From set to hyperset unification

Davide Aliffi Agostino Dovier Gianfranco Rossi

Abstract

In this paper we show how to extend a set unification algorithm—i.e., an extended unification algorithm incorporating the axioms of a simple theory of sets—to hyperset unification, that is to sets in which, roughly speaking, membership can form cycles. This is obtained by enlarging the domain from that of terms (hence, trees) to that of *graphs* involving free as well as interpreted function symbols (namely, the set element insertion and the empty set), which can be regarded as a convenient denotation of hypersets. We present a hyperset unification algorithm which (non-deterministically) computes, for each given unification problem, a finite collection of systems of equations in solvable form whose solutions represent a complete set of solutions for the given unification problem. The crucial issue of termination of the algorithm is addressed and solved by the addition of simple non-membership constraints. Finally, the hyperset unification problem dealt with is proved to be NP-complete, and the proposed algorithm to be non–deterministically polynomial.

1 Introduction

Sets are a basic data structure in programming languages and, more generally, in software specification languages (e.g., for data-base development, knowledge representation in A.I., etc.). Set data structures fit quite naturally in logic-based formalisms which provide the necessary equipment for representing (complex) properties and operations over sets. In this context, sets can be represented as terms, built out of other terms using a suitable set constructor operator (such as union, element insertion, etc.). From the operational viewpoint, a resolution-based inference mechanism is usually exploited, which in turn requires a suitable unification algorithm fulfilling the specific properties of the considered set constructor.

All set unification algorithms studied in the past (e.g., [28, 12, 3, 18, 35, 2]), however, consider well-founded sets only. An interesting extension of the concept of set is that of *hyperset* [9], that is sets in which, roughly speaking, membership can form cycles. Hypersets we are considering here are very closely related to Aczel's non-well-founded sets (cf. [1]), but they all have finite cardinality and are *hybrid*, in that their construction involves free Herbrand functors.

Usefulness of hypersets as a convenient data model has been investigated in [10] and [8]. Examples of hypersets arise when modeling a circular process or structure by means of sets. Moreover, hyperset theory has been applied in a number of areas of logic, linguistics, and computer science. As Barwise says in (Foreword to [1]) "[$\cdots$] *It seemed that in order to understand common knowledge (a crucial feature of communication), circular propositions, various aspects of perceptual knowledge and self-awareness* [$\cdots$] *we either had to give up the tools of set theory which are so well loved in mathematical logic, or we had to enrich the conception of set, finding one that admits of circular sets, at least.* [$\cdots$] "

Introducing hypersets as a data structure in a logic programming language requires a unification algorithm that is able to deal with objects denoting hypersets.

In this paper we show how to extend set unification to hyperset unification. We start with the set unification algorithm described in [18]. This algorithm extends standard (syntactic) unification by incorporating the axioms of a simple theory of sets that establish the (semantic) properties of the set constructor `with` (basically, permutativity and absorption, by which the order of elements and repeated elements in set terms can be ignored).

The key idea that allows set unification to be extended to hyperset unification is that of enlarging the domain of discourse from terms (i.e., finite trees) over the signature Σ to *directed labeled graphs* over Σ, possibly with cycles. This data structure, when involving the interpreted function symbol `with` used as the set constructor, can be regarded as a convenient way to denote hypersets. For instance, a solution to the equation

$$X = \{X\}$$

is a cyclic graph which can be interpreted as an hyperset containing itself as its only element.

The set unification algorithm of [18] is therefore modified so as to make it capable of dealing with graphs representing hypersets. Some of the required changes are the same needed when moving from standard unification to unification over (uninterpreted) rational trees, for which a number of efficient algorithms have been proposed in the literature, mostly in the context of logic programming languages (e.g., [14, 31, 23, 32]). We will consider, in particular, the algorithm in [31] (actually, a revised, simplified version of it).

Since the data structures involved may be cyclic, the new algorithm avoids performing full variable substitution (in the style of Martelli and Montanari's algorithm [30]). Moreover, simple *non-membership constraints*, avoiding the (possibly endless) repeated insertions of the same elements into hypersets, are added to this modified set unification algorithm, in order to guarantee termination.

The proposed algorithm is not simply a variant of the set unification algorithm in [18]. Actually, the (semantically) different kind of objects dealt with by the new algorithm causes non-trivial termination problems, requiring new, non-trivial solutions. Moreover—and in some sense, more importantly—, the resulting algorithm proves to have a polynomial non-deterministic complexity, which is not the case for the well-founded set unification algorithm.

An alternative approach could be using the general (and elegant) methods for combining unification algorithms [5] and, more generally, constraint domains—even non well-founded—and solvers [4]. According to these methods, we could concentrate our efforts on analyzing the "pure" case of hyperset (domains and) unification and then obtain automatically the algorithm for the general hybrid case.

However, as for the well-founded case [18], the real struggle is to write a *goal-driven* and *terminating* algorithm for the pure case. Then, the combined algorithm could be obtained, in our context, by adding two (deterministic) actions concerning standard term decomposition. Therefore, we have faced directly the more general hybrid problem without making use of the results of [5, 4].

The rest of the paper is organized as follows. Section 2 introduces the notion of set term and briefly presents the main ideas underlying the set unification algorithm of [18]. Section 3 gives an intuitive presentation of hypersets and briefly discusses the motivations underlying their introduction. Section 4 addresses the key issue of hyperset representation in the logical framework we are using. The hyperset unification algorithm is presented in Section 5. In particular, Section 5.2 introduces the key rewriting rule dealing with set-set equations which is also proved to be sound and complete, whereas Section 5.3 presents the complete hyperset unification algorithm, which is then proved to be always terminating (Section 5.4). Finally, Section 6 is devoted to proving the NP-completeness of the hyperset unification problem, and in particular the non–deterministic polynomiality of the proposed algorithm.

2 Set unification

2.1 Basic definitions and notation

We assume all the definitions and notations usually employed in unification theory (cf., e.g., [26, 30, 34]).

In particular, the following conventions will be adopted. Σ denotes a signature with *arity* function $ar : \Sigma \longrightarrow \mathbb{N}$. $\mathcal{V}$ is a denumerable set of logical variables disjoint from Σ; accordingly, $\tau(\Sigma)$ and $\tau(\Sigma \cup \mathcal{V})$ denote the collection of all ground and non-ground terms over Σ and $\Sigma \cup \mathcal{V}$. If t is a term, then $vars(t)$ denotes the set of variables occurring in t. Capital letters X, Y, Z, etc. represent variables; f, g, etc. stand for function symbols (i.e., elements of Σ); $\equiv$ denotes the syntactic identity relation between first-order terms over $\Sigma \cup \mathcal{V}$. If φ is a term or a first order formula, then φ^X_Y denotes the result of replacing every occurrence of the variable X by the variable Y in φ.

The following notions will also play a fundamental role in the sequel:

Definition 1 *A* Herbrand system *is a finite collection* $\{\ell_1 = r_1, \ldots, \ell_n = r_n\}$ *of first-order equations, where* $\ell_1, r_1, \ldots, \ell_n, r_n \in \tau(\Sigma \cup \mathcal{V})$.

Definition 2 *An equation* $X = t$ *of a Herbrand system* $\mathcal{E}$ *is said to be in* solved form *with respect to* $\mathcal{E}$ *if* X *is a variable which has no other occurrence in* $\mathcal{E}$. *When all the equations of a system* $\mathcal{E}$ *are in solved form with respect to* $\mathcal{E}$, $\mathcal{E}$ *itself is said to be in solved form (i.e.,* $\mathcal{E}$ *has the form* $\{X_1 = t_1, \ldots, X_n = t_n\}$, X_i*s pairwise distinct variables such that* X_i *does not occur in* t_j, *for* $i, j = 1, \ldots, n$*).*

2.2 Set terms

Following [18], we choose to represent a set as a list of its elements using the binary operator `with` as the set constructor. In accordance, we require that the signature Σ contains the function symbol `with` of arity two (left associating), and at least one constant symbol, say $\emptyset$. In view of the intended interpretation, $\texttt{with}$ will be interpreted as the *element insertion operation*, and $\emptyset$ as a *member-less* object.

Definition 3 *Variables and any term of the form* $s\,\texttt{with}\,t$*, where* s *and* t *are terms, are called* set terms*. All other terms are called non-set terms.*

Example 4 *The terms*

$$\emptyset\ \texttt{with}\ a\ \texttt{with}\ f(a)\ \texttt{with}\ X$$

and

$$S\ \texttt{with}\ \emptyset\ \texttt{with}\ (\emptyset\ \texttt{with}\ a)$$

are set terms. Also

$$f(X)\ \texttt{with}\ Y$$

where f *is a function symbol different from* $\texttt{with}$ *, is a set term.*

$$f(a, \emptyset\ \texttt{with}\ a)$$

is a term, but not a set term, because the main function symbol is different from $\texttt{with}$ *.*

Note that both *ground* and *non-ground* set terms are allowed. Moreover, set terms can be *nested* at any depth.

A set term like

$$f(X)\ \texttt{with}\ Y$$

describes what is called a *colored* set, i.e. a set built from any memberless object, called the *kernel* of the set.

This means that there is not a unique distinct constant for the empty set, and any non–set term can play the role of the empty set. The idea is that the kernel of an object is what remains when all its elements have been removed. In particular, a kernel itself is a colored set with no elements.

The main reason for considering colored set terms is keeping the term syntax as simple as possible.

A formal definition of the kernel of a term can be given as follows:

Definition 5 *The kernel of a term is defined as follows:*

- $\texttt{ker}(b\,\texttt{with}\,t) = b$*,* b *any constant symbol;*
- $\texttt{ker}(f(t_1, \ldots, t_n)) = f(t_1, \ldots, t_n)$*,* f *any function symbol different from* $\texttt{with}$ *;*
- $\texttt{ker}(t\,\texttt{with}\,s) = \texttt{ker}(t)$*.*

Example 6 $\texttt{ker}(\emptyset\,\texttt{with}\,a\,\texttt{with}\,b) = \emptyset$; $\texttt{ker}(b\,\texttt{with}\,a\,\texttt{with}\,b) = b$; $\texttt{ker}(f(a)) = f(a)$.

A precise semantics for terms is given in the following subsection.

2.3 Semantics of (well-founded) set terms

The following two properties are assumed as defining properties of the `with` operator:

$$
\begin{array}{lrlcl}
(E_p) & \forall XYZ & (X\ \texttt{with}\ Y\ \texttt{with}\ Z & = & X\ \texttt{with}\ Z\ \texttt{with}\ Y) \\
(E_a) & \forall XY & (X\ \texttt{with}\ Y\ \texttt{with}\ Y & = & X\ \texttt{with}\ Y).
\end{array}
$$

According to (E_p), the order of elements in a set term is immaterial (*permutativity* property); according to (E_a), also duplicates in a set term are not relevant (*absorption* property).

The interpretation domain is the ordinary Herbrand Universe H_Σ (or, equivalently, the set of finite trees over Σ), modulo the smallest congruence relation $\cong$ induced by permutativity and absorption (denoted $H_\Sigma/\cong$).

The interpretation of a term t is the *canonical representative* $[t]$ of the equivalence class to which t belongs A possible criterion to select a canonical representative $[t]$ from each class is the following (see [18]). Assume that a total ordering of Σ is given. Then a total ordering $<$ can be defined antilexicographically on the Herbrand universe H_Σ. This means that $r\,\texttt{with}\,t < s\,\texttt{with}\,u$ holds, recursively, when either $t < u$ or t coincides with u and $r < s$. A ground term g is said to be canonical if either it is a constant or every one of its subterms is canonical and, moreover, $t < u$ holds for every subterm of the form $s\,\texttt{with}\,t\,\texttt{with}\,u$ of g.

The interpretation of $s = t$ is the syntactic equality between $[s]$ and $[t]$. $s \in t$ is true in the interpretation if $[t]$ is of the form:

$$t_0\ \texttt{with}\ t_1\ \texttt{with}\ \ldots\ \texttt{with}\ [s]\ \texttt{with}\ \ldots\ \texttt{with}\ t_n.$$

The fundamental properties of the interpretations of set and non-set terms are described by an axiomatic first-order theory *Set*, whose basic axioms are:[1]

$$
\begin{array}{ll}
(K) & \forall x\, x \notin f(\cdots) \qquad f \in \Sigma, f \not\equiv \texttt{with} \\
(W) & \forall xyz\, (z \in x\,\texttt{with}\,y \leftrightarrow z \in x \vee z = y) \\
(E_k) & \forall xy\, ((\texttt{ker}(x) = \texttt{ker}(y) \wedge \forall z\, (z \in x \leftrightarrow z \in y)) \rightarrow x = y)
\end{array}
$$

[1] To be precise, the first axiom, (K), is an axiom schema.

in addition to the standard equality axioms and the Clark Equality Theory axioms for the non-set terms.

The interpretation $H_\Sigma/\cong$ can be proved to be a model of this simple theory of sets. Precisely, these axioms, along with the considered privileged model of $H_\Sigma/\cong$, describe the (abstract) data structure of the so called *hereditarily finite (and well-founded) colored sets* (see, e.g., [13], Chapter II), where *hereditarily finite* means containing a *finite* number of elements that can be either member-less objects (also called *ur-elements* in [36]) or *finite* sets.

Example 7 *The interpretation of the set term* $\emptyset\,\mathtt{with}\,a\,\mathtt{with}\,a\,\mathtt{with}\;(\emptyset\,\mathtt{with}\,b)$ *is the class whose canonical representative is the term* $\emptyset\,\mathtt{with}\,a\,\mathtt{with}\;(\emptyset\,\mathtt{with}\,b)$. *The intended meaning of both terms is the set containing the (member-less) element* a *and the singleton set containing the (member-less) element* b*, that is,* $\{a, \{b\}\}$.[2]

The interpretation of the term $c\;\mathtt{with}\;a\;\mathtt{with}\;(\emptyset\;\mathtt{with}\;b)$ *is the class whose canonical representative is the term itself, and the intended meaning is the set containing the same elements* a *and* $\{b\}$*, but based on the kernel* c*, that is, using the notation of [18],* $\{a, \{b\} \mid c\}$.

An alternative representation of sets as terms uses the binary union symbol $\cup$ as the set constructor [3, 12, 28]. In this case, the signature is composed only by constant symbols and a unique function symbol $\cup$, and set unification is dealt with as a problem of *ACI-unification*, i.e., unification under Associativity, Commutativity, and Idempotence. This approach, however, limits the kind of sets that can be represented to be *flat* sets. The list-like representation used in this paper (and in most papers dealing with sets in logic programming languages, such as [25, 11, 18, 22, 24]), on the contrary, naturally accommodates for *nested* sets (such as $\{\emptyset, \{a, \{\emptyset\}\}\}$).[3] On the other hand, the $\cup$-based representation allows to write set terms that cannot be directly expressed using a list-like representation, as for instance the term $X \cup a \cup Y$; the list-like representation, in fact, can only represent

[2]Note that duplicates do not occur in the set, of course, but they may occur in the concrete representations of it, i.e., in the set term denoting the set.

[3]Nested sets could be allowed in the $\cup$-based representation, too, provided a singleton (free) operator $\{\cdot\}$ is introduced. At our knowledge, no proposals have directly addressed this problem so far, although a non-deterministic unification procedure can be obtained from the results of [5].

the union of a sequence of singletons and a single variable. While some expressive convenience is sacrificed by the restrictions imposed by the `with` constructor—which, however, does not seem to affect significantly most practical cases—the same restrictions avoid the need to consider some complex unification problems with a large number of distinct solutions, which instead can be expressed using the $\cup$-based representation [3, 2].

2.4 A set unification algorithm à la Robinson

A set unification algorithm that extends standard unification so as to fulfill the desired properties of the set constructor operator `with` has been described in [18].

If T is the equational theory constituted by the permutativity and absorption axioms (E_p), (E_a) (see Section 2.3) this algorithm computes, for each given Herbrand system $\mathcal{E}$, a complete set of T-unifiers for $\mathcal{E}$ (refer to, e.g., [20] for the notions of T-unifier, complete set of T-unifiers, and related notions involved in the general theory of unification with respect to an equational theory T). In this context, $=$ will be used to express equality with respect to the theory T.

We briefly recall the main ideas underlying the definition of the algorithm in [18] since they will serve as well as the basis for the development of our hyperset unification algorithm.

The algorithm gets as input a Herbrand system $\mathcal{E}$ (possibly involving set terms) and non-deterministically produces a finite set of Herbrand systems in solved form $\mathcal{E}_1, \ldots, \mathcal{E}_n$. For each non-deterministic branch, the algorithm is basically in the style of Robinson's unification algorithm, using (full) variable substitution as one of its basic actions.

Set-set equations are dealt with by exploiting the following set equality axiom:

$$(E_k^s) \qquad \forall YZVW \left(\begin{array}{lr} V \;\texttt{with}\; Y = W \;\texttt{with}\; Z \;\; \leftrightarrow & \\ \qquad (Y = Z \wedge V = W) \vee & \text{(i)} \\ \qquad (Y = Z \wedge V = W \;\texttt{with}\; Z) \vee & \text{(ii)} \\ \qquad (Y = Z \wedge V \;\texttt{with}\; Y = W) \vee & \text{(iii)} \\ \qquad \exists N \; (V = N \;\texttt{with}\; Z \wedge W = N \;\texttt{with}\; Y) & \text{(iv)} \end{array} \right)$$

which is an equivalent, though more effective, criterion for testing set equality than that obtained either by the extensionality axiom (E_k) or by the permutativity and absorption axioms (E_p) and (E_a) (see [19] for discussions and complete proofs concerning selection of the set equality criteria).

Parts (ii) and (iii) deal with the presence of duplicates in the left-hand side and in the right-hand side term of the equation, respectively. Part (iv) has the following meaning: even if Y and Z are distinct objects, the two sets V `with` Y and W `with` Z can be equal, due to the permutativity property of the `with` operation. This happens if Y is an element of W and Z is an element of V, and the remaining parts are equal. These conditions can be expressed by the equalities $V = N \texttt{ with } Z$, $W = N \texttt{ with } Y$, where N is a new variable.[4]

Further transformation rules are required by equations of the form

$$X = X \texttt{ with } t_0 \texttt{ with} \ldots \texttt{with } t_n$$

and

$$X \texttt{ with } t'_0 \texttt{ with} \ldots \texttt{with } t'_m = X \texttt{ with } t_0 \texttt{ with} \ldots \texttt{with } t_n .$$

The first equation simply states that $t_0, \ldots, t_n$ must belong to X. It is dealt with by replacing it with the new equation

$$X = N \texttt{ with } t_0 \texttt{ with} \ldots \texttt{with } t_n,$$

N a new uninstantiated variable, which is more general than the given equation since N can be instantiated to any set term (i.e., the new equation states that X can be <u>any</u> set containing $t_0, \ldots, t_n$).

The second kind of equation involving set terms,

$$(1) \qquad X \texttt{ with } t'_0 \texttt{ with} \ldots \texttt{with } t'_m = X \texttt{ with } t_0 \texttt{ with} \ldots \texttt{with } t_n ,$$

must be dealt with in a special way since—as pointed out in [18]—a direct usage of the set equality axiom (E^s_k) leads to non-termination. The key idea in this case is that if two sets (in the intuitive sense, and using the common mathematical set notation) $X \cup \{t_0, \ldots, t_n\}$, $X \cup \{t'_0, \ldots, t'_m\}$ are equal, then there must exist subsets $\{t_{j_1}, \ldots, t_{j_k}\}$, $\{t'_{i_1}, \ldots, t'_{i_h}\}$ and N such that

$$\{t'_{i_1}, \ldots, t'_{i_h}\} = \{t_{j_1}, \ldots, t_{j_k}\},$$
$$X = (\{t'_0, \ldots, t'_m\} \setminus \{t'_{i_1}, \ldots, t'_{i_h}\}) \cup (\{t_0, \ldots, t_n\} \setminus \{t_{j_1}, \ldots, t_{j_k}\}) \cup N$$

[4] In fact, in the underlying set theory [18], one can prove that $\forall XY\,(Y \in X \leftrightarrow \exists Z(X = Z \texttt{ with } Y))$, which allows membership to be expressed in terms of equality.

The algorithm in [18], which basically repeatedly applies these rules to perform set unification, has been proved to be sound and complete, and to terminate for every input system $\mathcal{E}$. In order to ensure termination, however, some form of determinism in the control strategy is needed. In particular, (full) variable substitution must be applied as much as possible before applying rules involving set term handling.

3 Hypersets

In this section we give an intuitive definition of hypersets and briefly discuss the motivations underlying their introduction.

Roughly speaking, *hypersets* (or non-well-founded sets) [21, 8] are sets for which the membership relation can form cycles, or infinite descending chains. The original motivation for the introduction of non-well-founded sets was to give a set-theoretic mathematical model of concurrent processes (see [1] for the history of hypersets). Afterwards, interesting applications have been given by Barwise to the semantics of natural languages, and in particular to the semantics of languages containing a truth predicate (see [6] and [7]), where non-well-founded sets allow one to take into account the classical paradox of the liar.

The mathematical concept of set has two different aspects. On one hand, there is the idea that sets are built by means of the iterated application of the operation of insertion of an element (the `with` operator in our framework), starting from the empty set, or from some kind of previously defined non-set object. This approach leads to the cumulative hierarchy of sets introduced by von Neumann in the thirties, and now customary in logic and mathematics. Thus, given any set, we can go down the hierarchy and consider the elements of our set, the elements of the elements etc. It is a consequence of the previous construction that this can only be done a finite number of times, and that no membership loops can be encountered.

On the other hand, there is also the intuition that a set is obtained from some kind of structured object by means of forgetting the structure, and in fact it is this kind of conception that makes sets useful as a data structure. But this idea does not necessarily require that sets are organized in a well-founded cumulative hierarchy.

As an example, Barwise has shown ([6]) that to give a set–theoretical model of shared information in certain types of real situations requires the use of sets in which membership loops do exist.

Another interesting application of hypersets has been given by Milner and Tofte ([33] and [8], § 3.3) to the semantics of a simple functional language allowing recursive definitions. In [27] a promising hyperset view of web-like databases has been presented.

4 Representing hypersets

In this section we show how hypersets are introduced in the logic framework we are considering in this paper. For a more detailed and formal presentation of the first-order axiomatic theory of hypersets and its semantics, refer to [16, 17].

4.1 Graph representation of hypersets

The language is the same as in Section 2. Given a signature Σ, $\mathcal{G}_\Sigma$ is the set of *directed labeled graphs*, where for every graph G in $\mathcal{G}_\Sigma$, every node of G is labeled by a constant or function symbol in Σ, the edges going out of each node are ordered, and their number equals the arity of the node label.

Ground terms of the language are easily mapped to graphs of $\mathcal{G}_\Sigma$ by interpreting, as usual, the function symbols of Σ as graph constructors. Precisely, each function symbol $f \in \Sigma$ of arity n maps n graphs to a (rooted) graph whose root node is labeled by f and whose subgraphs are the arguments of the mapping.

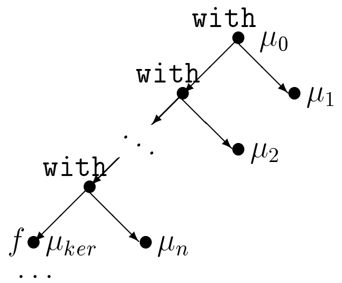
Graphs in $\mathcal{G}_\Sigma$ can be partitioned into equivalence classes according to a notion of *bisimulation* between directed labeled graphs, suitably adapted from Milner's classical one to our representation of hybrid hypersets. This notion will also account for the permutativity and absorption property of the `with` operator.

Definition 8 *Given two graphs $G_1 = \langle N_1, E_1 \rangle$ and $G_2 = \langle N_2, E_2 \rangle$, a relation $b \subseteq N_1 \times N_2$ is said to be a* bisimulation *if the following conditions hold:*

- *for each node $\mu \in N_1$ there is a node $\nu \in N_2$ such that $\mu b \nu$, and for each node $\nu \in N_2$ there is a node $\mu \in N_1$ such that $\mu b \nu$;*
- *if $\mu b \nu$ then μ and ν have the same label f;*

- *if $\mu b \nu$, μ and ν are labeled f, where f is a function symbol of arity n and different from* `with` *, $\langle \mu, \mu_1 \rangle, \ldots, \langle \mu, \mu_n \rangle$ are the edges that come out from μ in G_1 (G_2), and $\langle \nu, \nu_1 \rangle, \ldots, \langle \nu, \nu_n \rangle$ are the edges that come out from ν in G_2 (G_1), then for $i = 1, \ldots, n$, $\mu_i b \nu_i$;*

- *if a node $\mu \in N_1$ (N_2) is labeled by* `with` *, we need to identify the* `with`*-rooted graph Γ_0 containing μ as one of its* `with`*-labeled node. Let Γ_0 be the graph*



  and let $\nu_0, \nu_1, \ldots, \nu_m$, and possibly ν_{ker}, be the nodes of the corresponding sub-graph in G_2 (G_1); then

  – *all the nodes $\mu \in N_1$ (N_2) explicitly labeled by* `with` *in the diagram are such that $\mu b \nu_0$ (in particular $\mu_0 b \nu_0$),*

  – *for all $i = 1, \ldots, n$, there is $j = 1, \ldots, m$ such that $\mu_i b \nu_j$,*

  – *for all $j = 1, \ldots, m$, there is $i = 1, \ldots, n$ such that $\mu_i b \nu_j$;*

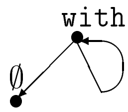
  – *$\mu_{ker} b \nu_{ker}$.*

The existence of a bisimulation between graphs induces an equivalence relation $\mathcal{B}$ over $\mathcal{G}_\Sigma$. The interpretation domain, therefore, is the set $\mathcal{G}_\Sigma$ modulo $\mathcal{B}$ (denoted $\mathcal{G}_\Sigma/\mathcal{B}$). Each Σ-term t is mapped to a member (i.e., the representative member) of the equivalence class to which the graph G_t obtained from t belongs.

By still interpreting `with` as the element insertion operation, and non-set terms as memberless objects, $\mathcal{G}_\Sigma/\mathcal{B}$ can be viewed as the abstract data structure of *hybrid hypersets* (precisely, hereditarily finite colored hybrid hypersets). *Hybrid* means that they can involve also free Herbrand functors, not only the `with` constructor and the empty-set symbol $\emptyset$.

In particular, any `with`-rooted graph G containing a `with`-labeled node whose right-hand outgoing arc is part of a cycle represents a *proper hyperset* (of course, $\mathcal{B}$-equivalent graphs in the same equivalence class represent the same hyperset).

Example 9 *The graph*



represents the (proper) hyperset which has itself as its only element (named Ω *by Aczel in [1]).*

Observe that we are assuming here that the domain of hypersets includes that of ordinary sets: hence, a set is also considered a special kind of hyperset. Therefore, in order to single out hypersets that are not ordinary sets we shall call them proper hypersets.

The set theory we are considering here (called *HSet*) is basically the simple first-order theory *Set* mentioned in Section 2.2, in which axiom (F_3^S) is replaced by a suitable form of the anti-foundation axiom (**AFA**). Basically, (**AFA**) states that for all systems of equations of a certain form, a solution always exists in the universe of hypersets (see [8], chapter 6). Moreover, such a solution is unique, when a mapping for certain special variables (the parameters) has been fixed.

The notion of system of equations and solution are given in the next subsection, as well as a more precise characterization of (**AFA**).

4.2 Hypersets as solutions of Herbrand systems

Differently from Herbrand models, in the domain $\mathcal{G}_\Sigma/\mathcal{B}$ not all the elements are denoted by terms of the language. Actually, terms denote only `with`-rooted trees, hence, well-founded sets. Proper hypersets, instead, can be described as solutions of Herbrand systems of equations (this is analogous to the case of unification over rational trees with respect to standard unification).

Intuitively, an equation of the form

$$X \;=\; \emptyset\,\mathtt{with}\,Y_1\;\cdots\;\mathtt{with}\,Y_n$$

where some of the Y_i can coincide with X, or it can be bound to a term containing X, admits a solution which is (up to bisimulation) a directed `with`-rooted graph of the kind we are interpreting as (hybrid) hypersets.

As an example, the hyperset Ω represented by the (cyclic) graph above can be obtained as solution of the Herbrand system $\mathcal{E} = \{X = \emptyset \texttt{ with } X\}$ (in fact, (**AFA**) states that there exists a unique solution to $\mathcal{E}$: the hyperset Ω).

More precisely:

Definition 10 *Let V be a set of variables. A* valuation *is a mapping $\theta : V \longrightarrow \mathcal{G}_\Sigma$. The* application *$t\theta$ of a valuation θ to a term t, $vars(t) \subseteq V$, can be defined recursively as follows:*

- *$X\theta = \theta(X)$ if X is a variable;*
- *$f(t_1, \ldots, t_n)\theta$ is a graph whose root μ is labeled by f, for $i = 1, \ldots, n$, there is an edge $\langle \mu, \mu_i \rangle$, and μ_i is the root of $t_i\theta$.*

Let $\mathcal{E}$ be a Herbrand system, and V a set of variables such that $vars(\mathcal{E}) \subseteq \mathcal{V}$. A solution *of $\mathcal{E}$ is a valuation $\theta : V \longrightarrow \mathcal{G}_\Sigma$ such that $s\theta =_\mathcal{B} t\theta$ for each equation $s = t$ in $\mathcal{E}$.*

Among Herbrand systems, a special role is played by Herbrand systems in solvable form.

Definition 11 *A Herbrand system $\mathcal{E}$ is said to be in* solvable form *if each equation in $\mathcal{E}$ has one of the forms:*

- *$X = Y$ and X does not occur elsewhere in $\mathcal{E}$;*
- *$X = f(t_1, \ldots, t_n)$, $f \in \Sigma$, and X does not occur as left-hand side of any other equation in $\mathcal{E}$.*

Observe that this definition allows solvable form systems containing equations of the form $X = f(t_1, \ldots, t_n)$, where X occurs in some of $t_1, \ldots, t_n$.

The importance of the solvable form is that a solution for a system in such a form can be obtained in a straightforward way. Here is a procedure that builds a solution for every Herbrand system in solvable form $\mathcal{E} = \{X_1 = t_1, \ldots, X_n = t_n\}$, by providing a graph $\eta(V)$ for each variable V occurring in $\mathcal{E}$ (actually an entry point to a node of a global graph $G = \langle N, E \rangle$).

Make solution

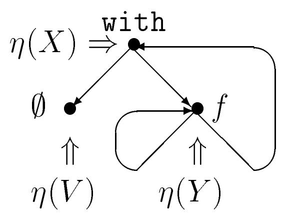
$N := \emptyset$; $E := \emptyset$;
for each variable V in $\mathcal{E}$ do $N := N \cup \{\mu_V\}$;
for each variable V occurring in $t_1, \ldots, t_n$ but not in $X_1, \ldots, X_n$,
    do label μ_V with $\emptyset$;
for each equation $X_i = t_i$ in $\mathcal{E}$, do $\mathtt{AddEdges}(\mu_{X_i}, t_i)$;
for each variable V in $\mathcal{E}$ do $\eta(V) = \mu_V$.

where the procedure `AddEdges` is defined as follows:

$\mathtt{AddEdges}(\mu : \mathtt{node}, t : \mathtt{term})$

if t is a variable V
then merge μ and μ_V
else (t is of the form $f(s_1, \ldots, s_m)$)
    label μ with f;
    for $i = 1$ to m do
        if s_i is a variable V
        then $E := E \cup \{\langle \mu, \mu_V \rangle\}$
        else let ν be a new node; $N := N \cup \{\nu\}$; $E := E \cup \{\langle \mu, \nu \rangle\}$;
            $\mathtt{AddEdges}(\nu, s_i)$.

Example 12 *From the solvable form system* $\{X = V \ \mathtt{with}\ Y, Y = f(Y, X)\}$ *we obtain the solution*



which involves a (proper) hyperset, containing a cyclic term.[5]

[5]Observe that, in the domain of *hybrid* hypersets, we can have an hyperset containing (possibly cyclic) non-set terms, as well as a (possibly cyclic) non-set term containing hypersets, such as, for instance, the one described by $\{X = f(Y), Y = \emptyset \ \mathtt{with}\ X\}$.

Given a Herbrand system in solvable form we can always obtain a solution for it. This is in accordance with (**AFA**) which assures that a solution exists and is unique. More precisely (cf. [1]):

Anti Foundation Axiom (**AFA**). Let $\mathcal{E}$ be a solvable form system $\{X_1 = t_1, \ldots, X_n = t_n\}$ such that $X_1, \ldots, X_n$ are distinct variables and t_i is either a variable or a set term $\emptyset$ `with` Y_1 `with` $\ldots$ `with` Y_h, $h \geq 0$, Y_i variables. Let $V_1, \ldots, V_m$ be the variables (called the *parameters*) occurring in $t_1, \ldots, t_n$ but not in $X_1, \ldots, X_n$. Then, once the parameters have been fixed mapping them to graphs $G_{V_1}, \ldots, G_{V_m}$, there exists a unique, modulo bisimulation, solution for $\{X_1 = t_1, \ldots, X_n = t_n\}$.

To this respect, observe that a system of the form

$$\{X = X \texttt{ with } Y_1 \texttt{ with } \ldots \texttt{ with } Y_n\}$$

which is in solvable form (but not in the form expected by (**AFA**)), simply states that $Y_1, \ldots, Y_n$ belong to X, and, hence, it has an infinite number of solutions, in contrast with (**AFA**).

Actually, in our unification algorithm the above equation will always be replaced by the equivalent equation

$$\{X = N \texttt{ with } Y_1 \texttt{ with } \ldots \texttt{ with } Y_n\}$$

where N is a new variable (a parameter of the system). In this way, once N has been instantiated to a graph in $\mathcal{G}_{\{\emptyset, \texttt{with}\}}$, uniqueness of the solution (modulo assignment to the other parameters and bisimulation) is ensured. As shown in [15], it is therefore possible to extend both axiom (**AFA**), and its counterpart for rational trees presented in [29], to an axiom (**AFA′**) that holds for our hybrid context:

$$(\mathbf{AFA'}) \quad \forall V_1 \ldots V_h \exists ! X_1 \ldots X_n \, (X_1 = t_1 \wedge \ldots \wedge X_n = t_n)$$

where $\mathcal{E} = \{X_1 = t_1, \ldots, X_n = t_n\}$ is in solvable form, $V_1 \ldots V_h$ are the parameters, and no situations of the form: $X_{i_1} = X_{i_2} \texttt{ with } \ldots, X_{i_2} = X_{i_3} \texttt{ with } \ldots, \ldots, X_{i_n} = X_{i_1} \texttt{ with } \ldots$, (also called "Cycle" in Section 5.2) occur in $\mathcal{E}$.

5 A hyperset unification algorithm

Once established a convenient representation of hypersets, we can develop our hyperset unification algorithm. The algorithm, like the one in [18], directly works on systems of equations whose solutions are members of $\mathcal{G}_\Sigma$ instead of working on graphs in $\mathcal{G}_\Sigma$ (basically, in the style of Martelli and Montanari's unification algorithm). For practical reasons, Herbrand systems are viewed as multisets of equations instead of sets of equations.

Solving an Herbrand system $\mathcal{E}$ means to rewrite it (non-deterministically) into a certain number of solvable form systems $\mathcal{E}_1, \ldots, \mathcal{E}_n$ which describe the solution space of $\mathcal{E}$.

The following definitions and notations will be useful in what follows.

Definition 13 *If* $\theta : V \longrightarrow \mathcal{G}_\Sigma$ *is a valuation, and* $V' \subseteq V$*, then*

$$\theta|_{V'} = [X \mapsto \theta(X) : X \in V'].$$

Let $\mathcal{E}$ *be a Herbrand system, then* $Soln(\mathcal{E})$ *is the set of all (graph) solutions of* $\mathcal{E}$ *(cf. Def. 10); if* $V \subseteq vars(\mathcal{E})$*, then*

$$Soln(\mathcal{E})|_V = \{\theta|_V : \theta \in Soln(\mathcal{E})\}.$$

Given a set V *of variables and two Herbrand systems* $\mathcal{E}_0, \mathcal{E}_1$*,* $V \subseteq vars(\mathcal{E}_0)$ *and* $V \subseteq vars(\mathcal{E}_1)$*,* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *are said to be* equivalent *in* V *if*

$$Soln(\mathcal{E}_0)|_V = Soln(\mathcal{E}_1)|_V .$$

Moreover, it is convenient to introduce the notion of a flat Herbrand system to make the algorithm description easier (in particular, to simplify the comparison of compound terms—cf., e.g., [14, 32]).

Definition 14 *A Herbrand system* $\mathcal{E}$ *is said to be* flat *if every equation* e *in* $\mathcal{E}$ *has either the form* $X = Y$*, or the form* $X = g(Y_1, \ldots, Y_n)$ *with* g *a function symbol in* Σ*,* $ar(g) = n$*, and* $Y_1, \ldots, Y_n$ *variables.*

Hereafter, we will consider flat systems only. Given a Herbrand system $\mathcal{E}$, it is always possible to obtain a flat system $\mathcal{E}'$ that is equivalent to $\mathcal{E}$ in $vars(\mathcal{E})$ by introducing a new distinct variable X_i in place of each non-variable subterm t_i occurring in $\mathcal{E}$, and adding the corresponding new equation $X_i = t_i$ to $\mathcal{E}'$.

5.1 Dealing with non-set terms

The following four rules are used in the unification algorithm to perform variable management and decomposition of terms other than set terms.

> Let $\mathcal{E}$ be a Herbrand system, and e be any equation occurring in $\mathcal{E}$.
> Let f and g be arbitrary symbols from Σ (including `with`).
>
> **Variable Elimination** [El]
>
> <u>if</u> $e \equiv \big(X = X\big)$ <u>then</u> $\mathcal{E} := \mathcal{E} \setminus \{e\}$;
>
> **Variable Replacement** [VR]
>
> <u>if</u> $e \equiv \big(X = Y\big)$ <u>and</u> X occurs somewhere else in $\mathcal{E}$
> <u>then</u> $\mathcal{E} := (\mathcal{E} \setminus \{e\})^X_Y \cup \{X = Y\}$;
>
> **Term Decomposition with Failure** [TDF]
>
> <u>if</u> $e \equiv \big(X = f(X_1, \ldots, X_n)\big)$ <u>and</u> there is an $e' \equiv \big(X = g(Y_1, \ldots, Y_m)\big)$ in $\mathcal{E}$ with $f \not\equiv g$ <u>or</u> $m \neq n$ <u>then</u> *failure*;
>
> **Term Decomposition** [TD]
>
> <u>if</u> $e \equiv \big(X = f(X_1, \ldots, X_n)\big)$, $f \not\equiv$ `with` <u>and</u> there is an $e' \equiv \big(X = f(Y_1, \ldots, Y_n)\big)$ in $\mathcal{E}$ <u>then</u> $\mathcal{E} := \mathcal{E} \setminus \{e'\} \cup \{X_1 = Y_1, \ldots, X_n = Y_n\}$.

Note that [VR] actually performs a simple replacement of variables by variables (not full variable substitution). Accordingly, the usual "occur-check" test is suppressed.

5.2 Dealing with set terms

The set equality axiom (E^s_k) presented in Section 2 can be easily expressed as a rewrite rule in our algorithm, adapted to the case of flat terms:

> <u>if</u> $e \equiv \big(X = V \,\texttt{with}\, Y\big)$ <u>and</u> there is an $e' \equiv \big(X = W \,\texttt{with}\, Z\big)$ in $\mathcal{E}$ <u>then</u> perform one of the following actions:
>
> (i) $\mathcal{E} := \mathcal{E} \setminus \{e'\} \cup \{Y = Z, V = W\}$;

(ii) $\mathcal{E} := \mathcal{E} \setminus \{e\} \cup \{Y = Z, V = X\}$;

(iii) $\mathcal{E} := \mathcal{E} \setminus \{e'\} \cup \{Y = Z, W = X\}$;

(iv) $\mathcal{E} := \mathcal{E} \setminus \{e'\} \cup \{V = N \texttt{ with } Z, W = N \texttt{ with } Y\}$

where N is a variable not occurring in $\mathcal{E}$.

This rule, however, is not always guaranteed to terminate, as shown by the following example.

Example 15 *Consider the system*

$$\{X = V \texttt{ with } Y, V = W \texttt{ with } Z, X = W \texttt{ with } Z\}.$$

By applying the above rule (case iv) to the first and third equation we get the new system

$$\{X = V \texttt{ with } Y, V = N_1 \texttt{ with } Z, W = N_1 \texttt{ with } Y, V = W \texttt{ with } Z\}.$$

Then, by applying the same rule (case iv) to the second and fourth equation we get

$$\begin{array}{ll}\{ & X = V \texttt{ with } Y, V = N_1 \texttt{ with } Z,\\ & W = N_1 \texttt{ with } Y, N_1 = N_2 \texttt{ with } Z, W = N_2 \texttt{ with } Z \ \}.\end{array}$$

The last three equations represent a situation as complex as the initial system.

Observe that, by applying variable substitutions, the set unification algorithm for the well-founded case would easily detect that the system of Example 15 requires two set terms with the same rest variables to be compared (see equation (1) in Section 2.4) and as such it could apply the rule for this special case. More generally, the whole termination proof for the set unification algorithm for the well-founded case heavily depends on the availability of full variable substitution. Since, on the contrary, in the new context we are dealing with in this paper, applying full variable substitution would clearly lead to termination problems we need to devise other techniques to guarantee termination of the algorithm.

From the above example, it is evident that the problem with termination originates from the fact that the same elements (viz., Y and Z) are repeatedly added to the same sets, by means of the newly generated variables $N_1, N_2, \ldots$, thus creating increasingly larger set terms.

The key idea to achieve termination in the new algorithm, therefore, is to use simple *non-membership constraints* that avoid the repeated insertion of the same element Y into the same hyperset X.

Definition 16 *A* constraint $\mathcal{C}$ *is a finite collection of simple* $\notin$*-constraints of the form* $(Y \notin X)$*, where* X *and* Y *are variables.*

Adding constraints to our algorithm requires the notion of *solution* to be suitably modified.

Definition 17 *Let* $\mathcal{E}$ *be a Herbrand system,* $\mathcal{C}$ *a constraint,* V *a set of variables, such that* $vars(\mathcal{E}) \subseteq V$ *and* $vars(\mathcal{C}) \subseteq V$*.*

A solution *of* $\langle \mathcal{E}, \mathcal{C} \rangle$ *is a mapping* $\theta : V \longrightarrow \mathcal{G}_\Sigma$ *such that* $s\theta =_\mathcal{B} t\theta$ *for each equation* $s = t$ *in* $\mathcal{E}$ *and such that* $X\theta \neq_\mathcal{B} (X \texttt{ with } Y)\theta$ *for each constraint* $Y \notin X$ *in* $\mathcal{C}$*.*

The new unification algorithm will get as input a flat Herbrand system $\mathcal{E}$ along with a constraint $\mathcal{C}$, which is assumed to be initially empty. $\mathcal{C}$ will be inspected at each iteration of the unification algorithm to detect possible inconsistencies as described by the following rule:

> **Membership Failure** [MeF]
>
> Let $\mathcal{C}$ be a constraint and c be a simple constraint occurring in $\mathcal{C}$.
>
> <u>if</u> $c \equiv \big(Y \notin X\big)$ <u>and</u> $\big(X = V \texttt{ with } Y\big)$ occurs in $\mathcal{E}$ <u>then</u> *failure*.

Moreover, to propagate constraints over variables the following rule is introduced:

> **Update constraints** [Upd]
>
> Let $\mathcal{C}$ be a constraint and c be a simple constraint occurring in $\mathcal{C}$.
>
> <u>if</u> $c \equiv \big(Z \notin X\big)$ <u>and</u> there is an equation $\big(X = V \texttt{ with } Y\big)$ in $\mathcal{E}$ <u>and</u> $\big(Z \notin V\big)$ is not yet in $\mathcal{C}$ <u>then</u> $\mathcal{C} := \mathcal{C} \cup \{Z \notin V\}$.

The presence of the constraint $\mathcal{C}$ requires that the variable replacement rule is modified so as to apply to $\mathcal{C}$ as well as to $\mathcal{E}$:

> **Variable Replacement** [VR']
>
> <u>if</u> $e \equiv \big(X = Y\big)$ <u>and</u> X occurs somewhere else in $\mathcal{E}$ or $\mathcal{C}$
> <u>then</u> $\mathcal{E} := (\mathcal{E} \setminus \{e\})^X_Y \cup \{X = Y\}$; $\mathcal{C} := (\mathcal{C})^X_Y$.

The remaining rules of the algorithm are specifically devoted to deal with set terms.

In particular, the following two rules are introduced to deal with equations of the form $X = X$ `with` t_0 `with` $\ldots$ `with` t_n.

Let $\mathcal{E}$ be a Herbrand system, e be any equation occurring in $\mathcal{E}$, and N be a variable not occurring in $\mathcal{E}$.

Cycle [Cy]

if $e \equiv \big(X = X\,\texttt{with}\,Y\big)$
then $\mathcal{E} := \mathcal{E} \setminus \{e\} \cup \{X = N\,\texttt{with}\,Y\}$ and $\mathcal{C} := \mathcal{C} \cup \{Y \notin N\}$;

Indirect Cycle [ICy]

if $e \equiv \big(X_0 = X_1\,\texttt{with}\,Y_0\big)$ and there are n equations in $\mathcal{E}$ $\big(X_1 = X_2\texttt{with}Y_1\big), \ldots, \big(X_{n-1} = X_n\texttt{with}Y_{n-1}\big), \big(X_n = X_0\texttt{with}Y_n\big), n \geq 1$, $X_0, \ldots, X_n$ distinct variables, and $\big(X_i = X_0\big)$ is not yet in $\mathcal{E}$ for some $i \in \{1, \ldots, n\}$, then $\mathcal{E} := \mathcal{E} \cup \{X_1 = X_0, X_2 = X_0, \ldots, X_n = X_0\}$.

[Cy] simply restates the corresponding rule for the set unification algorithm adapted to flat terms. [ICy], instead, is motivated by the fact that it is possible to have *indirect cycles* of the form

$$\big(X_0 = X_1\ \texttt{with}\ Y_0\big), \ldots, \big(X_{n-1} = X_n\ \texttt{with}\ Y_{n-1}\big), \big(X_n = X_0\ \texttt{with}\ Y_n\big),$$

$n \geq 1$, $X_0, \ldots, X_n$ distinct from one another. If full variable substitution were available, an indirect cycle could be transformed into a simple cycle of the type considered in [Cy] by repeatedly applying variable substitution to $X_n, X_{n-1}, \ldots, X_1$. Since, on the contrary, full variable substitution is not available, then indirect cycles must be dealt with explicitly by means of a new ad-hoc rule.[6]

The last, and most important case, is that of equations between two set terms. These equations could be dealt with directly by means of the set equality rule shown at the beginning of this section, suitably modified with the addition of the $\notin$-constraints $Z \notin N$ and $Y \notin N$ in case (iv).

[6]Note that the "$(X_i = X_0)$ is not yet in $\mathcal{E}$" test in rule [ICy] and the "$(Z \notin V)$ is not yet in $\mathcal{C}$" test in rule [Upd] are intended to avoid naive loops.

Unfortunately, this simple minded solution does not guarantee completeness. In fact, the presence of the $\notin$-constraints on N in (iv) prevents the situations in which the considered set terms contain duplicates and Y differs from Z to be dealt with properly. For example, no solution of the equation[7]

$$\emptyset \ \texttt{with} \ a \ \texttt{with} \ b \ \texttt{with} \ a = \emptyset \ \texttt{with} \ b \ \texttt{with} \ a$$

(where a and b are different constants) could be found using this rule. This is because $a \neq b$ prevents the application of rules (i), (ii), (iii), whereas the application of rule (iv) introduces a new variable N, that must be bound to $\emptyset$ with a, along with the constraint $\{a \notin N, b \notin N\}$, so that [MeF] detects a *failure*.

The solution we adopt is to take explicitly into account these situations by providing new cases in the rule dealing with equations between set terms (namely, actions (d) and (f–h) of the rule [STD] shown below).

Moreover, simple non-membership constraints, besides those on the new variables N mentioned above, are introduced in order to simplify the termination proof and make all actions of the rule [STD] mutually exclusive.

Set Term Decomposition [STD]

Let $\mathcal{E}$ be a Herbrand system, $\mathcal{C}$ be a constraint, e be any equation occurring in $\mathcal{E}$, and M and N be variables not occurring in $\mathcal{E}$.
if $e \equiv \big(X = V \ \texttt{with} \ Y\big)$ and there is an $e' \equiv \big(X = W \ \texttt{with} \ Z\big)$ in $\mathcal{E}$ then perform one of the following actions:

(a) $\mathcal{E} := \mathcal{E} \setminus \{e'\} \cup \{Y = Z, V = W\}$; $\mathcal{C} := \mathcal{C} \cup \{Z \notin W\}$;

(b) $\mathcal{E} := \mathcal{E} \setminus \{e\} \cup \{Y = Z, V = X\}$; $\mathcal{C} := \mathcal{C} \cup \{Z \notin W\}$;

(c) $\mathcal{E} := \mathcal{E} \setminus \{e'\} \cup \{Y = Z, W = X\}$; $\mathcal{C} := \mathcal{C} \cup \{Y \notin V, \}$;

(d) $\mathcal{E} := \mathcal{E} \setminus \{e, e'\} \cup \{X = N \ \texttt{with} \ Y, Y = Z, V = X, W = X\}$; $\mathcal{C} := \mathcal{C} \cup \{Z \notin N\}$;

(e) $\mathcal{E} := \mathcal{E} \setminus \{e'\} \cup \{V = N \ \texttt{with} \ Z, W = N \ \texttt{with} \ Y\}$; $\mathcal{C} := \mathcal{C} \cup \{Y \notin N, Z \notin N, Y \notin V, Z \notin W\}$;

(f) $\mathcal{E} := \mathcal{E} \setminus \{e\} \cup \{V = X, W = N \ \texttt{with} \ Y\}$; $\mathcal{C} := \mathcal{C} \cup \{Y \notin N, Z \notin W\}$;

(g) $\mathcal{E} := \mathcal{E} \setminus \{e'\} \cup \{V = N \ \texttt{with} \ Z, W = X\}$; $\mathcal{C} := \mathcal{C} \cup \{Z \notin N, Y \notin V\}$;

[7] For the sake of simplicity, we give the example in non-flat form.

(h) $\mathcal{E} := \mathcal{E} \setminus \{e, e'\} \cup \{X = M \texttt{ with } Y, M = N \texttt{ with } Z, V = X, W = X\}$; $\mathcal{C} := \mathcal{C} \cup \{Z \notin N, Y \notin M\}$.

Example 18 *Consider the system of Example 15:*

$$\{X = V \texttt{ with } Y, V = W \texttt{ with } Z, X = W \texttt{ with } Z\}.$$

By applying action (e) *of the new rule* [STD] *we get the same system of equations as in Example 15 but also the new two simple constraints* $Y \notin N_1$, $Z \notin N_1$. *A further application of* [STD](e) *will also produce the equation* $N_1 = N_3 \texttt{ with } Z$, *so that the subsequent application of rule* [MeF] *causes a failure to be detected.*

Other cases of rule [STD] *(e.g.,* [STD](h)*) can then be taken into account, and indeed at least one succeeds in finding a solution for the given system.*

Note that rule [STD] also covers the case of set-set equations (1) of Section 2.4, without requiring the use of a special rule as in the algorithm in [18].

The following two lemmas formally state correctness and completeness of the rewriting rules presented in this section. They will be used for proving correctness and completeness of the whole unification algorithm (Theorem 23).

The meaning of part (2) of the two lemmas is that a solution θ of the system $\mathcal{E}'$ obtained from the given system $\mathcal{E}$ by means of a rule is still a solution of the given system $\mathcal{E}$, *even if θ does not satisfy the new constraints added by the rule.* This means that the effect of the rewriting rule on the constraints, necessary for termination, is not needed for correctness and completeness. *This important consideration allows us to discard the constraints in the output systems.*

Lemma 19 *Let $\langle \mathcal{E}', \mathcal{C}' \rangle$ be the system and constraint obtained by applying rule* [rule] *to $\langle \mathcal{E}, \mathcal{C} \rangle$, where* rule $\in \{\mathsf{El}, \mathsf{VR}', \mathsf{TDF}, \mathsf{TD}, \mathsf{MeF}, \mathsf{Upd}, \mathsf{Cy}, \mathsf{ICy}\}$. *Then*

$$(1) \qquad Soln(\mathcal{E}, \mathcal{C})|_{vars(\mathcal{E},\mathcal{C})} = Soln(\mathcal{E}', \mathcal{C}')|_{vars(\mathcal{E},\mathcal{C})}\,.$$

Moreover, for rule $\in \{\mathsf{El}, \mathsf{VR}', \mathsf{TDF}, \mathsf{TD}, \mathsf{Upd}, \mathsf{Cy}, \mathsf{ICy}\}$, *it holds that:*

$$(2) \qquad Soln(\mathcal{E}, \mathcal{C})|_{vars(\mathcal{E},\mathcal{C})} = Soln(\mathcal{E}', \mathcal{C})|_{vars(\mathcal{E},\mathcal{C})}\,.$$

Proof. First, we prove part (1) for the first four rules. Actions [El] and [VR'] clearly do not affect the set of solutions.

By definition, there is no solution for $\{X = f(\ldots), X = g(\ldots), \ldots\}$ if $f \not\equiv g$. This justifies action [TDF].

For rule [TD], observe that θ is a solution of $\{X = f(X_1, \ldots, X_n), X = f(Y_1, \ldots, Y_n)\}$ if and only if $X\theta =_{\mathcal{B}} (f(X_1, \ldots, X_n))\theta$ and $X_i\theta =_{\mathcal{B}} Y_i\theta$ for all $i = 1, \ldots, n$. This fact happens if and only if θ is a solution of $\{X = f(X_1, \ldots, X_n), X_1 = Y_1, \ldots, X_n = Y_n\}$.

Observe that actions [El], [TDF], and [TD] neither use nor affect the constraint. Thus, part (2) trivially holds. Rule [VR'] modifies $\mathcal{C}$ by applying a substitution. Since $\mathcal{E}'$ keeps track of the substitution applied, again part (2) holds.

To prove (1) for rule [Cy], observe that if θ' is a solution of $X = N\,\mathtt{with}\,Y$, $Y \notin N$, then $Y\theta'$ is an element of the hyperset $X\theta'$, therefore $X\theta' =_{\mathcal{B}} (X\ \mathtt{with}\ Y)\theta'$ and θ' belongs to $Soln(\mathcal{E}, \mathcal{C})|_{vars(\mathcal{E},\mathcal{C})}$. On the other hand, if θ is a solution of $X = X\ \mathtt{with}\ Y$, then $Y\theta$ is an element of the hyperset $X\theta$; extend θ by means of the assignment $N\theta = G$, where G is the hyperset obtained from $X\theta$ by removing the element $Y\theta$; then θ belongs to $Soln(\mathcal{E}', \mathcal{C}')|_{vars(\mathcal{E},\mathcal{C})}$.

To prove (2) we only need to notice that, even if θ' is a solution of $X = N\ \mathtt{with}\ Y$ not fulfilling $Y \notin N$ (i.e., such that $N\theta' =_{\mathcal{B}} (N\ \mathtt{with}\ Y)\theta'$), then $\theta'|_{\{X,Y\}}$ is a solution of $X = X\,\mathtt{with}\,Y$, because in any case $Y\theta$ is an element of the hyperset X.

This justifies [Cy]. The proof for [ICy] is similar.

Part (1) for rule [Upd] states that θ is a solution of $X = V\,\mathtt{with}Y, Z \notin X$ if and only if θ is a solution of $X = V\,\mathtt{with}Y, Z \notin X, Z \notin V$. To prove this fact, notice that there are no solutions for: $X = V\ \mathtt{with}\ Y, Z \notin X, V = V\ \mathtt{with}\ Z$. Part (2) holds trivially.

To justify [MeF], assume there is a solution θ for $X = V\mathtt{with}Y$, i.e. $X\theta =_{\mathcal{B}} (V\ \mathtt{with}\ Y)\theta$. Then $(X\ \mathtt{with}\ Y)\theta =_{\mathcal{B}} (V\ \mathtt{with}\ Y\ \mathtt{with}\ Y)\theta =_{\mathcal{B}} (V\ \mathtt{with}\ Y)\theta =_{\mathcal{B}} X\theta$. Thus, there are no solutions for $X = V\ \mathtt{with}\ Y, Y \notin X$. □

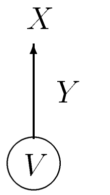
Lemma 20 *Let $\langle \mathcal{E}_i, \mathcal{C}_i \rangle$, for $i \in \{\mathsf{a}, \mathsf{b}, \ldots, \mathsf{h}\}$ be the systems and constraints obtained by applying rule [STD](i) to $\langle \mathcal{E}, \mathcal{C} \rangle$. Then*

$$(1) \qquad Soln(\mathcal{E}, \mathcal{C})|_{vars(\mathcal{E},\mathcal{C})} = \bigcup_{i \in \{\mathsf{a},\mathsf{b},\ldots,\mathsf{h}\}} Soln(\mathcal{E}_i, \mathcal{C}_i)|_{vars(\mathcal{E},\mathcal{C})}\,.$$

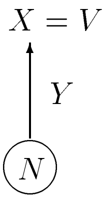
Moreover,

$$(2) \qquad Soln(\mathcal{E}, \mathcal{C})|_{vars(\mathcal{E},\mathcal{C})} = \bigcup_{i \in \{\mathsf{a},\mathsf{b},\ldots,\mathsf{h}\}} Soln(\mathcal{E}_i, \mathcal{C})|_{vars(\mathcal{E},\mathcal{C})} \,.$$

Proof. The theorem concerns the rewriting of equations of the form $X = V\ \mathtt{with}\ Y$. As explained in Def. 10, any solution of such an equation will map X into a (hyper)set containing the image of Y as one of its elements. Precisely, the equation $X = V\ \mathtt{with}\ Y$ has two different types of solutions, depending on whether Y already belongs to V or not, as shown pictorially by the following two diagrams:



which expresses the fact that $V \subset X$ (a proper subset) and X is obtained from V by means of the sole addition of the element Y;



which expresses the fact that Y already belongs to V and, hence, $X = V$ and N stands for the set $X - \{Y\}$.

Notice that, according to Definition 17, $\langle \{X = V\ \mathtt{with}\ Y\}, Y \notin V\rangle$ has only solutions of the form represented by the first diagram above.

The diagrams in Fig. 1 represent the 2^3 possible distinct solutions for the system of equations $\mathcal{E} = \{X = V\ \mathtt{with}\ Y,\ X = W\ \mathtt{with}\ Z\}$ according to the fact that $Y = Z$, $Y \in V$, $Z \in W$, or not.

To prove part (1), observe that, for each $i \in \{\mathsf{a}, \ldots, \mathsf{h}\}$, if θ is a solution of $\langle \mathcal{E}_i, \mathcal{C}_i\rangle$, then θ is also a solution of $\mathcal{E} = \{X = V\ \mathtt{with}\ Y, X = W\ \mathtt{with}\ Z\}$, precisely the solution depicted by diagram (i) of Fig. 5.2, provided variable N (actions d, e, f, g, h) is mapped to the hyperset R and M (action h) to the hyperset S. Conversely, if θ is a solution of $\mathcal{E} = \{X = V\ \mathtt{with}\ Y, X = W\,\mathtt{with}\,Z\}$, then there exists i such that θ is the solution depicted by diagram (i) of Fig. 5.2, and in such case θ is also a solution of $\langle \mathcal{E}_i, \mathcal{C}_i\rangle$.[8]

[8]Notice that, even if the constraint $Y \neq Z$ is not explicitly stated in actions $(\mathsf{e}) \ldots (\mathsf{h})$ of [STD], the other constraints occurring in them assure that only solutions satisfying $Y \neq Z$ can be actually obtained from these actions.

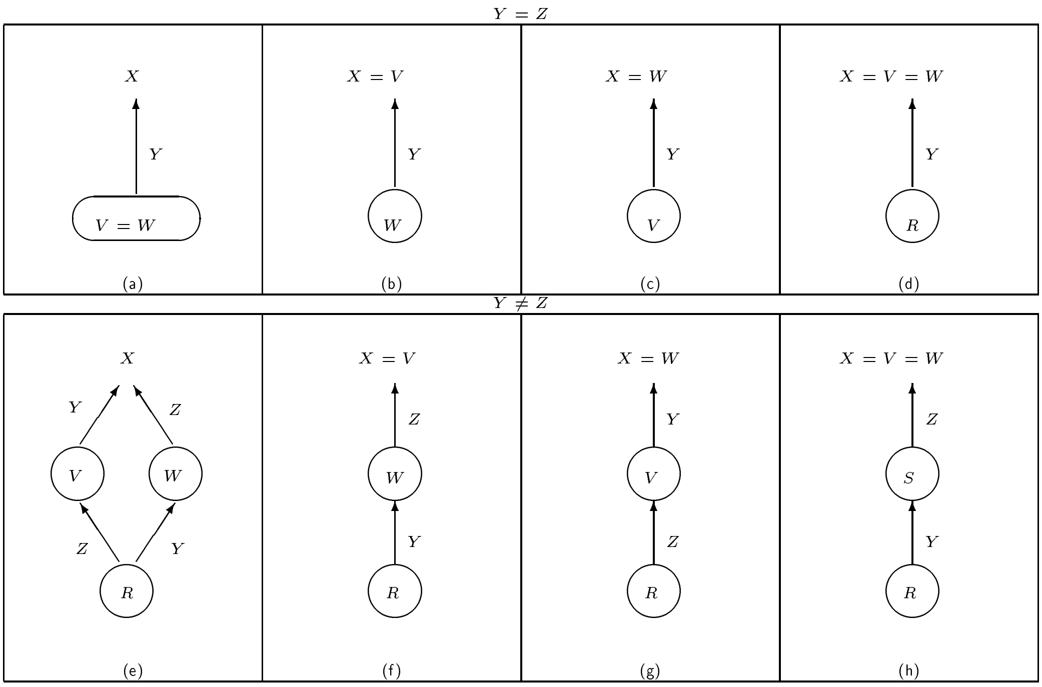


Figure 1: Solutions for $\{X = V \texttt{ with } Y, X = W \texttt{ with } Z\}$

To prove part (2), it remains to be seen that for each $i \in \{\mathsf{a}, \ldots, \mathsf{h}\}$, if θ is a solution of $\langle \mathcal{E}_i, \mathcal{C} \rangle$, then $\theta|_{vars(\mathcal{E})}$ is a solution of $\langle \mathcal{E}, \mathcal{C} \rangle$. Let us analyze each single action of [STD]:

a. Let θ be a solution of $\mathcal{E}_{\mathsf{a}}$. If the constraint $Z \notin W$ is fulfilled, then θ is represented by diagram (a) (i.e., it is a solution of $\mathcal{E}$). If on the contrary, θ forces $Z \in W$ to hold, then W is forced to be equal to X: we are in the situation described by diagram (d).

b. If θ is a solution of $\mathcal{E}_{\mathsf{b}}$ fulfilling $Z \notin W$, then θ is depicted by diagram (b). If θ forces $Z \in W$, then we are again in the situation described by diagram (d).

c. The situation is similar to the previous one, with the role of (b) taken by (c).

d. A solution θ of $\langle \mathcal{E}_{\mathsf{d}}, \mathcal{C} \rangle$ is clearly depicted by diagram (d) with N mapped to R. Assume that θ forces $Z \in N$. Then $N = W = V = X$: we are again in the situation (d) and R now represents the hyperset obtained from X by removing Z.

e. This is the most complex case. Let θ be a solution of $\mathcal{E}_{\mathsf{e}}$.

- If θ forces $Y = Z$, then we are in the situation of diagram (d).
- Assume θ forces $Y \neq Z$. Four cases are possible:
  - θ forces $Y \in N, Z \in N$: this implies $N = V = W = X$. We are in diagram (h).
  - θ forces $Y \notin N, Z \in N$: this implies that $N = V \neq X = W$. We are in diagram (g).
  - θ forces $Y \in N, Z \notin N$. Similarly, we are in diagram (f).
  - θ forces $Y \notin N, Z \notin N$: this implies that $V \neq N$, $W \neq N$ and $Y \notin V$ and $Z \notin W$. As a matter of fact, since $Y \neq Z$ and $Y \notin N, Z \notin N$, then X and N must differ by two elements. If it were, for instance, that $Y \in V$, then X and N would differ by a unique element: a contradiction. We are exactly in the situation of diagram (e).

f. If θ forces $Y = Z$, then the situation is depicted by diagram (d). If $Y \neq Z$ and $Z \notin W$, we are in the situation (f). Finally, if $Y \neq Z$ and $Z \in W$ we are in the situation (h).

g. The situation is similar to the previous one, with the role of (f) taken by (g).

h. If θ forces $Y = Z$ we are in the situation (d). If $Y \neq Z$, diagram (h) depicts the solution.

The overall schema is:

Diagram	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
Action(s)	a	b	c	a,b,c,d,e,f,g,h	e	e,f	e,g	e,f,g,h

which ensures correctness and completeness of rule [STD]. □

As already observed, constraints are introduced to ensure termination by avoiding that the same (or an equivalent) variable is repeatedly inserted as an element of the same hyperset. Therefore, we can simply remove them without affecting correctness and completeness of [STD].

However, not taking constraints into account may cause many *redundant solutions* to be generated by the algorithm. For example, by applying action [STD] to the system $\{X = V \texttt{ with } Y, X = W \texttt{ with } Z, Y = a, Z = a\}$ we get eight distinct systems in solvable form,[9] each one with its own constraints. By considering both equations and constraints, only four of them actually describe a solution of the given problem according to Definition 17 (the fact that Y and Z are equal, in fact, prevents the systems and constraints generated by actions [STD](e–h) to have a solution). On the other hand, by considering only equations (i.e., by discarding constraints) all the eight output systems have a solution, but four of them—those obtained by actions [STD](e–h)—are redundant (namely, they are less general than the other four).[10]

Taking constraints into account would require a more complete management of non-membership constraints. This could be done, for instance, by extending the procedures for testing set constraint satisfiability developed in [18] for the well-founded case, similarly to what has already done for set equality. However, this kind of non-trivial extension is outside the scope of this paper, and is left for future work.

[9] Namely, by restricting to variables V and W and using the non-flat form: (i) $V = W$; (ii) $V = W \texttt{ with } a$; (iii) $W = V \texttt{ with } a$; (iv) $V = N \texttt{ with } a, W = N \texttt{ with } a$; (v) $V = N \texttt{ with} a, W = N \texttt{ with} a$; (vi) $V = N \texttt{ with} a \texttt{ with} a, W = N \texttt{ with} a$; (vii) $V = N \texttt{ with} a, W = N \texttt{ with } a \texttt{ with } a$; $(viii)$ $V = N \texttt{ with } a \texttt{ with } a, W = N \texttt{ with } a \texttt{ with } a$.

[10] Strictly speaking, also the solution of system (iv) is redundant since it can be immediately obtained by that of system (i).

Also the whole problem of assuring minimality of the set of computed solutions is still an open problem, and we do not address it in this paper. Some results in the case of ordinary finite sets are presented in [2] and [35].

5.3 A hyperset unification algorithm

By using the rules introduced in the previous subsections we can finally define the complete hyperset unification algorithm.

In order to guarantee termination, however, some form of deterministic control in applying the rules needs also to be introduced. This is done by grouping the rules into actions, and splitting actions into two consecutive parts. *Actions of* PART I *are repeated as much as possible before executing actions of* PART II. Each action selects non-deterministically an equation e (or a simple constraint c) and applies to it one among a number of possible rules. The selected equation (resp., simple constraint) is one for which the condition part of at least one of the listed rules is satisfied.

Moreover, still for the sake of algorithm termination, the following rule is introduced to ensure that all the variables that are elements of a hyperset are from among the initial ones (i.e., no newly generated variable can occur as an element of a hyperset):

Variable Reordering [Re]

Let $\mathcal{E}$ be a Herbrand system and e be any equation occurring in $\mathcal{E}$.

<u>if</u> $e \equiv \big(X = Y\big)$ <u>and</u> Y (but not X) is a variable generated by rules [Cy] or [STD] <u>then</u> $\mathcal{E} := \mathcal{E} \setminus \{e\} \cup \{Y = X\}$.

The complete unification algorithm is shown in Fig. 2. It gets as its input a flat Herbrand system $\mathcal{E}$ along with a constraint $\mathcal{C}$ and it produces a finite set of Herbrand systems in solvable form $\mathcal{E}_1, \ldots, \mathcal{E}_n$.

<u>repeat</u>

<u>PART I</u>

<u>repeat</u>
select a simple constraint c in $\mathcal{C}$ enabling any of the rules
[Upd], [MeF], and apply one of them to c;

select an equation e in $\mathcal{E}$ enabling any of the rules
[EI], [VR'], [Re], [Cy], [ICy], and apply one of them to e;

until no rule applies or *failure*;

PART II
select an equation e in $\mathcal{E}$ enabling any of the rules
[TD], [TDF], [STD] and apply one of them to e;

until no rule applies or *failure*.

Figure 2: Hyperset unification algorithm

Example 21 *Let*

$$\{X = V \texttt{ with } X, X = W \texttt{ with } X, W = V \texttt{ with } X, V = \emptyset\}$$

be the initial system. By applying [STD](c) *to the first and second equation we get the new system* $\{X = V \texttt{ with } X, X = X, W = X, W = V \texttt{ with } X, V = \emptyset\}$ *(Notice that by using* [STD](a) *instead of* [STD](c) *we would obtain the new system* $\{X = V \texttt{ with } X, X = X, W = V, W = V \texttt{ with } X, V = \emptyset\}$ *and, from the third, fourth and fifth equations we would, after having replaced* W *with* V*, get a failure by* [TDF]*). Then, by applying* [EI] *and* [VR'] *we get the system* $\{X = V \texttt{ with } X, W = X, X = V \texttt{ with } X, V = \emptyset\}$*, and we can apply* [STD](a) *to the first and third equation, obtaining* $\{X = V \texttt{ with } X, X = X, V = V, W = X, V = \emptyset\}$*.*[11] *From the last system by* [EI] *finally we get the system in solvable form*

$$\{X = V \texttt{ with } X, W = X, V = \emptyset\}$$

whose solution is clearly the hyperset Ω*, i.e. the hyperset that has itself as its only element [1].*

The following lemma states that a system to which no rule applies is in solvable form.

Lemma 22 *Let* $\mathcal{E}$ *be a (flat) system to which no rule of the algorithm applies. Then* $\mathcal{E}$ *is in solvable form.*

[11] Note that we assume that our systems of equations and constraints are dealt with as multisets and, as such, they can contain duplicates.

Proof. Suppose $\mathcal{E}$ is not in solvable form. Then, according to Def. 11, either there is an equation in $\mathcal{E}$ of the form $X = Y$, with X occurring elsewhere in the system, and action [VR'] applies, or there is an equation of the form $X = f(t_1, \ldots, t_n)$, $f \in \Sigma$, with X occurring as left-hand side of some other equation in $\mathcal{E}$, and action [TD], [TDF] or [STD] applies. $\square$

Theorem 23 (Soundness and Completeness) *Assume that any non-deterministic branch of the computation of the algorithm terminates (possibly with failure) and let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be the returned solvable form systems. Then:*

$$Soln(\mathcal{E})|_{vars(\mathcal{E})} = \bigcup_{i \in \{1,\ldots,n\}} Soln(\mathcal{E}_i)|_{vars(\mathcal{E})}.$$

Proof. Each non-deterministic branch of the computation either terminates with failure (rules [TDF], [MeF]) or reaches a solvable form system. Since, by hypothesis, any computation is finite, then Lemmas 19 and 20 (part (1)) ensure that

$$Soln(\mathcal{E})|_{vars(\mathcal{E})} = \bigcup_{i \in \{1,\ldots,n\}} Soln(\mathcal{E}_i, \mathcal{C}_i)|_{vars(\mathcal{E})}.$$

To complete the proof, we only need to prove that if θ is a solution of $\mathcal{E}_i$ (but not necessarily of $\langle \mathcal{E}_i, \mathcal{C}_i \rangle$), then $\theta|_{vars(\mathcal{E})}$ is a solution of $\mathcal{E}$. This follows from part (2) of Lemmas 19 and 20, since, by hypothesis, action [MeF] (the unique rule that does not fulfill such part) has not affected such branch of the computation. $\square$

The hypothesis that "any non-deterministic branch of the computation of the algorithm terminates (possibly with failure)" is always fulfilled, as it will be shown in Section 5.4. Thus:

Corollary 24 *Let $\mathcal{E}_1, \ldots, \mathcal{E}_n$ be the solvable form systems returned by a non-deterministic computation of the algorithm. Then*

$$Soln(\mathcal{E})|_{vars(\mathcal{E})} = \bigcup_{i \in \{1,\ldots,n\}} Soln(\mathcal{E}_i)|_{vars(\mathcal{E})} .$$

To end this section, we want to point out that the algorithm of Fig. 2 can also be successfully employed as an NP-unification algorithm for well-founded hybrid sets simply by adding an occur-check test on the output systems at the end of the computation.

5.4 Termination

The idea of the termination proof is the following:

- First we prove that between two consecutive executions of an action in PART II only a finite number of actions in PART I can be performed (Lemma 27). Note that the "cleanup" actions in PART I are always applied as far as possible, in a loop. When PART II is entered, at most one action is performed; PART I is then entered again.

- Then we prove, by means of a complexity measure, that in every non–failing execution of the algorithm, only a finite number of actions [STD](a–h) are performed. It follows that every execution of the algorithm terminates (Theorem 34). In the next section we will also give a polynomial bound on the number of executions of PART II, that will be used in the complexity analysis of the algorithm.

The following two facts are the key point in proving that every branch in the search tree of the algorithm is finite:

- The only rules that introduce new variables are rules [Cy] and [STD](d–h).

- A new variable N is always introduced on the left-hand side of a `with` operator. The only way it can be extracted is by means of an application of rule [STD](a-h), so becoming one of the variables appearing in an equation of the form $Y = N$ or $N = Y$. Since the rules in PART I have precedence over those in PART II, an application of [Re] is mandatory in the first case. The result is that the new variable cannot be further propagated. As a consequence we have that: *After the cleanup phase* PART I *has been performed, any variable Y that occurs on the right-hand side of a* `with` *operator symbol inside $\mathcal{E}$ is one of the variables occurring in the original system. This means that only the variables already present in the initial system can represent elements of a set. The new variables generated by the execution of the algorithm can only denote subsets of already existing sets. Moreover, in a simple constraint $Y \notin V$ the variable Y must be one already present in the initial system.*

Definition 25 *Given a* flat *Herbrand system $\mathcal{E}$, let s denote the number of occurrences of function symbols in $\mathcal{E}$, and $\ell(\mathcal{E})$ denote the number of* distinct *variables occurring in the right-hand side of a* `with` *operator in some equation $X = V\,\mathtt{with}\,Y$ of $\mathcal{E}$ (i.e., the number of distinct variables used as "elements" in $\mathcal{E}$). We denote with ℓ the number $\ell(\mathcal{E})$, for the initial system $\mathcal{E}$.*

For what observed just before definition 25, it holds that for each system $\mathcal{E}$ generated during a computation of the algorithm, $\ell(\mathcal{E}) \leq \ell$.

Lemma 26 *The* total *number of applications of rule* [Cy] *is bounded by s, i.e., by the number of occurrences of function symbols in the* initial system*. The same holds for rule* [ICy]*.*

Proof. Equations of the form $X = X\,\mathtt{with}\,Y$ appear in $\mathcal{E}$ either because they were already present in the initial system, or because they have been obtained from a chain of substitutions applied to another equation $V_1 = V_2\,\mathtt{with}\,V_3$. However, if the equation $V_1 = V_2\,\mathtt{with}\,V_3$ was not present in the initial system, then it must necessarily have been introduced in $\mathcal{E}$ by either rule [Cy] or [STD](d–h). In both cases V_2 is a newly generated variable, hence the constraint $V_3 \notin V_2$ is in $\mathcal{C}$.

The subsequent applications of rule [VR'] which replace V_1 and V_2 by X, and V_3 by Y (both in $\mathcal{E}$ and in $\mathcal{C}$) cause rule [MeF] to detect a *failure.*

An application of action [ICy] generates a set of new equations $\{X_1 = X_0, X_2 = X_0, \ldots, X_n = X_0\}$. Since the rules in PART I have precedence, before going to any application of a rule in PART II, action [VR'] must be applied to the new equations and to the equations that constitute the cycle, transforming them into equations to which rule [Cy] applies. Hence, the total number of applications of [ICy] is bounded by the number of applications of [Cy], hence again by s. $\square$

Because of the previous lemma, we will not take into account actions [Cy] and [ICy] in what follows.

Moreover, we can observe that rules dealing with free functors, i.e. functors different from `with`, namely [TD] and [TDF], cause failure or decrease the number of occurrences of free functors in $\mathcal{E}$. No other rule can increase this number. Hence, the number of applications of rules [TD] and [TDF] is bounded by the number of occurrences of free functors in the initial system, hence by s. *For such a reason we will assume in the sequel that the only function symbol occurring in $\mathcal{E}$ is* `with`*.*

Lemma 27 *Between two consecutive executions of an action in* PART II *only a finite number of actions in* PART I *can be performed.*

Proof. Every action among [VR'], [EI], and [Re] uses an equation of the form $X = Y$ only once. The only action in PART I that can generate new equations of this form is [ICy], and their (total) number is polynomially bounded by s. Therefore, the number of applications of [VR'], [EI], and [Re] in an execution of PART I is bounded by the number of equations $X = Y$ in the current system plus s.

Every application of rule [Upd] uses a `with` equation and a simple constraint. No action in PART I can increase the number of `with` equations, and the only rules that can generate new constraints are [Cy] and [VR']. The total number of applications of [Cy] is bounded by s, and the number of applications of [VR'] is bounded by the number of equations $X = Y$ in the current system plus s. Therefore, the same holds for [Upd]. □

Definition 28 *For each equation $X = V \,\mathtt{with}\, Y$ of a system $\mathcal{E}$ occurring in a computation, we define:*

$$\mathit{weight}(X = V \,\mathtt{with}\, Y) = |\{Z : Z \notin V \mathit{\ in\ } \mathcal{C}\}| .$$[12]

Intuitively, if $\mathit{weight}(X = V \,\mathtt{with}\, Y) = k$, then we know that k (among the possible ℓ) elements cannot be inserted into the set represented by the variable V. To be clear, an "insertion" of the elements $Y_1, Y_2, \ldots$ in V is a chain of equations of the form:

$$V = X_1 \,\mathtt{with}\, Y_1, X_1 = X_2 \,\mathtt{with}\, Y_2, \ldots$$

If an insertion of one of the k elements in $\{Z : Z \notin V \mathit{\ in\ } \mathcal{C}\}$ in V happens, then immediately PART I will detect failure (actions [Upd] and [MeF]). Since at most ℓ elements (see Def. 25), represented by variables, can be introduced in sets, k cannot exceed ℓ. More precisely,

Lemma 29 *If $X = V \,\mathtt{with}\, Y$ belongs to $\mathcal{E}$ during the computation, then $\mathit{weight}(X = V \,\mathtt{with}\, Y) \leq \ell$.*

[12] Here, $|\cdot|$ denotes the cardinality of a set.

Now we want to associate to every system $\mathcal{E}$ a complexity measure $W(\mathcal{E})$, and show that every application of an action [STD](a–h) causes $W(\mathcal{E})$ to strictly decrease, and that no other action of the algorithm can increase it.

Definition 30 *Given a (flat) system $\mathcal{E}$ occurring in a computation, define*

$$W(\mathcal{E}) = \langle \ell(\mathcal{E}), x_0, \dots, x_\ell \rangle ,$$

where x_i denotes the number of equations of weight $= i$ in $\mathcal{E}$ (and i is bounded by ℓ because of Lemma 29). The ordering relation between complexity tuples is the (well founded) lexicographical ordering.

Notice that if an equation in $\mathcal{E}$ is replaced by one of greater weight, and no other equation changes its weight, then $W(\mathcal{E})$ decreases.

Lemma 31 *If a simple constraint $Y \notin V$ is in $\mathcal{C}$, then Y occurs in the right-hand side of a* `with` *operator in some equation $K = H \,\texttt{with}\, Y$ in $\mathcal{E}$.*

Proof. Constraints like $Y \notin V$ are not present in the initial system, they are introduced by rules [STD](a–h) (and [Cy], which we do not consider here) from an equation $K = H \,\texttt{with}\, Y$. Such equation can be deleted by a subsequent application of [STD](a–h), but whenever this happens, either a new equation $K = N \,\texttt{with}\, Y$ is created (actions [STD](d–h)) or an identification between variables is imposed, so that, after substitution, Y still appears on the right-hand side of a `with` operator (actions [STD](a–c)). □

Lemma 32 *No action in* PART I *can increase $W(\mathcal{E})$.*

Proof. This is clear for [MeF], [El], and [Re]. Action [VR'] can affect $W(\mathcal{E})$ only if it identifies two variables Y and Z occurring in constraints $Y \notin V$, $Z \notin V$. But in this case both Y and Z occur among the (initial) ℓ used as elements (Lemma 31), hence $\ell(\mathcal{E})$ is decreased by 1.

Action [Upd] clearly tends to increase the weight of equations, thus making $W(\mathcal{E})$ decrease. □

Lemma 33 *Every application of an action* [STD](a–h) *either causes $W(\mathcal{E})$ to strictly decrease, or forces immediate termination with failure.*

Proof. Every action [STD](a–h) eliminates at least one equation. Some of them introduce equations among variables (e.g., $Y = Z$). We have already proved that such equations do not cause the complexity to increase. Thus, for actions [STD](a–c), that do not introduce equations of the form $V_1 = V_2 \texttt{ with } V_3$, the claim is trivially true. It remains to be seen that when new equations of this form are added (actions [STD](d–h)), their weights become strictly greater than those of the eliminated ones, after subsequent applications of [Upd], unless immediate failure is caused by [MeF] (this is the case, for instance, when $Z \notin V$ and $Y \notin V$ are in $\mathcal{C}$ and we insert the equation $V = N \texttt{ with } Z$). This is a direct consequence of the updating of the constraint $\mathcal{C}$ associated with any action. □

Theorem 34 *Every execution of the algorithm terminates.*

Proof. It follows from Lemmas 27 and 33. □

6 Complexity

In this section we prove that the hyperset unification problem is NP-complete. To achieve this result we need a refinement of the termination results that have been established in Section 5.4. The upper bounds we compute are very rough in order to keep proofs as simple as possible.

Definition 35 *Given a flat Herbrand system $\mathcal{E}$, we denote by:*

- *$s(\mathcal{E})$ the number of occurrences of* `with` *in $\mathcal{E}$,*
- *$\ell(\mathcal{E})$ the number of distinct variables used as "elements" in $\mathcal{E}$ (see Def. 25),*
- *$v(\mathcal{E})$ the number of distinct variables in $\mathcal{E}$,*
- *$\#(\mathcal{E})$ the number of equations of $\mathcal{E}$.*

Moreover, s, ℓ, v, and $\#$ are used to denote the corresponding values for the initial system $\mathcal{E}$.

Remark 36 *The flat form of the system and the assumption that the only function symbol occurring in $\mathcal{E}$ is* `with` *ensure that $\ell(\mathcal{E}) \leq s(\mathcal{E}) \leq \#(\mathcal{E})$, and $v(\mathcal{E}) \leq 3\#(\mathcal{E})$.*

By a *step* we mean the execution of PART II of the algorithm, i.e. the execution of one action among [TD], [TDF], and [STD].

Lemma 37 *Suppose that in a branch of a non-deterministic computation of the algorithm, the total number of executions of actions* [STD](e–h) *of* PART II *is equal to* k*. Then such branch contains at most* $O(k(s+k))$ *steps.*

Proof. We split the sequence of steps performed by the algorithm into $k+1$ subsequent stages determined by the k applications of actions [STD](e–h).

The first stage is based on a system containing s occurrences of `with`, and since every execution of actions [STD](a–d) decreases the number of occurrences of `with` by one, it can contain no more than s steps. After the application of an action [STD](e–h) s can increase at most by one; hence the second stage can contain no more than $s+1$ steps. The total number of steps is therefore not greater than $\sum_{i=0}^{k}(s+i) \leq (k+1)(s+k)$. □

The following lemma refines Lemma 27, used to prove termination.

Lemma 38 *Let* $\mathcal{E}$ *be the system obtained after the i-th execution of* PART II*. Then, in the subsequent application of actions in* PART I*, no more than* $(4\#(\mathcal{E})+1)^2$ *actions can be executed.*

Proof. First we analyze the contribution of the various actions to the total number; then we find a simple bound for it.

1 Action [MeF] can be performed at most one time, causing immediate termination with failure.

$s(\mathcal{E})$ According to Lemma 26, actions [Cy] and [ICy] can be successfully performed at most s times in a non-deterministic branch of the computation. Here we do not distinguish between failing and successful computations, hence the two actions can be performed $s(\mathcal{E})$ times. Therefore both the number of equations and the number of variables of $\mathcal{E}$ can be increased by $s(\mathcal{E})$.

$\#(\mathcal{E})+s(\mathcal{E})$ Action [Re] can be performed no more than once for each equation; thus the number of executions of [Re] is bounded by the number of equations $\#(\mathcal{E})$, possibly increased by $s(\mathcal{E})$.

$\#(\mathcal{E}) + s(\mathcal{E})$ Actions [El] and [VR'] can be performed no more than once for each equation. Since they are mutually exclusive, the total number of their executions is bounded by the number of equations, possibly increased by $s(\mathcal{E})$.

$(v(\mathcal{E}) + s(\mathcal{E}))^2$ Constraint updating (action [Upd]) introduces new constraints of the form $Y \notin X$ between already existing variables X and Y. Hence it cannot be performed more than $(v(\mathcal{E})$ times (possibly increased by $s(\mathcal{E})$) to the square (actually, this number will be smaller, in general, than $(v(\mathcal{E}) + s(\mathcal{E}))^2$ because Y is always one among the ℓ initial variables).

Therefore, the total number of executions of actions in PART I is bounded by:

$$1 + 2\#(\mathcal{E}) + 3s(\mathcal{E}) + (v(\mathcal{E}) + s(\mathcal{E}))^2.$$

The claim now follows from the bounds, $s(\mathcal{E}) \leq \#(\mathcal{E})$, and $v(\mathcal{E}) \leq 3\#(\mathcal{E})$ (c.f. Remark 36). $\square$

Lemma 39 *After the execution of i steps, it holds that $\#(\mathcal{E}) \leq 2\# + 4i$.*

Proof. We prove that $\#(\mathcal{E}) \leq \# + s + 4i$; the result then follows from the inequality $s \leq \#$ (c.f. Remark 36). An action inPART II of the algorithm can increase the number of equations at most by two (actions STD(d) and STD(h)). This can happen in any step, so $2i$ must be added to the initial $\#$.

As can be realized from the proof of Lemma 38, the only actions in PART I that increase the number of equations are [Cy] and [ICy]. They can use initial `with` –equations (s) or new equations introduced by PART II. An action in PART II introduces at most two equations, so we again add $2i$ to the global sum. $\square$

Theorem 40 *Let k be the total number of executions of actions* [STD](e–h) *in a branch of a non-deterministic computation of the algorithm. Then the total number of actions executed in this branch is $O(k^6\#^3)$.*

Proof. From Lemma 37, we know that at most $O(k(s+k))$ steps are performed, where $s \leq \#$ (c.f. Remark 36). From Lemmas 38 and 39 we know that the number of performed actions after the completion of the i-th step depends only on the number of equations at the end of the i-th step, and that this number is a function of the initial number of variables and i. We need to add1 to this number to keep track of the execution of PART II. The total number of actions will be:

$$
\begin{array}{rcl}
O\left(\displaystyle\sum_{i=0}^{k(\#+k)}((4(2\#+4i)+1)^2+1)\right) & = & \\
O\left(\displaystyle\sum_{i=0}^{k(\#+k)}(\#^2+i^2+\#i)\right) & = & \\
O(\#^2k(\#+k)+k^3(\#+k)^3+\#k^2(\#+k)^2) & = & O(k^6\#^3).
\end{array}
$$

□

As a consequence of this result, we need only to find a polynomial bound for the number k of executions of actions [STD](e–h) in PART II.

The following concepts will be useful in the sequel for having a graphical counterpart to the notions of equation generation and weight introduced in the previous sections.

To a flat Herbrand system $\mathcal{E}$ can be associated a graph having as *nodes* the variables occurring as X and V in an equation of the form $X = V\,\mathtt{with}\,Y$ and, for each such equation, a labeled *edge* $X \stackrel{Y}{\leftarrow} V$. The *weight* of an edge is the weight of the corresponding equation (see Def. 28). Weights induce a (pre)ordering relation on the edges in the graph: an edge $X \stackrel{Y}{\leftarrow} V$ is of level i if $weight(X = V\,\mathtt{with}\,Y) = i$.

As an example, an application of action [STD](e) can be graphically represented by:

$$
\begin{array}{ccccccccc}
 & & V & & & & V & & \\
 & \stackrel{Y}{\swarrow} & & & & & & \stackrel{Z}{\nwarrow} & \\
X & & & \Rightarrow & X & & & & N \\
 & \stackrel{Z}{\nwarrow} & & & & \stackrel{Z}{\nwarrow} & & \stackrel{Y}{\swarrow} & \\
 & & W & & & & W & &
\end{array}
$$

The edges $V \stackrel{Z}{\leftarrow} N$ and $W \stackrel{Y}{\leftarrow} N$ are said to be *incoming edges* for the nodes V and W, respectively.

A successful non-deterministic computation starts with a system $\mathcal{E}_0 = \mathcal{E}$ and ends with a pair $\mathcal{E}_n, \mathcal{C}_n$, where $\mathcal{E}_n$ is a system of equations and $\mathcal{C}_n$ is a constraint. We have already noticed (Lemmas 32 and 33) that $\ell(\mathcal{E}_n) \leq$

$\ell(\mathcal{E}_0) = \ell$. When a new edge $V \overset{Y}{\leftarrow} N$ is introduced by actions $\mathsf{[STD](e–h)}$ in an intermediate system $\mathcal{E}_i$, a weight (implicitly stored in $\mathcal{C}_i$) is also associated with it. If $\ell(\mathcal{E}_n) < \ell(\mathcal{E}_i)$ than it is possible that $weight(V \overset{Y}{\leftarrow} N)$ is decreased in the remaining part of the computation; moreover, the edge itself could be subsequently deleted by an application of $\mathsf{[STD]}$.

For having a global level measure, we assume here that the algorithm keeps track of the deleted edges in an auxiliary variable $\mathcal{E}'$ and that $\mathsf{[VR']}$ also updates $\mathcal{E}'$.

Definition 41 *The* global weight *of an edge e introduced in $\mathcal{E}$ along the computation is the weight of its final descendant (i.e. the equation eventually obtained after (possibly) renaming of the variables of e) computed using the final systems $\mathcal{E}_n, \mathcal{E}'_n$, and constraint $\mathcal{C}_n$. Moreover, for a node (a variable) N, we define its* global weight *as $|\{Z : Z \notin N \text{ in } \mathcal{C}_n\}|$.*

To simplify the notation in what follows, the global weight is simply denoted by weight.

$\beta(i)$ represents the number of edges $V \overset{Y}{\leftarrow} N$ such that $weight(V) = i$ and $weight(N) = i + 1$, and N has been introduced by actions $\mathsf{[STD](e–h)}$. *Let $\bar{\beta} =_{Def} \sum_{i=0}^{\infty} \beta(i)$. Because of Lemma 29 we have that $\bar{\beta} = \sum_{i=0}^{\ell} \beta(i)$.*

Remark 42 *Since the generation of a new variable requires the creation of at least one edge, it is clear that the number of variables generated by actions* $\mathsf{[STD](e–h)}$ *is not greater than $\bar{\beta}$.*

Therefore, in order to show polynomiality we have only to work out a suitable upper bound for $\bar{\beta}$.

Theorem 43 *The number $\bar{\beta}$ of new variables generated in a non-deterministic computation of the algorithm is $O((s + \ell + 1)^3)$.*

Proof. Since every edge requires an occurrence of `with` , we have that $\beta(0) \leq s$. Moreover, an edge of $weight = 1$ requires a pair of edges of $weight = 0$. So we have that $\beta(1) \leq 2s$.

Assume for now that all the generated variables remain distinct during the computation. To fire action $\mathsf{[STD]}$ two edges incoming in the same node must be in $\mathcal{E}$. Therefore, the assumption that a new generated variable N does not unify with any other variable N' allows to say that for any generated variable N, at least one incoming edge $N \overset{Y}{\leftarrow} N''$ remains in $\mathcal{E}$ for the whole computation. Thus, for $j > 2$, $\frac{\beta(j-2)}{2}$ edges cannot be used to fire again action [STD]. In view of this observation, we can write the recursive rule:

$$\beta(j) \leq 2\left(\beta(j-1) - \frac{\beta(j-2)}{2}\right),$$

and, by induction on $j \geq 2$:

$$\beta(j) \leq 2js - (j-1)s \leq s(j+1).$$

Therefore,

$$\bar{\beta} = \sum_{i=0}^{\ell} \beta(i) \leq (s+\ell+1)^3.$$

Suppose now that a newly generated variable N is identified with a variable A, and that N belongs to an edge $V \overset{Z}{\leftarrow} N$, with the constraint $Z \notin N$ in $\mathcal{C}$. A simple inspection of actions [STD](a–h) shows that the only way this can happen (without encountering a failure situation) is an application of action [STD](a) to a pair of edges $V \overset{Z}{\leftarrow} N$ and $V \overset{Z'}{\leftarrow} A$. This means that the two edges $V \overset{Z}{\leftarrow} N$ and $V \overset{Z'}{\leftarrow} A$ cannot be combined together anymore in order to produce $N \overset{Z'}{\leftarrow} N'$ and $A \overset{Z}{\leftarrow} N'$ using actions [STD](d–h).

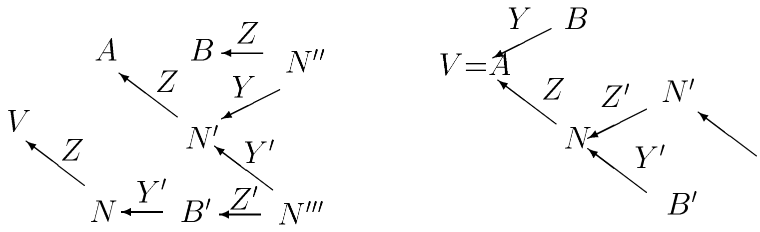
Assume A and N have at least one incoming edge:

$$\begin{tikzcd}
& A \arrow[dl, "Z'"'] & B \arrow[l, "Y"'] \\
V & & \\
& N \arrow[ul, "Z"] & B' \arrow[l, "Y'"']
\end{tikzcd}$$

If A and N are identified, then the two edges $A \overset{Y}{\leftarrow} B$ and $N \overset{Y'}{\leftarrow} B'$ can fire actions [STD](d–h). The maximum number of new edges is generated when action [STD](e) is applied, and the following situation is reached:

$$\begin{tikzcd}
& & B & \\
V & A{=}N \arrow[l, "Z{=}Z'"'] & & N' \arrow[ul, "Y'"'] \arrow[dl, "Y"] \\
& & B' \arrow[ul, "Y'"'] &
\end{tikzcd}$$

If A and N are not identified, only actions [STD](e) and [STD](f) can be applied (because of the constraint $Z \notin N$), and the following two situations are obtained:



In both cases, we have that the number of new variable generations is greater than in the case A and N are identified. $\square$

Corollary 44 (Polynomiality) *Every branch of a non-deterministic execution of the algorithm contains a number of executions of elementary actions that is polynomially bounded by # (the number of equations in the initial system).*

Proof. From Theorem 43 we know that the number of variables generated by actions [STD](e–h) (hence the number k of executions of such actions) is $O((s+\ell+1)^3)$, where $s \leq \#$ and $\ell \leq \#$ (c.f. Remark 36). The claim follows from Theorem 40. $\square$

In [18] we have proved the NP-hardness of the set unification problem by showing that the *3-SAT* problem can be reduced to a set unification problem. For instance, given the formula $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) = \texttt{true}$ it can be represented as the following unification problem:

$$\{\{X_1, Y_1\}, \{X_2, Y_2\}, \{X_3, Y_3\}, \{X_1, Y_2, Y_3, 0\}, \{Y_1, X_2, X_3, 0\}\} = \{\{0, 1\}\}$$

where 0 and 1 are constant terms or they are (set-theoretically built) as $\emptyset$ and $\{\emptyset\}$, respectively. If this problem admits a solution, then the solution is well-founded.

Since a hyperset unification algorithm must also be able to find well-founded solutions, this implies the NP-hardness also for the more general non-well-founded unification problem. Thus, the result of Corollary 44 is a proof of the *NP-completeness* of the hybrid hyperset unification problem.

7 Conclusions

We have presented a non-deterministic algorithm for extended unification incorporating the axioms of a simple theory of hybrid hypersets.

To this purpose we have introduced hypersets as directed labeled graphs and defined a notion of bisimulation which applies to this kind of graphs. The interpretation domain has been taken as the set of directed labeled graphs over the signature Σ modulo the equivalence relation induced by bisimulation. Elements of this domain, i.e. hypersets, can be obtained as solutions of Herbrand systems of equations, and the notion of Herbrand systems in solvable form has been defined as a convenient way to represent solutions.

An hyperset unification algorithm working on Herbrand systems of equations has been defined. It basically extends the set unification algorithm described in [18] by avoiding full variable substitution and adding simple non-membership constraints to guarantee termination.

Soundness and completeness of the algorithm, as well as the crucial issue of termination and the algorithm complexity, have been formally analyzed in detail.

The minimality problem (i.e., assuring that the set of computed solution is minimal), instead, has been left open for future research.

Acknowledgements

We are much obliged to Eugenio Omodeo and Alberto Policriti, whose qualified contribution emerges from a large part of the paper. Our referees suggested several substantial improvements to the paper. We really wish to thank them.

Acknowledgements of grants

Investigation supported by University of Bologna. Funds for selected research topics. The work has been partially supported by the MURST project: *Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software.*

References

[1] P. Aczel. *Non-well-founded sets.*, volume 14 of *Lecture Notes, Center for the Study of Language and Information.* Stanford, 1988.

[2] P. Arenas-Sánchez and A. Dovier. A minimality study for set unification. *Journal of Functional and Logic Programming*, 1997(7), December 1997.

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

[4] F. Baader and K. U. Schulz. On the Combination of Symbolic Constraints, Solution Domains, and Constraint Solvers. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP95*, volume 976 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1995.

[5] F. Baader and K. U. Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *Journal of Symbolic Computation*, 21(2):211–243, 1996.

[6] J. Barwise. Situations, Sets and the Axiom of Foundation. In J. B. Paris, A. J. Wilkie, and G. M. Wilmers, editors, *Logic Colloquium '84*, pages 21–36. North Holland, Amsterdam, 1986.

[7] J. Barwise and J. Etchemendy. *The Liar: An Essay on Truth and Circularity*. Oxford University Press, 1987.

[8] J. Barwise and L. Moss. *Vicious Circles. On the Mathematics of non-well-founded phenomena*. CSLI, Stanford, 1986.

[9] J. Barwise and L. Moss. Hypersets. *The Mathematical Intelligencer*, 13(4):31–41, 1991.

[10] J. Barwise and L. Moss. Applying AFA. Notes for a Tutorial on Non-Wellfounded Sets, 1993.

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

[12] W. Büttner. Unification in the Data Structure Sets. In J. K. Siekmann, editor, *Proceedings of the Eight International Conference on Automated Deduction*, volume 230, pages 470–488. Springer-Verlag, Berlin, 1986.

[13] D. Cantone, A. Ferro, and E. G. Omodeo. *Computable Set Theory, Vol. 1*. International Series of Monographs on Computer Science. Clarendon Press, Oxford, 1989.

[14] A. Colmeraurer. Prolog and Infinite Trees. In K. L. Clark and S-A. Tarnlund, editors, *Logic Programming*, pages 231–251. Academic Press, New York, 1982.

[15] A. Dovier. *Computable Set Theory and Logic Programming*. PhD thesis, Università degli Studi di Pisa, March 1996. TD–1/96.

[16] A. Dovier, E. G. Omodeo, and A. Policriti. Hyperset constraint handling. Rr 21/94, Dipartimento di Matematica ed Informatica, Univ. di Udine, December 1994. To appear in in Applicable Algebra in Engineering, Communication and Computing.

[17] A. Dovier, E. G. Omodeo, A. Policriti, and G. Rossi. Solving Systems of Equations over Hypersets. In M. Alpuente, R. Barbuti, and I. Ramos, editors, *GULP–PRODE'94 1994 Joint Conference on Declarative Programming*, pages 403–417, 1994.

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

[19] A. Dovier, A. Policriti, and G. Rossi. Integrating lists, multisets, and sets in a logic programming framework. In F. Baader and K. U. Schulz, editors, *First International Workshop on Frontiers of Combining Systems*, volume 3 of *Applied Logic*, pages 213–229. Kluwer Academic Publishers, 1996. March 26–29, Munich.

[20] F. Fages and G. Huet. Complete sets of unifiers and matchers in equational theories. *Theoretical Computer Science*, 43:189–200, 1986.

[21] M. Forti and F. Honsell. Set theory with free construction principles. Technical report, Annali Scuola Normale Superiore, Pisa, Classe di Scienze, 1883.

[22] P. M. Hill and J. W. Lloyd. The Gödel report. Technical Report TR-91-02, Computer Science Department, University of Bristol, 1991.

[23] J. Jaffar. Efficient Unification over Infinite Terms. *New Generation Computing*, 2:207–219, 1984.

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

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

[26] J. L. Lassez, M. J. Maher, and K. Marriot. Unification revisited. In *Lecture Notes in Computer Science*, volume 306, 1986.

[27] A. P. Lisitsa and V. Yu. Sazonov. Bounded hyperset theory and web-like data bases. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory, 5th Kurt Gödel Colloquium*, volume 1289 of *Lecture Notes in Computer Science*, pages 172–185. Springer-Verlag, Berlin, 1997.

[28] M. Livesey and J. Siekmann. Unification of Sets and Multisets. Technical report, Institut für Informatik I, Universität Karlsruhe, 1976.

[29] M. J. Maher. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In *Proceedings of 3rd Symposium Logic in Computer Science*, pages 349–357, Edinburgh, 1988.

[30] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[31] A. Martelli and G. Rossi. Efficient Unification with Infinite Terms in Logic Programming. In *Proceedings of FGCS'84: International Conference on Fifth Generation Computer Systems*, 1984.

[32] A. Martelli and G. Rossi. Stepwise Development of an Algorithm for Unification over Infinite Terms. *Computers and Artificial Intelligence*, 9(3):209–239, 1990.

[33] R. Milner and M. Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87:209–220, 1991.

[34] J. H. Siekmann. Unification theory. In C. Kirchner, editor, *Unification*. Academic Press, 1990.

[35] F. Stolzenburg. Membership-Constraint and Complexity in Logic Programming with Sets. In F. Baader and K. U. Schulz, editors, *First International Workshop on Frontiers of Combining Systems*, volume 3 of *Applied Logic*, pages 285–302. Kluwer Academic Publishers, 1996. March 26–29, Munich.

[36] A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, 1986.