AAECC **9**, 293–332 (1999)

AAECC
Applicable Algebra in Engineering, Communication and Computing

Solvable Set/Hyperset Contexts: II. A Goal-Driven Unification Algorithm for the Blended Case

Agostino Dovier,[1] Eugenio G. Omodeo[2], Alberto Policriti[3]

[1]Università di Verona, Dipartimento Scientifico-Tecnologico, Strada Le Grazie 3, I-37134 Verona (e-mail: dovier@sci.univr.it)
[2]Università di L'Aquila, Dipartimento di Matematica Pura ed Applicata, Via Vetoio Loc. Coppito, I-67100 L'Aquila (e-mail: omodeo@univaq.it)
[3]Università di Udine, Dipartimento di Matematica e Informatica. Via delle Scienze 208, I-33100 Udine (e-mail: policrit@dimi.uniud.it)

Received: March 4, 1996; revised version: August 24, 1998

Abstract. A universe composed by rational ground terms is characterized, both constructively and axiomatically, where the interpreted construct $with$, which designates the operation of adjoining one element to a set, coexists with free Herbrand functors. Ordinary syntactic equivalence must be superseded by a bisimilarity relation $\approx$, between trees labeled over a signature, that suitably reflects the semantics of $with$. Membership (definable as "$d \in t =_{\mathrm{Def}} (t\ with\ d) \approx t$") meets the non-well-foundedness property characteristic of *hyperset theory*. A goal-driven algorithm for solving the corresponding *unification problem* is provided, it is proved to be totally correct, and exploited to show that the problem itself is NP-complete. The results are then extended to the treatment of the operator $less$, designating the one-element removal operation. Applications to the automaton matching and type-finding problems are illustrated.

Keywords: Semantic unification, Bisimulations, Hypersets, Set theory, NP-completeness

Introduction

Graphs are widely used among basic data structures for representing both terms and sets (cf. [9, 1]). Such representation is particularly convenient and useful for algorithmic manipulations in fields like declarative programming, semantics of concurrency, and automated deduction. In view of applications in these and other fields, it is rather natural to assume the finiteness of graphs that are

to represent terms or sets; however, it is often convenient not to impose the additional constraint of acyclicity. By resorting to cyclic representations one is in fact enabled, to a limited but significant extent, to treat infinite terms (cf. [23]) and non-well-founded sets (cf. [1]).

Well-established applications of the latter, today often called *hypersets*, regard the handling of automata, and the modeling of linguistic self-referential situations [7]. Supposedly, many more applications will be found in the modeling of shared information systems: web-like databases [22], query languages for semi-structured information [11], etc. Even though issues of computational complexity still set a limit to such applications, hypersets and the related notion of bisimulation offer the best conceptual framework for the treatment of circular phenomena.

In this paper we study *rational* terms (cf. [24]) and circular sets, combined into a very expressive and self-referential abstract type. Instances of the latter type, called *blended (*or *hybrid) hypersets*, are represented by graphs labeled over a signature. The signature is, of course, to comprise one or more symbols denoting operations over hypersets; it will also comprise free symbols, including at least one constant. The share of the signature needed to represent purely set-theoretic terms consists of the $\emptyset$ constant and the binary $with$ operator whose semantics can be intuitively conveyed by the following identity:

$$x\ with\ y = x \cup \{y\}\,.$$

The very modest set-theoretic signature constituted by these two symbols alone, already suffices for the representation of the so-called *hereditarily finite* sets and hypersets.

The main focus of this paper is the design of a *goal-driven unification algorithm* for blended hypersets. After specifying such an algorithm, we show how to extend it into a more versatile set-constraint manager capable of dealing with negative information: to wit, we take into account the binary function symbol less whose intuitive semantics is conveyed by the identity

$$x\ less\ y = x \setminus \{y\}\,.$$

The unification algorithm to be presented has a rather traditional structure, *à la* Martelli-Montanari (cf. [25]): when applied to rational terms that do not involve the set-theoretic constructs, it in fact behaves very much like well-established algorithms, such as those proposed in [9, 26].

As already said, hypersets of the kind to be studied here result from a combination of entities from two domains. In order to precisely specify our unification problem, an axiomatization of the context under study is needed. We propose a theory of rational terms and hypersets, minimal in some sense, that can serve as a kernel for other, richer and accordingly more specialized, theories. In the context of such theories, our unification algorithm can still be exploited.

From the axiomatic part of our study, an interesting kinship emerges between the axiom that Maher introduced in [24] to guarantee completeness of the theory of infinite trees, and the axiom of bisimulation named **AFA** in [1]. Both axioms are adopted below, because our theory was aimed at achieving as much completeness as possible—though with least possible commitment.

It is worth emphasizing here that the theory to which our unification algorithm refers is not —nor would easily be— axiomatized in merely equational terms. The price we have to pay for this is that we cannot rely much on the vast literature on unification (cf., e.g., [5]), which is mainly focused on equational theories. To what extent and how our results can be recast in more conventional terms is a main issue left open by this paper. A promising approach for this is perhaps indicated by [19]; however, our current approach is closer to tableau-handling techniques than to rewriting techniques.

We also carry out a complexity analysis, leading to the conclusions that the proposed unification algorithm is in NP and that the unification problem under study is *NP-complete*. Various studies on the complexity of this problem, even restricted to sets (cf., among others, [21, 4, 32, 18]) show its intrinsic difficulty, so that proposals for more efficient set-unification algorithms are necessarily based on assumptions strongly limiting their field of application (cf. [31]).

The paper is organized as follows: after laying down in Section 1 an intuitive rationale for the hyperset notion, we characterize in Section 2 the universe of blended hypersets, both constructively and axiomatically. Section 3 presents hyperset unification as a very high-level paradigm to solve the deterministic finite-state automaton equivalence problem, along with a less classical, but nevertheless interesting, automaton matching problem. Section 4 is devoted to describing the blended hyperset unification algorithm, whose soundness, completeness, and termination are then proved, and whose complexity behavior is analyzed. In Section 5 it is shown how to extend the algorithm to deal with the negative information introduced by the removal operator $less$. Finally, we end with an application of blended hyperset unification to the type-finding problem, illustrated in Section 6: it turns out that while ordinary sets in type expressions suffice for the modeling of union data types, hypersets can play a role in the modeling of recursive types.

Although this article is a continuation of [28], it should be readable as a self-contained paper. There is, however, a dependency on a decision algorithm of [28] in the (rather marginal) Section 5.

1 Naive Approach to Hyperset Theory, Based on the Analogy with Terms

In this preliminary section, where we proceed in somewhat naive terms, we will contrast ordinary (nested) sets with entities of a much richer variety, to be called —after [6, 7]— *hypersets*. These are also known by the name *non-well-*

founded sets in the thorough study [1]. Indeed, hypersets violate a principle widely agreed upon in the field of Set Theory since von Neumann introduced, in 1928, the *regularity* axiom: traditional membership is a well-founded relation over sets—the situation, with hypersets, is antithetic.

In parallel, we will contrast ordinary terms of first-order logic with terms of the generalized kind treated, e.g., in [23], Ch. 6. The latter have become familiar to researchers in the field of Logic Programming by the name of *infinite terms*. We will recognize a tight analogy between terms and sets: to state it simply,

sets are to hypersets as ordinary terms are to infinite terms.

In view of this parallel, the unified (or "blended", or "hybrid") data structure to be investigated in this paper, encompassing both hypersets and generalized terms, will come out quite naturally.

After characterizing the domain of blended hypersets, $\overline{\mathbb{H}}_\Sigma$ (where Σ is a signature—see below), we will tackle the problem of solving systems of equations over $\overline{\mathbb{H}}_\Sigma$. In order to study this problem in greater generality, a convenient starting point is to consider particularly heavily constrained systems

$$\begin{cases} X_0 = \{ X_{01}, \ldots, X_{0m_0} \} \\ X_1 = \{ X_{11}, \ldots, X_{1m_1} \} \\ \quad \vdots \; \vdots \qquad\quad \vdots \\ X_n = \{ X_{n1}, \ldots, X_{nm_n} \} \end{cases}$$

of set equations. In this initial stage, we are assuming that the left-hand sides, namely $X_0, \ldots, X_n$, are distinct from one another (even though some of the sets they represent might coincide); moreover, each one of the X_{ij}s in a right-hand side must be the same as one of the symbols X_h on the left. As for n and the m_is, they simply are non-negative integers: $n, m_0, \ldots, m_n \in \mathbb{N}$, as infinite sets are beyond the scope of this paper.

By traditional methods, a system of this form admits a solution if and only if there is an ordering $X_{\pi_0}, \ldots, X_{\pi_n}$ of the X_hs such that any $X_{\pi_h j}$ occurring in the right-hand side of the π_h-th equation is one of $X_{\pi_{h+1}}, \ldots, X_{\pi_n}$, for $h = 0, 1, \ldots, n$.[1] Moreover, when this *non-circularity condition* holds, uniqueness of the solution ensues from *extensionality* (which is the postulate stating that two sets are equal if and only if they have the same members) and from the bi-implications $\forall z \, (z \in \{ v_1, \ldots, v_k \} \leftrightarrow \bigvee_{j=1}^{k} z = v_j)$, one for each $k \geq 0$, that summarize the semantics of the construct $\{ \_, \ldots, \_ \}$.

To move from ordinary sets to hypersets, we are to *postulate* that any system of the above form, whether circular or not, admits a solution, and that such a solution is always unique.

---

[1] A proof of this fact goes as follows. If a solution $X_0 = \xi_0, \ldots, X_n = \xi_n$ exists, the X_hs can be ordered so that the ranks of the corresponding ξ_hs do not increase: this ordering will meet the desired property. Conversely, if a permutation π with the stated property exists, then $m_{\pi_n} = 0$ and, accordingly, $X_{\pi_n} = \emptyset$; moreover, the value of X_{π_h} can be determined from those of $X_{\pi_{h+1}}, \ldots, X_{\pi_n}$, for all $h < n$.

For example, both the equation $X = \{ X \}$ and the system

$$\begin{cases} X_0 = \{ X_4, X_3 \} \\ X_1 = \{ X_2, X_3 \} \\ X_2 = \{ X_3 \} \\ X_3 = \{ X_1 \} \\ X_4 = \{ \} \end{cases}$$

have solutions. Indicating by Ω the hyperset such that $\Omega = \{ \Omega \}$, we readily recognize that $X_1 = X_2 = X_3 = \Omega$, $X_4 = \emptyset$, $X_0 = \{ \emptyset, \Omega \}$ is a solution to the system. This solution hence must be the only one—otherwise stated, one can infer $X_1 = X_2$ and $X_1 = X_3$ from the equalities $X_1 = \{ X_2, X_3 \}$, $X_2 = \{ X_3 \}$, $X_3 = \{ X_1 \}$, by virtue of the new postulate, stronger than extensionality. Of course $\emptyset \neq \Omega$ and $\Omega \neq \{ \emptyset, \Omega \}$, because $\forall z\, (z \notin \emptyset)$, $\forall z\, (z \in \Omega \leftrightarrow z = \Omega)$, and $\forall z\, (z \in \{ \emptyset, \Omega \} \leftrightarrow (z = \emptyset \vee z = \Omega))$ all three hold, and therefore $\emptyset$ has no members, Ω has exactly one, and $\{ \emptyset, \Omega \}$ has two.

Let us now switch to the realm of terms by considering a 'flat' system

$$\begin{cases} X_0 = f_0(X_{01}, \ldots, X_{0m_0}) \\ \quad \vdots \quad \vdots \qquad\qquad \vdots \\ X_n = f_n(X_{n1}, \ldots, X_{nm_n}) \,, \end{cases}$$

analogous to the one above, where $f_0, \ldots, f_n$ belong to a SIGNATURE Σ. This means that Σ is a collection of symbols, and that every symbol $s \in \Sigma$ has an associated DEGREE (or "*arity*") $ar(s) \in \mathbb{N}$, stating the legal number m of arguments of s in any well-formed expression of the form $s(t_1, \ldots, t_m)$. In particular, a system like the one above would be ill-formed unless $m_i = ar(f_i)$ held for $i = 0, \ldots, n$. This is the only new requirement: just as before, $X_0, \ldots, X_n$ must be distinct variables from whose collection each X_{ij} is drawn. Variables are now assumed to range over the domain $\mathfrak{G}_\Sigma$ commonly known as the *Herbrand universe* generated by Σ. This domain consists of all ground terms over the signature: accordingly, to solve the system one must substitute every X_h by a ground term γ_h so that every equality gets transformed into a syntactic identity.

The same non-circularity condition discussed earlier in connection with the solvability of a system by ordinary sets, is a necessary and sufficient condition for the solvability, over $\mathfrak{G}_\Sigma$, of the new system. As before, the solution $X_i \mapsto \gamma_i$, if one exists, is obviously unique.

One way to move from ordinary terms to infinite terms, is by postulating that every system of the above form —even one with infinitely many equations— admits a solution, and that the solution is unique. In circular cases —or when n is infinite— i.e., when the system contains equations $X_{i_0} = f_{i_0}(\ldots, X_{i_1}, \ldots)$, $X_{i_1} = f_{i_1}(\ldots, X_{i_2}, \ldots)$, ... that form an infinite chain, some of the γ_is, instead of being drawn from $\mathfrak{G}_\Sigma$, will be drawn from a larger domain $\overline{\overline{\mathfrak{G}}}_\Sigma$, known as the *completion of the Herbrand universe* over Σ (cf. [23]). If we allowed the number $n + 1$ of equations to range from one to infinity, then

the whole of $\overline{\overline{\mathbb{G}}}_\Sigma$ would be spanned by the solutions to flat systems over Σ; but since we will limit our consideration to finite systems, our γ_is will come from a domain $\overline{\mathbb{G}}_\Sigma$ intermediate between $\mathbb{G}_\Sigma$ and $\overline{\overline{\mathbb{G}}}_\Sigma$. The terms in $\overline{\mathbb{G}}_\Sigma$ are usually said to be *rational*, and various methods exist for specifying them (grammars and tree automata [16, 10], axioms [24], etc.).

In order to combine sets with ordinary terms and hypersets with infinite terms, the first step will be to assume that Σ comprises two symbols, $\emptyset$ and $with$, of respective degrees 0 and 2. By $with$, used as a left-associative infix operator, we intend to designate the operation of inserting an element into a set—or, more generally, into a hyperset. An expression of the form $\{ w_1, \ldots, w_k \}$ will be regarded, accordingly, as an abridged notation for $\emptyset\ with\ w_k\ with \cdots with\ w_1$; more generally,

$$\boxed{\begin{array}{l} \{ w_1, \ldots, w_k \mid z \} =_{\text{Def}} \underbrace{(\cdots(}_{k}\ z\ with\ w_k)\ with \cdots with\ w_1\ ) \\ \{ w_1, \ldots, w_k \} =_{\text{Def}} \{ w_1, \ldots, w_k \mid \emptyset \} \end{array}}$$

Any ground term whose outermost functor differs from $with$ will be regarded as a memberless entity, named a COLOR. Either $\emptyset$ (our name for the 'official' empty set) is the sole color, which happens only when $\Sigma = \{\emptyset,\ with\ \}$, or there are also other colors; these are the two cases called PURE and BLENDED, respectively. Quite unconventionally, we will allow insertions like $C\ with\ X$ for any color C distinct from $\emptyset$, regarding any hyperset that results from an insertion of this kind as something distinct from $\emptyset\ with\ X$.

Unlike other functors, which merely play the role of syntactic constructors, $with$ has an intended meaning; hence we can no longer insist that every flat system has a unique solution. For example, the equation $X_0 = \{ \emptyset \mid X_0 \}$ (which means $X_0 = X_0 \cup \{\emptyset\}$) is satisfied by any set to which $\emptyset$ belongs, and hence has the infinitely many solutions

$$X_0 = \Big\{ \emptyset,\ \underbrace{\{\cdots\{}_{k}\ \emptyset \}\cdots\} \Big\}, \quad k = 0, 1, 2, \ldots$$

—and many more. As will emerge from the next section, to circumvent the new difficulty it suffices to restrict one's consideration to systems

$$\bigwedge_{h=0}^{n} X_h = f_i(X_{h1}, \ldots, X_{h,ar(f_i)}) ,$$

as above, whence one cannot extract an infinite chain

$$X_{h_0} = \{ \_ \mid X_{h_1} \},\ X_{h_1} = \{ \_ \mid X_{h_2} \},\ X_{h_2} = \{ \_ \mid X_{h_3} \},\ \ldots$$

of equations (clearly with repetitions). In a theory of blended hypersets, one will postulate that every system of equations subject to this restriction admits one

and only one solution. In a theory of blended sets, the existence (and uniqueness) of a solution will presuppose non-circularity, as usual.

We have been approaching the notion of hyperset in terms of single-solution systems of equations. We now address the question: *what is* an hyperset, precisely? A reasonable way of answering would be to circumscribe, from among single-solution systems of equations, 'canonical' systems of a suitable kind. In each system, the left-hand side X_0 of the first equation could be taken as the variable of interest. Roughly, a system should be regarded as canonical when every equation in it irredundantly contributes to the determination of the value ξ of the variable of interest. One could identify *tout court* the canonical system with the hyperset ξ.

Irredundancy presupposes, among others, *injectivity*: that is, the values of distinct variables should be different. But how can one ascertain that this requirement is fulfilled? An answer is contained in the constructive notion of *bisimulation* in the next section. From now on, to clarify the presentation, single-solution systems will be represented by graphs labeled over Σ, while their variables will be represented by nodes of such graphs.

2 Characterization of Blended Hypersets

In this section the notion of *blended hyperset* will be defined precisely, both constructively (in Section 2.1) and axiomatically (in Section 2.2).

2.1 Intended hyperset model

The entities that form a Herbrand universe are sometimes characterized as being finite trees coherently labeled over a signature Σ. It is easy to adjust this abstract view of ground terms to the terms that form the *completion* of a Herbrand universe: to do this, it will suffice to withdraw the requirement that labeled trees must have finitely many nodes. From this graph-theoretical perspective, syntactic equivalence between terms turns out to coincide with the notion of isomorphism between labeled, ordered trees.

Given a term $\mathscr{T}$, one can 'fold' it by fusing two nodes ν, μ of $\mathscr{T}$ into a single node whenever the subterms rooted at ν, μ are equivalent to each other. This will yield an ordered multi-graph $\mathscr{P}$, both rooted and directed, retaining information of all essential features of $\mathscr{T}$: the PICTURE of $\mathscr{T}$, as we call it. If there are no infinite paths in $\mathscr{P}$, this indicates that the original $\mathscr{T}$ was already finite: this is the case of an ORDINARY term. When $\mathscr{P}$ is finite, $\mathscr{T}$ (which might be infinite) is said to be a RATIONAL term.

If one restricts one's own attention to rational terms and represents them suitably (e.g., by their pictures), then, at least assuming the signature Σ to be finite, even infinite terms can be algorithmically construed and manipulated.

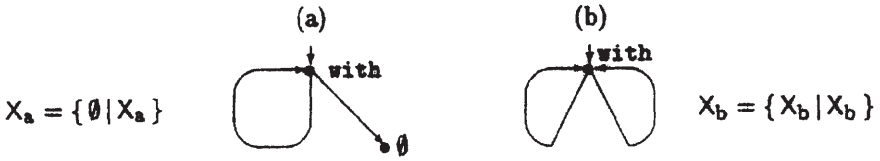


Fig. 1. The terms depicted by these graphs cannot be regarded as ground.

In the case under study, the construction of the universe is not entirely free. We are assuming, in fact, that Σ comprises a symbol, namely $with$, to which a special, fixed meaning is attributed (cf. Section 1). The intuitive semantics of this construct must reflect into the criteria we adopt for equivalencing labeled trees. Such criteria cease accordingly, in our specialized context, to be purely syntactic. At an even more fundamental level, we will have to discard certain trees labeled over Σ, that cannot be regarded as ground terms due to the semantics of $with$.

To proceed more formally, let us start by recalling a classical definition (cf. [23]), which still awaits a minor adaptation to our aims:

Definition 1 *A* GROUND TERM *(over Σ) is a mapping $\mathcal{T} : dom(\mathcal{T}) \longrightarrow \Sigma$ such that*

- *the domain $dom(\mathcal{T})$ of $\mathcal{T}$ is a non-empty ordered tree whose root is* [ ]*;*
- *for all v in $dom(\mathcal{T})$, $ar(\mathcal{T}(v)) = |\{i : [v, i] \text{ in } dom(\mathcal{T})\}|$, where $[v, i]$ stands for the i-th son of v.*[2]

□

To avoid *under-specified* situations, we then add:

Groundness restriction. *The requirement henceforth becomes integral part of the definition of (ground)* TERM *that there be no infinite sequence $v_0, v_1, v_2, \ldots$ of nodes with $\mathcal{T}(v_i) = with$ and $v_{i+1} = [v_i, 1]$ for all i.* □

To see why the presence of a path $v_0, v_1, v_2, \ldots$ as above, in a term, would conflict with the very notion of groundness, let us examine the two graphs of Fig. 1. Either of them is the picture of a labeled tree that violates the groundness restriction. The left arc in either graph indicates —if anything— self-inclusion; hence it conveys no information about the entity (X_a and X_b respectively) represented by the root. The second arc of Fig. 1(a) indicates that $\emptyset$ must belong to X_a, a property which is clearly insufficient to characterize X_a. The right arc of Fig. 1(b) indicates that X_b must belong to itself. If X_b were to be an ordinary set, this would be an absurdity, but we are dealing with hypersets here. Since membership can form cycles among such entities, we are again facing an under-specified situation.

---

[2] We follow [23] in identifying $[[a_1, \ldots, a_n], a_{n+1}]$ with $[a_1, \ldots, a_n, a_{n+1}]$, to keep the notation simple.

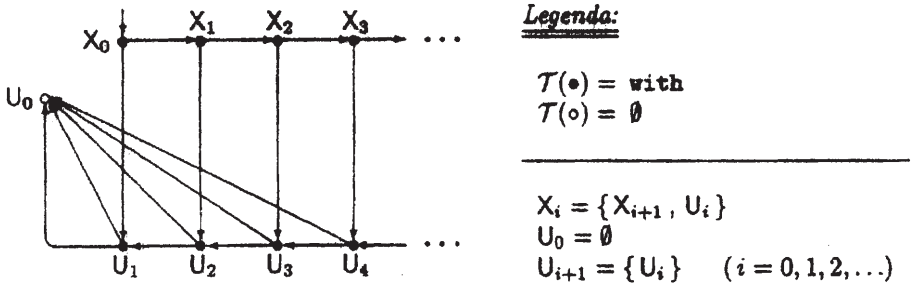


Fig. 2. This graph is the picture of an irrational term $\mathscr{T}$ representing a cycle-free hyperset X_0. Notice that X_0, as well as every element in its transitive closure, is finite.

Our next step will be to get rid of irrational terms (an example of set-theoretic irrational term is the one whose picture is the graph of Fig. 2). Preliminary to that, we need the notion of *bisimulation*, which in turn presupposes the following couple of auxiliary notions.

For every term $\mathscr{T}$ and any ν in $dom(\mathscr{T})$, let $\tau_0, \ldots, \tau_g$ and $\mu_0, \ldots, \mu_{g-1}$ be the sequences of nodes such that: $\tau_0 = \nu$; $\mathscr{T}(\tau_i) = with$, $\tau_{i+1} = [\tau_i, 1]$ and $\mu_i = [\tau_i, 2]$ for $i = 0, \ldots, g-1$; $\mathscr{T}(\tau_g) \neq with$. We denote by $Color(\nu)$ the node τ_g and say that the μ_is are the $\in$-PREDECESSORS of ν.

Definition 2 *Let $\mathscr{T}_0, \mathscr{T}_1$ be terms. A relation $\mathscr{B} \subseteq dom(\mathscr{T}_0) \times dom(\mathscr{T}_1)$ is said to be a* BISIMULATION *between $\mathscr{T}_0$ and $\mathscr{T}_1$ iff:* **i)** $[\,]\,\mathscr{B}\,[\,]$, **ii)** *when $\nu_0\,\mathscr{B}\,\nu_1$, the following hold:*

- $Color_0(\nu_0)\,\mathscr{B}\,Color_1(\nu_1)$, $\quad \mathscr{T}_0(\nu_0) = \mathscr{T}_1(\nu_1)$, *and moreover*
- *to every $\in$-predecessor ϱ_b of ν_b in $\mathscr{T}_b$ ($b = 0$ or $b = 1$), there corresponds at least one $\in$-predecessor ϱ_{1-b} of ν_{1-b} in $\mathscr{T}_{1-b}$ such that $\varrho_0\,\mathscr{B}\,\varrho_1$;*
- *if $\mathscr{T}_0(\nu_0) \neq with$, then $[\nu_0, i]\,\mathscr{B}\,[\nu_1, i]$ for $i = 1, \ldots, ar(\mathscr{T}_0(\nu_0))$.*

We write $\mathscr{T}_0 \approx \mathscr{T}_1$ iff there is a bisimulation $\mathscr{B}$ between $\mathscr{T}_0$ and $\mathscr{T}_1$. □

Bisimulations are, in a sense, isomorphisms complying with the intended (hyperset) semantics of $with$. Accordingly, $\mathscr{T}$ will be regarded as a *rational* term iff it has only finitely many subterms that cannot bisimulate one another. To make this idea precise, let us denote by $\mathscr{T}\lceil\nu$ the subterm of $\mathscr{T}$ issuing from a given node ν.

Definition 3 *A ground term $\mathscr{T}$ is said to be* RATIONAL *iff there are $\nu_0, \ldots, \nu_n$ in $dom(\mathscr{T})$ such that for every μ in $dom(\mathscr{T})$ there is an i, $0 \leq i \leq n$, fulfilling $\mathscr{T}\lceil\nu_i \approx \mathscr{T}\lceil\mu$.* □

In conclusion, indicating by $\overline{\mathbb{G}}_\Sigma$, $\mathbb{G}_\Sigma$ the family of all rational ground terms over Σ and its subfamily consisting of the terms that have finitely many nodes, our HYPERSET UNIVERSE and SET UNIVERSE will be

$$\overline{\mathbb{H}}_\Sigma =_{\text{Def}} \overline{\mathbb{G}}_\Sigma / \approx, \qquad \mathbb{H}_\Sigma =_{\text{Def}} \mathbb{G}_\Sigma / \approx$$

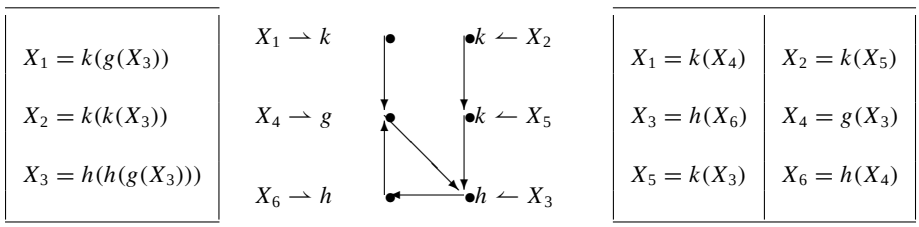


Fig. 3. Two renderings (the one on the right 'flat') of the same ground labeled graph.

respectively. Representing by $\mathcal{T}^{\approx}$ the $\approx$-class of $\mathcal{T}$, the element insertion operation and membership relation over these universes can be straightforwardly defined as

$$\mathcal{T}_0^{\approx}\ with_{\Sigma}\ \mathcal{T}_1^{\approx} =_{\text{Def}} \mathcal{W}^{\approx}, \qquad \mathcal{T}_1^{\approx} \in_{\Sigma} \mathcal{T}_0^{\approx} \text{ iff}_{\text{Def}}\ \mathcal{W} \approx \mathcal{T}_0,$$

where $\mathcal{W}$ is a tree whose root, labeled $with$, has left and right subtree isomorphic to $\mathcal{T}_0$, $\mathcal{T}_1$ respectively.

Every symbol f other than $with$ in Σ is interpreted *à la* Herbrand, that is to say, as the operation sending each m-tuple $\mathcal{T}_1^{\approx}, \ldots, \mathcal{T}_m^{\approx}$ with $m = ar(f)$ into $\mathcal{T}^{\approx}$, where $\mathcal{T}([\,]) = f$ and $\mathcal{T}\lceil[i] \approx \mathcal{T}_i$, for $i = 1, \ldots, m$.

As was done in [28] with reference to the *pure* case only, we could provide criteria for choosing a *canonical representative* out of each $\approx$-class: representatives could then be taken as hypersets proper.

Let us now generalize our discussion by adjoining to our former signature Σ a denumerably infinite collection $\mathcal{V}$ of new symbols of degree 0, named VARIABLES. Labeled graphs, and in particular terms, whose labeling may involve variables, or that may violate the above-stated groundness restriction, will be said to be HOLLOW. As will emerge from Section 4, every hollow, rooted and finite graph depicts a COLLECTION of ground terms, obtainable from it via substitutions.

As illustrated by Fig. 2 and Fig. 3, any ground labeled graph $\mathcal{G}$ (possibly with cycles) can be variously RENDERED, up to isomorphism, as a conjunction (consisting of a single element when $\mathcal{G}$ is finite and acyclic; infinite when $\mathcal{G}$ is not rational) of first-order equalities $\bigwedge_{\nu \text{ in } \mathcal{C}} (X_\nu = t_\nu)$ over the signature $\Sigma \cup \mathcal{V}$, where

- $\mathcal{C}$ is a collection of nodes of $\mathcal{G}$ comprising all nodes of $\mathcal{G}$ devoid of entering arcs, along with at least one node lying on ϖ for each infinite path ϖ of $\mathcal{G}$;
- the X_νs belong to $\mathcal{V}$, hence they are not used as labels in $\mathcal{G}$, and they are distinct from one another; the t_νs are first-order terms[3] over the signature $\Sigma \cup \{X_\nu : \nu \text{ in } \mathcal{C}\}$.

To do that, one views $\mathcal{G}$ as a collection $\{\mathcal{G}_\nu : \nu \text{ in } \mathcal{C}\}$ of finite acyclic rooted graphs labeled over $\Sigma \cup \{X_\nu : \nu \text{ in } \mathcal{C}\}$, each ν in $\mathcal{C}$ bearing the label X_ν and

[3] Notice the distinction we are making between first-order (concrete) terms and terms in the graph-theoretical sense.

each $\mathscr{G}_v$ being in a sense 'grafted' into v; then one takes as t_v the first-order term that straightforwardly corresponds to $\mathscr{G}_v$.[4]

Viewed this way, a rational ground labeled graph $\mathscr{G}$ is just a special case of what is usually called a *Herbrand system*:

Definition 4 *A* HERBRAND SYSTEM *is a finite collection* $\{\ell_1 = r_1, \ldots, \ell_n = r_n\}$ *of first-order equalities,*[5] *where* $\ell_1, r_1, \ldots, \ell_n, r_n$ *are terms over the signature* $\Sigma \cup \mathscr{V}$. *A Herbrand system* $\mathscr{E}$ *is said to be* FLAT *if every equality* e *in* $\mathscr{E}$ *has either the form* $X = Y$, *or the form* $X = g(Y_1, \ldots, Y_n)$ *with* g in Σ. □

Solving general systems of this kind over hypersets is the main unification task coped with in this paper, that we will tackle in Section 4.

2.2 Axiomatic view of the blended hyperset universe

To state our axioms about **hypersets**, we will use a first-order language comprising the constant $\emptyset$, infix operators $with$ and $less$, the (infix) predicates $=$ and $\in$, and a number of functors to which we will resort in order to express most of our axioms without using existential quantifiers. We often use $\{w_1, \ldots, w_k \mid z\}$ and $\{w_1, \ldots, w_k\}$ to shorten notation, as explained in Section 1.

It goes without saying that $=$ meets the usual properties of equality (see, e.g., [27]), which we collectively denote by the label **(=)**.

The symbols x, y, z, u, v, x_i, x_i^b, y_i will stand for distinct variables implicitly universally quantified in front of each axiom.

We begin with (a suitable adaptation of) the EXTENSIONALITY axiom, according to which any two entities that have the same color and the same elements are equal. Formally

$$\boxed{(\mathbf{E}) \quad \Big(\forall z\, (z \in x \leftrightarrow z \in y) \ \wedge\ color\_of(x) = color\_of(y) \Big) \longrightarrow x = y\,.}$$

Then we have axioms concerning the empty entities named COLORS (in particular the *null set* $\emptyset$) and the ELEMENT ADJUNCTION and ELEMENT REMOVAL operations, $with$ and $less$.

$(\mathbf{N}_{0,1})$	$v \notin color\_of(u)$	$color\_of(\emptyset) = \emptyset$
$(\mathbf{W}_{0,1})$	$v \in \{ y \mid u \} \leftrightarrow (v \in u \ \vee\ v = y)$	$color\_of(\{ y \mid u \}) = color\_of(u)$
$(\mathbf{L}_{0,1})$	$v \in u\ less\ y \leftrightarrow (v \in u \ \wedge\ v \neq y)$	$color\_of(u\ less\ y) = color\_of(u)$

[4] When $\mathscr{G}$ is rooted, finite and acyclic, so that its rendering is $X = t$, X not occurring in t, t itself is called *rendering* of $\mathscr{G}$.

[5] Depending on the context, a Herbrand system $\mathscr{E}$ may be seen at times as a set/system of equations, at times as a conjunction of equality atoms.

The ANTI–DIAGONAL and SELF–LOOP axioms below ensure, for example, that for any color y and any tuple $v_1, \ldots, v_m$ of hypersets, the system

$$x \notin v_1 \wedge \cdots \wedge x \notin v_m \wedge x \notin x \wedge color\_of(x) = y$$

of constraints, as well as the equation

$$x = \{x, v_1, \ldots, v_m \mid y\}$$

can be satisfied (one independently of the other).[6]

$(\mathbf{D}_{\notin})$	$\big(x = anti\_diagonal(u, y) \wedge (v \in u \vee v = x)\big) \longrightarrow$ $\big(color\_of(x) = color\_of(y) \wedge x \notin v\big)$
$(\mathbf{D}_{\in})$	$x = self\_loop(u, y) \longrightarrow$ $\big(color\_of(x) = color\_of(y) \wedge \big(v \in x \leftrightarrow (v \in u \vee v = x)\big)\big)$

We are arriving at the axioms essentially expressing our own weak version of Aczel's ANTI–FOUNDATION AXIOM **AFA** (see [1]). We name such axioms ANTI–REGULARITY, (**R**), and HYPER–EXTENSIONALITY, (**H**). To introduce these two schemes in informal terms close to the contents of Section 1, let us consider a system

$$\bigwedge_{j=0}^{n} x_j \equiv \{x_{j1}, \ldots, x_{jm_j}\},$$

where $n \geq 0$, $m_j \geq 0$ for all j, $x_0, \ldots, x_n$ are distinct variables, each x_{ij} is one of $x_0, \ldots, x_n$, and the j-th 'congruence' of the system is a short for $\forall z\,(z \in x_j \leftrightarrow \bigvee_{k=1}^{m_j} z = x_{jk})$. Anti-regularity states that each such system admits a solution with pre-assigned colors for the x_js. Hyper-extensionality states that the solution is uniquely determined by the colors, even if we consider a more general system of congruences

$$\bigwedge_{j=0}^{n} x_j \equiv \{x_{j1}, \ldots, x_{jm_j} \mid z_j\} \quad ,$$

where z is a fixed 'residue' (in the preceding system, $z_j = \emptyset$ for all j).

In the formal specification below, we employ distinct auxiliary variables $x_0^0, \ldots, x_n^0, x_0^1, \ldots, x_n^1$, and indicate by $x_{j1}^b, \ldots, x_{jm_j}^b$ the variables x_h^b such that x_h occurs inside the right-hand side of the j-th congruence.

---

[6] The role and importance of these two axioms first emerged from [29] (cf. also [28]).

(R̸)	$\exists x_0 \cdots \exists x_n \bigwedge_{j=0}^{n} \big(color\_of(x_j) = color\_of(y_j) \wedge \forall z \big(z \in x_j \leftrightarrow \bigvee_{k=1}^{m_j} z = x_{jk} \big) \big)$
(**H**)	$\Big(\bigwedge_{j=0}^{n} \big(color\_of(x_j^0) = color\_of(x_j^1) \wedge \big(\bigwedge_{b=0}^{1} \bigwedge_{i=0}^{n} (x_i^b \in x_j^b \leftrightarrow \bigvee_{k=1}^{m_j} x_i^b = x_{jk}^b) \big) \wedge \forall z \big((z \in x_j^0 \wedge \bigwedge_{i=0}^{n} z \neq x_i^0) \leftrightarrow (z \in x_j^1 \wedge \bigwedge_{i=0}^{n} z \neq x_i^1) \big) \big) \Big) \longrightarrow \bigwedge_{j=0}^{n} x_j^0 = x_j^1$

Let us incidentally observe that (R̸) is a sort of weak form of Aczel's **AFA$_1$**, while (**H**) corresponds to **AFA$_2$**. Notice also that the adoption of (**H**) makes the extensionality axiom (**E**) redundant (cf. [29, 28]).

The following five axioms are Clark's FREENESS assumptions (cf. [8]), an adaptation of Reiter's WEAK DOMAIN CLOSURE ASSUMPTION (cf. [30]), and a statement, antithetic to the OCCURS–CHECK scheme (cf. [8, 30]), which generalizes (R̸).

Here we assume that our signature Σ contains solely $\emptyset$, $with$ and the free functors mentioned in previous sections. In particular $less$, color_of, and any other functor introduced to state the axioms in this section, are not in Σ. We are to assume that f, g, and all f_νs appearing below, belong to Σ; also, g is to be distinct from both $with$ and f. Finally, let $A = \max\{ ar(h) : h \text{ in } \Sigma,\ h \neq with \}$.

Axiom (**U**$_0$) will only be included when Σ has finite cardinality. Every instance of (**¬OC**) results from a finite ground graph $\mathcal{G}$ labeled over Σ, where it is not restrictive to assume that $\mathcal{G}$ is cyclic. The variables $x_{\nu_0}, \ldots, x_{\nu_\ell}$ are in one-to-one correspondence with the nodes $\nu_0, \ldots, \nu_\ell$ of $\mathcal{G}$; we are indicating by f_ν the label of the node ν, by n_ν the degree of f_ν, and by μ_j^ν the j-th son of ν. The uniqueness requirement was introduced by Maher in [24] in order to achieve completeness of his theory of infinite trees: this makes it very similar to **AFA**. Such kinship will be exploited in a continuation of this paper to carry out a reduction of the *blended* hyperset unification problem to the *pure* hyperset unification problem (cf. [13]).

(**F**$_0$)	$f(x_1, \ldots, x_n) \neq g(y_1, \ldots, y_m)$
(**F**$_1$)	$g(x_1, \ldots, x_n) = g(y_1, \ldots, y_n) \longrightarrow (x_1 = y_1 \wedge \cdots \wedge x_n = y_n)$
(**U**$_0$)	$x = color\_of(u) \longrightarrow \exists x_1 \cdots \exists x_A \bigvee_{h \text{ in } \Sigma \setminus \{ with \}} x = h(x_1, \ldots, x_{ar(h)})$
(**U**$_1$)	$color\_of(f(x_1, \ldots, x_n)) = f(x_1, \ldots, x_n)$
(**¬OC**)	$\exists ! \, x_{\nu_0} \cdots x_{\nu_\ell} \bigwedge_{\nu \text{ node of } \mathcal{G}} x_\nu = f_\nu(x_{\mu_1^\nu}, \ldots, x_{\mu_{n_\nu}^\nu})$

Remark 1

1. *To obtain from the preceding theory of hypersets a corresponding theory of* **sets**, *we should drop* ($\mathbf{D}_{\in}$), ($\not\mathbf{R}$) *and* ($\neg\mathbf{OC}$), *adopting a classical* REGULARITY *axiom and a suitable version of the* OCCURS–CHECK *scheme of axioms (cf. [14]).*
2. *It is immediate to see that* ($\mathbf{U_1}$) *generalizes* ($\mathbf{N_1}$)*; also, it is plain to deduce* ($\mathbf{N_0}$) *from the remaining axioms, including* ($\mathbf{U_0}$). *Hence one shall not list* ($\mathbf{N_{0,1}}$) *among the axioms when* Σ *is finite.* □

3 Expressiveness of Equation Systems Over Hypersets: An Example

One of the most common exploitations of hypersets is as a means to model deterministic finite-state automata. As we will now see, the notions introduced so far provide a well-suited framework for conceiving automaton manipulation algorithms based on this kind of modeling. Another, rather distant, application of hypersets will be considered in Section 6.

A *deterministic finite automaton* (DFA for short) consists of a set $\mathcal{Q} = \{q_0, \ldots, q_N\}$ of *states*, an *alphabet* $\mathcal{S} = \{s_1, \ldots, s_M\}$ (with $0 < M < \infty$), a partial *transition function* $d : \mathcal{Q} \times \mathcal{S} \longrightarrow \mathcal{Q}$. Moreover, one of the states —say q_0— is singled out as the *initial state*, and there is a set $\mathcal{F} \subseteq \mathcal{Q}$ of *accepting states* (for a full explanation of DFAs see, for instance, [2, 12]).

Given a DFA A as just said, one can construct a corresponding Herbrand system $\mathcal{E}_A$ involving as many unknowns and equations as A has states, in the signature $\Sigma = \{\, with\ , \emptyset, false, true\,\} \cup \mathcal{S}$ where all symbols save $with$ have degree 0. $\mathcal{E}_A$ consists of the equations (one for each $i = 0, \ldots, N$)

$$X_i = \{f_i\} \cup \bigcup_{h=1}^{M} \delta_{ih}$$

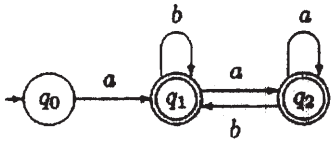
where

$$\begin{cases} f_i \ \text{ is the truth value of } \ q_i \in \mathcal{F}\ , \\ \delta_{ih} = \emptyset \ \text{ if } \ d(q_i, s_h) \ \text{ is undefined}\ , \\ \delta_{ih} = \{\{\, s_h,\ X_j\,\}\} \ \text{ if } \ d(q_i, s_h) = q_j\ . \end{cases}$$

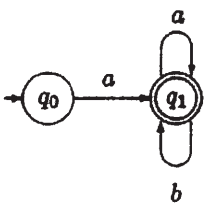
Even though $\mathcal{E}_A$ is not written in flattened form, it should be clear from the discussion in Section 1 —echoed by the axioms ($\not\mathbf{R}$) and (**H**) in Section 2.2— that $\mathcal{E}_A$ has a unique solution over hypersets: we can take the hyperset value of X_0 in this solution as a representation, more essential than the automaton itself, of the regular language $\mathcal{L}(A)$ accepted by A.

This is essentially the same technique for translating finite automata into hypersets provided in [7], where it is shown that graph bisimilarity (cf. our Def. 2 of $\approx$) faithfully reflects equivalence between deterministic automata.

To illustrate how one can test whether or not $\mathcal{L}(A) = \mathcal{L}(B)$, for given DFAs A and B, let us consider the following automata with alphabet $\{a, b\}$:



DFA A



DFA B

Here, the sets of accepting states are $\mathscr{F}_A = \{q_1, q_2\}$ and $\mathscr{F}_B = \{q_1\}$, respectively.

In our framework, these automata can be modeled by the systems:

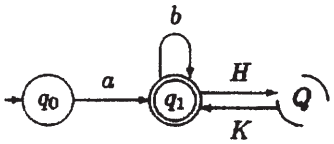
$$\begin{aligned}\mathscr{E}_A = \{\ & X_0 = \{false, \{a, X_1\}\},\\ & X_1 = \{true, \{a, X_2\}, \{b, X_1\}\},\\ & X_2 = \{true, \{a, X_2\}, \{b, X_1\}\}\ \}\ ,\\ \mathscr{E}_B = \{\ & X_0' = \{false, \{a, X_1'\}\},\\ & X_1' = \{true, \{a, X_1'\}, \{b, X_1'\}\}\ \}\ .\end{aligned}$$

Both of these admit a solution, but it is not obvious that the system $\mathscr{E}_A \cup \mathscr{E}_B \cup \{\ X_0 = X_0'\ \}$ resulting from their combination has a solution as well: the latter system contains in fact a pair of equations sharing the same left-hand side, X_0.

We are facing an instance of the *blended* hyperset unification problem to be discussed in full generality later on. In the case at hand, the answer to the problem will be affirmative, because we can directly check on the automata that $\mathscr{T}_A \approx \mathscr{T}_B$ under a bisimulation $\mathscr{B}$, where $\mathscr{T}_A$ and $\mathscr{T}_B$ are the values yielded for X_0 and X_0' by $\mathscr{E}_A$ and $\mathscr{E}_B$.

The problem of testing the equivalence of two deterministic automata is NL-complete (cf. [20]) and can be solved by very fast algorithms (cf. [2, 17]). However, it would be wrong to expect from this deceptively simple example that hyperset unification could be performed by suitably adapted versions of those algorithms. Via a reduction to hyperset unification, we can solve the more general problem of establishing whether a *partially defined* DFA can be completed into a given deterministic automaton, a problem that we will show to be NP-complete.

As an example of this kind of one-way matching, let us consider the problem of establishing whether the following partially defined automaton C,[7]



where H, K, and Q are unknown, can be instantiated so as it accepts the same language as the previously seen *DFA A*. In other words, we are to solve the system $\mathscr{E}_A \cup \mathscr{E}_C \cup \{X_0 = Q_0\}$, where $\mathscr{E}_C$ reads:

---

[7] Note that *a priori* this automaton could even be non-deterministic.

$$
\begin{aligned}
Q_0 &= \{false, \{a, Q_1\}\},\\
Q_1 &= \{true, \{b, Q_1\}, \{H, Q_2\}\},\\
Q_2 &= \{F_2, \{K, Q_1\}\},\\
\{F_2 \mid \_\} &= \{false, true\},\\
\{H, K \mid \_\} &= \{a, b\}\ .
\end{aligned}
$$

Two solutions indeed exist: one of them yields $Q_0 = X_0$, $Q_1 = Q_2 = X_1$, $H = a$, $K = b$.

4 Hyperset Unification

The following notation is used below. Capital letters X, Y, Z, etc. represent variables; f, g, etc. stand for functional symbols (i.e. elements of Σ); $\equiv$ denotes the syntactic identity relation between first-order terms over $\Sigma \cup \mathscr{V}$; φ_Y^X denotes the result of replacing every occurrence of the variable X by Y in a quantifier-free first-order expression φ, and $vars(\varphi)$ denotes the set of all variables occurring in φ; $dom(\lambda)$ and $ran(\lambda)$ denote the set of first, respectively second, components of a binary relation λ. Hereafter we only need to consider substitutions of the following kind:

Definition 5 *A (ground)* SUBSTITUTION *is a mapping γ from a finite subset of $\mathscr{V}$ to the universe $\overline{\mathbf{G}}_\Sigma$.* □

4.1 The Unification Problem

One can APPLY a substitution γ to a hollow graph $\mathscr{G}$, thereby obtaining a ground $\mathscr{G}^\gamma$, when $\mathrm{dom}(\gamma) \supseteq \mathscr{V} \cap \mathrm{ran}(\mathscr{G})$: to do that, one will graft an isomorphic copy of $\gamma(X)$ in place of each node v labeled $\mathscr{G}(v) = X$, for all X in $dom(\gamma)$. After observing that every first-order term t is the concrete rendering of an acyclic rooted graph $\mathscr{G}_t$ labeled over $\Sigma \cup \mathscr{V}$ (see ending remarks of Section 2.1), one realizes that the notation t^γ makes sense too, provided $dom(\gamma) \supseteq vars(t)$. Thus we are ready to define:

Definition 6 *A* SOLUTION *to a Herbrand system $\mathscr{E}$ is a substitution γ that solves all equations in $\mathscr{E}$ at once. That is, for all $\ell = r$ in $\mathscr{E}$, both $dom(\gamma) \supseteq vars(\ell) \cup vars(r)$ and $\ell^\gamma \approx r^\gamma$ hold.* □

There are systems of equations of special forms for which a solution can be determined quite easily.

Definition 7 *A Herbrand system $\mathscr{E}$ is said to be in* SOLVABLE FORM *if each equation in it has one of the forms:*

- *$X = Y$, with Y distinct from X and X not occurring elsewhere in $\mathscr{E}$;*

- $X = f(Y_1, \ldots, Y_n)$, *or in particular* $X = \{ Y_2 \mid Y_1\}$, *with* X *not occurring as left-hand side of any other equation in* $\mathscr{E}$.

A Herbrand system in solvable form is said to be EXPLICIT *if it contains no subsystem of the following* ZIPPER *form:*

$$X_0 = \{Y_0|X_1\}, \ldots, X_{m-1} = \{Y_{m-1}|X_m\}, \quad X_m = \{Y_m|X_0\}$$

(for $m = 0$ *this reduces to the single equation* $X_0 = \{Y_0|X_0\}$*).* □

To justify this nomenclature, let us see how readily a solution to a system in solvable form can be found when the system is EXPLICIT. One can proceed to enlarge the system with all equations $Y = \emptyset$, where each Y is a variable that, although occurring in the system, does not occur as left-hand side of any of its equations. The ground image $\gamma(X)$ of each variable X in a solution γ, can thus be read directly off the system. (Trivially, the solution to the enlarged system is unique up to $\approx$; however the original system could have had many other solutions.)

More generally, a system in solvable form can be modified until it becomes explicit. Every modification step will introduce new variables; however each solution to the modified system can be restricted to the old variables giving a solution to the previous system. Thus, at the end, any solution to the explicit system will also be a solution to the original system. To see this, note that $\gamma(X_0) \approx \gamma(X_1) \approx \cdots \approx \gamma(X_m)$ must hold in any solution γ, when there is a zipper $X_0 = \{Y_0|X_1\}, \ldots, X_{m-1} = \{Y_{m-1}|X_m\}, \quad X_m = \{Y_m|X_0\}$. Therefore, as long as a there is a subsystem of this kind, we can replace it by the equations

$$K_1 = \{Y_0|K_0\}, \ldots, K_m = \{Y_{m-1}|K_{m-1}\},$$

$$X_0 = \{Y_m|K_m\}, \ldots, X_m = \{Y_m|K_m\}\ ,$$

where $K_0, \ldots, K_m$ are new variables. While the system resulting from this modification is still in solvable form, it is closer to explicit form, because the number of zippers has been reduced.

We are now ready to state the unification problem in very specific terms. Systems in solvable form can, for that sake, be employed as templates of the solutions to a given system:

Definition 8 *Given a Herbrand system* $\mathscr{E}$, SOLVING $\mathscr{E}$ *amounts to producing a finite set of Herbrand systems in solvable form* $\mathscr{E}_1, \ldots, \mathscr{E}_m$*, such that*

- *for every solution* γ *to* $\mathscr{E}$*, at least one of the* $\mathscr{E}_i$*s has a solution* σ *such that* $\gamma(X) = \sigma(X)$ *for all* X *in* $vars(\mathscr{E}) \cap vars(\mathscr{E}_i)$*;*
- *for any solution* σ *to any of the* $\mathscr{E}_i$*s, every substitution* γ *such that* $dom(\gamma) \supseteq vars(\mathscr{E})$ *and* $\gamma(X) = \sigma(X)$ *for all* X *in* $vars(\mathscr{E}) \cap dom(\sigma)$*, is a solution to* $\mathscr{E}$. □

Each system in solvable form covers a family of solutions. The intuitive meaning of the definition above, apart from the technical requirement concerning variables, is the following: the set $\{\mathscr{E}_1, \ldots, \mathscr{E}_m\}$ solves $\mathscr{E}$ if any solution to $\mathscr{E}$ is also a solution for at least one of the $\mathscr{E}_i$s; moreover, every solution to any cal E_i is also a solution to $\mathscr{E}$. Notice that we are not requiring that the various $\mathscr{E}_i$s be independent: this means that there can be solutions to $\mathscr{E}$ that are solutions to more than one of the $\mathscr{E}_i$s. The features of our theory make it hard to develop here a syntactic criterion for comparing the generality of two systems in solvable form, and, accordingly, finding suitable notions of *most general unifier (mgu)* and of *complete set of mgus* (see, e.g., [5]). Of course, a preorder can be imposed on systems as follows: given two systems E and E', E is more general than E' if any solution to E is also a solution to E'. From this semantic standpoint, the above set $\{\mathscr{E}_1, \ldots, \mathscr{E}_m\}$ contains (possibly properly) a minimal set of *most general systems* in solvable form (or *templates of solutions*) for $\mathscr{E}$.

4.2 NP-hardness of the problem of finding a single hyperset unifier

One-way matching between automata, that is, the problem of instantiating a partially defined automaton so as to make it equivalent to a fully specified DFA (see end of Section 3), will turn out to be NP-complete. For the moment let us prove its NP-hardness, by reducing to it the propositional 3-SAT problem: it will emerge from Section 4.5, after the hyperset unification problem has been algorithmically settled, that the problem can be solved non-deterministically in polynomial time.

Given the formula

$$\Phi = (\ell_{11} \vee \ell_{12} \vee \ell_{13}) \wedge \cdots \wedge (\ell_{n1} \vee \ell_{n2} \vee \ell_{n3})$$

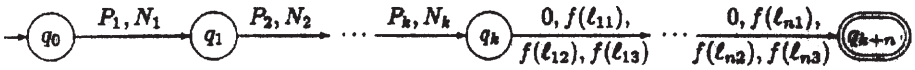
where, for $i = 1, 2, 3$ and $j = 1, \ldots, n$,

$$\ell_{ji} \equiv a_{ji} \text{ or } \ell_{ji} \equiv \neg a_{ji} \text{ and } a_{ji} \in \{d_1, \ldots, d_k\}$$

for a suitable $k \leq 3 \cdot n$, we introduce distinct unknowns $P_1, N_1, \ldots, P_k, N_k$, and define the transformation function f as follows:

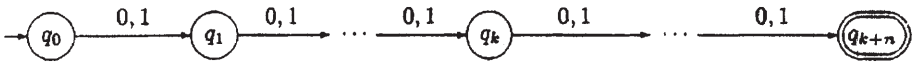
$$f(\ell) = \begin{cases} P_i \text{ if } \ell \equiv d_i, \\ N_i \text{ if } \ell \equiv \neg d_i. \end{cases}$$

Thus, in order to solve the instance Φ of 3-SAT, it is sufficient to check whether there exists an instantiation making the automaton



equivalent to the fully specified automaton below:[8]

[8] To stay strictly inside the realm of pure (hyper)sets, one could replace 0 by $\emptyset$, and 1 by $\{\emptyset\}$.



Notice that the technique presented in Section 3 for expressing automaton equivalence as a unification problem would not work properly for nondeterministic automata: the equivalence problem for such automata is known to reside at a higher level of complexity (it is in fact *PSPACE*-complete—cf. [15]).

4.3 A unification algorithm, Hyper_unify

Special chains of inclusions, introduced by the following definition, will play an important role in our subsequent discussion:

Definition 9 *A* PATH *in a Herbrand system $\mathscr{E}$ is a sequence $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_n}{\leftarrow} X_{n+1}$ of labeled edges such that $X_i = \{Y_i | X_{i+1}\}$ is in $\mathscr{E}$ for all i* in $\{0, \ldots, n\}$. □

We now discuss the unification algorithm Hyper_unify described in Fig.4. This algorithm gets an input $\mathscr{E}$ which, without loss of generality, is assumed to be *flat* (cf. Def. 4).

Hyper_unify performs a non-deterministic search. When reaching the leaf of a successful branch of the search tree, it will output a system $\mathscr{E}_i$ in solvable, explicit, form. The whole search tree will be finite.

The algorithm makes also use of an auxiliary data structure, $\mathscr{C}$, to keep track of a number of temporary assumptions of the form $W \notin V$. Action Fail_1 may detect a failure situation by checking whether one of the constraints $Y \notin X$ in $\mathscr{C}$ conflicts with the fact that Y must belong to X, by $\mathscr{E}$.

In its present version, Hyper_unify initializes $\mathscr{C}$ internally, and does not produce any information about it in the output. We will see, however, in Section 5, that Hyper_unify can be enhanced to deal with negated membership literals. Then an initial value for $\mathscr{C}$ will be submitted as part of the input, and the final value of $\mathscr{C}$ will be retained with the output.

Before we undertake analyzing the complexity of Hyper_unify —after which, the correctness proof will be supplied— let us develop a little nomenclature and some preliminary remarks and comments. It will be useful to think of a (nondeterministic) execution of Hyper_unify as consisting of segments classified as follows:

Phase: An iteration of the outer repeat, consisting of a preamble (i.e., a full execution of the inner repeat), followed by the checks performed at the turning point, and (unless a termination has occurred) by one of the five actions (a), (b.1)–(b.4) (when more than one of these is viable, one is arbitrarily chosen).

Hyper_unify($\mathscr{E}$: Herbrand_system);
$\mathscr{C} := \emptyset$;
repeat
Preamble: repeat

1. discard any reflexive equations $X = X$ from $\mathscr{E}$;
2. for each equation $e \equiv X = Y$ in $\mathscr{E}$ such that X occurs somewhere else in $\mathscr{E}$, do
   begin $\mathscr{E} := (\mathscr{E}\,{}^{X}_{Y} \cup \{ X = Y \}) \setminus \{ Y = Y \}$; $\mathscr{C} := \mathscr{C}\,{}^{X}_{Y}$ end;
3. as long as there is a zipper $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_{n-1}}{\leftarrow} X_n \overset{Y_n}{\leftarrow} X_0$
   (with $X_0, \ldots, X_n$ distinct from one another) in $\mathscr{E}$, do
   begin
   $\mathscr{E} := (\mathscr{E} \setminus \{ X_0 = \{Y_0|X_1\}, \ldots, X_{n-1} = \{Y_{n-1}|X_n\}, \quad X_n = \{Y_n|X_0\} \}) \cup$
   $\{ X_0 = \{Y_0|K_0\}, \ldots, X_0 = \{Y_n|K_n\}, X_1 = X_0, \ldots, X_n = X_0 \}$;
   $\mathscr{C} := \mathscr{C} \cup \{ Y_0 \notin K_0, \ldots, Y_n \notin K_n \}$;
   end;
4. as long as there is a path $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_n}{\leftarrow} X_{n+1} \overset{Y_0}{\leftarrow} X_{n+2}$
   (with $Y_0, \ldots, Y_n$ distinct from one another) in $\mathscr{E}$, do
   $\mathscr{E} := (\mathscr{E} \setminus \{ X_0 = \{Y_0|X_1\} \}) \cup \{ X_1 = X_0 \}$;
5. close $\mathscr{C}$ with respect to the rule: $X \overset{Y}{\leftarrow} V$ in $\mathscr{E}$, $Z \notin X$ in $\mathscr{C}$ implies $Z \notin V$ in $\mathscr{C}$;

until nothing has been modified by the last iteration;
Turning point:

Fail_1: if there is an edge $X \overset{Y}{\leftarrow} V$ in $\mathscr{E}$ with $Y \notin X$ in $\mathscr{C}$,
then exit with failure; /* this check could immediately follow action 5*/

Fail_2: if there are equations $X = f(X_1, \ldots, X_n)$, and $X = g(Y_1, \ldots, Y_m)$ in $\mathscr{E}$ with $f \not\equiv g$,
then exit with failure;

Succeed: if $\mathscr{E}$ is in solvable form, then exit with success returning $\mathscr{E}$;

Actions: select arbitrarily an equation e in $\mathscr{E}$ enabling one of the following actions

(a) $e \equiv X = f(X_1, \ldots, X_n)$, $f \not\equiv with$, and there is an $e' \equiv X = f(Y_1, \ldots, Y_n)$ in $\mathscr{E} \setminus \{e\}$:
$\mathscr{E} := (\mathscr{E} \setminus \{e'\}) \cup \{ X_1 = Y_1, \ldots, X_n = Y_n \}$; /* removing e' is inessential */

(b) $e \equiv X = \{Y|V\}$, and there is an $e' \equiv X = \{Z|W\}$ in $\mathscr{E} \setminus \{e\}$

(note that $X \not\equiv V$ and $X \not\equiv W$, thanks to action 3); perform one of the following actions:

(b.1) $\mathscr{E} := (\mathscr{E} \setminus \{e'\}) \cup \{ Y = Z, V = W \}$; /* removing e' is inessential */

(b.2) $\mathscr{E} := \mathscr{E} \cup \{ X = V \}$;

(b.3) $\mathscr{E} := \mathscr{E} \cup \{ X = W \}$;

(b.4) $\mathscr{E} := (\mathscr{E} \setminus \{e'\}) \cup \{ V = \{Z|N\}, W = \{Y|N\} \}$; $\mathscr{C} := \mathscr{C} \cup \{ Y \notin N, \; Z \notin N \}$;

forever.

Fig. 4. Hyperset unification algorithm

Stage: A series of consecutive phases that

- immediately follows either initialization or an execution of action (b.4);

- either goes on forever, or terminates at the turning point, or ends with an execution of action (b.4);
- does not comprise any execution of action (b.4), save, possibly, at the end.

(An important fact we will discover later on is that no stage consists of infinitely many phases: this will rule out the second of the above three possibilities.)

Notice that new variables are introduced into $\mathscr{E}$ by actions 3 and (b.4) (they are the K_is and the Ns, respectively).

As we are about to discuss, we can think that an equivalence relation between variables is being implicitly calculated by Hyper_unify. Initially, each variable makes an equivalence class by itself; then, the equivalence relation gets refined by the preamble of each phase. Once they have become equivalent, two variables are to represent the same hyperset; accordingly, as soon as a variable formerly generated by action 3 or (b.4) becomes equivalent to an initial variable, we identify the two with one another and cease to regard the generated variable as 'new' any more.

Let us now clarify the main purpose of each preamble, which is to decompose the system $\mathscr{E}$ into subsystems of the form

$$\left.\begin{matrix} X_1 \\ \vdots \\ X_\ell \end{matrix}\right\} = R = \left\{\begin{matrix} f_1(Y_{11}, \ldots, Y_{1a_1}) \\ \vdots \qquad\qquad \vdots \\ f_m(Y_{m1}, \ldots, Y_{ma_m}) \end{matrix}\right.$$

with $\ell + m \geq 1$ (hopefully with $f_i \equiv f_j$ for all pairs of i and j). These subsystems will be mutually independent in the sense that

- no left-hand variable X_i appears, globally, more than once;
- no *representative* variable R occurs as left-hand side of a variable-variable equation; apart from this, R is entirely free to occur in right-hand sides.

Of course we may assume that representative variables are distinct in different subsystems; thus, when $m \leq 1$ holds for each one of the subsystems, a solvable form results from the preamble.

As for the equivalence relation over variables hinted at above, it can be characterized at the end of each preamble as being the reflexive and transitive closure of the relation $\{[X, R] : X = R \text{ in } \mathscr{E}\}$.

4.4 Termination of the Hyper_unify algorithm

In this subsection we prove the termination of Hyper_unify; that is, we show that every branch of the search tree of Hyper_unify eventually breaks off, reporting failure or success. A rather coarse assessment of the maximum length of a branch will result from initial analysis; this will be refined into a polynomial bound on the overall complexity of the algorithm in Section 4.5.

To measure the size of the input system $\mathscr{E}$, we adopt the following two parameters:

- v_0, the number of distinct variables in $\mathscr{E}$;
- s_0, the number of occurrences of functional symbols in $\mathscr{E}$ (including with).

For reasons to become clear soon, both the termination proof and the complexity analysis will mainly consists in determining upper bounds for the number of stages, i.e. for the number of actions (b.4) performed along a branch.

Remark 2

1. A new variable Q always shows up, at creation time, in a context $V = \{Y|Q\}$ (where $V \not\equiv Q$ and $Y \not\equiv Q$); later on, it can be moved to a different context by actions 2, or (b.2)–(b.3). Anyway, it will never come to occupy a label position over an edge $X \overset{Q}{\leftarrow} W$: that is to say, no equality $X = \{Q|W\}$ will ever go into $\mathscr{E}$ unless after Q has become equivalent to a pre-existing label, in which case we no longer regard it as 'new'.
2. It follows from the preceding observation and from the presence of actions 3 and 4 in Hyper_unify that no path $X_0 \overset{Y_1}{\leftarrow} X_1 \overset{Y_2}{\leftarrow} \cdots \overset{Y_n}{\leftarrow} X_n$ can have length $n > v_0$ at the end of any preamble.
3. If an edge $X_0 \overset{Y_0}{\leftarrow} X_1$ drawing its origin from an action 3 or (b.4) becomes part of a zipper $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_0}{\leftarrow} X_0$, a failure will take place at the next turning point. This follows from the fact that the creation of the edge $X_0 \overset{Y_0}{\leftarrow} X_1$ causes the constraint $Y_0 \notin X_1$ to be put in $\mathscr{C}$; however the path in question will cause action 3 to place the equation $X_0 = X_1$ in $\mathscr{E}$, which will fire a Fail_1 termination.
4. The preceding observation implies that the overall number of arcs introduced by action 3 is less than the number of occurrences of $with$ in the initial system, bounded by s_0. □

The following lemma implies that the algorithm terminates provided action (b.4) is performed a finite number k of times (cf. Corollary 2). Later on, with Lemma 3, we will place a finite bound on k.

Lemma 1 *Consider a stage of a computation of $Hyper\_unify(\mathscr{E})$. Let v and s be the number of inequivalent variables (initial or not) and the number of occurrences of functional symbols in $\mathscr{E}$, as they are at the beginning of that stage. Then the stage comprises at most $1 + v + s_0 + s$ phases.*

Proof. First note that no action other than (b.4) increases s.

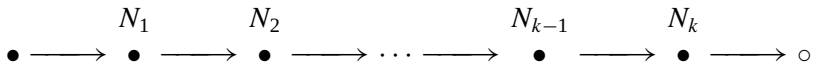
The number v might increase due to executions of action 3, which however cannot be exploited more than s_0 times, as noticed in Remark 2.4. This potential increase of v accounts for the addendum s_0 in the thesis of the lemma.

We can henceforth focus on actions (a), (b.1)–(b.3), namely the ones from which a non-deterministic choice is performed in every non-final phase.

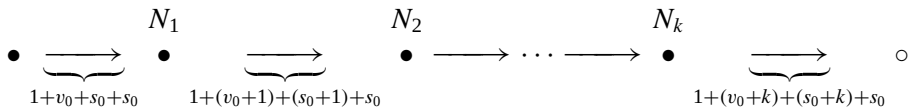
(a) and (b.1) cause s to decrease; actions (b.2) and (b.3), although leaving s unchanged, set the ground for either a reduction of v in the next preamble or a Fail_1 termination at the next turning point. ■

Corollary 2 *Suppose action (b.4) is performed a finite number k of times during a computation of $Hyper\_unify(\mathscr{E})$. Then the computation comprises at most $(k+1)\cdot v_0 + (k+2)\cdot s_0 + (k+1)^2$ phases.*

Proof. Let $N_1, \ldots, N_k$ be the variables introduced by the successive executions of action (b.4). We can split the sequence of phases performed by Hyper_unify($\mathscr{E}$) into $k+1$ successive stages determined by the introduction of the N_is.



The first stage is based on a system with size parameters v_0 and s_0; hence, by Lemma 1, it contains no more than $1+v_0+s_0+s_0$ phases. After the application of action (b.4), both s and v get increased at most by one; therefore the second stage, again by Lemma 1, contains no more than $1+(v_0+1)+(s_0+1)+s_0$ phases. The overall situation is summarized by the following diagram:



The number of phases forming a whole computation is hence bounded by $\sum_{i=0}^{k}(1+(v_0+i)+(s_0+i)+s_0)$; however, it is appropriate to consider the addendum s_0 only once, because it represents an upper bound on the number of edges introduced by action 3, which is a global quantity. Thus, no more than $(k+1)\cdot v_0 + (k+2)\cdot s_0 + (k+1)^2$ phases can be performed. ■

It should be clear, already, that each phase lasts finitely long: we will be more specific on this by assessing the complexity of a preamble in Section 4.5.

Let us now indicate by E_i and C_i the values of $\mathscr{E}$ and $\mathscr{C}$ when the turning point is reached for the $(i+1)$-st time. Any E_i induces an equivalence relation $\sim_i$ among variables, as explained at the end of Section 4.3.

To place a bound on the number k of times (b.4) gets executed, we will define in terms of $\sim_i$ a tuple τ_i of natural numbers so that modifications of $\mathscr{E}$, $\mathscr{C}$ made during the i-th phase may cause τ_{i+1} to differ from τ_i. Changes of τ will invariably lower it w.r.t. the lexicographic well-ordering of tuples, without affecting its length. τ gets lowered whenever a new variable is introduced, which happens, in particular, with (b.4). This by itself ensures —in view of Corollary 2— the termination of Hyper_unify.

Notice that any lowering of a tuple $[x_0, \ldots, x_\ell]$ w.r.t. this ordering can be achieved by elementary *decrease actions* of the following kind:

- a component x_j, with $x_j > 0$, gets decremented by one; simultaneously (if $j < \ell$)
- a component x_i, with $i > j$, is incremented by h units, with $h \geq 0$.

In the case at study, we will be able to show that such decrease actions will always have $0 \leq h \leq 2$; correspondingly, a decrease action will be named:

destruction: if $h = 0$,
climbing: if $h = 1$,
generation: if $h = 2$.

To introduce τ, let us consider the set

$$L_i = \{\ Y\ :\ X = \{\, Y \mid W \,\}\ \text{ in }\ E_i\}$$

of all Ys occurring in a context $X \overset{Y}{\leftarrow} W$ within E_i. By Remarks 2.1–2, $|L_i| \leq |L_0|$ for all $i \geq 0$. Moreover, as easily seen, $|L_i| = |L_0/\sim_i|$. We define τ_i to be the tuple $\tau_i = [x_0, \ldots, x_{|L_0|}]$, where each x_j is the cardinality of the following set:

$$\Big\{ \text{equations of the form } \_ = \{\, \_ \mid W \,\} \text{ in } E_i\ :\ |L_0| - |L_i| + |\{\ Y\ :\ Y \notin W \text{ in } C_i\ \}| = j \Big\}.$$

The initial value τ_0 of τ clearly has $x_0 \leq s_0$ and $x_1 = \cdots = x_{|L_0|} = 0$.

Lemma 3 *A stage always lowers the value of τ except, possibly, when a Fail_1 exit is about to take place.*

Proof. Actions 1, (a), (b.2), and (b.3) do not affect τ. Action (b.1) always affects τ as a destruction action.

The complexity function $compl(\tau)$ has been carefully chosen in order that action 2, whose effect is a sequence of destruction and generation actions, cannot increase it.

Actions 3 and 5 may leave τ unchanged or affect it as climbing actions; action 4 might cause τ to increase, but in this case Fail_1 will immediately cause termination.

If neither $Y \notin W$ nor $Z \notin V$ belongs to $\mathscr{C}$, then action (b.4) makes τ lower (it can be viewed as a generation action). Unless this is the case, an immediate combination of action 3 with Fail_1 will lead to termination. ■

Theorem 4 (termination) *Let $\mathscr{E}$ be a flat system. Then $Hyper\_unify(\mathscr{E})$ always terminates, no matter what sequence of non-deterministic choices has been made.*

Proof. Lemma 3 ensures that a only finite number of actions (b.4) occur along a branch of the computation. The claim follows, by Corollary 2. ■

One may wonder whether the above translation of Hyper_unify actions into chains of destruction, climbing, and generation actions, may disclose a time complexity assessment sharper than a simple termination result. Unfortunately, this is not the case.

It can be shown that, starting with a tuple $[x_0, \ldots, x_\ell]$ of natural numbers, no more than

$$3^\ell \cdot x_0 + 3^{\ell-1} \cdot x_1 + \cdots + 3^0 \cdot x_\ell \leq (x_0 + \cdots + x_\ell) \cdot 3^\ell$$

consecutive destruction, climbing, and generation actions can be performed.

This exponential bound cannot be improved significantly without refining the technique: if, as in our case, the initial tuple has the form

$$[s_0, \underbrace{0, \ldots, 0}_{\ell-1}] ,$$

a chain consisting of $s_0 \cdot (2^\ell - 1)$ generation actions followed by $s_0 \cdot 2^\ell$ destructions is *a priori* conceivable. However, as we will now see, a similar chain does not reflect the behavior of any concrete Hyper_unify computation.

4.5 NP-completeness of the problem of finding a single hyperset unifier

In this subsection we prove that Hyper_unify belongs to the complexity class NP. It hence follows that the hyperset unifiability problem is NP-complete, in view of the reduction of 3-SAT to pure hyperset matching seen in Section 4.2.

For the termination proof, an adequately abstract view of the data structure $\mathscr{E}$, $\mathscr{C}$ manipulated by Hyper_unify was provided above by the tuple τ: the decrease actions performed on τ somehow mimicked the basic actions of Hyper_unify. Here, before undertaking the complexity analysis, we must resort to a subtler abstract interpretation.

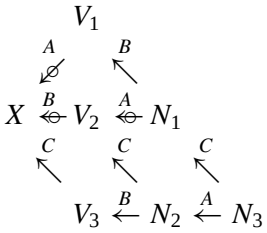
We will represent $\mathscr{E}$ by a multi-graph $\mathscr{G}_{\mathscr{E}}$, and $\mathscr{C}$ by a function $lev : vars(\mathscr{E}) \longrightarrow I\!N$. We momentarily defer the characterization of lev. As for $\mathscr{G}_{\mathscr{E}}$, its constituents are:

- nodes $\mathscr{N}_{\mathscr{E}} = \left\{ V_1, V_2 \ : \ V_1 = \{ \_ \mid V_2 \} \text{ in } \mathscr{E} \right\}$;
- labels $\mathscr{L}_{\mathscr{E}} = \left\{ Y \ : \ \_ = \{ Y \mid \_ \} \text{ in } \mathscr{E} \right\}$;
- directed labeled edges $\mathscr{A}_{\mathscr{E}} = \left\{ V_1 \stackrel{Y}{\leftarrow}_2 : \ V_1 = \{Y|V_2\} \text{ in } \mathscr{E} \right\}$.

Conceptually, the pair $\mathscr{G}_{\mathscr{E}}$, lev gets updated at every turning point, like τ. As we have already proved termination, we are in a position to refer to the final value of either of these structures. (Obviously, $\mathscr{G}_{\mathscr{E}}$ and lev, as well as their final values, depend on the course of a non-deterministic computation.)

The following two examples are aimed at conveying an intuitive grasp of why each non-deterministic branch contains a number of phases (equivalently, a number of (b.4) actions) polynomially related to the size $v_0 + s_0$ of $\mathscr{E}$.

Example 3 Starting with the set $\mathscr{E} = \{ X = \{A|V_1\}, X = \{B|V_2\}, X = \{C|V_3\} \}$ of membership constraints, a branch that maximizes the number of stages will yield the following final value for $\mathscr{G}_{\mathscr{E}}$. Indicating by the arrow $\overset{\ominus}{\leftarrow}$ any edge removed by action (b.4), we have

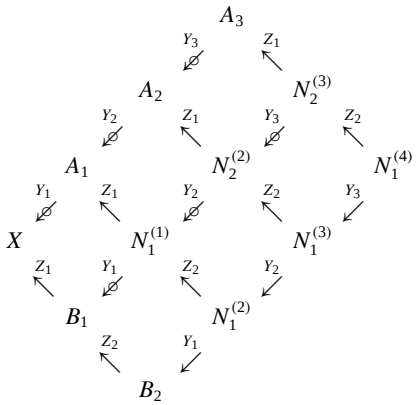


Notice that the unification algorithm does not generate a variable corresponding to each subset of a three-element set, a reason being that once the edge $X \overset{A}{\leftarrow} V_1$ has been removed, it becomes impossible to re-exploit it in conjunction with $X \overset{C}{\leftarrow} V_3$ to fire an action (b.4). □

Example 4 A branch that maximizes the number of phases for the Herbrand system

$$\mathscr{E} = \{ X = \{ Y_1 \mid A_1 \},\ A_1 = \{ Y_2 \mid A_2 \},\ A_2 = \{ Y_3 \mid A_3 \},\\ X = \{ Z_1 \mid B_1 \},\ B_1 = \{ Z_2 \mid B_2 \} \ \},$$

expressing the set unification problem $\{Y_1, Y_2, Y_3 \mid A_3\} = \{Z_1, Z_2 \mid B_2\}$, will yield the following final value for $\mathscr{G}_{\mathscr{E}}$:



□

Inspection of these examples leads to the following observations:

Remark 5

1. If one subtracts the initial number of edges in $\mathscr{G}_{\mathscr{E}}$ from the number of edges in the final $\mathscr{G}_{\mathscr{E}}$—counting also those that were removed by (b.4)— one obtains twice the overall number $\overline{\beta}$ of (b.4) actions.

2. The evolution of $\mathcal{G}_{\mathcal{E}}$ progressively limits the possibility to apply action (b.4): calling into play again the decrease actions of Section 4.4, this means that generation actions (the only potential source of exponentiality) become less and less viable.
   Sometimes the termination guarantee does not come from this phenomenon, but, rather, from the presence of $\mathcal{C}$-constraints. This does not emerge from the two examples just seen, but can be seen from studying a system like the following: $X = \{ \_ \mid A \}, X = \{ \_ \mid B \}, Y = \{ \_ \mid A \}, Y = \{ \_ \mid B \}$.
3. If a node X eventually inserted into $\mathcal{G}_{\mathcal{E}}$ has incoming edges in some phase, at least one incoming edge will always be alive, till the end of the computation.

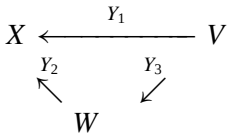
□

It is now time to introduce lev:

Definition 10 *Given the pair $\mathcal{E}, \mathcal{C}$, a* LEVEL–MAPPING *is a function $lev : vars(\mathcal{E}) \longrightarrow I\!N$, satisfying the following conditions:*

- *for all edge $V_1 \stackrel{Y}{\leftarrow} V_2$ of $\mathcal{G}_{\mathcal{E}}$, if $Y \notin V_2$ belongs to $\mathcal{C}$, then $lev(V_2) = lev(V_1) + 1$, otherwise $lev(V_2) \leq lev(V_1) + 1$;*
- *for all equations $V_1 = V_2$ in $\mathcal{E}$, $lev(V_1) = lev(V_2)$.* □

Remark 6 In the absence of negative information, a trivial level-mapping can be obtained by simply setting to 0 all variables. On the other hand, it may be the case that a level-mapping does not exist, as the following example shows:



$Y_3 \notin V,\ Y_2 \notin W$ both belonging to $\mathcal{C}$.

Plainly, if a level-mapping lev for $\mathcal{E}, \mathcal{C}$ exists at all, it is not unique: one may 'tune' its construction so as it meet the condition

$$\text{for all variables} V, \text{lev}(V) \leq \mid L_{\mathcal{E}} \mid$$

(cf. the definition of the L_is in Section 4.4). This new condition —to be taken from now on as part of the definition of a level-mapping— can easily be fulfilled, e.g., by setting $lev(W) = 0$ for at least one node W in each connected component of (the undirected graph underlying) $\mathcal{G}_{\mathcal{E}}$. □

W.r.t. a level-mapping lev, we say that an edge $X \stackrel{Y}{\leftarrow} V$ (or, more generally, an ordered pair X, V) is of level i if $lev(V) = i$.

Definition 11 *Let $\mathcal{E}'$ be the system in solvable form resulting from a (non-deterministic) computation of Hyper_unify with input $\mathcal{E}$: hence any variable generated in such a computation occurs in $\mathcal{E}'$. Let, moreover, lev be a level-mapping for $\mathcal{E}'$ (hence for $\mathcal{E}$). Then,*

- *$\alpha(i)$ stands for the number of edges of level i in $\mathcal{G}_{\mathscr{E}}$. Also, $\overline{\alpha} =_{Def} \sum_{i=0}^{\infty} \alpha(i) = \sum_{i=0}^{|L_{\mathscr{E}}|} \alpha(i)$.*
- *$\beta(i)$ stands for the number of edges of level i introduced in the computation by action (b.4) (regardless of whether such edges survive in $\mathcal{G}_{\mathscr{E}'}$). Let also $\overline{\beta} =_{Def} \sum_{i=0}^{\infty} \beta(i) = \sum_{i=0}^{|L_{\mathscr{E}}|} \beta(i)$.* □

In view of Corollary 2, in order to guarantee the NP-completeness of Hyper_unify, it is sufficient to place polynomial bounds both on the total number $\frac{\overline{\beta}}{2}$ of executions of (b.4) and on the number of actions taking place within a single phase.

Theorem 5 *In a successful branch of $Hyper\_unify(\mathscr{E})$, if no variable X generated by action (b.4) ever joins another variable Y (in the sense that either $X = Y$ or $Y = X$ is put into $\mathscr{E}$), then $\overline{\beta}$ is $O(\overline{\alpha}^3)$.*

Proof. First of all, let us focus on a couple of conditions necessary for an action (b.4) to take place:

- two 'parent' edges of level j concur to the action, where $j+1$ is the level of the two edges to be created—one of these parent edges will be deleted by the action;
- since the parent edges must enter the same node, the latter cannot have a single incoming edge when the action is fired.

Based on these remarks, we derive the following:

- $\beta(0) = 0$ and $\beta(1) \leq 2 \cdot \alpha(0)$.
- The presence of two generated edges at level $j-1$ implies that there is a node at level $j-1$.
  The overall number of generated nodes of level $j-1$ is $\frac{\beta(j-2)}{2}$; hence the assumption that generated nodes never join —implying that nodes, once generated, persist in $\mathcal{G}_{\mathscr{E}}$ retaining an incoming edge— ensures, for $j \geq 2$, that
$$\beta(j) \leq 2 \cdot \left(\alpha(j-1) + \beta(j-1) - \frac{\beta(j-2)}{2} \right).$$

By unfolding these constraints on β, we obtain
$$\beta(j) \leq 2 \cdot \left(\sum_{i=1}^{j} i \cdot \alpha(j-i) \right).$$

Since $| L_{\mathscr{E}} | \leq \alpha$, we conclude:
$$\overline{\beta} = \sum_{j=0}^{|L_{\mathscr{E}}|} \beta(j) \leq 2 \cdot \left(\sum_{j=1}^{|L_{\mathscr{E}}|} \sum_{i=1}^{j} i \cdot \alpha(j-i) \right) = O(\overline{\alpha}^3).$$

■

We are left with the task of showing that the assumption that generated variables never join can be discarded from the statement of Theorem 5. To this end, notice that two variables of level $j+1$ becoming equivalent might generate a situation in which action (b.4) can be performed. However, an inspection of Hyper_unify (cf. proof of Theorem 6 below) shows that at the same time two edges of level j come to coincide. As a consequence, the overall number of actions (b.4) (and hence $\overline{\beta}$) cannot increase.

Theorem 6 *In any computation of Hyper_unify, no more than* $O(\overline{\alpha}^3) = O\left((s_0)^3\right)$ *new variables can be generated.*

Proof. In view of Theorem 5, we only need to show the following fact: if a variable N introduced by action (b.4) as the tail of two edges $V \stackrel{Y}{\leftarrow} N$ and $W \stackrel{Z}{\leftarrow} N$ of level i joins another variable A, then a pair of edges of level i has not been used to generate two edges of level $i+1$.

In fact, in order for N to be unified with A, a series $N = A_1, A_1 = A_2, \ldots, A_n = A$ of equalities must be inferred by Hyper_unify. To get the first equation $N = A_1$, the unification algorithm must perform action (b) with one of the two equations $V = \{ Y \mid N \}$ or $W = \{ Z \mid N \}$, together with an equation $V = \{ C \mid B \}$ or $W = \{ C \mid B \}$. A simple inspection of the four subcases of action (b) shows that (b.1) is the only subcase that introduces an equation $N = B$ without leading to a subsequent Fail_1 (note that $B \equiv A$). This means that in such branch of a computation, the edges $V \stackrel{C}{\leftarrow} B$ (or $W \stackrel{C}{\leftarrow} B$) and $V \stackrel{Y}{\leftarrow} N$ (or $W \stackrel{Z}{\leftarrow} N$) of level i cannot be used to fire action (b.4) and hence to generate two edges of level $i+1$. ■

Corollary 7 (NP-completeness) *Let* $\mathscr{E}$ *be a flat system. Then every single branch generated during the execution of* $Hyper\_unify(\mathscr{E})$ *consists in a number of phases polynomially bounded in* $v_0 + s_0$.

Proof. From Theorem 6 we know that the number k of variables generated by action (b.4) of the algorithm is polynomially bounded in $v_0 + s_0$. The thesis follows from Corollary 2. ■

As already said, for a full NP-completeness proof we must place a bound also on the number of actions within a phase. This is relatively easy, as we know from the last propositions that a polynomial bound exists for the number of new variables and functional symbols (edges) that can be introduced:

- action 1 can be performed ad most once for each variable; the same happens with 2;
- action 3 can be performed *in total* a number of times not exceeding s_0, as noticed in Remark 2.4;
- the number of times action 4 gets executed is bounded by the number of edges;

- action 5 consists of an update of a system of constraints that relate to one another a polynomially bounded number of variables.

4.6 Soundness and completeness of the Hyper_unify algorithm

In this section we will prove that the set unification algorithm Hyper_unify presented above is sound and complete with respect to the axiomatic set theory introduced in Section 2.2. These important properties of the algorithm can be phrased as follows (proof to be supplied later on):

Theorem 8 *Let $\mathcal{E}_1, \ldots, \mathcal{E}_k$ be the systems in solvable form produced as output by $Hyper\_unify(\mathcal{E})$. Then the following holds:*

$$(\mathbf{W}), (\mathbf{E}), (\mathbf{L}), (=), (\mathbf{F}_0), (\mathbf{F}_1) \vdash \left(\mathcal{E} \leftrightarrow \exists Q_1 \cdots \exists Q_m \bigvee_{i=1}^{k} \mathcal{E}_i \right),$$

where $Q_1, \ldots, Q_m$ are all variables in the $\mathcal{E}_i$s that do not occur in $\mathcal{E}$. □

In order to achieve greater generality, we will prove the above result as a consequence of an analogous result concerning a variant procedure named $Hyper\_unify_*(\mathcal{E}, \mathcal{C})$—see Lemma 9 below. Although strictly akin to Hyper_unify, Hyper_unify$_*$ will not be guaranteed to terminate. The only differences between the two procedures are:

- $\mathcal{C}$ is regarded as a new input parameter. Therefore, it shall not be initialized to $\emptyset$ inside Hyper_unify$_*$;
- in actions 3 and (b.4), situations where Hyper_unify places a new pair into $\mathcal{C}$, Hyper_unify$_*$ *is free to do the same or to leave $\mathcal{C}$ unchanged*. (Note that the only remaining action that may affect $\mathcal{C}$, which is action 5, remains as before.)

Preliminary to stating the generalization of Theorem 8, we need a few definitions.

Definition 12 *A* UNIFY–TREE *is a directed unordered, non-empty tree, each of whose nodes ν bears labels $\mathcal{E}_\nu, \mathcal{C}_\nu$ meeting the following conditions:*

- *$\mathcal{E}_\nu$ is a flat Herbrand system;*
- *$\mathcal{C}_\nu$ is a collection of literals of the form $Y \notin Q$, with Y and Q variables.*
- *A relationship reflecting the behavior of Hyper_unify$_*$ holds between $\mathcal{E}_\nu, \mathcal{C}_\nu$ on the one hand and the tuple $\mathcal{E}_{\mu_1}, \mathcal{C}_{\mu_1}, \ldots, \mathcal{E}_{\mu_p}, \mathcal{C}_{\mu_p}$, where $\mu_1, \ldots, \mu_p$ are the distinct children of ν, on the other. The relationship is as follows:*
  If the values of $\mathcal{E}$ and $\mathcal{C}$ are set to $\mathcal{E}_\nu$ and $\mathcal{C}_\nu$ at the beginning of a phase, then, depending on the first action that will affect either of them in the subsequent execution of Hyper_unify$_*$, one has that:

1, 2, 3, 4, 5, (a): $p = 1$ *and* $\mathscr{E}_{\mu_1}$ *and* $\mathscr{C}_{\mu_1}$ *are possible values of* $\mathscr{E}, \mathscr{C}$ *after the execution of the modifying action;*
(b): $p = 4$ *and* $\mathscr{E}_{\mu_i}$ *and* $\mathscr{C}_{\mu_i}$ *are possible values of* $\mathscr{E}, \mathscr{C}$ *after the execution of (b.i);*
none: $p = 0$. *In this case,* ν *will be called a failure or a success leaf in agreement with the kind of exit that takes place (cf. actions Fail_1, Fail_2, and Succeed).*

A FRINGE *of a Unify-tree is a set S of nodes such that: every maximal path of the tree contains at most one node from S, and, if it contains none, it ends with a failure leaf.* □

It should be clear from this definition that for any given pair $\mathscr{E}_*, \mathscr{C}_*$, a Unify-tree whose root is labeled $\mathscr{E}_*, \mathscr{C}_*$ exists. Unify-trees correspond in fact to the parallel executions of $Hyper\_unify_*(\mathscr{E}_*, \mathscr{C}_*)$: there are two sources of parallelism in Hyper_unify$_*$, namely action (b) and some freedom in putting literals into $\mathscr{C}$. However, it is only the branching caused by action (b) that gets represented by a single Unify-tree.

Among others, a Unify-tree originates from the specific execution of Hyper_unify$_*$ that is entirely alike to an execution of Hyper_unify, except for the different initialization of $\mathscr{C}$. In this execution, $\mathscr{C}$ literals are added whenever possible; moreover, the Unify-tree will be finite in this special case (cf. Theorem 4), and its set of success leaves will constitute a fringe.

Lemma 9 *Let* $\mathscr{T}$ *be a Unify-tree, with root* ϱ. *Then, for any fringe S of* $\mathscr{T}$, *the following holds:*

$$(\mathbf{W}), (\mathbf{E}), (\mathbf{L}), (=), (\mathbf{F}_0), (\mathbf{F}_1) \vdash \left((\mathscr{E}_\varrho \wedge \mathscr{C}_\varrho) \leftrightarrow \exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S} (\mathscr{E}_\nu \wedge \mathscr{C}_\nu) \right),$$

where $Q_1, \ldots, Q_m$ *are all the variables in the* $\mathscr{E}_\nu$*s that do not occur in* $\mathscr{E}_\varrho \wedge \mathscr{C}_\varrho$.

Proof. To prove the thesis, since

$$(\mathbf{W}), (\mathbf{E}), (=), (\mathbf{F}_0) \vdash \neg(\mathscr{E}_\mu \wedge \mathscr{C}_\mu)$$

trivially holds for every failure leaf μ, it will suffice to check that every single action of $Hyper\_unify_*$ leads from a node ν to nodes $\mu_1, \cdots, \mu_p$ such that

$$(\mathbf{W}), (\mathbf{E}), (\mathbf{L}), (=), (\mathbf{F}_1) \vdash \left((\mathscr{E}_\nu \wedge \mathscr{C}_\nu) \leftrightarrow \exists Z_1 \cdots \exists Z_\ell \bigvee_{i=1}^{p} (\mathscr{E}_{\mu_i} \wedge \mathscr{C}_{\mu_i}) \right),$$

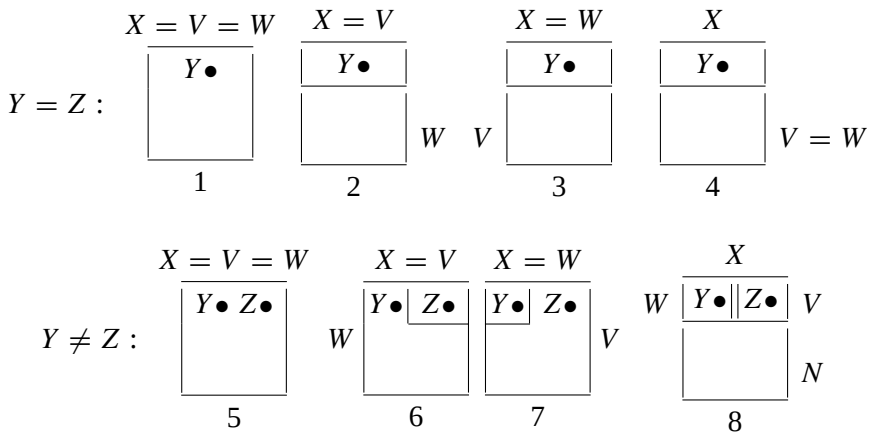
where $Z_1, \ldots, Z_\ell$ are the new variables (if any). In the following, each number on the left indicates the action being analyzed.

1. $X = X$, being true by **(=)**, can be discarded from $\mathscr{E}_\nu$.

2. For any formula φ, $X = Y \wedge \varphi(X, Y)$ is equivalent to $X = Y \wedge \varphi(Y, Y)$ by **(=)**.
3. Viewed as a formula, a zipper $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_{n-1}}{\leftarrow} X_n \overset{Y_n}{\leftarrow} X_0$ yields $X_i = X_0$, hence $Y_i \in X_0$, for $i = 0, \ldots, n$—by (**E**) and (**W**). Accordingly, by assigning X_0 *less* Y_i to K_i, one has both $X_0 = \{Y_i | K_i\}$ and $Y_i \notin K_i$ for all i—by (**W**) and (**L**).
   It follows that
$$(\mathbf{W}), (\mathbf{E}), (\mathbf{L}) \vdash (\mathscr{E}_\nu \wedge \mathscr{C}_\nu) \rightarrow \exists K_0 \cdots \exists K_n \left(\mathscr{E}_{\mu_1} \wedge \mathscr{C}_\nu \wedge \bigwedge_{i=0}^{n} Y_i \notin K_i \right) .$$
   Conversely, as is easily seen,
$$(\mathbf{W}), (=) \vdash (\mathscr{E}_{\mu_1} \wedge \mathscr{C}_\nu) \rightarrow (\mathscr{E}_\nu \wedge \mathscr{C}_\nu) .$$
4. From $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_n}{\leftarrow} X_{n+1} \overset{Y_0}{\leftarrow} X_{n+2}$, by (**W**) and (**E**), it follows that $X_0 = X_1$. In fact, $Y_0 \in X_{n+1}$, whence $Y_0 \in X_n, \cdots, Y_0 \in X_1$. Hence, the insertion of Y_0 into X_1 has no effect and, thanks to **(=)**, the equality $X_0 = \{Y_0 | X_1\}$ can be simplified into $X_0 = X_1$.
5. $\mathscr{E}_\nu \wedge \mathscr{C}_\nu$ is equivalent to $\mathscr{E}_{\mu_1} \wedge \mathscr{C}_{\mu_1}$ by virtue of **(W)** alone.

(a) $X = f(X_1, \ldots, X_n) \wedge X = f(Y_1, \ldots, Y_n)$ is equivalent to $X = f(X_1, \ldots, X_n) \wedge X_1 = Y_1 \wedge \cdots \wedge X_n = Y_n$. One direction follows from ($\mathbf{F}_1$), and the other from **(=)**.

(b) It is easy to infer from (**W**), (**E**), (**L**), and **(=)**, that the following Venn diagrams describe all possible situations under which $e \wedge e' \equiv X = \{Y | V\} \wedge X = \{Z | W\}$ holds:



Moreover, the same axioms yield that:

(b.1) diagrams 1, 4 describe all possibile situations under which ($e \wedge Y = Z \wedge V = W$) holds;

(b.2) diagrams 1, 2, 5, 6 describe all possibile situations under which ($e \wedge e' \wedge X = V$) holds;

(b.3) diagrams 1, 3, 5, 7 describe all possibile situations under which $(e \wedge e' \wedge X = W)$ holds;

(b.4) diagrams 1, 8 describe all possibile situations under which $\exists N\,(e \wedge V = \{Z|N\} \wedge W = \{Y|N\} \wedge Y \notin N \wedge Z \notin N)$ holds.

We conclude from this diagram analysis that the following holds

$$(e \wedge e') \leftrightarrow \Big((e \wedge Y = Z \wedge V = W) \vee (e \wedge e' \wedge X = V) \vee (e \wedge e' \wedge X = W) \vee \exists N\,(e \wedge V = \{Z|N\} \wedge W = \{Y|N\} \wedge Y \notin N \wedge Z \notin N) \Big) .$$

Either one of the literals $Y \notin N$, $Z \notin N$, or both, can be removed from the right-hand side of this bi-implication, because $\exists N\,(e \wedge V = \{Z|N\} \wedge W = \{Y|N\})$ easily yields e'.[9] ■

Proof of Theorem 8 Let us consider the Unify-trees $\mathscr{T}_0, \mathscr{T}_1$ corresponding to the executions of
$Hyper\_unify_*(\mathscr{E}, \emptyset)$ that respectively add $\mathscr{C}$ literals

- whenever possible,
- never.

$\mathscr{T}_0$ corresponds to the execution of $Hyper\_unify(\mathscr{E})$; hence it is finite, by Theorem 4. The preceding lemma, referred to the fringe S_0 consisting of all success leaves of $\mathscr{T}_0$, gives

$$\mathscr{E} \leftrightarrow \exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} (\mathscr{E}_\nu \wedge \mathscr{C}_\nu) .$$

$\mathscr{T}_1$ clearly contains an isomorphic copy $\mathscr{T}'_0$ of $\mathscr{T}_0$: every node μ in $\mathscr{T}_0$ is labeled $\mathscr{E}_\mu, \mathscr{C}_\mu$, while the corresponding node μ' is labeled $\mathscr{E}_\mu, \emptyset$ in $\mathscr{T}'_0$. We consider the fringe S_1 of $\mathscr{T}_1$ consisting of all nodes ν' s.t. ν either belongs to S_0 or is a Fail_1 leaf of $\mathscr{T}_0$. Again by Lemma 9, we have

$$\mathscr{E} \leftrightarrow \exists Q_1 \cdots \exists Q_m \exists Q_{m+1} \cdots \exists Q_{m+k} \bigvee_{\nu' \text{ in } S_1} \mathscr{E}_\nu .$$

Assuming $\exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} \mathscr{E}_\nu$, one readily obtains $\exists Q_1 \cdots \exists Q_{m+k} \bigvee_{\nu' \text{ in } S_1} \mathscr{E}_\nu$, whence $\mathscr{E}$ follows. Conversely, from $\mathscr{E}$, one derives $\exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} (\mathscr{E}_\nu \wedge \mathscr{C}_\nu)$, which entails $\exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} \mathscr{E}_\nu$.

[9] Note that since the pairs $(1, 8)$, $(4, 5)$, $(6, 7)$ of diagrams are incompatible, none of the alternatives (b.1)–(b.4) in Hyper_unify is superfluous.

We conclude with the desired thesis: $\mathscr{E} \leftrightarrow \exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} \mathscr{E}_\nu$. ■

An easy corollary of Theorem 8 is that Hyper_unify solves $\mathscr{E}$ in the sense of Definition 8. As a matter of fact, any solution to $\mathscr{E}$ is also a solution to (at least) one of the $\mathscr{E}_i$s (provided a mapping for the new variables $Q_1, \ldots, Q_m$ is made explicit). As said after Definition 8, the set $\{\mathscr{E}_1, \ldots, \mathscr{E}_k\}$ can be seen as a complete set of *templates of solutions* for $\mathscr{E}$. Unfortunately, the various $\mathscr{E}_i$s are not necessarily independent, as it emerges from the following example:

Example 7 Let $\mathscr{E}$ consist of the equations:

$$X = \{A \mid C\},\ X = \{B \mid D\},\ C = \{B \mid Z\},\ D = \{A \mid Z\},\ Z = \emptyset$$

Two possible non-deterministic computations return the systems:

$$\mathscr{E}_1 = \{A = B, C = D, X = \{B \mid D\}, D = \{B \mid Z\}, Z = \emptyset\},$$
$$\mathscr{E}_2 = \{X = \{A \mid C\}, C = \{B \mid Z\}, D = \{A \mid Z\}, Z = \emptyset, N = Z\}$$

(to get the former, start by applying action (b.1); for the latter, apply first (b.4) —which generates a new variable N— and then (b.1) twice). It is clear that $\mathscr{E}_2$ covers all the solutions of $\mathscr{E}_1$ plus those in which A and B are mapped into different hypersets. Thus, $\mathscr{E}_1$ is more general than $\mathscr{E}_2$. □

Algorithms for detecting overlaps between solvable form systems will hopefully emerge from ongoing research. More specifically, let us assume that $E_0, \ldots, E_{n+1}$ are solvable form systems and that $\mathscr{W}$ are the 'relevant' variables they share. That is, one regards solutions γ_i, γ_j for E_i, E_j as being the 'same' solution when $\gamma_i|_{\mathscr{W}} = \gamma_j|_{\mathscr{W}}$.One would like to

1. determine whether every solution to E_0 also solves at least one of $E_1, \ldots, E_{n+1}$;
2. determine whether all solutions of E_0 also are solutions to E_1;
3. determine whether E_0 and E_1 have any solutions in common.

A solution to 2 seems to be at hand, whereas the more general problems 1 and 3 remain as yet somewhat puzzling.

5 Hyper_unify as a Set-Constraint Manager

We have seen in the preceding section that any system $\mathscr{E}$ involving $with$, $\emptyset$, and free functors, can be rewritten, equivalently, as a finite disjunction $\mathscr{E}_1, \ldots, \mathscr{E}_k$ of systems in solvable form.

We now extend this result by showing that the unification algorithm introduced in Section 4 is in fact sufficiently general to support the treatment of the operator $less$ (to be interpreted in agreement with ($\mathbf{L}_{0,1}$) of Section 2.2). In the

new broader context, however, the negative information contained in $\mathscr{C}$ must be retained as output of the system.

In order to handle flat systems of equations that involve $less$, we might choose to install into Hyper_unify the explicit treatment of pairs e, e' of equations of the following forms:

$$\begin{array}{ll} e \equiv X = V\ less\ Y, & e' \equiv X = W\ less\ Z; \\ e \equiv X = V\ less\ Y, & e' \equiv X = \{\, Z \mid W \,\}; \\ e \equiv X = V\ less\ Y, & e' \equiv X = f(Y_1, \ldots, Y_n), \\ & \text{where } f \not\equiv less\,,\ f \not\equiv with\,. \end{array}$$

A simpler line of attack is to resort to the following non-deterministic preprocessing technique:

for every equation $V = X\ less\ Y$, add to $\mathscr{E}$ either

1. $X = \{\, Y \mid V \,\}$; or
2. $X = V$;

moreover, do $\mathscr{C} := \mathscr{C} \cup \{\, Y \notin V \,\}$.

Clearly, 1 reflects the case when $Y \in X$ holds, while 2 corresponds to the case $Y \notin X$. The correctness of our approach is based on the following lemma:

Lemma 10 (**W**), (**E**), (**L**) $\vdash V = X\ less\ Y \leftrightarrow$

$$\big(\,(\,Y \notin V \wedge X = \{\, Y \mid V \,\}\,) \vee (\,Y \notin X \wedge V = X\,)\,\big)\,.$$

□

It is easy to modify the proof of Theorem 8 so as to show that the input system $\mathscr{E}$ is provably equivalent to the disjunction $\bigvee_i \exists N_{i_1} \cdots \exists N_{i_{k_i}} (\mathscr{E}_i \wedge \mathscr{C}_i)$ taken over all pairs $\mathscr{E}_i, \mathscr{C}_i$ that result from the successful branches, and $N_{i_1}, \ldots, N_{i_{k_i}}$ denote all variables, present in $\mathscr{E}_i \wedge \mathscr{C}_i$ but not in the original $\mathscr{E}$, introduced along the i-th such branch.

Somewhat disturbingly, such conjunctions are *not* guaranteed to be useful in general. Consider the following *unsatisfiable* system: $\mathscr{E} = \big\{X = \{\, Y,\ X \,\}, Y = \{\, X \,\}, Z = \{\, B \mid A \,\}, Z = X\ less\ Y\big\}$.[10] By using the rewriting of $less$ (case 1.) and two applications of action (b.1) of Hyper_unify we obtain the system $\big\{B = X = \{\, Y,\ X \,\}, Z = Y = \{\, X \,\}, A = \emptyset\big\}$ together with the constraint $Y \notin Z$. The system alone is satisfiable and, by Aczel's **AFA**, implies $X = Y = Z$. Hence, its conjunction with the constraint $Y \notin Z$ placed into $\mathscr{C}$ by the preprocessor, contradicts, very much like $\mathscr{E}$ does, **AFA**—in our restricted axiomatization of Section 2.2 it contradicts **(H)**. An approach to eliminate useless disjuncts would be to exploit the decision test specified in Section 6 of [28], but the latter works only in a context where free functors are not allowed to occur. Therefore, given $\mathscr{E}_i$ in solvable form and $\mathscr{C}_i$, we test for satisfiability the following formula, implicitly keeping into account the freeness axioms:

[10] For the sake of readability, we are not rewriting $\mathscr{E}$ into equi-satisfiable flat form.

$$\mathscr{C}_i \cup \Big\{ \text{equations of the form } X = Y \text{ or of the form}$$
$$X = \{Y|V\} \text{ in } \mathscr{E}_i \Big\} \cup \mathscr{F}_0(\mathscr{E}_i) \cup \mathscr{F}_1(\mathscr{E}_i) \cup \mathscr{U}(\mathscr{E}_i)\ ,$$

where the formulae placed in $\mathscr{F}_0(\mathscr{E}_i)$, $\mathscr{F}_1(\mathscr{E}_i)$, and $\mathscr{U}(\mathscr{E}_i)$, as indicated below, force the correct interpretation w.r.t. the axioms $(\mathbf{F}_0)$, $(\mathbf{F}_1)$, and $(\mathbf{U}_{0,1})$, respectively:[11]

- $\mathscr{F}_0(\mathscr{E}_i) =_{\text{Def}} \{\, X \neq Y \,:\, X = f(V_1, \ldots, V_m), Y = g(Z_1, \ldots, Z_n)$ both in $\mathscr{E}_i$, and $f \not\equiv g \,\}$.
- $\mathscr{F}_1(\mathscr{E}_i) =_{\text{Def}} \bigcup_{f \in \Sigma \setminus \{with\}} \mathscr{F}_1(f, \mathscr{E}_i)$, where for each f, assuming that $X_1 = f(Y_1^{(1)}, \ldots, Y_n^{(1)}), \ldots, X_k = f(Y_1^{(k)}, \ldots, Y_n^{(k)})$ are all equations of the form $X = f(Y_1, \ldots, Y_{ar(f)})$ in $\mathscr{E}_i$, we have
$$\mathscr{F}_1(f, \mathscr{E}_i) =_{\text{Def}} \left\{ \left(\bigwedge_{h=1}^{n} Y_h^{(i)} = Y_h^{(j)} \right) \leftrightarrow X_i = X_j \,:\, 0 < i < j \leq k \right\}\ .$$
- Let $\{X_1 = t_1, \ldots, X_m = t_m\}$ be the entire set $\bigcup_{f \in \Sigma \setminus \{with\}} \{$ equations of the form $X = f(Y_1, \ldots, Y_{ar(f)})$ in $\mathscr{E}_i \}$; then
$$\mathscr{U}(\mathscr{E}_i) =_{\text{Def}} A = \{A_1, \ldots, A_m\} \wedge \bigwedge_{\_ = \{W|\_\} \text{ in } \mathscr{E}_i} W \notin A \ \wedge \bigwedge_{j=1}^{m} X_j = \{A_j\}$$
where $A, A_1, \ldots, A_m$ are new variables not occurring in $\mathscr{E}_i$.

The formulae in $\mathscr{U}(\mathscr{E}_i)$ reflect the fact that different uninterpreted terms must denote different colors, by introducing sets $A_1, \ldots, A_m$ suitably witnessing differences.

It is easy to prove that the above translation preserves satisfiability.

6 Type-Finding Through Hyperset Unification

This section is devoted to illustrating, through an example, how hyperset unification can be exploited for the task (more engaging than type-checking) of *type-finding*. As in Chapter 6 of [3] —with slight adaptations to our needs— we can inductively define TYPE EXPRESSIONS as follows:

1. a basic type (such as boolean, char, integer, real) is a type expression;
2. there are type variables at will, each of them being a type expression;
3. if t is a type expression and n_1, n_2 are natural numbers, then $array(\langle n_1, n_2 \rangle, t)$ is a type expression;

[11] A set $\{\varphi_1, \cdots, \varphi_\ell\}$ of formulae is being identified with the conjunction $\bigwedge_{j=1}^{\ell} \varphi_j$, as usual.

4. if $t_{11}, \ldots, t_{1m_1}, \ldots, t_{n1}, \ldots, t_{nm_n}$ are type expressions and $i_1, \ldots, i_n$ are field names (i.e. identifiers), then $record(\{\langle i_1, \{t_{11}, \ldots, t_{1m_1}\}\rangle, \ldots, \langle i_n, \{t_{n1}, \ldots, t_{nm_n}\}\rangle\})$ is a type expression;
5. if t is a type expression, then $pointer(t)$ is a type expression.

Analogues of type expressions in our sense —and, in fact, of our *ground* type expressions— can be found in the declarations of a most typical programming language such as Pascal.[12] We may think, for example, that the Pascal declaration type vector = array [1..10] of char binds the identifier vector to the type expression $array(\langle 1, 10\rangle, char)$. The rationale for admitting type variables in the context of type-finding is that the problem, here, is to detect from the way constructs are used in a program the types of program variables whose declarations are missing. An automatic inference process aimed at solving this, must somehow represent partially specified types. *Hollow* types, that is, type expressions involving variables, represent incompletely known types. Moreover, they can represent polymorphic data types.

Sets serve two purposes in the type language. As argument of the record constructor, a set reflects our intention to attribute no significance to the ordering of fields; for instance, both of the Pascal types

```
record                    record
    next : list;              data : char;
    data : char               next : list
end                       end
```

would be rendered by the same type expression in our framework.

Moreover, sets conveniently represent the union of types (the very same idea was exploited, e.g., in [33, 34]). In this brief discussion type union has a limited import, as we are using it only for multiply-valued record fields.

For recursive data types (such as lists or trees), a representation by rational terms is appropriate. For such a representation, Herbrand systems are, of course, very convenient.

A *type constraint* is a system of equations interrelating type expressions. Among others, a *type system* comprises rules for associating type constraints with the various parts of a program: it is in this fashion that specific instances of the type-finding problem are generated.

Figure 5 illustrates how a type solver works. For the sake of simplicity, we are only considering straight code inside a single procedure body, and our type analysis will be based on the following assumptions:

- next is a keyword, whose meaning is: the expression $\langle id\rangle\uparrow.next$ has the same data type as $\langle id\rangle$;
- the data type of the keyword nil matches any pointer data type.

[12] Our ongoing illustration is based on a Pascal-like language only for the purpose of readability: it is well-known that the applications of type-finding based on unification go well beyond the tiny example to be elaborated here, and that this approach does not presuppose the language, to which type-finding is applied, to be imperative.

Pascal-like program	Type constraint
$new(p);$	$\Rightarrow P = pointer(P_1),$
$p\uparrow.next := nil;$	$\Rightarrow P_1 = record(\{\langle next, \{P\}\rangle \mid R_1\}),$
$p\uparrow.mode := true;$	$\Rightarrow P_1 = record(\{\langle mode, \{boolean \mid C_1\}\rangle \mid R_2\}),$
$p\uparrow.date := 1965;$	$\Rightarrow P_1 = record(\{\langle date, \{integer \mid D_1\}\rangle \mid R_3\}),$
$new(q);$	$\Rightarrow Q = pointer(Q_1),$
$q\uparrow.mode := false;$	$\Rightarrow Q_1 = record(\{\langle mode, \{boolean \mid C_2\}\rangle \mid R_4\}),$
$q\uparrow.date := p;$	$\Rightarrow Q_1 = record(\{\langle date, \{P \mid D_2\}\rangle \mid R_5\}),$
$q\uparrow.next := p;$	$\Rightarrow Q_1 = record(\{\langle next, \{Q\}\rangle \mid R_6\}),$
	$P = Q$

Fig. 5. A type system in action

$$P = pointer(record(\{\langle next, \{P\}\rangle, \langle mode, \{boolean \mid C_1\}\rangle, \langle date, \{integer, P \mid D_1\}\rangle, \mid R_1\}))$$

next	$\circ\!\rightarrow$	*next*	$\circ\!\rightarrow$
mode	*boolean*	*mode*	*boolean*
date	*integer*	*date*	$\circ\!\rightarrow$

Fig. 6. A solution to the type constraint

P and Q represent the types of p and q, respectively.

Figure 6 shows a system in solvable form returned by the Hyper_unify algorithm, upon elaboration of the type constraint above. One observes that recursive types are modeled through membership cycles.

Acknowledgments. We are grateful to Giorgio Levi, who encouraged us in pursuing to an end the research reported in this paper. We are much obliged to Gianfranco Rossi, whose qualified contribution emerges from a large part of the paper. Davide Aliffi and Enrico Pontelli actively contributed to the early phases of the discussion. Angelo Monti acted as a consultant on the complexity issues related to finite automata. Partial funding came from the Italian National Research Council (CNR coordinated project SETA), from Compulog 2 (Esprit project 6810), from MURST 40% projects (*Calcolo algebrico e simbolico, Logica matematica e applicazioni, Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software*), and from MURST 60% (*Programmazione logica con insiemi*).

References

1. Aczel., P. Non-well-founded sets. Lecture Notes, Vol. 14 Center for the Study of Language and Information, Stanford, 1988
2. Aho, A. V., Hopcroft, J. E., Ullman, J. D.: The Design and Analysis of Computer Algorithms. Reading Mass: Addison-Wesley, 1974

3. Aho, A. V., Sethi, R., and Ullman, J. D. Compilers: Principles, Techniques, and Tools. Reading Mass: Addison-Wesley, 1985
4. Arenas-Sánchez, P., Dovier, A.: A Minimality Study for Set Unification. The Journal of Functional and Logic Programming 7, pp. 1–49, 1997
5. Baader, F., and Schulz, K. U. Unification Theory. In: Bibel W. Schmitt P.H. eds., Automated Deduction: a basis for applications, Vol 1, pp. 211–243, Hingham MA: Kluwer 1998
6. Barwise, J., Moss, L.: Hypersets. The Mathematical Intelligencer, 13(4), 31–41 (1991)
7. Barwise, J., Moss, L.: Vicious Circles: On the Mathematics of Non-Wellfounded Phenomena, Lecture Notes, Center for the Study of Language and Information. Stanford, 1996
8. Clark, K. L. Negation as Failure. In: Gallaire H. and Minker, J. (eds.), Logic and Databases, pp. 293–321 New York: Plenum Press, 1978
9. Colmeraurer, A. Prolog and Infinite Trees. In: Clark, K. L. Tärnlund, S.-Å . (eds.), Logic Programming, pp. 231–251 New York: Academic Press, 1982
10. Comon, H., Dauchet, M., Gilleron, R., Lugiez, D., Tison, S., Tommasi, M. Tree automata techniques and applications (Forthcoming book)
11. Cortesi, A., Dovier, A., Quintarelli, E., Tanca, L.: Operational and Abstract Semantics of a Query Language for Semi-Structured Information. In: Fraternali, P. Geske, U. Ruiz, C. Seipel, D. (eds.), Proceedings of 6th International Workshop on Deductive Databases and Logic Programming DDLP'98, p 127–139. GMD Report 22, June 1998
12. Davis, M. D., Sigal, R., and Weyuker, E. J. Computability, complexity, and languages - Fundamentals of theoretical computer science. Computer Science ad scientific computing, New York: Academic Press, 1994
13. Dovier, A. Computable Set Theory and Logic Programming. PhD thesis, University of Pisa, 1996
14. Dovier, A., Omodeo, E. G., Pontelli, E., Rossi, G.-F. {log}: A Language for Programming in Logic with Finite Sets. The Journal of Logic Programming, 28(1), 1– 44, 1996
15. Garey, M. R., Johnson, D. S. Computers and Intractability — A Guide to the Theory of NP-Completeness. New York: Freeman 1979
16. Gecseg, F., Steinby, M. Tree automata. Akadmiai Kiado, Budapest, 1984
17. Hopcroft, J.E. An $n \log n$ algorithm for minimizing states in a finite automaton, In Kohavi, Z., Paz, A. (eds.), Theory of Machines and Computations. pp. 189–196. New York Academic Press 1971
18. Hibti, M. Décidabilité et complexité de systèmes de contraintes ensemblistes. PhD thesis N.464, Université de Franche-Comté, 1995
19. Inverardi, P., and Nesi, M. Deciding Observational Congruence of Finite-State CCS Expressions by Rewriting, Theoretical Computer Science, 139(1 & 2), 315–354 (1995)
20. Jones, N. D., Lien, Y. E. New problems complete for nondeterministic log space, Mathematical Systems Theory, 10, 1–17 (1976)
21. Kapur D., Narendran P. NP-Completeness of the Set Unification and Matching Problems. In: Siekmann J. H. (ed.), CADE 1986, pp. 489-495 LNCS 230, Berlin: Springer, 1986
22. L isitsa, A. P., Sazonov, V. Yu: Bounded hyperset theory and web-like data bases. In: Gottlob, G. Leitsch, A. Mundici, D. (eds), Computational Logic and Proof Theory, 5th Kurt Gödel Colloquium, LNCS, 1289 p 172–185, Berlin: Springer 1997
23. LLloyd, J. W. Foundations of Logic Programming. Berlin: Springer 1987
24. Maher, M. J. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In Proceedings of 3rd Symposium Logic in Computer Science Edinburgh, 1988, pp. 349–357
25. Martelli, A., Montanari, U. An efficient unification algorithm. ACM Transactions on Programming Languages and Systems 4, 258–282, (1982)
26. Martelli, A., Rossi, G. Stepwise Development of an Algorithm for Unification over Infinite Terms. Computers and Artificial Intelligence 9(3), 209–239 (1990)
27. Mendelson, E. Introduction to Mathematical Logic. Princeton, NJ; Van Nostrand 1979

28. Omodeo, E. G., Policriti, A. Solvable set/hyperset contexts: I. Some decision procedures for the pure, finite case. Comm. Pure Appl. Math., 48(9/10) (special issue dedicated to J. T. Schwartz), 1123–1155 (1995)
29. Omodeo, E. G., and Policriti, A. Decision procedures for set/hyperset contexts. In: Miola, A. (ed.), Design and implementation of symbolic computation systems (1993), Vol. 722 of Lecture Notes in Computer Science, 722 pp. 192–215, Berlin: Springer
30. Shepherdson, J. C. Negation in Logic Programming. In: Minker, J. (ed.), Foundations of Deductive Databases and Logic Programming, pp. 19–88, Los Altos, CA: Morgan Kaufmann 1988
31. Shmueli, O., Tsur, S., Zaniolo, C. Compilation of Set Terms in the Logic Data Language (LDL). Journal of Logic Programming 12 (1), 89–120 (1992)
32. Stolzenburg, F. Membership-Constraint and Complexity in Logic Programming with Sets. In: Baader, F. Schulz, K. U. (eds.), Proc. First Int'l Workshop on Frontier of Combining Systems, p 285–302, Hingkham MA: Kluwer 1996
33. Tenenbaum, A. M. Type determination for very high level languages. Courant Computer Science Rep. 3, Courant Inst. of Mathematical Sciences, New York University New York, 1974
34. Weiss, G. Recursive data types in SETL: automatic determination, data language description, and efficient implementation. Computer Science Dept., Tech. Rep. 201, Courant Inst. of Mathematical Sciences, New York University, New York, 1986