Set Unification

Agostino Dovier
Università di Udine
Dip. di Matematica e Informatica
Via delle Scienze 206
33100 Udine (Italy)
dovier@dimi.uniud.it

Enrico Pontelli
New Mexico State University
Dept. Computer Science
Box 30001, Dept. CS
Las Cruces, NM 88003 (USA)
epontell@cs.nmsu.edu

Gianfranco Rossi
Università di Parma
Dip. di Matematica
Via M. D'Azeglio, 85/A
43100 Parma (Italy)
gianfr@prmat.math.unipr.it

Abstract

The goal of this paper is to provide a uniform overview of the unification problem in algebras capable of describing *sets*. The problem 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. Facets of this problem have been addressed by several authors, but the various solutions proposed are spread across a large literature. In this paper we provide a uniform presentation of unification of sets. The problem is formalized 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. Uniform unification algorithms are 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. However, the major contribution of this work is to provide the first uniform presentation of the set unification problem, covering all its different instances, and surveying the different solutions proposed.

Keywords: Unification Theory, Set Theory, $ACI1$ Unification.

1 Introduction

Sets are well-known mathematical objects and are also often used as an high-level abstraction for possibly complex data structures. Sets can be used to model collections of data in which the order and repetitions of elements do not matter.

According to the traditional *extensionality* axiom [39], 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 as 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, (uninitialized) variables can occur in set terms 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 the 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 (in other words, one-way set unification). Set unification can be thought of as a particular case of E-unification, that is unification modulo an equational theory E, where E describes the (semantic) properties of the interpreted symbols used to represent sets through equalities.

Two main approaches for representing sets as terms are described 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 $\{\cdot\,|\,\cdot\}$). List-like representation has been frequently used in the context of logic languages embedding sets. It is used for instance in [40], in [13], where $\{\cdot\,|\,\cdot\}$ is called **scons**, in the language {log} [19], and also in the Gödel language [31]. In set-theoretic papers $\{\cdot\,|\,\cdot\}$ is used and called `with` [16]. [32] uses the $\cup$ operator but its behavior is that of the $\{\cdot\,|\,\cdot\}$ operator. The union-based representation, on the contrary, has been often used when dealing with the problem of set unification on its own [14, 45], where set unification is dealt with as an *ACI unification* problem (i.e, unification under Associativity, Commutativity, and Idempotence). In [41] sets are represented as in the union-based approach. However, since set operations are evaluated only when applied to *ground sets*, set unification is not required at all. Indeed, equations between sets containing uninitialized variables are delayed and possibly returned as part of the computed answer.

Complexity of the set unification/matching problem has been investigated by Kapur and Narendran [34, 35], who established that these decision problems are NP-complete. Complexity of the set unification/matching operation, however, depends on which form of set terms are allowed: e.g., flat or nested sets, with zero, one, or more set variables. The form of set terms in turn is influenced by the set constructor we use to build them. Thus, different complexity results (possibly better than NP) can be obtained for different classes of set terms.

In this paper we propose a 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 unification problem considered are presented and analyzed. These algorithms are either drawn from the literature or 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 Using Set Unification

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

Declarative programming languages with sets. Various declarative programming languages relying on sets as *first-class objects* use some, possibly restricted, form of set unification as the basic operation on sets. Most of these languages belong to the (constraint) logic programming paradigm [19, 22, 67] or to the functional-logic paradigm [32, 6]. The specification language Z [62] makes use of sets as data abstraction; attempts have been made to produce executable versions of Z, such as the ZAP compiler [28].[1]

Deductive databases. Various proposals of deductive databases offering sets as a primitive data structure provide also set unification, or more often set matching as a built-in mechanism for set manipulation [44, 1, 51, 59, 42, 36]. In this framework it is common to have to deal with sets involving unions of variables.

AI and Automated deduction. Set descriptions 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 [46]. Set unification is frequently adopted in works on computational linguistics. In particular, unification based grammars augmented with set descriptions (cf., e.g., [56, 55]) are likely to require set unification. Set unification is also used in discovery procedures for determining categorial grammars from linguistic data (cf., e.g., [47]). Set data structures have also been used in pattern matching and pattern directed invocation in various AI languages [45]. Proposals dealing with computability properties and algorithmic manipulation of set structures have recently appeared also in the area of automated deduction, e.g., to reduce the length of proofs [54].

[1] Nevertheless, the current implementation of ZAP does not embed a set unification algorithm.

Program analysis. Codish and Lagoon [17] describe an application of elementary $ACI1$ unification to the problem of *sharing analysis* of logic programs.

1.2 Unification Algorithms

Set unification has been tackled mainly as ACI unification. The first work proposing a viable solution to ACI unification—hence, to set (and bag) unification—is [45]. Actually, [45] mostly deals with AC unification—by reducing it to the solution of Diophantine equations—and only towards 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 [14] and Baader and Büttner [9]. More recently, Baader and Schulz [10] provided a general methodology allowing the unification with constants algorithms proposed for ACI to be extended to general 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 [19]). A first proposal in this direction is the algorithm sketched by Jayaraman and Plaisted in [33]. A more general and complete algorithm is the one in [19]. The problem of set unification in this context has been tackled by different authors [5, 23, 63, 64, 18]. In particular, the algorithms presented in [5] and [64] provide solutions which are optimal in terms of number of unifiers for large classes of unification problems. The algorithms in [5, 18] 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 (e.g., [59]) and unification of *Bound Simple* set terms, i.e., bound set terms of the form $\{s_1, \dots, s_n\}$, where each s_i is either a constant or a variable [7, 27]. A parallel algorithm for such restricted $(Ab)(C\ell)$ unification has been presented in [42]. Set matching is also discussed in [34].

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 discuss the complexity of the set unification decision problem for each syntactic class of set terms listed in the previous section. In Section 4 we introduce the basic notions and notation concerning E-unification and the equational theories used in E-unification with sets. In Section 5 we describe the problem of $ACI1$ *unification with constants* and its impact on set unification. In Section 6 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 constructor $\{\cdot\,|\,\cdot\}$, and we present a new algorithm for this case. Next, in Section 7 we tackle the most general problem of unification of terms containing both $ACI1$ and free function symbols. A new general $ACI1$ unification algorithm is presented. A collection of examples of unification problems are presented in Section 8 and some related topics are discussed in Section 9. Concluding remarks are presented in Section 10.

2 Sets and the Set Unification Problem

We introduce sets as mathematical objects. We present a well-defined set universe, 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 collection of elements that can be specified intensionally, by means of the property characterizing exactly the elements, or extensionally, by enumeration of all its elements. In this paper we will restrict our attention to extensional sets only. For instance, $\{a, b, c\}$ is the set containing 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 [39] 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. $\{a, c, b\}$, $\{b, a, c\}$, ... are alternative ways for writing 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.[2] 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. 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) [2, 12]; they are very important in some areas, such as concurrency theory, but they are not accepted in traditional set theory, where sets are 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 $\emptyset$, and
- *sets with individuals*, in which there is a collection $\mathcal{U}$ of individuals, and each individual is not a set (hence, $\emptyset \notin \mathcal{U}$) and it does not contain elements.

In the second approach the extensionality axiom should be revised for individuals, since (i) two different individuals are different even if they contain the same elements (none), and (ii) all of them 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 $\mathcal{U} = \emptyset$).

Let us introduce the universe of sets we are interested in (see also [16], page 89). If s is a set, $\wp_{\text{fin}}(s) = \{x : (\forall y \in x)(y \in s) \land x \text{ is finite}\}$ denotes the set of all its finite subsets.

Definition 2.1 *The* Universe $\mathbb{HF}^{\mathcal{U}}$ of hereditarily finite sets based on $\mathcal{U}$ *is obtained as follows:*

$$\left\{ \begin{array}{lcl} \mathbb{HF}_0^{\mathcal{U}} & \stackrel{\mathcal{D}ef}{=} & \wp_{fin}(\mathcal{U}) \\ \mathbb{HF}_{i+1}^{\mathcal{U}} & \stackrel{\mathcal{D}ef}{=} & \mathbb{HF}_i^{\mathcal{U}} \cup \wp_{fin}(\mathbb{HF}_i^{\mathcal{U}}) \\ \mathbb{HF}^{\mathcal{U}} & \stackrel{\mathcal{D}ef}{=} & \bigcup_{i \in \mathbb{N}} \mathbb{HF}_i^{\mathcal{U}} \end{array} \right.$$

The sets in $\mathbb{HF}_0^{\mathcal{U}}$ are the finite subsets of the set of individuals: these particular sets are called *flat* sets. Sets introduced in $\mathbb{HF}_i^{\mathcal{U}}$, with $i > 0$, contain elements that are sets themselves. We refer to such sets as *nested sets*. For instance, if $\mathcal{U} = \{a, b, c\}$, then $\mathbb{HF}_0^{\mathcal{U}}$ 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] A precise, formal, characterization of the notion of finiteness is outside the scope of this work. For a theoretical analysis of this topic see, e.g., [65].

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 terms of the language. For this purpose, 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 $\{\cdot\}$. $s \cup t$ is the set containing the elements of the sets s and t, that is,
$$s \cup t = \{x \,:\, x \in s \vee x \in t\},$$
while $\{t\}$ is the set containing the single element t. With this approach, the finite set $\{t_0, \ldots, t_n\}$ is represented as 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 $\{\cdot \,|\, \cdot\}$. $\{t \,|\, s\}$ is 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
$$\{t \,|\, s\} = \{x \,:\, x \in s \vee x = t\}.$$
Thus, the finite set $\{t_0, \ldots, t_n\}$ is represented as a sequence of element insertions: $\{t_0 \,|\, \{\cdots \{t_n \,|\, \emptyset\} \cdots\}\}$, where $t_0, \ldots, t_n$ are either sets or individuals. The empty set is represented by a distinguished constant $\emptyset$.

In both approaches, the elements of the sets are represented either by constants, by compound terms $f(t_1, \ldots, t_n)$ where f is not a set constructor and $t_1, \ldots, t_n$ are terms, or by other set terms. Observe that the element insertion constructor $\{\cdot \,|\, \cdot\}$ can be implemented using $\cup$ as $\{s \,|\, t\} \stackrel{\mathcal{D}ef}{=} s \cup \{t\}$. However, in [20] it is proved that, without the possibility of building singleton sets (e.g., using $\{\cdot\}$ as the singleton set constructor) the two symbols are not mutually definable without using formulas with universal quantifiers.

Furthermore, note that, being based on membership, the definition of $\cup$ makes sense on sets, not on individuals. For instance, the union of two individuals a and b would be a memberless object. But 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, the $\cup$ constructor is used only on sets. Similar considerations apply to the second argument of the $\{\cdot \,|\, \cdot\}$ operator.

For the sake of simplicity, in the rest 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 2.2 *An* abstract set term *is a term of the form*
$$\{X_1, \ldots, X_m, a_1, \ldots, a_n, s_1, \ldots, s_p\} \cup Y_1 \cup \cdots \cup Y_q$$
where X_i, Y_i are variables, a_i are individual terms, and s_i are non-variable abstract set terms, $m \geq 0, n \geq 0, p \geq 0, q \geq 0$. When $m, n, p,$ or q are 0, the corresponding list is empty. In particular when $m = n = p = q = 0$ we write $\emptyset$. 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 will be written using capital letters.

When $q \leq 1$, 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 $\{X_1 \,|\, \{X_2 \,|\, \{a \,|\, \{b \,|\, \{c \,|\, Y_1\}\}\}\}\}$ 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 representation (1) is feasible.

In the rest of this section we will typically talk about abstract set terms. When clear from the context, however, we will omit the word 'abstract', referring to abstract set terms simply as set terms.

Set terms may contain variables, both as individuals (variables X_i's) and as sets (variables Y_j's). A set term containing variables actually 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 will take the value a or b then the set will have only 2 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.

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.

Definition 2.3 *If $s \equiv \{a_1, \ldots, a_n, s_1, \ldots, s_p\}$ is a ground set term, then $s^{\mathbb{HF}}$ 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 $a_1, \ldots, a_n$ and $s_1^{\mathbb{HF}}, \ldots, s_p^{\mathbb{HF}}$*

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, t, then $\mathbb{HF} \models \vec{\exists}(s = t)$ if and only if there exists an assignment σ of ground abstract set terms to $X_1, \ldots, X_n$ such that $\mathbb{HF} \models (s = t)\sigma$.

Definition 2.4 *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 2.5 *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 individuals for the variables $X_1, \ldots, X_n$, s.t. $\mathbb{HF} \models (s = t)\sigma$.*[3]

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 concerns nested set terms involving variables it becomes NP-complete (see Section 3.4). Thus, in order to classify the set unification problem, we split set terms into various syntactic classes.

Definition 2.6 *For $m \geq 0, n \geq 0, p \geq 0, q \geq 0$, the class $\mathsf{set}(m, n, p, q)$ is the set 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 $0 \leq m' \leq m$, $0 \leq n' \leq n$, $0 \leq p' \leq p$, $0 \leq q' \leq q$, and $s_i \in \mathsf{set}(m, n, p, q)$.

[3] In Section 4 we give a more standard and complete definition of the unification problem.

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

$\mathsf{ground} = \bigcup_{n\geq 0, p\geq 0} \mathsf{set}(0,n,p,0)$: $\{a_1,\dots,a_n,s_1,\dots,s_p\}$, with a_i individuals and s_i ground set terms.

$\mathsf{gflat}(q) = \bigcup_{n\geq 0} \mathsf{set}(0,n,0,q)$: $\{a_1,\dots,a_n\} \cup Y_1 \cup \cdots \cup Y_{q'}$, with a_i individual terms and Y_i variables ranging over $\mathsf{gflat}(q)$ sets (i.e., sets denoted by $\mathsf{gflat}(q)$ set terms) $(0 \leq q' \leq q)$.

$\mathsf{flat}(q) = \bigcup_{m\geq 0, n\geq 0} \mathsf{set}(m,n,0,q)$: $\{X_1,\dots,X_m,a_1,\dots,a_n\} \cup Y_1 \cup \cdots \cup Y_{q'}$, with a_i and X_i denoting individuals, and Y_i ranging over $\mathsf{flat}(q)$ sets $(0 \leq q' \leq q)$.

$\mathsf{nested}(q) = \bigcup_{m\geq 0, n\geq 0, p\geq 0} \mathsf{set}(m,n,p,q)$: $\{X_1,\dots,X_m,a_1,\dots,a_n,s_1,\dots,s_p\} \cup Y_1 \cup \cdots \cup Y_{q'}$, with a_i individual set terms, s_i non-variable $\mathsf{nested}(q)$ set terms, X_i ranging over individuals or $\mathsf{nested}(q)$ sets, Y_i ranging over $\mathsf{nested}(q)$ sets $(0 \leq q' \leq q)$.

$\mathsf{gflat}(q)$ and $\mathsf{flat}(q)$ denote flat sets only, while ground and $\mathsf{nested}(q)$ account for nested sets. Of course, for the same q, we have that $\mathsf{gflat}(q) \subseteq \mathsf{flat}(q) \subseteq \mathsf{nested}(q)$. Moreover, ground is included in $\mathsf{nested}(q)$ (actually, in $\mathsf{nested}(0)$) but it is not included in the other classes since ground accounts also for nested sets.

Observe that in the concrete representation of sets, whenever $q \leq 1$ the union constructor is not required. Therefore, we can use for these sets the representation (2), based 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 scheme (1)) while it has no corresponding representation in the scheme (2). These observations will play an important role when we will address the various unification problems.

3 The Set Unification Decision Problem and its Complexity

We show the complexity of the SUD problem for each one of the syntactic classes of set terms listed in the previous section.

3.1 SUD for ground

The set equivalence test for two ground abstract set terms s and t can be solved in linear time w.r.t. the size $||s|| + ||t||$ of the two terms. The proof is based on the graph representation of a set and on the existence of a fast algorithm for proving graph bisimulation. We first focus on the *pure* case (without individuals).

An *accessible pointed graph (*apg*)* $\langle G, \nu \rangle$ is a directed graph $G = \langle N, E \rangle$ together with a distinguished node $\nu \in N$ (the *point*), such that all the nodes in N are reachable from ν. Pure sets can be seen as *accessible pointed graphs*. Edges represent membership, namely $\langle m, n \rangle$ means that m has n as an element, and the nodes in the graph denote all the sets which contribute in the construction of the represented set. A node without outgoing edges represents the empty set $\emptyset$.

It is possible to write a procedure that translates in linear time a ground set term denoting a pure set into an acyclic apg. An example showing two apg's obtained in this way is shown in Figure 1. From the figure it is possible to observe the sets associated to the subgraph pointed at each node.

Given two graphs $G_1 = \langle N_1, E_1 \rangle$ and $G_2 = \langle N_2, E_2 \rangle$, a *bisimulation* between G_1 and G_2 is a relation $b \subseteq N_1 \times N_2$ such that [2]:

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

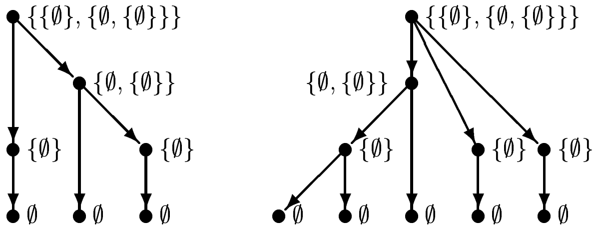


Figure 1: Two bisimilar apg's obtained from $\{\{\emptyset\},\{\emptyset,\{\emptyset\}\}\}$ and $\{\{\{\emptyset,\emptyset\},\emptyset\},\{\emptyset\},\{\emptyset\}\}$

3. $u_1 \, b \, u_2 \wedge \langle u_2, v_2 \rangle \in E_2 \Rightarrow \exists v_1 \in N_1 (v_1 \, b \, v_2 \wedge \langle u_1, v_1 \rangle \in E_1)$.

Two apg's $\langle G_1, \nu_1 \rangle$ and $\langle G_2, \nu_2 \rangle$ are bisimilar if and only if there exists a bisimulation b between G_1 and G_2 such that $\nu_1 \, b \, \nu_2$. It is easy to check that the two pointed graphs of Figure 1 are bisimilar. Observe that conditions (2) and (3) above resembles the extensionality axiom (Section 2.1). As a matter of fact, it is well-known that (pure) sets are equal if and only if their apg representations are bisimilar [2]. In [21] it is proved that bisimilarity between *acyclic* graphs can be tested in linear time. This result is based on an algorithm that guesses an initial partition of the nodes and refines it using a suitable computation strategy. In particular, at the beginning, all leaf nodes are put in the same class and this class is not further split.

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

3.2 SUD for gflat(q)

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

$$\begin{array}{rclrcl} V_1 & = & vars(s) \setminus vars(t) & C_1 & = & consts(s) \setminus consts(t) \\ V_2 & = & vars(t) \setminus vars(s) & C_2 & = & consts(t) \setminus consts(s) \\ V_3 & = & vars(s) \cap vars(t) & C_3 & = & consts(s) \cap consts(t) \end{array} \quad (1)$$

where $vars(r), consts(r)$ return the set of variables and individual terms occurring in the term r, respectively. Let us observe that these sets can be determined in time linear in $||s|| + ||t||$.

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

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 is trivially obtained by assigning the set $\{a_1, \ldots, a_n, b_1 \ldots, b_{n'}\}$ to all variables in s and t.

If exactly one between q' and q'' is 0, for instance $q' = 0$, the problem admits a solution if and only if $C_1 = \emptyset$ (if $q'' = 0$, then $C_2 = \emptyset$).

Thus, to solve the SUD problem for gflat(q) terms we need to compute only the sets C_i and V_i that can be done in time $O(||s|| + ||t||)$.

3.3 SUD for flat(q)

Let $s = \{X_1, \ldots, X_m, a_1, \ldots, a_n\} \cup Y_1 \cup \cdots \cup Y_{q'}$, and $t = \{W_1, \ldots, W_{m'}, b_1, \ldots, b_{n'}\} \cup Z_1 \cup \cdots \cup Z_{q''}$ ($q' \leq q$ and $q'' \leq q$). Let V_i, C_i as in formula (1).

Problem	$\mathsf{gflat}(q)$, $q > 0$	$\mathsf{flat}(q)$, $q = 0, 1$	$\mathsf{flat}(q)$, $q > 1$
SUD Complexity	$O(\|\|s\|\| + \|\|t\|\|)$	$O(\|\|s\|\| + \|\|t\|\|)$	$O(\|\|s\|\| + \|\|t\|\|)$
Theory	$ACI1$ with constants	$(Ab)(C\ell)$	gen. $ACI1$
Problem	ground	$\mathsf{nested}(q)$, $q = 0, 1$	$\mathsf{nested}(q)$, $q > 1$
SUD Complexity	$O(\|\|s\|\| + \|\|t\|\|)$	NP	NP
Theory	$(Ab)(C\ell)$	$(Ab)(C\ell)$	gen. $ACI1$

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

If $m = m' = 0$ we are in the case $\mathsf{gflat}(q)$ studied before. Otherwise, we need to analyze the different possibilities for q' and q''.

If q' and q'' are both greater than 0, then a trivial solution always exists, as in the $\mathsf{gflat}(q)$ case.

If $q' = q'' = 0$, it is immediate to see that a necessary condition for the existence of a solution is that:

$$|V_1| + |V_2| + |V_3| \geq |C_1| + |C_2|, |V_1| + |V_3| \geq |C_2|, |V_2| + |V_3| \geq |C_1|$$

This condition is also sufficient. If the condition holds, we can 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 element of C_1 and C_2 (if any) to the variables in V_3. The above condition guarantees that there are enough variables in V_3. If some variables have not been assigned by this algorithm, then it is easy to complete the valuation. E.g., when $|V_1| > |C_2|$, assign any value from C_2 or C_3 to the remaining variables of V_1.

If exactly one of q' and q'' is 0 (w.l.o.g., assume $q'' = 0$), let V_i, C_i computed as in the previous cases, but without considering the variables Y_i, Z_i. Then the problem admits a solution if and only if $|V_2| + |V_3| \geq |C_1|$.

Thus, the SUD problem for $\mathsf{flat}(q)$ set terms is reduced to the problem of computing sets V_i, C_i. This can be done in time $O(||s|| + ||t||)$. As we observe in Section 6.5 the class of problems $\mathsf{flat}(0)$ is studied in [27] where these set terms are called *Bound Simple set terms*.

3.4 SUD for $\mathsf{nested}(q)$

The set unification test for nested sets with non-ground elements has been proved to be NP-hard in [19], even for the simpler case of $q = 0$. 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 $\mathsf{nested}(0)$ set terms:

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

(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 the literature [52, 35].

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.

3.5 Equation vs. Systems

We conclude this section with a final remark. We have defined the SUD and SUS problem on a unique equation. The notions can be extended to deal with conjunctions (systems) of equations: we need to check if all the equations in the system are simultaneously unifiable. In the ground case nothing changes: each equation is analyzed independently.

For $\mathsf{gflat}(q)$ we know from [35, 29] 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 5 we prove the equivalence of this problem with the $\mathsf{gflat}(q)$ problem.

As far as $\mathsf{flat}(q)$ class is concerned, we can adapt the reduction done for $\mathsf{nested}(q)$. The instance of 3SAT is mapped to the system:

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

Thus, the $\mathsf{flat}(q)$ SUD problem for a single equation requires linear time, while for systems is NP-complete.

Regarding the $\mathsf{nested}(q)$ 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.

4 E-Unification

E-unification is concerned with solving term equations modulo an equational theory E. Set unification is a particular case of E-unification where the underlying equational theory contains the identities that capture the properties of set terms, i.e. the first-order terms used to represent sets (basically 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 must be considered to describe their basic properties.

We assume that the reader is familiar with the notions of equational theory, E-unification, E-unifier and related topics (see, e.g., [61, 11]). 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 Σ-terms (it is called the *term algebra*—function symbols of Σ are the operators). Given a sequence of terms $t_1, \ldots, t_n$, $vars(t_1, \ldots, t_n)$ denotes the set of variables occurring in some of the terms. $vars$ is naturally extended to equations and sets of equations.

A *substitution* $\sigma : \mathcal{V} \longrightarrow \mathcal{T}(\Sigma, \mathcal{V})$ is represented by its graph $[X_1/t_1, \ldots, X_n/t_n]$, 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 $T(\Sigma, \mathcal{V})$ 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 defined by a finite collection of identities E, written $s \approx t$, s, t terms belonging to $\mathcal{T}(\Sigma, \mathcal{V})$. $=_E$ is the *least congruence relation* on the term algebra $\mathcal{T}(\Sigma, \mathcal{V})$ which contains E and is closed under substitution [11]. 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:

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

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

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

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 finite set (or a *system*) of equations $\mathcal{E} = \{s_1 = t_1, \ldots, s_n = t_n\}$ between Σ-terms. A substitution μ which is a E-unifier of all the equations in $\mathcal{E}$ is said a E-unifier (or a *E-solution*) of $\mathcal{E}$. The set of all E-unifiers of $\mathcal{E}$ is denoted by $\mathcal{U}_E(\mathcal{E})$. In the rest of this paper we will equivalently consider systems of equations as *sets* of equations or as *conjunctions* of equations.

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 ordered with respect to the pre-order $\leq_E^{\mathcal{W}}$. Given two substitutions σ_1, σ_2:

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

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

It is well-known that, differently from the case of syntactic unification, unifiable E-unification problems may not have a most general unifier. Indeed, 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 E-unification problem $\mathcal{E}$ is a set $\mathcal{C}$ of E-unifiers (i.e., a subset of $\mathcal{U}_E(\mathcal{E})$) that satisfies the additional condition:

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

A complete set of E-unifiers $\mathcal{C}$ that also fulfills the following *minimality* condition:

- for any pair μ_1, μ_2 in $\mathcal{C}$, if $\mu_1 \leq_E \mu_2$, then $\mu_1 = \mu_2$

is said a *minimal complete set of E-unifiers*. When the minimal complete set of E-unifiers of $\mathcal{E}$ contains only one substitution σ we say that σ is the *most general E-unifier*.

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 $\mathcal{E}$ if X does not occur neither in t nor elsewhere in $\mathcal{E}$. In this case, X is said to be a *solved* variable in $\mathcal{E}$. A system $\mathcal{E}$ is said to be in solved form if, for all e in $\mathcal{E}$, e is in solved form with respect to $\mathcal{E}$. From the system in solved form $\{X_1 = t_1, \ldots, X_n = t_n\}$ it is immediate to obtain the most general E-unifier $[X_1/t_1, \ldots, X_n/t_n]$.

E-unification problems can be classified according to whether their signatures Σ contain 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 equational theory considered;
- *unification with constants*, where the terms to be unified are built by using symbols in the equational theory plus 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 3 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 problem of determining a *complete set* of E-unifiers of an equation $s = t$ or of a system of equations $\mathcal{E}$.

5 $ACI1$ with Constants

According to the classification presented in Section 2.2 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.

5.1 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 5.1 *An $ACI1$* 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 $ACI1$ 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 4. Thanks to the associativity property (A), $ACI1$ 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.

5.2 Which Kind of Set Unification

The $ACI1$ with constants language allows us to describe the set unification problem for $\mathsf{gflat}(q)$ set terms. Let $(\cdot)^*$ be the function that translates an $ACI1$ with constants term into a $\mathsf{gflat}(q)$ set term; $(\cdot)^*$ is defined as follows:

$$\begin{cases} X^* & \stackrel{\mathcal{D}ef}{=} & X \\ c^* & \stackrel{\mathcal{D}ef}{=} & \{c\} \\ (s \cup t)^* & \stackrel{\mathcal{D}ef}{=} & s^* \cup t^* \end{cases}$$

where a constant symbol c on the right is viewed as an element of the set $\mathcal{U}$ of the individuals. The function is extended to substitutions: $\sigma^*(X) = (\sigma(X))^*$. An $ACI1$ unifier μ of two terms s and t is *non-redundant* if for each $X \in \mathit{vars}(s) \cup \mathit{vars}(t)$ it holds that $\mathit{consts}(\mu(X)) \setminus (\mathit{consts}(s) \cup \mathit{consts}(t)) = \emptyset$. It follows from [9] that if there is an $ACI1$ unifier μ for two terms s and t, then there is also a non-redundant unifier σ for s and t such that $\sigma \leq \mu$. In [9] it is shown how to compute a complete set of unifiers which contains only non-redundant substitutions—see also Section 5.3.

Lemma 5.2 *σ is a non-redundant $ACI1$ unifier of $s = t$ if and only if σ^* is a solution of the SUS problem $s^* = t^*$.*

Proof. Let s and t be of the form

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

where C_i and V_i are determined according to formula (1). σ is a non-redundant 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 $\sigma(X) = a_i \cup \cdots$ and
- for each b_j in C_2 there is X in $V_2 \cup V_3$ such that $\sigma(X) = b_j \cup \cdots$ and
- each variable in $V_1 \cup V_2 \cup V_3$ is mapped on a union of constants in $C_1 \cup C_2 \cup C_3$ (non-redundant hypothesis).

This means that $\sigma^*(s^*)$ is a set that contains the same elements as $\sigma^*(t^*)$. The vice versa is similar. 5.2 □

Example 5.3 *The following are set terms and set unification problems which are allowed in* $ACI1$ *with constants.*

- $X_1 \cup X_2 \cup X_3 = X_4 \cup X_1$,
- $X_1 \cup a \cup b \cup X_2 = X_3 \cup c$—*which is equivalent, by applying* $(\cdot)^*$, *to* $\{a, b\} \cup X_1 \cup X_2 = \{c\} \cup X_3$.

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. Details of this result can be found in [20].

5.3 Unification Algorithm

A general algorithm capable of computing a minimal complete set of $ACI1$-unifiers for $ACI1$ with constants problems has been presented in [9]. This algorithm is described in the rest of this section.

Given two Σ-terms s and t we compute a complete set S of $ACI1$-unifiers of $\{s = t\}$. 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 the schema of boolean ACI-matrices in Figure 2. Let $V_1, V_2, V_3, C_1, C_2, C_3$ be as in formula (1) from Section 3.2. With $I_{n,n}$ we denote the $n \times n$ *identity* matrix ($I_{n,n}[x, x] = 1$, $I_{n,n}[x, y] = 0$ if $x \neq y$). With $0_{m,n}$ we denote the $m \times n$ *zero* matrix ($0_{m,n}[x, y] = 0$). With $Col^j_{m,n}$ we denote the $m \times n$ *column* matrix that has the j-th column set to 1 ($Col^j_{m,n}[x, j] = 1$, $Col^j_{m,n}[x, y] = 0$ if $x \neq j$). We omit the dimensions of the matrices when these are clear from the context.

Each ACI-matrix associated to an instance $\{s = t\}$ of the unification problem is composed of a *fixed part* and of a *variable (non-deterministic) part*. The *fixed part* includes the rows associated to the new sets of variables Z_i, W_j, U_k, Y, where the ranges for i, j, k depend on $|V_1|, |V_2|$, and $|V_3|$—see Figure 2. This part is built using a combination of matrices $I, 0$, and Col^j.

The *variable (non-deterministic) part* includes all the rows associated with the constants C_1, C_2, C_3. This part uses *zero* matrices and boolean vectors $\vec{v}_i, \vec{v}'_i, \vec{w}_i, \vec{w}'_i, \vec{u}_i, \vec{u}'_i$, whose values are non-deterministically determined according to the following rules:

- For $c \in C_1$ $(1 \leq i \leq |\ C_1\ |)$:
  - Pick arbitrarily $\vec{v}_i$ in $\{0, 1\}^{|V_2|}$;
  - if $\vec{v}_i \neq (0, \ldots, 0)$ then $\vec{v}'_i = (0, \ldots, 0)$;
  - if $\vec{v}_i = (0, \ldots, 0)$ and $V_3 \neq \emptyset$ then pick arbitrarily $\vec{v}'_i$ in $\{0, 1\}^{|V_3|}$ such that $\vec{v}'_i$ contains a single 1.
- For $c \in C_2$ $(1 \leq i \leq |\ C_2\ |)$:

  - Pick arbitrarily $\vec{w}_i$ in $\{0,1\}^{|V_1|}$;
  - if $\vec{w}_i \neq (0,\ldots,0)$ then $\vec{w}'_i = (0,\ldots,0)$;
  - if $\vec{w}_i = (0,\ldots,0)$ and $V_3 \neq \emptyset$ then pick arbitrarily $\vec{w}'_i$ in $\{0,1\}^{|V_3|}$ such that $\vec{w}'_i$ contains a single 1.

- For $c \in C_3$ $(1 \leq i \leq |\, C_3 \,|)$:
  1. Pick arbitrarily $\vec{u}_i$ in $\{0,1\}^{|V_1|}$;
  2. if $\vec{u}_i = (0,\ldots,0)$ then pick arbitrarily $\vec{u}'_i$ in $\{0,1\}^{|V_2|} \setminus \{(0,\ldots,0)\}$
  3. else $\vec{u}'_i = (0,\ldots,0)$.

Note that in the cases for C_1 (C_2) if $V_3 = \emptyset$ then we are not allowed to choose $\vec{v}_i = (0,\ldots,0)$ ($\vec{w}_i = (0,\ldots,0)$). These vectors are used to distribute the constants between the various partitions of sets created in the solution. Multiple choices for the values of the vectors $\vec{v}_i, \vec{v}'_i, \vec{w}_i, \vec{w}'_i, \vec{u}_i, \vec{u}'_i$, lead to different matrices; each of them represents a distinct solution to the unification problem.

$$
\begin{array}{|ccc|ll}
\multicolumn{1}{c}{V_1} & \multicolumn{1}{c}{V_2} & \multicolumn{1}{c}{V_3} & & \\
\hline
Col^1 & I & O & Z_1 & (|\, V_2 \,| \text{ new variables}) \\
\hline
Col^2 & I & O & Z_2 & (|\, V_2 \,| \text{ new variables}) \\
\hline
 & \vdots & & & \\
\hline
Col^{|V_1|} & I & O & Z_{|V_1|} & (|\, V_2 \,| \text{ new variables}) \\
\hline
Col^1 & O & I & W_1 & (|\, V_3 \,| \text{ new variables}) \\
\hline
Col^2 & O & I & W_2 & (|\, V_3 \,| \text{ new variables}) \\
\hline
 & \vdots & & & \\
\hline
Col^{|V_1|} & O & I & W_{|V_1|} & (|\, V_3 \,| \text{ new variables}) \\
\hline
O & Col^1 & I & U_1 & (|\, V_3 \,| \text{ new variables}) \\
\hline
O & Col^2 & I & U_2 & (|\, V_3 \,| \text{ new variables}) \\
\hline
 & \vdots & & \vdots & \\
\hline
O & Col^{|V_2|} & I & U_{|V_2|} & (|\, V_3 \,| \text{ new variables}) \\
\hline
O & O & I & Y & (|\, V_3 \,| \text{ new variables}) \\
\hline
 & \vec{v_1} & \vec{v'_1} & & \\
 & \vec{v_2} & \vec{v'_2} & & \\
O_{|C_1|,|V_1|} & \vdots & \vdots & C_1 & \\
 & \vec{v_{|C_1|}} & \vec{v'_{|C_1|}} & & \\
\hline
\vec{w_1} & & \vec{w'_1} & & \\
\vec{w_2} & & \vec{w'_2} & & \\
\vdots & O_{|C_2|,|V_2|} & \vdots & C_2 & \\
\vec{w_{|C_2|}} & & \vec{w'_{|C_2|}} & & \\
\hline
\vec{u_1} & \vec{u'_1} & & & \\
\vec{u_2} & \vec{u'_2} & & & \\
\vdots & \vdots & O_{|C_3|,|V_3|} & C_3 & \\
\vec{u_{|C_3|}} & \vec{u'_{|C_3|}} & & & \\
\hline
\end{array}
$$

Figure 2: Schema for *ACI*-matrices

Let M be an *ACI*-matrix; let n denote the number of rows in M and let r_i be the variable or constant labeling the row i. A substitution θ, such that $dom(\theta) = V_1 \cup V_2 \cup V_3 = \{X_1, \ldots, X_m\}$, is computed from the *ACI*-matrix M as follows:

$$\theta(X_j) = \bigcup_{i \in \{1,\ldots,n\} \wedge M[i,j]=1} r_i$$

where X_j is the variable labeling the column j. Observe that $\theta(X_j) = \emptyset$ if $(\forall i \in \{1, \ldots, n\})M[i, j] = 0$.

$$
\begin{array}{cc|cc|c|cc}
\multicolumn{2}{c}{\overbrace{\qquad\quad}^{V_1}} & \multicolumn{2}{c}{\overbrace{\qquad\quad}^{V_2}} & \multicolumn{1}{c}{\overbrace{\quad}^{V_3}} & & \\
S_1 & S_2 & T_1 & T_2 & X & & \\
\hline
1 & 0 & 1 & 0 & 0 & R_1 & \multirow{2}{*}{$\}\ Z_1$} \\
1 & 0 & 0 & 1 & 0 & R_2 & \\
\hline
0 & 1 & 1 & 0 & 0 & R_3 & \multirow{2}{*}{$\}\ Z_2$} \\
0 & 1 & 0 & 1 & 0 & R_4 & \\
\hline
1 & 0 & 0 & 0 & 1 & R_5 & \}\ W_1 \\
0 & 1 & 0 & 0 & 1 & R_6 & \}\ W_2 \\
\hline
0 & 0 & 1 & 0 & 1 & R_7 & \multirow{2}{*}{$\}\ U_1$} \\
0 & 0 & 0 & 1 & 1 & R_8 & \\
\hline
0 & 0 & 0 & 0 & 1 & R_9 & \}\ Y \\
\hline
\end{array}
$$

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

Example 5.4 *Consider the instance:* $S_1 \cup S_2 \cup X = T_1 \cup T_2 \cup X$. *Here,* $V_1 = \{S_1, S_2\}, V_2 = \{T_1, T_2\}, V_3 = \{X\}$, *and* $C_1 = C_2 = C_3 = \emptyset$. *The ACI1-matrix is reported in Fig. 3. In absence of constants, the matrix consists of its fixed part. Thus, it represents an mgu:*

$$[S_1/R_1 \cup R_2 \cup R_5, S_2/R_3 \cup R_4 \cup R_6, T_1/R_1 \cup R_3 \cup R_7, T_2/R_2 \cup R_4 \cup R_8, X/R_5 \cup R_6 \cup R_7 \cup R_8 \cup R_9]$$

The two 1 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,* $R_1 = S_1 \cap T_1$*).*

Example 5.5 *Consider the instance:* $X_1 \cup X_2 \cup X_3 = a \cup b$ *where* $V_1 = \{X_1, X_2, X_3\}, C_2 = \{a, b\}$, $V_2 = V_3 = C_1 = C_3 = \emptyset$. *We can compute 49 ACI-matrices. Two of them are:*

$$
\begin{array}{ccc|cc}
\multicolumn{3}{c}{\overbrace{\qquad\qquad}^{V_1}} & & \\
X_1 & X_2 & X_3 & & \\
\hline
0 & 1 & 1 & a & \multirow{2}{*}{$\}\ C_2$} \\
\hline
1 & 0 & 1 & b & \\
\hline
\end{array}
\qquad\qquad
\begin{array}{ccc|cc}
\multicolumn{3}{c}{\overbrace{\qquad\qquad}^{V_1}} & & \\
X_1 & X_2 & X_3 & & \\
\hline
0 & 0 & 1 & a & \multirow{2}{*}{$\}\ C_2$} \\
\hline
0 & 1 & 0 & b & \\
\hline
\end{array}
$$

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

5.4 Results for the $ACI1$ with Constants Unification Algorithm

Soundness and completeness of this unification procedure has been proved in [9]. Proving the termination of this algorithm is trivial. Each matrix can be computed in polynomial time. Any solution is immediately computed from the matrix as indicated above. The number of matrices 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 $O(2^{(||s||+||t||)^2})$ [9].

The detection of a solution of a unification problem clearly implies solving the related decision problem. Thus, generally speaking, the complexity of finding a solution can be no better than the complexity of solving the corresponding decision problem. Specifically, the complexity of finding an arbitrary solution for the unification problem presented in this section can be done in linear time. This can be achieved as follows. The first step is to verify that the decision problem $s = t$ has a positive answer; this can be done in linear time thanks to the results in Lemma 5.2 and Section 3.2. 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 the $s = t$.

5.5 Discussion

A simpler unification problem—called the *elementary* $ACI1$ unification problem—has been considered in the literature. This problem involves terms which are constructed using only variables, the constant $\emptyset$, and the binary constructor $\cup$ (i.e, a subcase of $\mathsf{gflat}(q)$ with $n = 0$ and $q \geq 0$). 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, complexity of finding an arbitrary solution is $O(1)$. Furthermore, each elementary $ACI1$ unification problem admits a single most general unifier—since the ACI-matrix contains only the fixed part. In Appendix B we show a variant for the ACI-matrix for this simplified problem.

As a final remark, in [29, 35] it is shown how the result presented in this section can be extended to provide a polynomial time solution to systems of $ACI1$ with constants unification problems.

6 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 Lemma 5.2). 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 $q \leq 1$. As already observed at the end of Section 2.3, in this case we can rely on the element insertion operator $\{\cdot\,|\,\cdot\}$ 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 (concrete) logic and functional-logic programming languages (e.g., $CLP(\mathcal{SET})$ [25, 22], SEL [32], SETA [6]).

The algorithm we propose here is similar to the one presented in [19]—but with considerably simpler termination proof. The underlying equational theory comprises the two identities (Ab) and $(C\ell)$ shown in Section 4, stating the fundamental properties of the set constructor $\{\cdot\,|\,\cdot\}$.

6.1 Language and Semantics

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

Definition 6.1 *An* $(Ab)(C\ell)$ set term *is either a variable, or the constant* $\emptyset$*, or a* Σ*-term of the form* $\{t\,|\,s\}$ *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* $f \not\equiv \{\cdot\,|\,\cdot\}$ *and* $t_1, \ldots, t_n$ *are* Σ*-terms.*

The function symbol $\{\cdot\,|\,\cdot\}$ has the properties described by the identities (Ab) and $(C\ell)$ introduced in Section 4. Hence, set terms denote hereditarily finite sets based on $\mathcal{U}$, while individual terms denote individuals, i.e., elements of the universe $\mathcal{U}$. As a notational convenience $\{\, s_1 \,|\, \{\, s_2 \,|\, \cdots \{\, s_n \,|\, t\,\} \cdots\}\}$ will be written as $\{s_1, \ldots, s_n \,|\, t\}$ or simply by $\{s_1, \ldots, s_n\}$ when t is $\emptyset$.

6.2 Which Kind of Set Unification

The $(Ab)(C\ell)$ language allows us to describe the SUD and SUS problems for $\mathsf{nested}(1)$ set terms—hence, also for all classes of set terms that are included in $\mathsf{nested}(1)$, namely ground, $\mathsf{gflat}(1)$, and $\mathsf{flat}(1)$. That is, it allows the representation of arbitrary nested sets with at most a single set variable per set term. As already noted set terms that allow more than one set variable to occur as arguments of the union operator (i.e., abstract set terms with $q > 1$) are not expressible in this language.

Example 6.2 *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* 2.2—$\{\{X_1, a\}\} \cup Y_1 = \{X_3\} \cup Y_2$*)*

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

6.3 Unification Algorithm

The algorithm consists of three parts: the first part (`AbCl_unify`—see Figure 4) chooses one equation at a time using a partially deterministic strategy, while the second (`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 but dealt with as a stack. Given a system of equations $\mathcal{E}_{\mathtt{in}}$, the algorithm non-deterministically returns either `fail` or a collection of systems in solved form.

```
AbCl_unify(ℰ_in) :
   let ℰ = ⟨ℰ_s , ℰ_ns , ℰ_aux⟩;
   ℰ_s := ∅; ℰ_ns := ℰ_in; 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)
```

Figure 4: General $(Ab)(C\ell)$ Unification Algorithm (main)

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

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

The core of the unification algorithm (Figure 5) is very similar in structure to the traditional unification algorithms for standard Herbrand terms (e.g., [48]). The main difference is represented by the presence of action (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 [23]:

$$(E^s_k) \quad \forall Y_1 Y_2 V_1 V_2 \quad \left(\begin{array}{c} \{Y_1 \mid V_1\} = \{Y_2 \mid V_2\} \ \leftrightarrow \\ (Y_1 = Y_2 \wedge V_1 = V_2) \vee \\ (Y_1 = Y_2 \wedge V_1 = \{Y_2 \mid V_2\}) \vee \\ (Y_1 = Y_2 \wedge \{Y_1 \mid V_1\} = V_2) \vee \\ \exists K\, (V_1 = \{Y_2 \mid K\} \wedge V_2 = \{Y_1 \mid K\}) \end{array} \right)$$

(E^s_k) is a "syntactic version" of the extensionality axiom and it considers also equations of the form

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

where the two sides are set terms with the same variable as tail element. Unfortunately, a blind application of the rewriting rule naively obtained from (E^s_k) would lead to non-termination in this

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

(1)	$X = X$	$\mapsto$	$\mathcal{E}_{\mathtt{ns}} := \mathcal{E}_{\mathtt{ns}}$
(2)	$t = X$ t is not a variable	$\mapsto$	$\mathcal{E}_{\mathtt{ns}} := \mathcal{E}_{\mathtt{ns}} \wedge (X = t)$
(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	$\mapsto$	$\mathcal{E}_{\mathtt{s}} := \mathcal{E}_{\mathtt{s}}[X/t] \wedge (X = t)$; $\mathcal{E}_{\mathtt{ns}} := \mathcal{E}_{\mathtt{ns}}[X/t]; \mathcal{E}_{\mathtt{aux}} := \mathcal{E}_{\mathtt{aux}}[X/t]$
(6)	$f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$ $f \not\equiv g$	$\mapsto$	`fail`
(7)	$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ $f \not\equiv \{\cdot \mid \cdot\}$	$\mapsto$	$\mathcal{E}_{\mathtt{ns}} := \mathcal{E}_{\mathtt{ns}} \wedge (s_1 = t_1 \wedge \ldots \wedge s_n = t_n)$
(8)	$\{t \mid s\} = \{t' \mid s'\}$	$\mapsto$	`AbCl_step`$(\mathcal{E}, \{t \mid s\} = \{t' \mid s'\})$

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

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

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

Figure 5: General $(Ab)(C\ell)$ Unification Rewriting Rules

situation. This is the reason why it has been isolated and dealt with as special case in the algorithm. A call to `AbCl_step` introduces equations in the stack $\mathcal{E}_{\mathtt{aux}}$ 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`.

Lemma 6.3 *The global effect of a call to* `AbCl_step` *on* $e \equiv \{t_1, \ldots, t_m \,|\, h\} = \{t'_1, \ldots, t'_n \,|\, k\}$*, where* h, k *are either variables or terms with outermost symbol* $f \neq \{\cdot \,|\, \cdot\}$*, is to replace* e *by a conjunction of equations* $t_i = t'_j$ *between elements of the two sets (added directly to the system* $\mathcal{E}_{\mathtt{ns}}$*) plus:*

- *if* h *and* k *are the same variable, a set of membership equations of the form* $h = \{\cdots \,|\, h\}$*.*
- *otherwise, the equation(s) obtained from one of the following cases:*
  1. *an equation* $h = \{t'_{j_1}, \ldots, t'_{j_r} \,|\, k\}$*,* $r \leq n$ *or,*
  2. *an equation* $k = \{t_{i_1}, \ldots, t_{i_\ell} \,|\, h\}$*,* $\ell \leq m$ *or,*
  3. *the equations* $h = \{t'_{j_1}, \ldots, t'_{j_r} \,|\, N\}, k = \{t_{i_1}, \ldots, t_{i_\ell} \,|\, N\}$*,* $\ell \leq m, r \leq n$*,* N *a new variable, the same for the two equations.*
- *A number of (passive) solved form equations* $N_i = s$ *for temporarily introduced variables* N_i*.*

Observe that all solved form equations generated above are immediately transformed into substitutions and immediately applied. Note also that the substitutions obtained from the $N_i = s$ *will not affect the rest of the system.*

Proof. By induction on $\langle m, n \rangle$ (see Appendix A). 6.3 □

Moreover, equations of the form $X = \{t_0, \ldots, t_n \,|\, X\}$ (membership equations) 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 $(C\ell)$. These equations are processed at the end 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 (actions (3) and (4)).

Example 6.4 *Consider the unification problem*

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

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

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

In general, the algorithm `AbCl_unify` may open a large—though finite—number of alternatives, possibly leading to redundant solutions. In [5, 64] it is shown how to improve the algorithm to minimize the number of redundant unifiers.

6.4 Results for the General $(Ab)(C\ell)$ Unification Algorithm

Theorem 6.5 (correctness and completeness) *Given a system $\mathcal{E}$,* `AbCl_unify`$(\mathcal{E})$ *computes a complete set of $(Ab)(C\ell)$ unifiers for $\mathcal{E}$.*

Proof. The result follows from the algorithms presented in [19], from which the algorithm is derived. We only need to consider the missing case of membership equations, dealt with by `AbCl_unify_final`. Correctness and completeness of this procedure is a consequence of the fact that the unique information contained in the equations

$$X \;=\; \{t_0^0,\dots,t_{n_0}^0 \,|\, X\},\quad \dots,\quad X \;=\; \{t_0^k,\dots,t_{n_k}^k \,|\, X\}$$

is that $t_j^i \in X$, for $i = 0,\dots,k$, $j = 0,\dots,n_i$. The same information is encoded in the equation:

$$X = \{t_0^0,\dots,t_{n_0}^0,\dots,t_0^k,\dots,t_{n_k}^k \,|\, X\}$$

6.5 □

Theorem 6.6 (termination) *For any Herbrand system $\mathcal{E}$, and for any possible sequence of non-deterministic choices,* `AbCl_unify(`$\mathcal{E}$`)` *terminates.*

Proof. We first show the termination of the algorithm, except for the `AbCl_unify_final` part.

We associate a complexity measure to a system and we show that the effect of the algorithm causes the complexity to decrease. Termination will derive from the fact that the ordering between complexity measures is well-founded.

The complexity measure associated to a system $\mathcal{E}$ is the pair $\langle A, B\rangle$ where:

- A is the number of non-solved variables in $\mathcal{E}$
- B is the multiset of integer numbers $\{\![size(\ell) : \ell = r \ in\ \mathcal{E}_{\mathtt{ns}}]\!\}$, where $size$ is the function returning the number of occurrences of constant and function symbols in a term. A well-founded ordering between such multisets of integer numbers is the reflexive and transitive closure of the following rule:

$$\{\![a_1,\dots,a_{i-1},b_1,\dots,b_m,a_{i+1},\dots,a_n]\!\} \prec \{\![a_1,\dots,a_i,\dots,a_n]\!\} \quad \text{if} \quad m \geq 0 \wedge b_1 < a_i \wedge \cdots \wedge b_m < a_i$$

The ordering between two complexity pairs is the lexicographic ordering in which $<$ is used for the integer numbers of the first argument and $\prec$ for the second.

By case analysis, we observe that:

- actions $(3), (4)$ and (6) cause immediate termination;
- actions (1), (2), and (7) cannot increase A, while B always decreases;
- action (5) decreases A;
- action (8) is slightly more complicated to analyze, since it calls the auxiliary function `AbCl_step`. Let us consider the global effect of this function. Given the equation $\{t_1,\dots,t_m \,|\, h\} = \{t'_1,\dots,t'_n \,|\, k\}$, four cases must be distinguished and studied according to the results from Lemma 6.3:
  1. h and k are both terms with outermost symbol $f \not\equiv \{\cdot\,|\,\cdot\}$: 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 the tails, leading to a smaller B.
  2. exactly one of h and k is a variable. Assume h is a variable: a substitution for it is computed and applied: A decreases. The situation when k is a variable is perfectly symmetrical.

3. h and k are different variables. One of the following cases happens:
   (a) a substitution $h = \{\cdots \mid k\}$ is computed,
   (b) a substitution $k = \{\cdots \mid h\}$ is computed,
   (c) a substitution $h = \{\cdots \mid N\}$ and a substitution $k = \{\cdots \mid N\}$ (N a new variable, the same for the two equations) are computed.

   In all the three cases the application of the substitution causes A to decrease.
4. h and k are the same variable, say X. In this case membership equations $X = \{\cdots \mid X\}$ are added to $\mathcal{E}$ together with a certain number of equations between elements of the sets. Anyway, all these equations lead to a decrease in the B component of the measure. In particular, one should note that the membership equations provide a contribution of 0 in the B component of the measure, replacing a non-zero value present before this process.

The termination of the `AbCl_unify_final` part is straightforward. For each variable X occurring in an equation $X = \{\cdots \mid X\}$ we perform at most one rewriting and an application of substitution. Equations in solved form remain in solved form and do not activate any new actions. 6.6 □

6.5 Discussion

The $(Ab)(C\ell)$-unification algorithm we present here is derived from those presented in [19, 22], but it is simpler to understand and it offers a simpler termination proof. With respect to the algorithm in [19], it makes explicit some deterministic choices that turn out to be crucial for the termination proof. Moreover, with respect to both [19, 22] it provides a separate treatment of membership equations. Basically, in the new algorithm presented in this paper we avoid the application of the (anyway correct) 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 [19, 22]. This change, together with further small changes, allows the number of variables in the system to be kept under control.

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 in [19]. To be precise, NP is a class of decision problems. Here we mean that there exists an algorithm on a non-deterministic machine that can also find the answer (the correct class is FNP). In [52], a methodology to guess a solution of a conjunction of literals where $\mathcal{F} = \{\emptyset, \{\cdot \mid \cdot\}\}$ and $\Pi_c = \{=, \in, \cup, \cap, \setminus\}$ is proposed. Any *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 constraint can be done in polynomial time. In [52], it is also shown how this technique can be extended to the hybrid problem—the one we deal with in this paper. A non-deterministic algorithm based on a "guess-and-verify" technique is also proposed in [34].

The algorithm presented here, as well as those in [19, 5], have the common drawback that, due to the explicit application of substitutions during the solving process (e.g., action (5) of the algorithm), they have a computational complexity which falls outside of the FNP class. Nevertheless, it is possible to encode this algorithm using well-known techniques [48, 53]—such as multi-equations or graphs with structure sharing—that allow us to maintain a polynomial time complexity along each non-deterministic branch of the computation. For instance, in [4] 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 [18]. 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 a call to `AbCl_step`. This number is no greater than the size of the minimal complete set for the problem:

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

This value, as proved in [5], has the rough upper bound $O(2^{n \lg n})$. Since this process can be repeated once for each nesting, a rough upper bound to the number of solution 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 [44, 1, 51, 42]. Typical restrictions which have been considered are flat and completely specified, i.e., elements either of $\mathsf{gflat}(q)$ or $\mathsf{flat}(0)$. 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 [27], 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 3.

Let us illustrate the results in the simple case of matching [7] (the approach has been generalized to sequential unification in [27] and to parallel unification in [42]). 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 3.2), $b_i \in C_2$, $c_i \in C_3$, and $X_i \in V_1$ ($C_1 = \emptyset$ otherwise the problem has no solutions). The two terms unify iff $h \geq k$ (cf. Section 3.3). From [7] 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 seen as all h-tuples containing elements from $\{b_1, \ldots, b_k, c_1, \ldots, c_r\}$, which contain all the elements of the set $\{b_1, \ldots, b_k\}$. This can be achieved by

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

An algorithm capable of producing these substitutions is shown in Appendix C of [24]. This algorithm 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.

7 General $ACI1$ Unification

The unification problem considered in Section 5 is capable of dealing with flat set terms containing an arbitrary number of set variables. On the other hand, the unification problem of Section 6 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 of unification problems which do not fall in any of the two above categories—i.e., 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 5 and 6 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.

7.1 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 function symbols with arbitrary arities.

Definition 7.1 *An $ACI1$* set term *is either a variable, 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 $t_1, \ldots, t_n$ are Σ-terms.*

The function symbols $\cup$ and $\emptyset$ have the properties described by the identities (A), (C), (I) and (1) introduced in Section 4. Hence, set terms denote hereditarily finite sets based on $\mathcal{U}$, while individual terms denote individuals, i.e., elements of the universe $\mathcal{U}$.

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, then we are in the case discussed in Section 5. As a notational convenience, we will denote with $\{s\}$ the term $f(s, \emptyset, \ldots, \emptyset)$—intuitively, we use this term to encode singleton sets. Moreover, we will denote the term $\{s_1\} \cup \cdots \cup \{s_n\}$ with $\{s_1, \ldots, s_n\}$.

7.2 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 $q \geq 2$ are handled in this framework (and not in any of the previous ones).

Example 7.2 *The following are set terms and set unification problems which are allowed in general $ACI1$:*

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

7.3 Unification Algorithm

In this section we propose a novel algorithm to directly solve the general $ACI1$ unification problem. This unification algorithm can be seen as a *set* unification algorithm. The algorithm is composed of a main procedure `general_aci`, shown in Figure 6, and of a set-equation rewriting function, named `aci_step` (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, $\mathcal{E}_{\mathtt{s}}$ and $\mathcal{E}_{\mathtt{ns}}$: 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 actions dealing with set terms (i.e., terms containing occurrences of $\cup$ at the outermost level):

- action (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 are not in the Herbrand case;
- action (8) which is used to solve equalities 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 $s_1 = t_1, \ldots, s_n = t_n$.

The `aci_step` function receives as input the set equation to be solved and non-deterministically produces as result two systems of equations (corresponding to the $\mathcal{E}_{\mathtt{s}}$ and $\mathcal{E}_{\mathtt{ns}}$ of the main unification procedure) and a substitution for the variables instantiated during the resolution of the equation. The

general_aci($\mathcal{E}$) :
  $\mathcal{E}_{\mathtt{s}} := \emptyset; \mathcal{E}_{\mathtt{ns}} := \mathcal{E}$ (i.e., the initial system of equations);
  while $\mathcal{E}_{\mathtt{ns}} \neq \emptyset$ do
    select arbitrarily an equation e from $\mathcal{E}_{\mathtt{ns}}$ and remove it;
    case e of

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

aci_step(e):
  $\mathcal{E}^n :=$ Normalization(e);
  $\langle \mathcal{E}_1^{ACI}, \mathcal{E}_2^{ACI} \rangle :=$ Elementary_$ACI1$_Solution($\mathcal{E}^n$) ;
  $\langle \mathcal{E}_1, \mathcal{E}_2 \rangle :=$ Term_Propagation($\mathcal{E}_1^{ACI}, \mathcal{E}_2^{ACI}$) ;
  $\rho :=$ Variables_Removal($\mathcal{E}_1, \mathcal{E}_2$) ;
  return $\langle \mathcal{E}_1, \mathcal{E}_2\rho, \rho \rangle$

Figure 6: General $ACI1$ Unification Procedure and the function aci_step

function `aci_step` performs its task in four successive steps, as shown in Figure 6. Term Propagation is the only 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

$$f_1(\bar{l}_1) \cup \cdots \cup f_{k_1}(\bar{l}_{k_1}) \cup L_1 \cup \cdots \cup L_{k_2} = g_1(\bar{r}_1) \cup \cdots \cup g_{h_1}(\bar{r}_{h_1}) \cup R_1 \cup \cdots \cup R_{h_2} \tag{2}$$

where L_i, R_j $(0 \leq i \leq k_2,\ 0 \leq j \leq h_2)$ are variables and f_i, g_j $(0 \leq i \leq k_1,\ 0 \leq j \leq h_1)$ are function symbols different from $\cup$;
output: A system

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

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

This step, following the idea used in [43, 10], performs a normalization of the problem $\mathcal{E}$ into the problem $\mathcal{E}^n$—producing a pure set equation, that contains only variables.

Elementary $ACI1$ Solution:

input: The system $\mathcal{E}^n$ produced by the Normalization phase;
output: A pair of systems $\mathcal{E}_1^{ACI}$ and $\mathcal{E}_2^{ACI}$ obtained by solving the elementary $ACI1$ unification problem

$$N_1^L \cup \cdots \cup N_{k_1}^L \cup L_1 \cup \cdots \cup L_{k_2} = N_1^R \cup \cdots \cup N_{h_1}^R \cup R_1 \cup \cdots \cup R_{h_2} \tag{3}$$

of $\mathcal{E}^n$. This problem can be directly solved by using the techniques seen in Section 5. The result of the computation is a collection of equations of the form $V = \ldots$ for all the variables occurring in the two terms. The solved form equations associated to L_j and R_i form the set $\mathcal{E}_1^{ACI}$. $\mathcal{E}_2^{ACI}$ is composed of the equations concerning the variables N_j^L and N_i^R; these variables are immediately replaced with the terms they have been set equal to during the Normalization phase.

Observe that this step is completely deterministic.

Term Propagation:

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

The equations in $\mathcal{E}_2^{ACI}$ can be simplified using the semantics properties of $\emptyset$ and $\cup$. As a matter of fact, the equations in $\mathcal{E}_2^{ACI}$ 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 $\mathcal{E}_2^{ACI}$.

More precisely, a substitution λ describing the solution of the equations in $\mathcal{E}_2^{ACI}$ can be build as follows. Let us assume that an ordering has been fixed on the equations in $\mathcal{E}_2^{ACI}$ and on the variables $A_{i,j}$. Thus, for each $A_{i,j}$ occurring in $\mathcal{E}_2^{ACI}$ 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 $\mathcal{E}_2^{ACI}$. Once all the $A_{i,j}$ occurring in $\mathcal{E}_2^{ACI}$ 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 $h \geq 1$ (without loss of generality, we may assume that all 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 $\mathcal{E}_2$ consists of all equations of the form:

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

for each equation in $\mathcal{E}_2^{ACI}$. An algorithm to perform the Elementary $ACI1$ Solution step and the Term Propagation step is presented in Appendix B.

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

$$\begin{array}{rcl} \mathcal{E}_1 & = & \bigwedge_{1 \leq j \leq k_2} L_j = \bigcup_{i=1}^{h_1} \lambda(A_{i,k_1+j}) \cup \bigcup_{i=h_1+1}^{h_1+h_2} A_{i,k_1+j} \wedge \\ & & \bigwedge_{1 \leq i \leq h_2} R_i = \bigcup_{j=1}^{k_1} \lambda(A_{h_1+i,j}) \cup \bigcup_{j=k_1+1}^{k_1+k_2} A_{h_1+i,j} \end{array}$$

Variables Removal:

input: The pair $\mathcal{E}_1$ and $\mathcal{E}_2$ computed in the previous step;
output: The substitution ρ.

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

Let us define the following relation:

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

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

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

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

If $\mathcal{E}_1$ is solvable, 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 the removal of all the variables $\{L_1, \ldots, L_{k_1}, R_1, \ldots, R_{h_1}\}$.

Example 7.3 *Consider the unification problem:*

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

The Normalization phase leads to the system

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

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 $ACI1$ Solution phase), leading to:

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

A possible substitution λ produced by the Term Propagation phase 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

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

and the substitution $\rho = [X/A_{2,3} \cup A_{3,3},\ Y/\{\{a\}\} \cup \{b\} \cup A_{2,3},\ Z/A_{3,3}]$.

7.4 Results for the General $ACI1$ Unification Algorithm

7.4.1 Soundness and Completeness

Lemma 7.4 *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 $\langle \mathcal{E}_1^i, \mathcal{E}_2^i, \rho^i \rangle$, 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 $\mathcal{E}_1^i \cup \mathcal{E}_2^i$ then σ is a unifier of e and $\sigma \leq \rho^i$*
- *if σ is a unifier of e then there exists $1 \leq i \leq k$ and a substitution γ such that $\sigma \cup \gamma$ is a unifier of $\mathcal{E}_1^i \cup \mathcal{E}_2^i$.*

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

- For the *Normalization* phase, it is trivial to show that σ is a unifier for e if and only if $\sigma \cup \gamma$ is a unifier for $\mathcal{E}^n$, 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 $[N_i^L/f_i(\bar{l}_i)\sigma \mid 1 \leq i \leq k_1] \cup [N_j^R/g_j(\bar{r}_j)\sigma \mid 1 \leq j \leq h_1]$.
- For the *Elementary ACI* phase the result follows from the results for elementary $ACI1$ unification [9]. In this case we have that σ is a unifier of $\mathcal{E}^n$ if and only if $\sigma \cup \gamma$ is a unifier for $\mathcal{E}^{ACI}$, where $dom(\gamma) \subseteq \{A_{i,j} \mid 1 \leq i \leq h_1 + h_2 \wedge 1 \leq j \leq k_1 + k_2\}$.
- Let us consider the *Term Propagation* phase. We prove that $\sigma' = \sigma \cup \gamma$ is a unifier for $\mathcal{E}^{ACI}$ (where $dom(\sigma) \cap dom(\gamma) = \emptyset$ and $dom(\gamma) = \{A_{i,j} \mid 1 \leq i \leq h_1 + h_2 \wedge 1 \leq j \leq k_1 + k_2\}$) iff $\sigma\gamma'$ is a unifier for $\mathcal{E}_1 \cup \mathcal{E}_2$ (where γ' is the restriction of γ to $\{A_{i,j} \mid h_1 + 1 \leq i \leq h_1 + h_2 \wedge k_1 + 1 \leq j \leq k_1 + k_2\}$).
  Let $\sigma' = \sigma \cup \gamma$ be a unifier for $\mathcal{E}^{ACI}$ and let us consider the equations e in $\mathcal{E}_2^{ACI}$ in the same arbitrary order used to build $\mathcal{E}_2$. Such equations have the form $f(\bar{s}) = \bigcup A_i \cup \bigcup B_j$. If σ' is a unifier for $\mathcal{E}^{ACI}$, 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 $\mathcal{E}_1 \cup \mathcal{E}_2$ (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 $\mathcal{E}_2$. It is straightforward to observe that σ' is a unifier for $\mathcal{E}_2$. Observe also that $\sigma' \leq \lambda$, i.e., $\sigma' = \lambda \circ \theta$. This last fact, together with the fact that σ is a unifier for $\mathcal{E}_1^{ACI}$, is sufficient to conclude that σ' is a unifier for $\mathcal{E}_1$.

Vice versa, let σ' be a unifier for a certain $\mathcal{E}_1 \cup \mathcal{E}_2$ produced by the algorithm. Since the construction was possible, then there is a substitution λ which has been used to convert $\mathcal{E}_2^{ACI}$ into $\mathcal{E}_2$. If σ' is a solution of the equations $\bar{s} = \bar{s}_1, \ldots, \bar{s} = \bar{s}_h$ present in $\mathcal{E}_2$, then clearly σ' is also a unifier for the equation $f(\bar{s}) = (\bigcup A_i \cup \bigcup B_j)\lambda$ which produced such elements of $\mathcal{E}_2$. Thus, $\sigma' \cup [A/A\lambda \circ \sigma' \mid A \in dom(\lambda)]$ is a unifier for $\mathcal{E}_2^{ACI}$. The result for $\mathcal{E}_1^{ACI}$ is obvious.

- Correctness for the *Variables Removal* phase follows from the fact that we are not interested in solutions over infinite terms. 7.4 □

Theorem 7.5 (correctness and completeness) *The unification procedure* `general_aci` *is correct and complete with respect to the general ACI1 theory.*

Proof. Immediate from the above Lemma 7.4 concerning the auxiliary function `aci_step`, and from the classical results regarding Herbrand unification for the remaining actions. 7.5 □

7.4.2 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 [10]. Fages proposed [26] a termination proof for general *AC* unification. The complexity measure developed by Fages to prove termination 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 admits solutions in *ACI*1.

The key point in the termination proof is the fact that `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 formally prove termination, we introduce the following notions. First of all, we define an *iteration* to be a single application of an action of the procedure `general_aci`.

Definition 7.6 *Let $\mathcal{E}$ be a set of equations, and let us consider a function lev* : $vars(\mathcal{E}) \longrightarrow \mathbb{N}$. *This function can be extended over elements of* $T(\Sigma, \mathcal{V})$ *as follows:*

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

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

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

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* $\mathcal{E}$.

Lemma 7.7 *Given a system of equations $\mathcal{E}$, and given p the number of occurrences of elements of Σ in $\mathcal{E}$, exactly one of the following conditions holds:*

- *there exists a complete p-level for $\mathcal{E}$*

- *for any natural number q, there are no complete q-levels for $\mathcal{E}$.*

Proof. See Appendix A. 7.7 □

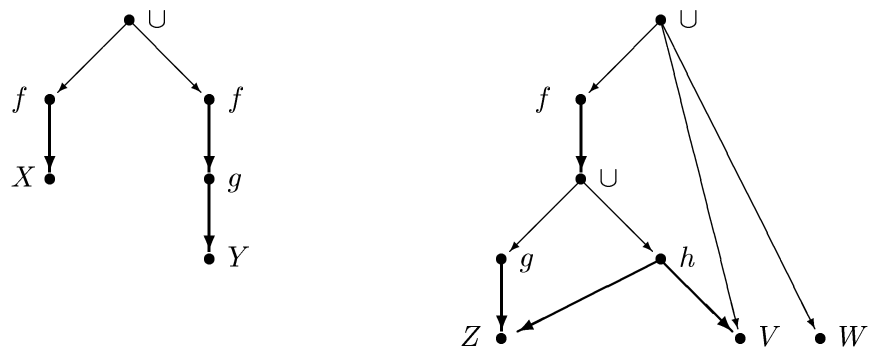
The notion of p-level has a direct interpretation on a graph-encoding of the system of equations. The unification algorithm itself can be mapped on a collection of graph manipulation operations. The mapping of the unification algorithm on graphs allows us to intuitively demonstrate that for each intermediate 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 $\mathcal{E}_0$ we define the directed labeled graph G_0 as follows:

- G_0 contains a node for each occurrence of a function symbol in $\mathcal{E}_0$; without loss of generality we assume to replace any occurrence of a constant c 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 $\mathcal{E}_0$
- for each term $f(t_1, \ldots, t_n)$ (f different from $\cup$) in $\mathcal{E}_0$, if μ is the node created for the specific occurrence of f, and ν_i is the node created for the main functor of t_i (or for the variable t_i), then the edge (μ, ν_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 $\mathcal{E}_0$; or
  - there exists a term $f(t_1, \ldots, t_n)$ in $\mathcal{E}_0$ 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., $t_1 \boxed{\cup} \cdots \cup t_n$) and let ν_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, ν_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 7.8 *Let $\mathcal{E}_0$ 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):*



Each action of the unification algorithm can be mapped onto an operation on the graph. If $\mathcal{E}_i$ 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 actions are the following:

1. if $\mathcal{E}_{i+1}$ is obtained by removing an equation $X = X$ from $\mathcal{E}_i$, then $G_{i+1} = G_i$
2. if $\mathcal{E}_{i+1}$ is obtained by replacing $t = X$ with $X = t$ in $\mathcal{E}_i$, then $G_{i+1} = G_i$

4. if $\mathcal{E}_{i+1}$ is obtained by replacing each occurrence of X with t in $\mathcal{E}_i$, then G_{i+1} is obtained by adding the edge (μ, ν) with label 0, where μ is the node associated with the variable X and ν is the node created for the main functor of term t (or for the variable t)

5. if $\mathcal{E}_{i+1}$ 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 ν 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, ν)

7. if $\mathcal{E}_{i+1}$ is obtained by replacing the equation $f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)$ in $\mathcal{E}_i$, then $G_{i+1} = G_i$

8. let us assume that $\mathcal{E}_{i+1}$ 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 $\mathcal{E}_i$ 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 ν_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, ν_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 7.9 *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 cycle including 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.

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. 7.9 □

Lemma 7.10 *Assume there is a non-failing sequence of k non-deterministic choices such that* `general_aci`$(\mathcal{E})$ *generates (one per each successive iteration) the systems $\mathcal{E} = \mathcal{E}^{(0)}, \mathcal{E}^{(1)}, \mathcal{E}^{(2)}, \ldots, \mathcal{E}^{(k)}$. Let p be the number of occurrences of function symbols in $\mathcal{E}^{(0)}$. Then, there exists $lev : vars\left(\bigcup_{j=0}^{k} \mathcal{E}^{(j)}\right) \longrightarrow \mathbb{N}$ such that:*

- *it fulfills condition $(*)$ of Def. 7.6 for all systems of equations $\mathcal{E}^{(j)}$ (i.e., $lev(\ell) \leq p$ and $lev(r) \leq p$ for all equations $\ell = r$ in $\mathcal{E}^{(j)}$), and*
- *any time a substitution $[X/t]$ has been applied, then $lev(X) = lev(t)$.*

Proof. Consider the graphs G_j associated to the systems $\mathcal{E}^{(j)}$. First of all observe that if there is a function fulfilling the requirements for the system $\mathcal{E}^{(j)}$, then the same function works for all graphs $\mathcal{E}^{(i)}$ with $i < j$. This allows us to concentrate on $\mathcal{E}^{(k)}$. By Lemma 7.9, we know that G_k is acyclic and it contains the same number (p) of $+1$ edges as $\mathcal{E}^{(0)}$. 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. 7.10 □

Theorem 7.11 (termination) *Given a system of equations $\mathcal{E}$, all the non-deterministic branches of the computation of* `general_aci`$(\mathcal{E})$ *terminate in a finite amount of time.*

Proof. Assume that there is a non-failing sequence of non-deterministic choices $\mathcal{E}^{(0)}, \mathcal{E}^{(1)}, \mathcal{E}^{(2)}, \ldots, \mathcal{E}^{(k)}$ (they are the values of $\mathcal{E}$ 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 $\mathcal{E}^{(0)}$. We know from Lemma 7.10 that there exists a function $lev : vars\left(\bigcup_{j\geq 0} \mathcal{E}^{(j)}\right) \longrightarrow \mathbb{N}$ such that

- it fulfills condition $(*)$ for all systems of equations $\mathcal{E}^{(j)}$, and
- any 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 $\mathcal{L}_{\mathcal{E}}$ for the system of equations $\mathcal{E}$:

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

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

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

- action (1) clearly reduces the complexity by removing one equation
- action (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 $X = t$)
- action (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
- action (5) will lead in one additional iteration to an action (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
- action (7) replaces an equation of complexity $2 + l_1 + r_1$ with a collection of equations each having complexity $l + r \leq l_1 + r_1$, leading to a smaller total complexity (thanks to lexicographical ordering)

- action (8) is a complex action 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 $\mathcal{E}_{\mathtt{ns}}$. 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 action application decreases the complexity measure $\mathcal{L}_{\mathcal{E}}^{(lev)}$. 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 actions. Moreover, we do not know if the function lev fulfills condition (α) for the successive systems $\mathcal{E}^{(k+1)}, \mathcal{E}^{(k+2)}, \ldots$.

To prove termination, a further measure is needed: let

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

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

$\mathcal{M}_{\mathcal{E}^{(0)}}$ 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 $vars(\mathcal{E}^{(0)})$ to $\{0,\ldots,p\}$ that fulfills condition (α).

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 fewer 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 fewer than t (the new variables N are introduced in equations of the form $X = \cdots \cup N$ and thus, $\ell'(N) \leq \ell(X)$).
- 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.

7.11 □

7.5 Discussion

An alternative approach to solve general $ACI1$ unification problems can be derived by specializing general methodologies for solving unification problems in *union* of disjoint equational theories. Combining unification procedures for different unification problems has been a major topic of investigation for years [60]. Various proposals have been put forward to allow combination of unification procedures under different conditions on the equational theories [68, 30, 66, 37, 57]. Baader and Schulz [10] have proposed a general technique for combining unification procedures over disjoint theories under very simple restrictions—i.e., *constants restrictions*. 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 unification procedure for $ACI1$ with constants and a standard Herbrand unification algorithm to obtain a unification procedure for general $ACI1$.

Another approach to solve general $ACI1$ unification problem can be found in the work by Kapur and Narendran [35]; the authors propose a guess-and-verify algorithm as part of their proofs of NP-completeness of this unification problem.

In terms of complexity, the general $ACI1$ problem is known to be FNP but the algorithm that we proposed does not provide a non-deterministic polynomial time solution; this is due to the explicit application of substitutions.

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 resolution of a $ACI1$ with constants problem—and we have seen earlier that this leads to $O(2^{n^2})$ solutions. Since this process can be repeated once for each nesting, a rough upper bound to the number of solution is $O(2^{n^3})$. Observe that this number of solutions is greater than those computed by the $(Ab)(C\ell)$. This fact suggest that the general $ACI1$ unification should be used only when the problem is really not expressible by general $(Ab)(C\ell)$ unification and the full range of solutions is required.

8 Examples

This section presents a series of unification problems, each encoded in one of the different frameworks presented in the previous sections. This allows us to give an intuitive idea of the expressive power of the different frameworks considered.

- Determine 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). In its abstract syntax, we can encode the problem as:

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

  The problem can either be encoded as a flat $(Ab)(C\ell)$ problem or by an $ACI1$ with constants problem, i.e., by encoding as:

$$c \cup e \cup g \cup b\flat = g \cup g \cup e \cup b\flat \cup c \cup e$$

- Solve the simple problem of verifying whether two teachers are covering three courses in their current course assignment:

$$\mathit{Teacher1} \ \cup \ \mathit{Teacher2} \ = \mathit{course1} \ \cup \ \mathit{course2} \ \cup \ \mathit{course3}$$

  Note that this problem is characterized by the fact that the solutions of the variables range over union of elements (subsets of $\mathit{course1} \cup \mathit{course2} \cup \mathit{course3}$) rather than simply over elements, as it can be expressed by $(Ab)(C\ell)$ unification.

- Consider the graph-coloring problem consisting of a 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

$$\{c_1, c_2, c_3\}$$

  This problem can be easily encoded as a general $(Ab)(C\ell)$ problem in the following way:

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

The solution of this equation provides a solution of the corresponding graph coloring problem. A possible solution (actually, the first one returned by the $CLP(\mathcal{SET})$ interpreter [22]) is:

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

Also solutions with two colors, such as $X_1 = X_3 = c_1$, $X_2 = X_4 = c_2$ and $R = \{\{c_1, c_3\}, \{c_2, c_3\}\}$ are computed.

- Let us consider a deterministic finite state automata on the alphabet $\{0, 1\}$, containing a set of states $Q = \{q_n \mid Q_1\}, Q_1 = \{q_0, \dots, q_{n-1}\}$ where q_0 is the initial state and q_n is the unique final state. Q_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 can be encoded as follows. The set of transitions D is:

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

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

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

that forces the transitions $(q_0, a_0, Y_1), (Y_1, a_1, Y_2), \dots, (Y_k, a_k, q_n)$ to belong to D. If $b_0 \cdots b_h$ is a string that should not be accepted, then we need to add the equations:

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

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

For example, if we want a four states automata that accepts the strings 000 and 001 and rejects the strings 011 and 10, then we write the $(Ab)(C\ell)$ equation system

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

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

$$D = \{(q_0, 0, q_1), (q_0, 1, q_2), (q_1, 0, q_3), (q_1, 1, q_2), (q_2, 0, q_0), (q_2, 1, q_0), (q_3, 0, q_3), (q_3, 1, q_3)\}$$

- Consider the problem of computing all the subgraphs of an undirected graph $G = \langle \{c_1, \dots, c_n\}, E \rangle$ 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 general $ACI1$ problem. In fact, all the subgraphs of G are given by the solutions for G_1 of the equation

$$G_1 \cup G_2 = \bigcup_{\{c_i, c_j\} \in E} \{\{c_i, c_j\}\} \tag{4}$$

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

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

Observe that G_1 is temporarily extended to G_3 by introducing artificial loops, thus allowing to recognize paths of length less than n. As another example, to find the paths with exactly, say three, edges, one could add the following equation to (4):

$$G_1 = \{\{c_1, X_1\}, \{X_1, X_2\}, \{X_2, c_n\}\} \cup G_1$$

If, in particular, E may contain self-loops and we want to ignore them from the path, then we simply need to add the equation:

$$\{X_1\} \cup \{X_2\} \cup \cdots \cup \{X_n\} = \{c_1\} \cup \cdots \cup \{c_n\}$$

9 Related Work

Substantial part of the related proposals has already been addressed throughout the paper, e.g., in Sections 1, 5.5, 6.5, and 7.5. In this section we provide a brief overview of other related efforts.

Boolean unification. Boolean unification is a very powerful framework that allows one, in particular, to mimic the $ACI1$ with constants unification problems. The richer language of Boolean unification, however, allows the various solutions of a given $ACI1$ problem—described by ACI-matrices in Section 5—to be encoded in a very compact way, as a single complex solution. A fundamental work in this area is [50], which surveys both the *Boole's method* and the *Löwenheim's method.* The former has been originally described in [15] while the second has been proposed in [49]. 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 $\mathcal{U}$. 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 for the decision problem of Boolean unification is studied in [8]. In the elementary case, i.e., without constants, it is NP-complete. In the case with constants it is 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.* A *unique* most general unifier can be proved to be sufficient to cover all solutions (see [15]). 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. Computable Set Theory [16] was mainly developed at the New York University to potentiate the inference engine for Theorem provers and for the implementation of the imperative set-based programming language SETL [58]. 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.

Set constraints. Set constraints [38, 3] 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 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 the terms in the language is such to prevent to encode large classes of set unification problems, e.g., problems involving nested sets. In order to represent nested sets in set constraints, we need to make use of a distinguished function symbol $\{\cdot\}$ (as described also in Section 7.1); using the set constraint interpretation of expressions, the two expressions $\{\{s,t\}\}$ and $\{\{s\},\{t\}\}$ are mapped to the same set.

Alternative Representations of Sets. Other syntactic representations of sets are also feasible. For instance as $\{\}_n(t_0,\dots,t_n)$, 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 in [59]. 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, like in $\{\}_3(X,Y,Z) = \{\}_2(a,b)$.

This representation scheme allows one to express only set terms with a known upper bound on their cardinality, such as

$$|\,\{\}_n(t_1,\dots,t_n)\,| \leq n\ .$$

10 Conclusions

In this paper we 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 we have shown the 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

The research presented in this paper has benefited from discussions with A. Formisano, G. Gupta, E. G. Omodeo, C. Piazza, A. Policriti, and D. Ranjan all of whom we would like to thank.

E. Pontelli is partially supported by NSF Grant, CCR 9875279, CCR 9900320, EIA 9810732, and HRD 9906130. A. Dovier and G. Rossi are partially supported by MIUR project: *Aggregate—and number—reasoning for computing.*

References

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

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

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

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

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

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

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

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

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

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

[11] Baader, F., and Schulz, K. U. Unification Theory. In *Automated Deduction — A Basis for Applications, Vol. I: Foundations — Calculi and Methods* (1998), W. Bibel and P. Schmidt, Eds., vol. 8 of *Applied Logic Series*, Kluwer Academic Publishers, pp. 225–263.

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

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

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

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

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

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

[18] Dantsin, E., and Voronkov, A. 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* (1999), W. Thomas, Ed., vol. 1578 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 180–196.

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

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

[21] Dovier, A., Piazza, C., and Policriti, A. A Fast Bisimulation Algorithm. In *Proc. of Int. Conference on Computer Aided Verification (CAV'01)* (2001), G. Berry, H. Comon, and A. Finkel, Eds., vol. 2102 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 79–90.

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

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

[24] Dovier, A., Pontelli, E., and Rossi, G. Set Unification Revisited. Technical Report NMSU-CSTR-98-17, Department of Computer Science, New Mexico State University, USA, October 1998.

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

[26] Fages, F. Associative-Commutative Unification. *Journal of Symbolic Computation 3* (1987), 257–275.

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

[28] Grieskamp, W. *Notes on ZAP*. Technical University of Berlin, July 2000. `uebb.cs.tu-berlin.de/zeta/zetadistdoc/zapnotes.html`.

[29] Hermann, M., and Kolaitis, P. 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* (1997), G. Smolka, Ed., vol. 1330 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 282–296.

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

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

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

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

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

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

[36] Kifer, M., and Lausen, G. 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* (1989), ACM Press, pp. 134–146.

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

[38] Kozen, D. Set Constraints and Logic Programming. *Information and Computation 142*, 1 (1998), 2–25.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[56] Rounds, W. C. 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, 1988.

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

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

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

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

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

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

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

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

[65] Tarski, A. Sur les ensembles fini. *Fundamenta Mathematicae VI* (1924), 45–95.

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

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

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

A Proofs

Lemma 6.3 *The global effect of a call to* `AbCl_step` *on*

$$e \equiv \{t_1, \ldots, t_m \mid h\} = \{t'_1, \ldots, t'_n \mid k\},$$

where h, k are either variables or terms with outermost symbol $f \neq \{\cdot \mid \cdot\}$, is to replace e by a conjunction of equations $t_i = t'_j$ between elements of the two sets (added directly to the system $\mathcal{E}_{\mathtt{ns}}$) plus:

- *if h and k are the same variable, a set of membership equations of the form $h = \{\cdots \mid h\}$.*
- *otherwise, the equation(s) obtained from one of the following cases:*

  type 1: *an equation $h = \{t'_{j_1}, \ldots, t'_{j_r} \mid k\}$, $r \leq n$ or,*

  type 2: *an equation $k = \{t_{i_1}, \ldots, t_{i_\ell} \mid h\}$, $\ell \leq m$ or,*

  type 3: *the equations $h = \{t'_{j_1}, \ldots, t'_{j_r} \mid N\}, k = \{t_{i_1}, \ldots, t_{i_\ell} \mid N\}$, $\ell \leq m, r \leq n$, N a new variable, the same for the two equations.*

- *A number of (passive) solved form equations $N_i = s$ for temporarily introduced variables N_i.*

Observe that all solved form equations generated above are immediately transformed into substitutions and immediately applied. Note also that the substitutions obtained from the $N_i = s$ will not affect the rest of the system.

Proof. We prove the lemma by induction on the pair $\langle m, n\rangle$, where $\langle m, n\rangle < \langle m', n'\rangle$ iff ($m \leq m'$ and $n < n'$) or ($m < m'$ and $n \leq n'$).

Let us start by looking at the case where $h = k = X$ for some variable X.

$\langle 0, n\rangle$, $\langle m, 0\rangle$: the input is already a membership equation (of the type 1 or 2) or an equation of the form $X = X$ (which is a special case of type 1 equations).

$\langle m, n\rangle, m > 0, n > 0$: the property holds by inductive hypothesis in all the cases arising from the alternatives (i)–(iv) of the algorithm.

Assume now that h and k are not the same variable.

$\langle 0, n\rangle$, $\langle m, 0\rangle$: the input is already an equation of the type 1 or 2.

$\langle m, n\rangle, m > 0, n > 0$: the property holds immediately by inductive hypothesis in the cases (i)–(iii) of the algorithm. If choice (iv) is selected, then the equation $\{t_1, \ldots, t_m \mid h\} = \{t'_1, \ldots, t'_n \mid k\}$ will introduce two new equations in the stack $\mathcal{E}_{\mathtt{aux}}$:

$$e_1 \equiv \{t_2, \ldots, t_m \mid h\} = \{t'_1 \mid N\}, e_2 \equiv \{t_1 \mid N\} = \{t'_2, \ldots, t'_n \mid k\}$$

e_2 is selected first. N and k are not the same variable, since N has just been introduced. By inductive hypothesis, it returns equations of the form $t_1 = t'_j$ plus one or two equations in one of the following forms:

1. an equation $N = \{t'_{j_1}, \ldots, t'_{j_r} \mid k\}$, $r \leq n - 1$ or,
2. an equation $k = \{t_1 \mid N\}$ (or $k = N$) or,
3. the equations $N = \{t'_{j_1}, \ldots, t'_{j_r} \mid N'\}, k = \{t_1 \mid N'\}$, (or $k = N$), $r \leq n - 1$.

These equations are selected next and the resulting substitutions are applied.

1. If the substitution $[N/\{t'_{j_1}, \ldots, t'_{j_r} \mid k\}]$ is applied, e_1 becomes $\{t_2, \ldots, t_m \mid h\} = \{t'_1, t'_{j_1}, \ldots, t'_{j_r} \mid k\}$, $r \leq n - 1$ and it is selected next. The claim is then proved by inductive hypothesis.

2. If the substitution $[k/\{t_1 \mid N\}]$ is applied (similarly, if $k = N$), then e_1 remains:[4]

$$\{t_2, \ldots, t_m \mid h\} = \{t'_1 \mid N\}$$

and it is selected. By inductive hypothesis, the algorithm will generate equations of the type $t_i = t'_1$ along with equations in one of the following forms:

   (a) an equation $h = \{t'_1 \mid N\}$ (or $h = N$) is generated. In this case, we have generated a situation of type 3 (or 2).

   (b) an equation $N = \{t_{p_1}, \ldots, t_{p_\ell} \mid h\}$, $\ell \leq m - 1$ is generated: by substitution application, we obtain the equation: $k = \{t_1, t_{p_1}, \ldots, t_{p_\ell} \mid h\}$ (type 2) plus the solved form (passive) equation for N.

   (c) the equations $h = \{t'_1 \mid N'\}$ (or $h = N'$), and $N = \{t_{p_1}, \ldots, t_{p_\ell} \mid N'\}$, $\ell \leq m-1$ are generated. Globally, we have the equations: $h = \{t'_1 \mid N'\}, k = \{t_1, t_{p_1}, \ldots, t_{p_\ell} \mid N'\}$ plus the solved form (passive) equation for N (similarly for $k = N'$).

3. If the substitutions induced by the equations $N = \{t'_{j_1}, \ldots, t'_{j_r} \mid N'\}, k = \{t_1 \mid N'\}$, (or $k = N$) $r \leq n-1$, are applied, e_1 becomes: $\{t_2, \ldots, t_m \mid h\} = \{t'_1, t'_{j_1}, \ldots, t'_{j_r} \mid N'\}$. Applying the inductive hypothesis, we have three cases:

   (a) $h = \{t'_{q_1}, \ldots, t'_{q_{r'}} \mid N'\}$, $r' \leq r + 1 \leq n$: globally, we are in case type 3.

   (b) $N' = \{t_{p_1}, \ldots, t_{p_{\ell'}} \mid h\}$, $\ell' \leq m-1$. By applying the substitution we obtain: $k = \{t_1, t_{p_1}, \ldots, t_{p_{\ell'}} \mid h\}$. Globally, we are in case type 2.

   (c) $h = \{t'_{q_1}, \ldots, t'_{q_{r'}} \mid N''\}$, $r' \leq n$ and $N' = \{t_{p_1}, \ldots, t_{p_{\ell'}} \mid N''\}$, $\ell' \leq m - 1$. Apply the substitution and obtain: $k = \{t_1, t_{p_1}, \ldots, t_{p_{\ell'}} \mid N''\}$, plus a passive equation for N'. Globally, we are in case type 3.

6.3 □

Lemma 7.7 *Given a system of equations $\mathcal{E}$, and given p the number of occurrences of elements of Σ in $\mathcal{E}$, exactly one of the following conditions holds:*

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

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

1. $X = Y$
2. $X = f(Y_1, \ldots, Y_n)$, $f \in \Sigma$ and $f \not\equiv \cup$
3. $X = Y_1 \cup Y_2$

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

The goal is to map $\mathcal{E}'$ to a set of linear integer constraint systems. Each possible complete p-level for $\mathcal{E}'$ (and thus for $\mathcal{E}$) 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 $\mathcal{E}'$, for a suitable q. Such mapping is realized as follows: for each (term) variable X in $\mathcal{E}'$ we introduce a corresponding (integer) variable x; then we add equations and disequations according to the following rules:

[4]To be precise, t_i, t'_1 could be modified by the substitution, but this modification will not affect the structure of the sets.

$$\begin{aligned} y_2 &= 0 \\ y_2 &\leq x \\ y_2 &\leq y_1 \\ x &= y_1 + 1 \\ y_1 &= y_2 + 1 \\ y_3 &\leq x \end{aligned}$$

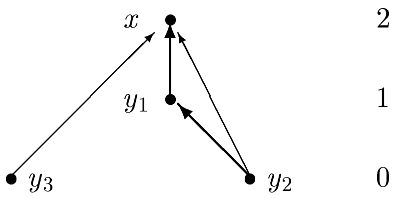


Figure 7: A simplified integer system, the corresponding graph, and a solution

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

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:

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

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 exists one solution σ of S_i such that for each variable x we have $\sigma(x) \leq p$.

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.
- 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 Fig. 7; 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 $x \leq y$ 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 does 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 $\sigma(x) \leq \sigma(y)$

- 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 $\phi(X) \leq r$, where r is the number of $+1$ edges in G_{S_i}—in particular $r \leq p$.

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 outcoming 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 $h + k \geq 1$. 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 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 $k \geq 0$: in each consistent labeling of G_{S_i} we must have that $\sigma(v_1) = \cdots = \sigma(v_h) = \sigma(x) - 1$. 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) $\sigma(w_i) \leq \sigma(x)$ for $i = 1, \ldots, k$, thus $\sigma(w_i) = \sigma(x)$ or $\sigma(w_i) \leq \sigma(v_1)$

   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 graph 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 $\sigma(x) = \sigma(v_1) + 1$. 7.7 □

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 7.3). 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:

$$\begin{array}{cccccccc}
S_1 & \dots & S_n & X_1 & \dots & X_p & \\
\hline
\multicolumn{1}{|c}{A_{1,1}} & \dots & \multicolumn{1}{c|}{A_{1,n}} & A_{1,n+1} & \dots & \multicolumn{1}{c|}{A_{1,n+p}} & T_1 \\
\multicolumn{1}{|c}{\vdots} & \ddots & \multicolumn{1}{c|}{\vdots} & \vdots & \ddots & \multicolumn{1}{c|}{\vdots} & \vdots \\
\multicolumn{1}{|c}{A_{m,1}} & \dots & \multicolumn{1}{c|}{A_{m,n}} & A_{m,n+1} & \dots & \multicolumn{1}{c|}{A_{m,n+p}} & T_m \\
\hline
\multicolumn{1}{|c}{A_{m+1,1}} & \dots & \multicolumn{1}{c|}{A_{m+1,n}} & A_{m+1,n+1} & \dots & \multicolumn{1}{c|}{A_{m+1,n+p}} & X_1 \\
\multicolumn{1}{|c}{\vdots} & \ddots & \multicolumn{1}{c|}{\vdots} & \vdots & \ddots & \multicolumn{1}{c|}{\vdots} & \vdots \\
\multicolumn{1}{|c}{A_{m+p,1}} & \dots & \multicolumn{1}{c|}{A_{m+p,n}} & A_{m+p,n+1} & \dots & \multicolumn{1}{c|}{A_{m+p,n+p}} & X_p \\
\hline
\end{array}$$

The most general unifier for the elementary problem can be obtained as follows:

$$\begin{aligned}
S_j &= \bigcup_{i=1}^{m+p} A_{i,j} \qquad\qquad T_i = \bigcup_{j=1}^{n+p} A_{i,j} \\
X_j &= \bigcup_{i=1}^{m} A_{i,n+j} \cup \bigcup_{k=1}^{n} A_{m+j,k} \cup A_{m+j,n+j}
\end{aligned}$$

One can easily prove that this method provides the same solution as the $ACI1$ unification with constants algorithm presented in Section 5.

Example B.1 *Let us consider the same unification problem $S_1 \cup S_2 \cup X = T_1 \cup T_2 \cup X$ as in Example 5.4; the elementary ACI-matrix is*

$$\begin{array}{cccc}
S_1 & S_2 & X & \\
\hline
\multicolumn{1}{|c}{R_1} & \multicolumn{1}{c|}{R_3} & \multicolumn{1}{c|}{R_7} & T_1 \\
\multicolumn{1}{|c}{R_2} & \multicolumn{1}{c|}{R_4} & \multicolumn{1}{c|}{R_8} & T_2 \\
\hline
\multicolumn{1}{|c}{R_5} & \multicolumn{1}{c|}{R_6} & \multicolumn{1}{c|}{R_9} & X \\
\hline
\end{array}$$

Let us observe that the variables in the matrix have been named to show the correspondence with the new variables used in Example 5.4.

Given the unification problem:

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

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

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

Under the assumption that

$$\{L_1, \ldots, L_{k_2}\} \cap \{R_1, \ldots, R_{h_2}\} = \emptyset$$

any (non-deterministic) solution can be described using a $(h_1 + h_2) \times (k_1 + k_2)$ matrix B such that

- for $h_1 + 1 \leq i \leq h_1 + h_2$ and $k_1 + 1 \leq j \leq k_1 + k_2$ we have $B[i, j] = \perp$
- all the other components of B have a value taken from $\{0, 1\}$
- for each $1 \leq i \leq h_1$ $\sum_{j=1}^{k_1+k_2} B[i, j] \geq 1$ and for each $1 \leq j \leq k_1$ $\sum_{i=1}^{h_1+h_2} B[i, j] \geq 1$.

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

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

Additionally, B generates the new set of equations:

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

The discussion can be easily generalized to the case where the two sides of the equation have variables in common. 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 8. The table in Figure 8 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 9 it is depicted 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 $\perp$;
- the non-zero entries in quadrant 9 are assigned $\perp$; 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 $1 \leq j \leq k_1$ we have

$$\sum_{i=1}^{h_1+h_2} B[i, j] + \sum_{i=1}^{c} B[i, j] \geq 1$$

- for $1 \leq i \leq h_1$ we have

$$\sum_{j=1}^{k_1+k_2} B[i,j] + \sum_{j=1}^{c} B[i,j] \geq 1$$

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

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

Figure 8: Elementary ACI-matrix

N_1^L $\dots$ $N_{k_1}^L$	L_1 $\dots$ L_{k_2}	Com_1 $\dots$ Com_c	
Quad 1	Quad 2	Quad 3	N_1^R $\vdots$ $N_{h_1}^R$
Quad 4	Quad 5	Quad 6	R_1 $\vdots$ R_{h_2}
Quad 7	Quad 8	Quad 9 $I_{c,c}$	Com_1 $\vdots$ Com_c

Figure 9: Extended Boolean Matrix B