there exists a term $f(t_1,\ldots,t_n)$ in $\mathscr{E}_0$ such that $t \equiv t_i$ and f is different from $\cup$.

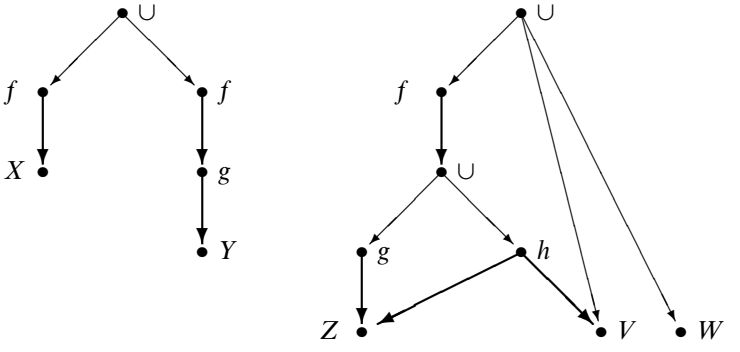
  Let μ be the node introduced for the first occurrence of $\cup$ in t, i.e.,

$$t_1 \underbrace{\cup}_{\Uparrow} t_2 \cup \cdots \cup t_n$$

  and let v_i be the node created for the main functor of t_i (or for the variable t_i); the graph G_0 contains the edges (μ, v_i) with label 0 for $i = 1,\ldots,n$.
- remove from G_0 all the nodes created for occurrences of $\cup$ which do not have any outgoing edges.

Example 9

Let $\mathscr{E}_0$ be the system $f(X) \cup f(g(Y)) = f(g(Z) \cup h(Z,V)) \cup V \cup W$. Then G_0 is the graph (thick lines are used for $+1$ edges while thin lines are used for 0 edges):



Let us define an *iteration* to be a single application of a rule of the procedure `general_aci`. Each rule of the unification algorithm can be mapped onto an operation on the graph. If $\mathscr{E}_i$ is the system obtained after i iterations of the unification algorithm, then we denote with G_i the corresponding graph. The graph operations corresponding to the different non-failing unification rules are the following:

1. if $\mathscr{E}_{i+1}$ is obtained by removing an equation $X = X$ from $\mathscr{E}_i$, then $G_{i+1} = G_i$
2. if $\mathscr{E}_{i+1}$ is obtained by replacing $t = X$ with $X = t$ in $\mathscr{E}_i$, then $G_{i+1} = G_i$
4. if $\mathscr{E}_{i+1}$ is obtained by replacing each occurrence of X with t in $\mathscr{E}_i$, then G_{i+1} is obtained by adding the edge (μ, v) with label 0, where μ is the node associated

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

with the variable X and v is the node created for the main functor of term t (or for the variable t)

5. if $\mathscr{E}_{i+1}$ is obtained by replacing the equation $X = t$ where $t \equiv f_1(\bar{s}_1) \cup \cdots \cup f_n(\bar{s}_n) \cup V_1 \cup \cdots \cup V_m \cup X$ (assumed in this ordered form as explained in the unification algorithm – note that this simplification is not needed in the graph representation) with the equation $X = f_1(\bar{s}_1) \cup \cdots \cup f_n(\bar{s}_n) \cup V_1 \cup \cdots \cup V_m \cup N$, N new variable, then G_{i+1} is obtained by adding a new node v for N, by removing the edge (μ, ξ) where μ is the node for the functor of t and ξ the node for X, and by adding the new edge (μ, v)
7. if $\mathscr{E}_{i+1}$ is obtained by replacing the equation $f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)$ in $\mathscr{E}_i$, then $G_{i+1} = G_i$
8. let us assume that $\mathscr{E}_{i+1}$ is obtained by replacing the equation

$$f_1(\bar{s}_1) \cup \cdots \cup f_n(\bar{s}_m) \cup X_1 \cup \cdots \cup X_h = g_1(\bar{t}_1) \cup \cdots \cup g_n(\bar{t}_n) \cup Y_1 \cup \cdots \cup Y_k$$

in $\mathscr{E}_i$ with a family of equations:

$$f_i(\bar{s}_i) = g_j(\bar{t}_j) \qquad \text{for some } i, j$$

and by substituting X_i (Y_i) with terms of the form:

$$X_i = g_{i_1}(\bar{t}_{i_1}) \cup \cdots \cup g_{i_r}(\bar{t}_{i_r}) \cup N_1 \cup \cdots \cup N_s$$

(similarly for Y_i). G_{i+1} is obtained from G_i as follows:

— introducing a new node v_i for each new variable N_i

— if μ is the node for X_i (Y_i) and η_j is the node for the main functor of term $g_j(\bar{t}_j)$ ($f_j(\bar{s}_j)$), then add the edge (μ, v_w) and (μ, η_j) with label 0 for each $g_j(\bar{t}_j)$ ($f_j(\bar{s}_j)$) and for each N_w present in the substitution for X_i (Y_i).

Lemma 4

Let $One(G)$ be the set of $+1$ edges present in the graph G. Then for each G_i obtained from the above transformations we have $One(G_i) = One(G_0)$. Furthermore, G_i does not contain any cycles which include edges labeled $+1$.

Proof

The first property is obvious from the definition of the transformations.

The second property is straightforward for the cases (1), (2), and (7) of the unification algorithm, since they do not add edges – and thus cannot generate cycles. Case (5) adds a new edge, but the destination of the edge is a new variable which has no outgoing edges.

Case (4) can be seen as follows: let us assume, by contradiction, that the addition of the edge from the node of X to the node of t generates a cycle with $+1$ edges. This means that, before this operation, there exists a path from the root of t to the node of X (with at least one $+1$ edge). This path can be only the result of a sequence of edge additions leading from a node reachable from the root of t to the node of X. Each of these edges has been introduced during previous variable substitutions – and each of the nodes reachable using this path identifies a sub-term of t. Thus, X is a sub-term of t. This contradicts the possibility of applying case (4), since this situation is explicitly handled by case (3) and leads to a failure.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

Case (8) can be seen as a combination of cases (7) (new equations of the type $f_i(\bar{s}_i) = g_j(\bar{t}_j)$ which do not modify the graph), (5) for the new variables N_i, and (4) for the substitution of existing variables. □

Lemma 5
Let us assume that there is a non-failing sequence of k non-deterministic choices, such that `general_aci`($\mathscr{E}$) generates (one per each successive iteration) the systems $\mathscr{E} = \mathscr{E}^{(0)}, \mathscr{E}^{(1)}, \mathscr{E}^{(2)}, \ldots, \mathscr{E}^{(k)}$. Let p be the number of occurrences of function symbols in $\mathscr{E}^{(0)}$. Then, there exists

$$lev : vars\left(\bigcup_{j=0}^{k} \mathscr{E}^{(j)}\right) \longrightarrow \mathbb{N}$$

such that:

- it fulfills condition (∗) of Def. 11 for all systems of equations $\mathscr{E}^{(j)}$ (i.e., $lev(\ell) \leqslant p$ and $lev(r) \leqslant p$ for all the equations $\ell = r$ in $\mathscr{E}^{(j)}$), and
- any time a substitution $[X/t]$ has been applied, then $lev(X) = lev(t)$.

Proof
Let us consider the graphs G_j associated to the systems $\mathscr{E}^{(j)}$. First of all, observe that if there is a function fulfilling the requirements for the system $\mathscr{E}^{(j)}$, then the same function works for all graphs $\mathscr{E}^{(i)}$ with $i < j$. This allows us to concentrate on $\mathscr{E}^{(k)}$. By Lemma 4, we know that G_k is acyclic and it contains the same number (p) of $+1$ edges as $\mathscr{E}^{(0)}$. From this fact, starting from leaf nodes and going back on edges, augmenting a value only if a $+1$ edge is encountered, it is natural to find a function *lev* fulfilling the required property. □

Theorem 3 (*termination*)
Given a system of equations $\mathscr{E}$, all the non-deterministic branches of the computation of `general_aci`($\mathscr{E}$) terminate in a finite number of steps.

Proof
Assume that there is a non-failing sequence of non-deterministic choices $\mathscr{E}^{(0)}, \mathscr{E}^{(1)}$, $\mathscr{E}^{(2)}, \ldots, \mathscr{E}^{(k)}$ (they are the values of $\mathscr{E}$ at the $0^{th}, 1^{st}, 2^{nd}, \ldots, k^{th}$ iteration, respectively), and let p be the number of occurrences of function symbols in $\mathscr{E}^{(0)}$. We know from Lemma 5 that there exists a function $lev : vars\left(\bigcup_{j\geqslant 0} \mathscr{E}^{(j)}\right) \longrightarrow \mathbb{N}$ such that

- it fulfills condition (∗) for all the systems of equations $\mathscr{E}^{(j)}$, and
- each time a substitution $[X/t]$ has been applied, then $lev(X) = lev(t)$,

We call this property condition (α).

Picking such a *lev*, we define a measure of complexity $\mathscr{L}_{\mathscr{E}}$ for the system of equations $\mathscr{E}$:

$$\mathscr{L}_{\mathscr{E}}^{(lev)} \quad = \quad [\#(2p), \#(2p-1), \#(2p-2), \ldots, \#(1), \#(0)]$$

where $\#(j)$ returns the number of equations *not in solved form* $\ell = r$ in $\mathscr{E}$ such that $lev(\ell) + lev(r) = j$. The ordering between two lists of this form is the usual well-founded lexicographical ordering.

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

Let h be the number of equations in the initial system. The initial tuple $\mathscr{L}^{(lev)}_{\mathscr{E}^{(0)}}$ is necessarily less than or equal to $[h, 0, \ldots, 0]$. Let us consider how the various rules in Figure 6 modify the complexity measure tuple:

- rule (1) clearly reduces the complexity by removing one equation
- rule (2) does not affect the complexity but can be safely ignored (we could easily rewrite the algorithm without it by adding explicit cases for equations $t = X$ wherever we analyze $X = t$)
- rule (4) reduces the complexity: in fact one equation of complexity $2lev(X)$ is removed, while the rest of the system is unaffected, since X is replaced by a term with the same level
- rule (5) will lead in one additional iteration to a rule (4), which means that the complexity of the original equation must be $2lev(X)$; by assigning $lev(N) = lev(X)$ we have that after two reductions the complexity will decrease
- rule (7) replaces an equation of complexity $2 + l_1 + r_1$ with a collection of equations each having complexity $l + r \leqslant l_1 + r_1$, leading to a smaller total complexity (thanks to lexicographical ordering)
- rule (8) is a complex rule which leads to the execution of the `aci_step` function. Let e be the equation communicated to `aci_step`. The only equations in non-solved form that are generated by `aci_step` are the equations $\bar{s} = \bar{s}'$ present in $\mathscr{E}_{\mathtt{ns}}$. Such an equation $\bar{s} = \bar{s}'$ originates from simplifying an equation $f(\bar{s}) = f(\bar{s}') \cup \cdots$. Observe that in this equation $f(\bar{s})$ and $f(\bar{s}')$ originally appeared on distinct sides of the equation e – in the general structure of the equation, one of the two is a $f_j(\bar{l}_j)$ and the other is a $g_i(\bar{r}_i)$. Thus, the equation $f(\bar{s}) = f(\bar{s}')$ has a complexity which is less or equal than that of e, which implies also that the complexity of $\bar{s} = \bar{s}'$ is strictly lower than that of e. Thus the original equation is replaced by a collection of equations of smaller complexity (assuming, as stated earlier, that the equations of the form $L_i = t_i$ and $R_j = s_j \cdots$ that lead to ρ are all such that $lev(L_i) = lev(t_i)$ and $lev(R_j) = lev(s_j)$).

Thus, every rule application decreases the complexity measure $\mathscr{L}^{(lev)}_{\mathscr{E}}$. The lexicographical ordering on constant-length lists of non-negative integers is a well-founded ordering, and thus this activity cannot be done indefinitely.

However, this is not sufficient for termination, since we are not sure that the complexity measure tuple reaches the value $[0, \ldots, 0]$ within k rule applications. Moreover, we do not know if the function lev fulfills condition (α) for the successive systems $\mathscr{E}^{(k+1)}, \mathscr{E}^{(k+2)}, \ldots$.

To prove termination, a further measure is needed: let

$$\mathscr{M}_{\mathscr{E}} = \{[\mathscr{L}^{(\ell)}_{\mathscr{E}} : \ell \text{ is a function from } vars(\mathscr{E}) \text{ to } \{0, \ldots, p\} \text{ that fulfills condition } (\alpha)]\}$$

Multisets of tuples are governed by (well-founded) multiset ordering.

$\mathscr{M}_{\mathscr{E}^{(0)}}$ is finite. All the initial tuples are less than or equal to $[r, 0, \ldots, 0]$; each of them is associated to a function from $vars(\mathscr{E}^{(0)})$ to $\{0, \ldots, p\}$ that fulfills condition (α).

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

Let us consider this multiset and the effects of an iteration over each of its tuples. After one iteration it holds that:

- The function ℓ fulfills condition (α) for the successive systems. In this case t is replaced by a smaller tuple (see the proof above).
- The function ℓ does not assign values to new variables. However, it is possible to extend ℓ into ℓ' in order to assign values for these variables. In this case the tuple t is replaced by a certain (finite) number of tuples smaller than t (the new variables N are introduced in equations of the form $X = \cdots \cup N$ and thus, $\ell'(N) \leqslant \ell(X)$).
- The function ℓ does not fulfill condition (α) for the new system and, moreover, it is not possible to extend ℓ into ℓ' in order to assign values for these variables to fulfill condition (α). In this case the tuple t is simply removed from the multiset.

Since multiset ordering is well-founded, this ensures termination. □

Appendix B Matrix for *Term Propagation*

In this section we briefly show how it is possible to compute automatically the output equations of the *Term Propagation* phase of the General ACI unification algorithm (Section 8.3). The method we propose builds on the solution of the ACI unification with constants problem based on ACI-matrices; the novelty is the use of a simplified form of ACI-matrix that takes advantage of the format of the equations to be dealt with in this context, i.e., elementary $ACI1$ equations.

Given an elementary $ACI1$ unification problem

$$S_1 \cup \cdots \cup S_n \cup X_1 \cup \cdots \cup X_p = T_1 \cup \cdots \cup T_m \cup X_1 \cup \cdots \cup X_p$$

the *elementary ACI-matrix* is as follows:

S_1	$\dots$	S_n	X_1	$\dots$	X_p	
$A_{1,1}$	$\dots$	$A_{1,n}$	$A_{1,n+1}$	$\dots$	$A_{1,n+p}$	T_1
$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$A_{m,1}$	$\dots$	$A_{m,n}$	$A_{m,n+1}$	$\dots$	$A_{m,n+p}$	T_m
$A_{m+1,1}$	$\dots$	$A_{m+1,n}$	$A_{m+1,n+1}$	$\dots$	$A_{m+1,n+p}$	X_1
$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$A_{m+p,1}$	$\dots$	$A_{m+p,n}$	$A_{m+p,n+1}$	$\dots$	$A_{m+p,n+p}$	X_p

However, variables $A_{m+i,n+j}$ with $i > 0, j > 0, i \neq j$ are not used and thus we can avoid their introduction. The most general unifier for the elementary problem can be obtained as follows:

$$S_j = \bigcup_{i=1}^{m+p} A_{i,j} \qquad T_i = \bigcup_{j=1}^{n+p} A_{i,j}$$

$$X_j = \bigcup_{i=1}^{m} A_{i,n+j} \cup \bigcup_{k=1}^{n} A_{m+j,k} \cup A_{m+j,n+j}$$

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

One can easily prove that this method provides the same solution as the ACI unification with constants algorithm based on Boolean ACI matrices of (Baader and Büttner 1988), briefly recalled in Section 6.

Example 10
Let us consider the same unification problem $S_1 \cup S_2 \cup X = T_1 \cup T_2 \cup X$ as in Example 3; the elementary ACI-matrix is

S_1	S_2	X	
R_1	R_3	R_7	T_1
R_2	R_4	R_8	T_2
R_5	R_6	R_9	X

Let us observe that the variables in the matrix have been named to show the correspondence with the new variables used in Example 3.

Given the unification problem:

$$\begin{aligned}\mathscr{E}^n \equiv\ & N_1^L = f_1(\bar{l}_1) \wedge \cdots \wedge N_{k_1}^L = f_{k_1}(\bar{l}_{k_1}) \wedge \\ & N_1^R = g_1(\bar{r}_1) \wedge \cdots \wedge N_{h_1}^R = g_{h_1}(\bar{r}_{h_1}) \wedge \\ & N_1^L \cup \cdots \cup N_{k_1}^L \cup L_1 \cup \cdots \cup L_{k_2} = N_1^R \cup \cdots \cup N_{h_1}^R \cup R_1 \cup \cdots \cup N_{h_2}^R\end{aligned}$$

we solve the elementary $ACI1$ problem on the equation:

$$N_1^L \cup \cdots \cup N_{k_1}^L \cup L_1 \cup \cdots \cup L_{k_2} = N_1^R \cup \cdots \cup N_{h_1}^R \cup R_1 \cup \cdots \cup R_{h_2}$$

We build an auxiliary Boolean matrix B that allows us to reduce the non-determinism. We deal with two cases:

- If $\{L_1, \ldots, L_{k_2}\} \cap \{R_1, \ldots, R_{h_2}\} = \emptyset$ any (non-deterministic) solution can be described using a $(h_1 + h_2) \times (k_1 + k_2)$ matrix B such that
  - — for $h_1 + 1 \leqslant i \leqslant h_1 + h_2$ and $k_1 + 1 \leqslant j \leqslant k_1 + k_2$ we have $B[i, j] = \bot$
  - — all the other components of B have a value taken from $\{0, 1\}$
  - — for each $1 \leqslant i \leqslant h_1$ $\sum_{j=1}^{k_1+k_2} B[i, j] \geqslant 1$ and for each $1 \leqslant j \leqslant k_1$ $\sum_{i=1}^{h_1+h_2} B[i, j] \geqslant 1$.

  Thus, B is a boolean matrix with the exception of the fourth quadrant, where the matrix contains only the value $\bot$. The matrix B can be used to describe the substitution λ:

$$\lambda(A_{i,j}) = \begin{cases} A_{i,j} & \text{if } B[i, j] = \bot \\ \emptyset & \text{if } B[i, j] = 0 \\ h(r_i) & \text{if } B[i, j] = 1 \wedge j > h_1 \\ h(l_j) & \text{if } B[i, j] = 1 \wedge j \leqslant h_1 \end{cases}$$

  Additionally, B generates the new set of equations:

$$E^{conf} = \bigwedge_{B[i,j]=1 \wedge 1 \leqslant i \leqslant k_1 \wedge 1 \leqslant j \leqslant h_1} h(l_j) = h(r_i)$$

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

- Assume now that the two sides of the equation share some variables. I.e., let us assume that the problem at hand is

$$N_1^L \cup \cdots \cup N_{k_1}^L \cup L_1 \cup \cdots \cup L_{k_2} \cup Com_1 \cup \cdots \cup Com_c = N_1^R \cup \cdots \cup N_{h_1}^R \cup R_1 \cup \cdots \cup R_{h_2} \cup Com_1 \cup \cdots \cup Com_c$$

The solution of the problem in this case can be built around the elementary ACI-matrix shown in Figure B 1. The table in Figure B 1 assumes $h = h_1 + h_2$ and $k = k_1 + k_2$. The solution of the ACI problem, in this case, will be composed of equations of the form:

$$\begin{aligned} L_j &= \bigcup_{i=1}^{h_1} A_{i,k_1+j} \cup \bigcup_{i=h_1+1}^{h_1+h_2} A_{i,k_1+j} \cup \bigcup_{i=h_1+h_2+1}^{h_1+h_2+c} A_{i,k_1+j} \\ R_i &= \bigcup_{j=1}^{k_1} A_{h_1+i,j} \cup \bigcup_{j=k_1+1}^{k_1+k_2} A_{h_1+i,j} \cup \bigcup_{j=k_1+k_2+1}^{k_1+k_2+c} A_{h_1+i,j} \\ Com_v &= \bigcup_{i=1}^{h_1+h_2+c} A_{i,k_1+k_2+v} \cup \bigcup_{j=1}^{k_1+k_2+c} A_{h_1+h_2+v,j} \end{aligned}$$

In Figure B 2, we depict the boolean matrix B which will be used in this case. The matrix B should satisfy the following properties:

— quadrant 5, 6, and 8 are filled with $\bot$;
— the non-zero entries in quadrant 9 are assigned $\bot$; observe that the quadrant 9 is a diagonal matrix with non-zero elements only along the main diagonal;
— quadrant 1, 2, 3, 4, and 7 are boolean matrices;
— for $1 \leqslant j \leqslant k_1$ we have $\sum_{i=1}^{h_1+h_2} B[i,j] + \sum_{i=1}^{c} B[i,j] \geqslant 1$
— for $1 \leqslant i \leqslant h_1$ we have $\sum_{j=1}^{k_1+k_2} B[i,j] + \sum_{j=1}^{c} B[i,j] \geqslant 1$

The substitution λ and the collection of new equations E^{conf} are defined exactly as above.

N_1^L	…	$N_{k_1}^L$	L_1	…	L_{k_2}	Com_1	…	Com_c	
$A_{1,1}$	…	A_{1,k_1}	A_{1,k_1+1}	…	$A_{1,k}$	$A_{1,k+1}$	…	$A_{1,k+c}$	N_1^R
	…			…			…		⋮
$A_{h_1,1}$	…	A_{h_1,k_1}	A_{h_1,k_1+1}	…	$A_{h_1,k}$	$A_{h_1,k+1}$	…	$A_{h_1,k+c}$	$N_{h_1}^R$
$A_{h_1+1,1}$	…	A_{h_1+1,k_1}	A_{h_1+1,k_1+1}	…	$A_{h_1+1,k}$	$A_{h_1+1,k+1}$	…	$A_{h_1+1,k+c}$	R_1
	…			…			…		⋮
$A_{h,1}$	…	A_{h,k_1}	A_{h,k_1+1}	…	$A_{h,k}$	$A_{h,k+1}$	…	$A_{h,k+c}$	R_{h_2}
$A_{h+1,1}$	…	A_{h+1,k_1}	A_{h+1,k_1+1}	…	$A_{h+1,k}$	$A_{h+1,k+1}$	…	$A_{h+1,k+c}$	Com_1
	…			…			…		⋮
$A_{h+c,1}$	…	A_{h+c,k_1}	A_{h+c,k_1+1}	…	$A_{h+c,k}$	$A_{h+c,k+1}$	…	$A_{h+c,k+c}$	Com_c

Fig. B 1. Elementary ACI-matrix

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press

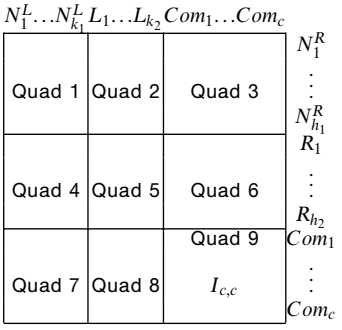


Fig. B 2. Extended Boolean Matrix B

https://doi.org/10.1017/S1471068406002730 Published online by Cambridge University Press