Decidability Results for Sets with Atoms*

Agostino Dovier† Andrea Formisano§,‡ Eugenio G. Omodeo§

Abstract

Formal Set Theory is traditionally concerned with *pure* sets; consequently, the satisfiability problem for fragments of set theory was most often addressed (and in many cases positively solved) in the pure framework. In practical applications, however, it is common to assume the existence of a number of primitive objects (sometimes called *atoms*) that can be members of sets but behave differently from them. If these entities are assumed to be devoid of members, the standard extensionality axiom must be revised; then decidability results can sometimes be achieved via reduction to the pure case and sometimes can be based on direct goal-driven algorithms. An alternative approach to modeling atoms, that allows one to retain the original formulation of extensionality, was proposed by Quine: atoms are self-singletons. In this paper we adopt this approach in coping with the satisfiability problem: we show the decidability of this problem relativized to $\exists^*\forall$-sentences, and develop a goal-driven unification algorithm.

Key words: *Set-hyperset theories, Satisfiability problem, Syllogistics, Prenex sentences, Quantifier elimination, Unification.*

Introduction

> *··· a transformational programming methodology that includes a fully operational set-theoretic proof checker ··· Based on our experience with these derivations, we believe that mechanical verification of program transformations is the most important missing ingredient to the successful use of transformational programming as part of a viable program development technology.* J.-P. Keller and R. Paige [KP95]

Many practical problems can be tersely formulated within Set Theory which, indeed, constitutes a powerful framework for both formalization and mechanization of mathematical notions [Far01, PG96, Gor96]. Consequently, specialized inference techniques for axiomatic set theories, and, in particular, decision procedures for fragments of them, are central goals in several fields of research. Fields which include automated deduction [PS95, Qua92, Bel99, FOT01], interactive theorem proving [Mizar, Pau97, Saa97], symbolic model checking [BC*92], hardware and software specification [Lam02, MS96], certified hardware design [Ras95, Ras96], program analysis and verification [Pau95, Noë93], rapid prototyping and certified program derivation [SDDS86, DF89, KP95], declarative programming [HL94], to mention some among many.

In weak set theories where simple notions such as union, intersection, difference, and so on (cf. Figure 1) are involved, a crucial property holds. Namely, satisfiability of set-theoretic constraints can be reduced to satisfiability of $\exists^*\forall$-sentences that predicate over equality and membership only. This is possible because all of the basic constructs/constraints of (weak) set theory can be expressed as formulas of this class. To be more specific, let us consider a conjunction ψ

*Research funded by MURST/MIUR project *Aggregate- and number-reasoning for computing: from decision algorithms to constraint programming with multisets, sets, and maps.* This research benefited from collaborations fostered by the European action COST n. 274 (TARSKI, see `www.tarski.org`).

†Università di Udine, Dipartimento di Matematica e Informatica. Email: `dovier@dimi.uniud.it`

‡Università di Perugia, Dipartimento di Matematica e Informatica. Email: `formisano@di.univaq.it`

§Università di L'Aquila, Dipartimento di Informatica. Email: `omodeo@di.univaq.it`

of literals of the forms $x = \{y, z\}$, $y \subseteq x$, $x \cap y = \emptyset$, $x = \{\emptyset\}$, $x = y \setminus z$, and $x = y \cup z$. The satisfiability of ψ is equivalent to the satisfiability of a sentence $\exists x_1 \cdots \exists x_n \forall v\, \varphi$ where $x_1, \ldots, x_n$ are all the variables occurring in ψ and φ is a Boolean combination of literals of the forms $x_i = x_j$ and $x_i \in x_j$. Such a sentence can be obtained by first exploiting the specifications shown in Figure 1, then factoring out the universal quantifier, and then bounding existentially all variables other than v. Several results concerning this class of formulas are well-known [COP01].

Definiendum	*Definiens*
$x = \{y, z\}$	$\forall v \left(y \in x \wedge z \in x \wedge \big(v \in x \rightarrow (v = y \vee v = z)\big) \right)$
$y \subseteq x$	$\forall v \left(v \in y \rightarrow v \in x \right)$
$x \cap y = \emptyset$	$\forall v \left(v \in x \rightarrow v \notin y \right)$
$x = \{\emptyset\}$	$x = \{y, y\} \wedge y \cap y = \emptyset$
$x = y \setminus z$	$\forall v \left(\big(v \in y \rightarrow (v \in x \leftrightarrow v \notin z)\big) \wedge (v \in x \rightarrow v \in y) \right)$
$x = y \cup z$	$y \subseteq x \wedge z \subseteq x \wedge \forall v \left(v \in x \rightarrow (v \in y \vee v \in z) \right)$

Figure 1: Translation of basic set-constructs

Gogol showed [Gog78] that the classical set theory ZFC is complete with respect to $\forall^*\exists$- / $\exists^*\forall$-sentences, namely that for every sentence φ of these classes either ZFC $\vdash \varphi$ or ZFC $\vdash \neg\varphi$ holds. With this proof he was one of the few forerunners of a very fruitful stream of research on computable set theory initiated over twenty-five years ago.

Reverse logic revealed that Gogol's decidability result does not depend on strong assumptions (such as the axiom of choice or the subset axioms [End77]), while algorithmic analysis assessed that his $\exists^*\forall$-provability problem belongs to the NP complexity class, and, in particular, it is NP-complete (cf. [COP01, pp. 167–168]).

Subsequent research yielded varied generalizations of Gogol's result. Major improvements have been obtained in

- treating larger and larger collections of formulas in set theory (cf., e.g., [COU02]), by naïve reference to a standard and well-understood universe of sets such as the von Neumann hierarchy, or its sub-hierarchy of the hereditarily finite sets first investigated by Ackermann [Ack37, Lev79];
- seeking a syllogistic decomposition (cf., e.g., [CGO88, OPP93]) of the input formula r, so as to fully classify those set assignments for the existential variables that make r true;
- adjusting the decision algorithm to, or enhancing it [BP96] under, varied axiomatic assumptions, e.g. by shifting the focus from customary set theory to hyperset theory [OP95].

Significant applications of Gogol's result include the design of a unification algorithm for sets that nicely fits into Constraint Logic Programming with Sets [DPPR00]. The latter algorithm was shown in [DPR98] to be generalizable to a variety of set-theoretic contexts, and in particular to a hyperset context [DOP99].

Formal Set Theory is generally concerned with *pure* sets. Namely, members of sets are sets themselves and there exists a unique set devoid of elements (usually denoted by $\emptyset$).

Nevertheless, in practical applications it is common to assume the existence of a number of primitive objects (sometimes called *atoms*), that can be members of sets but, generally, are not supposed to possess elements. Problems related to sets with atoms (also called *hybrid sets*) can be reduced to the pure case using rather standard reductions (see, e.g., [Jec79, Chapter 3, p. 198]). However, when developing a constraint solver to deal with the hybrid case, practical considerations lead to a direct solution of the problems. This is the approach adopted, for instance, in [DPPR00]

where the language $CLP(\mathcal{SET})$ is presented: there atoms are viewed as being standard first-order terms that have no members.

A third approach to the handling of hybrid sets is well-explained in the following passage:

> *Thereupon the question arises how to interpret '$y \in z$' where z is an individual. The convention that first suggests itself and has commonly been adopted in the literature is that in such a case '$y \in z$' is simply false for all y; individuals do not have members. But there is a different convention that proves much more convenient $\cdots$ None of the utility of class theory is impaired by counting an individual, its unit class, the unit class of that unit class, and so on, as one and the same thing.* W. V. Quine [Qui63, pp. 30–31]

The decision algorithm for $\exists^*\forall$-sentences of [Gog78] was already adapted in [OP95] to very weak set/hyperset theories devoid of atoms. In the same frame of mind of [Qui63], in this paper we tune up such a decision algorithm to deal with sets with atoms. Atoms are treated here as self-singleton sets, as suggested above, and this constitutes a novelty; indeed, to date the work in computable set theory treats atoms as member-less entities (which are not sets, and hence differ from $\emptyset$). One can see as a drawback in the usual approach that it forcibly brings into play 'colored' sets (cf. [COP01, pp. 79–81]) in order to ensure a smooth treatment of unification, thus disrupting both elegance and intuitive appeal of the extensionality axiom. The contribution in this paper eliminates such a drawback without limiting the realm of applicability of the revised formal framework. In particular, as a specialization of the $\exists^*\forall$-decision algorithm, we will design and implement a unification algorithm which can deal with terms denoting sets with self-singleton atoms. This result discloses a viable alternative to the approach adopted in Constraint Logic Programming with Sets [DPPR00]. Indeed, it can be seen as a new foundational contribution to the design of theories suitably supporting programming with sets.

In Section 1 we characterize the weak set theory underlying this study. In the subsequent Section 2 we show how to develop a satisfiability decision test for Gogol's collection of sentences in the language used. Then, in Section 3 we develop a goal-driven unification algorithm for our theory, and we report on some experimental results. Conclusions are drawn in Section 4.

1 Set Axioms

In this section we will introduce an axiomatic system which is well-suited for the purposes of the ongoing study. Its axioms can be considered as characterizing

- a subtheory of standard Zermelo-Skolem-Fraenkel (cf. [End77, Jec79]),
- a subtheory of Aczel's hyperset theory (cf. [Acz88, BM96]), and
- a tiny theory of sets ultimately based on atoms.

Despite being rather weak, the theory to be specified will be able to 'judge' every $\exists^*\forall$-sentence r, by entailing either r or $\neg r$ as a theorem.

We will characterize atoms by a monadic predicate ur. Atoms are viewed here, in agreement with [Qui63], as self-singletons in a universe which, save for what regards them, is well-founded by membership. Accordingly,

$$\mathsf{ur}(a) \ \leftrightarrow_{\text{Def}} \ a = \{a\}.$$

A weak theory of sets can be based on the axioms of *extensionality*, *null set*, single-element *addition* and *removal* (see, e.g., [OP95]), and stated as follows using uppercase letters to denote implicit universal quantifiers for variables:

$$\begin{array}{ll}
\textbf{(E)} & \forall v\,(v \in X \leftrightarrow v \in Y) \rightarrow X = Y, \\
\textbf{(N)} & \exists z\,\forall v\; v \notin z, \\
\textbf{(W)} & \exists w\,\forall v\,(v \in w \leftrightarrow v \in X \vee v = Y), \\
\textbf{(L)} & \exists \ell\,\forall v\,(v \in \ell \leftrightarrow v \in X \wedge v \neq Y).
\end{array}$$

To inject atoms into a set theory, usually one needs to restate **(E)** in terms slightly less stringent than the formulation seen above, because for example, if atoms were viewed as entities devoid of members, then any atom and the null set $\emptyset$, although being distinct, would have the 'same' members. A safe way of stating **(E)**, in general, is

$$\forall\, v\, (v \in X \leftrightarrow v \in Y) \rightarrow \big(X \,=\, Y \vee \mathsf{atom}(X) \vee \mathsf{atom}(Y)\big),$$

where 'atom' is perhaps taken as a primitive predicate, unlike our derived predicate ur. An advantage of the 'self-singleton approach' is that **(E)** requires no change.

However, something must be stated about well-foundedness (or non well-foundedness) of sets. A usual way of asserting well-foundedness is via the *regularity* axiom:

$$\textbf{(R)} \qquad \exists\, r\, \forall\, v\, \big((r = X \vee r \in X) \wedge \neg(v \in r \wedge v \in X)\big).$$

We need to revise **(R)** because this formulation would allow us to infer $\neg\exists\, x\ \mathsf{ur}(x)$. Indeed, assuming by contradiction that $\mathsf{ur}(a)$, so that $a \in a$, and instantiating X as $\{a\}$ in **(R)**, the only potential values for r are $\{a\}$ and a; however, they both intersect $\{a\}$, a contradiction which proves our claim.

The revised version **(R′)** of **(R)**, and a *plenitude* axiom (ensuring that there exist atoms at will), are as follows:

$$\begin{array}{ll} \textbf{(R}')\textbf{(R)} & \exists\, r\, \Big(\big(r = X \vee (\neg\, \mathsf{ur}(r) \wedge r \in X)\big) \wedge (\forall\, v \in X)\big(v \in r \;\rightarrow\; \mathsf{ur}(v)\big)\Big), \\ \textbf{(D}_{\in}\textbf{)} & \exists\, u\big(\mathsf{ur}(u) \wedge (\forall\, v \in X)\ (u \notin v)\big). \end{array}$$

For a very weak, and yet useful, theory of sets with atoms, only the axioms **(E)**, **(N)**, **(W)**, and **(L)** need to be added.

Notice that if we were to postulate scarcity $\neg\exists\, x\ \mathsf{ur}(x)$ instead of plenitude, this new **(R′)** would become equivalent to **(R)**. At any rate, even in the new context the rôle of **(R)** remains the one of forbidding the formation of genuine membership cycles:

Lemma 1.1 *It follows from the definition of* ur *and from the axioms* **(E)**, **(N)**, **(W)**, **(L)**, *and* **(R′)**, *that for every natural number* n,

$$\textbf{(A}^{(n)}\textbf{)} \qquad X_0 \in X_1 \in \cdots \in X_n \in X_0 \rightarrow \mathsf{ur}(X_0) \wedge X_0 \,=\, X_1 \,= \cdots =\, X_n.$$

Conversely,

$$\mathsf{ur}(X) \rightarrow X \in X.$$

Proof: (Sketch) After constructing (by **(N)**,**(W)**) the set $X_\star = \{X_0, \ldots, X_n\}$, one can treat the two cases $(\exists\, v \in X_\star)\mathsf{ur}(v)$ and $(\neg\exists\, v \in X_\star)\mathsf{ur}(v)$ separately. Notice that **(R′)** intervenes only in the treatment of the latter case. □

The following law $\textbf{(D}_{\notin}\textbf{)}$, called *anti-diagonal* axiom in [COP01], will be used to show the existence of a decision procedure:

$$\textbf{(D}_{\notin}\textbf{)} \qquad \exists z\, \Big(z \notin z \wedge (\forall x \in Y)(z \notin x)\Big)\ .$$

Unlike in the hyperset theory treated in [COP01], in our present context this law follows from the available axioms:

Lemma 1.2 *The statement* $\textbf{(D}_{\notin}\textbf{)}$ *ensues from* **(E)**, **(N)**, **(W)**, **(L)**, *and* **(R′)**.

Proof: (Sketch) Given Y, we must find a z such that both $z \notin z$ and $\neg\exists x(z \in x \wedge x \in Y)$ hold. In view of $\textbf{(A}^{(1)}\textbf{)}$, it suffices to take $z = Y \setminus \{Y\}$. □

For convenience of the reader, Figure 2 summarizes the laws on sets with self-singletons.

A consequence of $\textbf{(D}_{\in}\textbf{)}$ and $\textbf{(D}_{\notin}\textbf{)}$ which will play a crucial role in our decision technique is the following:

Lemma 1.3 (Diagonalization) *Sets with atoms fulfill the two laws*

$$\textbf{(Y}_{\in}\textbf{)} \qquad (\exists\, y \in y)\Big(\bigwedge_{0<i\leqslant n} (X_i \neq y \wedge y \notin X_i)\Big)$$

and

$$\textbf{(Y}_{\notin}\textbf{)} \qquad \begin{array}{l} \big(\bigwedge_{0<j\leqslant m} \bigwedge_{m<g\leqslant n} X_j \neq X_g \big) \;\rightarrow\; (\exists\, y \notin y)\Big((\bigwedge_{0<i\leqslant n} y \notin X_i) \wedge \\ \qquad (\bigwedge_{0<i\leqslant n} y \neq X_i) \wedge (\bigwedge_{0<j\leqslant m} X_j \in y) \wedge (\bigwedge_{m<g\leqslant n} X_g \notin y) \Big) \end{array}$$

with m, n integers, $0 \leqslant m \leqslant n$.

Proof: To meet **(Y$_{\in}$)** it suffices to choose a y satisfying the condition

$$y \in y \;\wedge\; \big(\forall x \;\in\; \{\, X_1, \ldots, X_n, \{\, X_1 \,\}, \ldots, \{\, X_n \,\}\,\}\big)\big(y \;\notin\; x\big).$$

Note that the existence of such a y is ensured by **(D$_{\in}$)**.

Given sets $X_1, \ldots, X_n$ satisfying the antecedent of **(Y$_{\notin}$)**, one can exploit **(D$_{\notin}$)** and **(N)**, **(W)** to choose $z_0, z_1, \ldots, z_n$ such that $z_0 \notin z_0$, $z_i \notin z_i$,

$$\forall x \left(x \;\in\; \{\, X_1, \ldots, X_n, \{\, X_1 \,\}, \ldots, \{\, X_n \,\}\,\} \;\rightarrow\; z_0 \;\notin\; x \right),$$

and

$$\forall x \left(x \;\in\; X_i \;\vee\; x = \{X_1\} \vee \cdots \vee x = \{X_n\} \;\rightarrow\; z_i \;\notin\; x \right),$$

for $i = 1, \ldots, n$. Exploiting **(N)** and **(W)** again to put $y = \{\, X_1, \ldots, X_m, z_0, \ldots, z_n \,\}$, one has that: y differs from each X_i (because $z_0 \in y, z_0 \notin X_i$); $y \notin y$ (for the opposite would imply $y = z_h$ for some $h \geqslant 0$, whereas $z_h \notin z_h$); $y \notin X_i$ for any X_i (because $z_i \in y$); $X_1, \ldots, X_m$ belong to y whereas $X_{m+1}, \ldots, X_n$ (each one of which is distinct from $X_1, \ldots, X_m$ and from $z_0, \ldots, z_n$) do not. Thus y is as required by the consequent of **(Y$_{\notin}$)**, and **(Y$_{\notin}$)** follows. □

		$\mathsf{ur}(X) \;\leftrightarrow_{\mathrm{Def}}\; \forall v\,(v \in X \leftrightarrow v = X)$
Extensionality	**(E)**	$\forall v\,(v \in X \leftrightarrow v \in Y) \;\rightarrow\; X = Y$
Null-set existence	**(N)**	$\exists z\, \forall v\; v \notin z$
Add an element	**(W)**	$\exists w\, \forall v\,(v \in w \;\leftrightarrow\; v \in X \vee\; v = Y\,)$
Remove an element	**(L)**	$\exists \ell\, \forall v\,(v \in \ell \;\leftrightarrow\; v \in X \wedge v \neq Y)$
Regularity	**(R′)**	$\exists r \Big(\big(r = X \vee (\neg\, \mathsf{ur}(r) \wedge r \in X)\big) \wedge (\forall v \in X)\big(v \in r \rightarrow \mathsf{ur}(v)\big)\Big)$
Plenitude	**(D$_{\in}$)**	$(\exists\, u \in u)(\forall v \in X)\,(u \notin v)$
Acyclicity	**(A$^{(n)}$)**	$X_0 \in X_1 \in \cdots \in X_n \in X_0 \rightarrow \mathsf{ur}(X_0) \wedge X_0 = X_1 = \cdots = X_n$
Anti-diagonality	**(D$_{\notin}$)**	$\exists z \,\big(z \notin z \wedge (\forall x \in Y)(z \notin x)\big)$

Figure 2: Abbreviation, axioms, and laws for a theory of sets with self-singletons

Remark 1.4 A discussion on the von Neumann-Bernays-Gödel theory NBG of sets an classes lies well beyond our current aims. Nevertheless, for the informed reader, we momentarily digress here to observe that adjusting the results of this paper to NBG would presumably require substantial reworking. On the one hand, **(E)**, **(N)**, **(W)**, and **(L)** all hold in the NBG theory (save for the fact that the variable Y in **(W)** must be restricted to sets). However, on the other hand, if one does allow the variables to refer to proper classes as well as sets, then the plenitude axiom **(D$_{\in}$)** would be false when the variable X refers to the class of all sets.

Similar comments apply to the anti-diagonal law, which is the basis for ongoing decidability results. In the context of the NBG theory, **(D$_{\notin}$)** says that for any Y, the Russell class (of all sets that do not belong to themselves) is not a subclass of the union-class $\bigcup Y$ of all members of Y. If Y is taken to be the universal class, this statement is simply false. If Y is restricted to sets, then this law is a theorem in NBG (even without assuming the axiom of regularity): the Russell class, being a proper class, cannot be a subclass of the set $\bigcup Y$. ■

2 A Decision Technique for $\exists^*\forall$-Sentences on Sets

In this section we present a two-phase decision technique to test $\exists^*\forall$-sentences for satisfiability over sets with atoms. First, we show how to reduce the provability problem for $\exists^*\forall$-sentences to the same problem for (equivalent) formulas where universal quantifiers are *bounded*. Secondly, in Sec. 2.2 we describe a decision algorithm for proving or refuting formulas of the latter kind.

2.1 Quantifier-bounding technique

We show a technique for re-expressing any given sentence $\exists\, x_1 \cdots \exists\, x_n\, \forall\, y\, p$ in the predicates $=$ and $\in$, with p devoid of quantifiers and functors, as a sentence which involves bounded universal quantifiers $\forall\, y \in x_j$ instead of the unbounded $\forall\, y$. The three main phases of our technique are as follows.

Atom elimination. Let $\Phi_0 \equiv \exists\, x_1 \cdots \exists\, x_n\, \forall\, y\, p$ be the input formula, where we admit constant symbols denoting atoms. Let $a_1, \ldots, a_k$ be all the distinct constants occurring in p and let $x_{n+1}, \ldots, x_{n+k}$ be new distinct variables. We obtain a formula Φ_1 equivalent to Φ_0 as follows:[1]

$$\Phi_1 \quad \equiv_{\text{Def}} \quad \exists\, x_1 \cdots \exists\, x_{n+k}\, \forall\, y \quad \underbrace{\left(\begin{array}{c} p[a_1 \mapsto x_{n+1}, \ldots, a_k \mapsto x_{n+k}] \wedge \\ \bigwedge_{i=n+1}^{n+k} (x_i \in x_i \wedge \bigwedge_{j=i+1}^{n+k} x_i \neq x_j) \end{array} \right)}_{\tilde{p}}.$$

Quantifier bounding. In this phase we rewrite Φ_1 in the form

$$\Phi_2 \quad \equiv_{\text{Def}} \quad \exists\, x_1 \cdots \exists\, x_{n+k} \quad \left(\begin{array}{c} \bigwedge_{i=1}^{n+k} \tilde{p}[y \mapsto x_i] \wedge \\ \bigwedge_{j=1}^{n+k} (\forall y \in x_j) q_j \wedge \\ (\forall y \in y) r \wedge \\ (\forall y \notin y) s \end{array} \right),$$

where $\tilde{p}$ is the matrix of Φ_1 as indicated in the previous phase description. This rewriting moves the universal quantifier inwards while performing a case-splitting analysis. In particular, we start by considering Φ_1 and observing that $\tilde{p}$ is a propositional combination of atomic formulas $u\, \pi\, v$ with u, v in $\{x_1, \ldots, x_{n+k}, y\}$ and π a predicate symbol in $\{=, \in\}$. For any given assignment of values to the variables $x_1, \ldots, x_{n+k}$ satisfying Φ_1, the formula must hold for each possible value of y. A particular y may be equal to one among $x_1, \ldots, x_{n+k}$, or it may be the case that $y \in x_i$, for any $i = 1, \ldots, n+k$, and so on. We proceed by case analysis, by first splitting $\forall y\, \tilde{p}$ into various mutually exclusive sub-cases and then processing each of them. More precisely, we start by observing that the tautology $A_1 \vee \cdots \vee A_{n+k} \vee (\neg A_1 \wedge \cdots \wedge \neg A_{n+k})$ yields that

$$\vdash \forall x_1 \cdots \forall x_{n+k} \forall y \big(\, y = x_1 \;\vee \cdots \vee\; y = x_{n+k} \;\vee\; \underbrace{(y \neq x_1 \wedge \cdots \wedge y \neq x_{n+k})}_{\alpha} \,\big).$$

Similarly, we have the equivalence

$$\vdash \alpha \leftrightarrow \Big(\;(y \in x_1 \wedge \alpha) \;\vee \cdots \vee\; (y \in x_{n+k} \wedge \alpha) \;\vee\; \underbrace{(y \notin x_1 \wedge \cdots \wedge y \notin x_{n+k} \wedge \alpha)}_{\alpha'} \;\Big),$$

and, moreover, we have that

$$\vdash \alpha' \leftrightarrow \Big(\;(y \in y \wedge \alpha') \;\vee\; (y \notin y \wedge \alpha') \;\Big).$$

[1] By $p[r \mapsto s]$ we denote the formula obtained by replacing all occurrences of r in p by s.

Summing up:

$$\vdash \forall x_1 \cdots \forall x_{n+k} \forall y \left(\begin{array}{l} \bigvee_{i=1}^{n+k} (y = x_j) \vee \\ \bigvee_{j=1}^{n+k} \Big(y \in x_j \wedge \bigwedge_{i=1}^{n+k} (y \neq x_i) \Big) \vee \\ \Big(y \in y \wedge \bigwedge_{i=1}^{n+k} (y \notin x_i) \wedge \bigwedge_{i=1}^{n+k} (y \neq x_i) \Big) \vee \\ \Big(y \notin y \wedge \bigwedge_{i=1}^{n+k} (y \notin x_i) \wedge \bigwedge_{i=1}^{n+k} (y \neq x_i) \Big) \end{array} \right).$$

Hence, by suitably instantiating the tautological scheme

$$\bigvee_{i=1}^{m} A_i \rightarrow \Big(B \leftrightarrow \bigwedge_{i=1}^{m} (A_i \rightarrow B) \Big)$$

with $m = 2 \cdot (n + k + 1)$ and $B \equiv \tilde{p}$, we easily get (by exploiting laws on equality and quantification) $\vdash \Phi_1 \leftrightarrow \Phi_2$, where Φ_2 is in the desired form:

$$\Phi_2 \ \equiv_{\mathrm{Def}} \exists x_1 \cdots \exists x_{n+k} \left(\begin{array}{l} \bigwedge_{i=1}^{n+k} \Big(\tilde{p}[y \mapsto x_i] \Big) \wedge \\ \bigwedge_{j=1}^{n+k} \Big((\forall y \in x_j) \big(\bigwedge_{i=1}^{n+k} (y \neq x_i) \rightarrow \tilde{p} \big) \Big) \wedge \\ (\forall y \in y) \Big(\big(\bigwedge_{i=1}^{n+k} (y \notin x_i) \wedge \bigwedge_{i=1}^{n+k} (y \neq x_i) \big) \rightarrow \tilde{p} \Big) \wedge \\ (\forall y \notin y) \Big(\big(\bigwedge_{i=1}^{n+k} (y \notin x_i) \wedge \bigwedge_{i=1}^{n+k} (y \neq x_i) \big) \rightarrow \tilde{p} \Big) \end{array} \right).$$

Simplifications of the last two conjuncts will be carried out by the subsequent quantifier-elimination phase.

Elimination of unbounded quantifiers. Conjuncts $\tilde{p}[y \mapsto x_i]$ and $(\forall y \in x_j) q_j$, in the formula Φ_2 resulting from the preceding phase, are already in acceptable form. In this phase we get rid of the quantifiers $(\forall\, y \in y)$ and $(\forall\, y \notin y)$ (which restrict y to be an atom or a set proper, respectively), because they fail to be legal bounded quantifiers. The elimination of the quantified conjuncts $(\forall\, y \in y) r$ and $(\forall\, y \notin y) s$ relies on the diagonalization lemma already seen.

The elimination of $(\forall y \in y)$ ("treatment of self-singletons") is relatively easy, and we will examine it before the (slightly more challenging) elimination of $(\forall y \notin y)$.

Treatment of self-singletons proceeds by transforming $(\forall\, y \in y)\, r$ into a formula which no longer involves y. Recall that r has the form

$$\Big(\bigwedge_{i=1}^{n+k} (x_i \neq y \wedge y \notin x_i) \Big) \rightarrow r'.$$

Therefore, since the set-theoretic axioms yield

$$(\forall\, y \in y) \Big(\bigwedge_{i=1}^{n+k} y \notin x_i \ \rightarrow \ \bigwedge_{g=1}^{n+k} x_g \notin y \Big),$$

within r' we can replace all occurrences of the forms $x_g \in y$, and $y \in x_i$, and $x_i = y$ by `false`, all occurrences of the forms $y = y$, and $y \in y$ by `true`, which will reduce r' to a formula r'' devoid of occurrences of y.

In view of the instance

$$(\exists\, y \in y) \Big(\bigwedge_{i=1}^{n+k} (x_i \neq y \wedge y \notin x_i) \Big)$$

of $(\mathbf{Y}_{\in})$, we can conclude that $(\forall\, y \in y)\, r$ is equivalent, under the set-theoretic axioms, to r''.

The elimination of y from the formula $(\forall\, y \notin y)\, s$ will be carried out by exploiting a very similar—though apparently more complex—idea. Let us describe the algorithm for this elimination.

1. Consider the sub-formula
$$(\forall y \notin y)\big(\bigwedge_i (y \neq x_i \wedge y \notin x_i) \rightarrow s'\big)$$
of Φ_2, where s' is a propositional combination of atomic formulas involving only the variables $x_1, \ldots, x_{n+k}, y$ and the predicate symbols $=$ and $\in$. Without loss of generality, we can assume s' to be in conjunctive normal form, namely $s' \equiv D_1 \wedge \cdots \wedge D_d$.
2. This allows us to distribute the $\forall$ over $\wedge$ and to consider formulas of the form
$$\psi \equiv \forall y \Big(y \notin y \wedge \bigwedge_i (y \neq x_i \wedge y \notin x_i) \rightarrow D\Big)$$
separately, where D is a finite disjunction of literals of the form $u\ \pi\ v$, with π in $\{=, \in, \neq, \notin\}$ and u, v variables drawn from $x_1, \ldots, x_{n+k}, y$.
3. The consequent D can be simplified as follows in each ψ, as dictated by the antecedents of the implication:
   (a) The literals $y = y$ and $y \neq y$ are replaced by true and false, respectively.
   (b) The literals of the forms $y \in y$, $y \notin y$, $y \in x_i$, $y \notin x_i$, $y = x_i$, $y \neq x_i$ are replaced by true or false, depending on the antecedents of the implication.
   (c) If at least one of the disjuncts of D has been replaced by true, then the whole ψ can be replaced by true.
   (d) Any disjunct in D in which y does not occur (i.e., any literal of the form $x_i\ \pi\ x_j$, with π in $\{=, \in, \neq, \notin\}$), can be moved outside the scope of the quantifier $\forall y$ in ψ.

   Unless ψ has been reduced to true, let ψ' be the universally quantified formula resulting from performing the steps (3a)–(3d). Moreover, let D' be the "residue" of D within ψ':
$$\psi' \equiv \forall y \Big(y \notin y \wedge \bigwedge_i (y \neq x_i \wedge y \notin x_i) \rightarrow \underbrace{\big(\bigvee_{j \in \mathcal{F}_1} x_j \in y \vee \bigvee_{j \in \mathcal{F}_2} x_j \notin y\big)}_{D'}\Big),$$
   where $\mathcal{F}_1$ and $\mathcal{F}_2$ are subsets of the set $\{1, \ldots, n+k\}$ of indices. Two situations may occur:
   (a) If one between $\mathcal{F}_1$ and $\mathcal{F}_2$ is empty, it is immediate to check that the overall formula ψ is false.
   (b) Otherwise, we replace ψ' by the disjunction $\bigvee_{i \in \mathcal{F}_1, j \in \mathcal{F}_2} x_i = x_j$.

   Correctness of this rewriting step follows from $\mathbf{(Y_{\notin})}$ of the diagonalization lemma.

Example 2.1 As an example of how $\mathbf{(Y_{\notin})}$ enters into play, consider the simple case of the formula:
$$\forall y \big(y \notin y \wedge (y \neq x_1 \wedge y \notin x_1) \wedge (y \neq x_2 \wedge y \notin x_2) \rightarrow x_1 \in y \vee x_2 \notin y\big).$$
Two cases may occur:

- If $x_1 = x_2$ then the consequent of the implication becomes equivalent to $x_1 \in y \vee x_1 \notin y$, hence to true;
- Otherwise, if $x_1 \neq x_2$, by $\mathbf{(Y_{\notin})}$ the whole formula is equivalent to false.

■

2.2 Decision technique for sentences with bounded universal quantifiers

In this section we present a satisfiability decision procedure for finite conjunctions of formulas

$$(\forall y_1 \in w_1) \cdots (\forall y_m \in w_m)\, p$$

where $m \geqslant 0$, $y_1, \ldots, y_m$ are variables distinct from one another and distinct from the variables w_j, and p is a propositional combination of atomic formulas of the two kinds $u = v$, $u \in v$ (u, v variables). Basically, we adapt the technique described in [COP01, pp. 153–160] to sets with self-singleton atoms. As in Sec. 2.1, we also start with constants representing atoms in the input formula and eliminate them at the very beginning.

From now on, we indicate by r the result of atom elimination and by $x_1, \ldots, x_n$ the distinct free variables in r. Moreover, we put $x_0 \equiv_{\text{Def}} \emptyset$.

2.2.1 Formulas without quantifiers

Let us begin by considering the case when r is quantifier-free. Without loss of generality, we can assume that r is a conjunction of literals $u = v$, $u \neq v$, $u \in v$, and $u \notin v$, where each u and v stands for either a variable or $\emptyset$. We determine the least equivalence relation $\sim$ between the symbols $x_0, x_1, \ldots, x_n$ such that

- $x_i \sim x_j$ when $x_i = x_j$ belongs to r;
- $x_{j_0} \sim x_{i_0}$ when there is a membership cycle $x_{i_0} \in x_{i_1}, x_{i_1} \in x_{i_2}, \ldots, x_{i_{h-1}} \in x_{i_h}, x_{i_h} \in x_{i_0}$ in r, and $x_{j_0} \in x_{j_1}$, $x_{j_1} \in x_{j_2}$, $\ldots$, $x_{j_{k-1}} \in x_{j_k}$, and $x_{j_k} \in x_{i_0}$ belong to r, for some h and k.

With $[x]$ we denote the equivalence class of the symbol x modulo $\sim$.

The satisfiability test consists in checking that r contains no literals $u \neq v$ with $u \sim v$; no literals $u \in v$ with $v \sim \emptyset$; and no pair $u \in v$, $u' \notin v'$ of literals with $u \sim u'$ and $v \sim v'$. If the test is successful, then sets $\mathcal{A}(x)$ can be assigned to variables so as to satisfy r, in the manner described below. Let $\mathcal{G} = \langle N, E \rangle$ be the directed graph whose

- set N of nodes consists of the equivalence classes $[x]$ of $\sim$;
- set E of edges consists of all pairs $\langle [x], [y] \rangle$ such that $x' \in y'$ is in r for some $x' \sim x$ and $y' \sim y$.

Notice that $\mathcal{G}$ is acyclic save for self-loops. For each node ν, we shall indicate by $ip(\nu)$ the set

$$ip(\nu) = \{\eta : \langle \eta, \nu \rangle \text{ is in } E\}$$

of all immediate predecessors of ν in $\mathcal{G}$.

To obtain a new graph $\mathcal{G}' = \langle N', E' \rangle$ with $N \subseteq N'$ and $E \subseteq E'$ in which no two nodes ever share the same immediate predecessors, we submit $\mathcal{G}$ to the following enrichment algorithm (to be illustrated in Example 2.3 and Figure 4). We initially put $N' = N$ and $E' = E$. While scanning each node ν in $N \setminus \{[\emptyset]\}$ once, we do the following:

- if $ip(\nu) = ip(\mu)$ for some still unscanned node μ in $N \setminus \{\nu\}$, and moreover $ip(\nu) \neq \{\nu\}$, we insert a new node ν' into N', and the two edges $\langle \nu', \nu' \rangle$, $\langle \nu', \nu \rangle$ into E';
- if in particular $ip(\nu) = \emptyset$ $(= ip([\emptyset]))$, then we also add another new node ν'' to N', and edges $\langle \nu'', \nu'' \rangle$, $\langle \nu'', \nu \rangle$ to E'.

We are thus ready to single out the desired assignment $\mathcal{A}$ for the variables x of r. We define $\mathcal{A}(x) = \Im([x])$, where for each node ν of the enriched graph $\mathcal{G}'$ we put

$$\Im(\nu) = \begin{cases} \text{an atom } a_\nu \text{ uniquely associated with } \nu & \text{if } \langle \nu, \nu \rangle \text{ is an edge of } \mathcal{G}', \\ \{\Im(\mu) : \langle \mu, \nu \rangle \text{ is an edge of } \mathcal{G}'\} & \text{otherwise.} \end{cases}$$

Observe that an $\Im$ complying with this specification exists; indeed, in order to determine its values one can process the nodes according to any topological sort of $\mathcal{G}'$ obtained by ignoring self-loops.

Assuming that $a_{[x]} = b$ whenever the variable originated from the elimination of an atom b in the **Atom Elimination** phase, as described above (see Sec. 2.1, page 6), we easily get the following lemma:

Lemma 2.2 *For $i, j = 0, 1, \ldots, n$, $\mathcal{A}(x_i) = \mathcal{A}(x_j)$ holds if and only if $x_i \sim x_j$. Consequently, $\mathcal{A}$ correctly models all literals in r.*

Proof: The first claim readily follows from the more general statement that $\Im(\nu) \neq \Im(\mu)$ when the nodes ν, μ of $\mathcal{G}'$ are different. This is proved as follows. Consider a topological sort $\nu_1, \ldots, \nu_k$ of $\mathcal{G}'$ obtained by ignoring all self-loops. Without loss of generality, we can assume that all nodes having a self-loop on $\mathcal{G}'$ are $\nu_1, \ldots, \nu_h$ for some $h \leqslant k$. By definition of $\Im$, thanks to the plenitude axiom, $\Im(\nu_i) \neq \Im(\nu_j)$ whenever $i \neq j$ with $i, j \in \{1, \ldots, h\}$. By effect of the enrichment process yielding $\mathcal{G}'$ from $\mathcal{G}$, it follows that for all ν_{i_0}, ν_{i_1} with $i_0, i_1 \in \{h+1, \ldots, k\}$ and $i_0 \neq i_1$, there exists $i_2 \in \{1, \ldots, \max(i_0, i_1)\} \setminus \{i_0, i_1\}$ such that $\langle \nu_{i_2}, \nu_{i_b} \rangle$ is an edge of $\mathcal{G}'$ whereas $\langle \nu_{i_2}, \nu_{i_{1-b}} \rangle$ is not (with $b = 0$ or $b = 1$). Hence, again by definition of $\Im$, we have $\Im(\nu_{i_0}) \neq \Im(\nu_{i_1})$.

Assuming r to be satisfiable, let us prove that $\mathcal{A}$ models all literals in r. Observe that the graph $\mathcal{G}'$ fulfills the following conditions:

- if r entails $x_i = x_j$, then $[x_i] = [x_j]$;
- if r entails $x_i \in x_j$, then the edge $\langle [x_i], [x_j] \rangle$ is in $\mathcal{G}$ and consequently in $\mathcal{G}'$;
- if r does not entail $x_i = x_j$, then $[x_i]$ and $[x_j]$ do not have the same immediate predecessors in $\mathcal{G}'$;
- if r does not entail $x_i \in x_j$, then $\langle [x_i], [x_j] \rangle$ is not an edge of $\mathcal{G}'$;

Moreover, by definition, for all x_i, x_j, $\mathcal{A}(x_i) \in \mathcal{A}(x_j)$ if and only if $\Im([x_i]) \in \Im([x_j])$ if and only if $\langle [x_i], [x_j] \rangle$ is in $\mathcal{G}'$. Hence, $\mathcal{A}$ models all literals in r of the forms $x_i = x_j$ and $x_i \in x_j$, whereas it does not falsify any of the literals of the forms $x_i \neq x_j$ and $x_i \notin x_j$. □

Figure 3 summarizes the above decision algorithm and Lemma 2.4 states its correctness.

```
procedure Find_model(r);
    determine the equivalence relation ∼ induced by r;
    if Trivial_contradiction(r,∼)
        then return unsatisfiable;
        else determine G, G', ℑ, and A;
         return A;
    end_if;
end Find_model;
```

```
procedure Trivial_contradiction(r,∼);
    if there exist u, v, u', v' such that
      u ≠ v occurs in r and u ∼ v
      or   u ∈ v occurs in r and ∅ ∼ v
      or   u ∈ v and u' ∉ v' occur in r and u ∼ v and u' ∼ v'
        then return true;
        else return false;
    end_if;
end Trivial_contradiction;
```

Figure 3: Decision procedure for quantifier-free conjunctions

Example 2.3 Let us consider the following formula r:

$$r \equiv x_1 \in x_2 \wedge x_3 \in x_2 \wedge x_2 \in x_3 \wedge x_1 \in x_4 \wedge x_1 \in x_5 \wedge x_4 \neq x_5 \wedge x_6 \neq x_7 \wedge \emptyset \notin x_6 \,.$$

We readily see that $x_1 \sim x_2 \sim x_3$, because x_2 and x_3 form a membership cycle, and hence $x_1 \sim x_i$ holds for $i = 2, 3$. To see that r is satisfiable we construct the graph $\mathcal{G}$ depicted in Figure 4 and enrich it as explained above. The enrichment process introduces 6 new nodes $\nu'_1, \ldots, \nu'_4$ and ν''_3, ν''_4. By virtue of the definition of $\Im$ and $\mathcal{A}$, we have

$$\begin{array}{ll}
\Im([x_0]) = \emptyset, & \Im(\nu'_i) = a_{\nu'_i}, \quad (\text{for } i = 1, \ldots, 4) \\
\Im([x_1]) = \Im([x_2]) = \Im([x_3]) = a_{[x_1]}, & \Im(\nu''_i) = a_{\nu''_i}, \quad (\text{for } i = 3, 4) \\
\Im([x_4]) = \{a_{[x_1]}, a_{\nu'_1}\}, & \Im([x_5]) = \{a_{[x_1]}, a_{\nu'_2}\}, \\
\Im([x_6]) = \{a_{\nu'_3}, a_{\nu''_3}\}, & \Im([x_7]) = \{a_{\nu'_4}, a_{\nu''_4}\}.
\end{array}$$

$\mathcal{A}$ is then determined accordingly.

Extend, then, the above formula into $r_1 \equiv x_1 \neq x_3 \wedge r$. After computing the equivalence relation $\sim$ for r_1, one immediately detects unsatisfiability, since $x_1 \sim x_3$ whereas $x_1 \neq x_3$ is a conjunct of r_1. ■

Lemma 2.4 *The procedure* Find_model *determines a model of the conjunction r if and only if r is satisfiable.*

Proof: On the one hand, it is immediate to see that if the procedure Trivial_contradiction returns true, then any possible assignment of sets to the variables in r cannot satisfy all conjuncts of r, together with all axioms of Figure 2. On the other hand, Lemma 2.2 ensures that if Trivial_contradiction returns false, then the assignment $\mathcal{A}$ determines a model of r. □

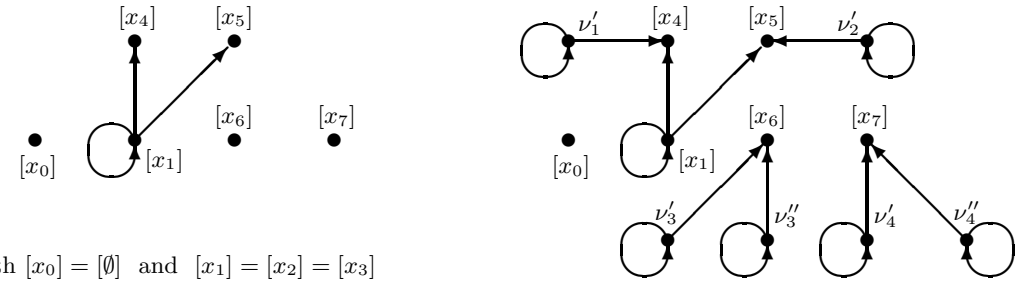


Figure 4: The graph $\mathcal{G}$ and its enrichment $\mathcal{G}'$ for the Example 2.3

2.2.2 Quantified formulas

Let us now move to the case when at least one conjunct of the input formula r is quantified. Again, let $x_0 = \emptyset$ and let $x_1, \ldots, x_n$ be the free variables of r. In order to decide r:

- We obtain a formula r' by conjoining with r the formula

$$\bigwedge_{i=0}^{n-1} \bigwedge_{j=i+1}^{n} \Big(x_i \neq x_j \rightarrow \bigvee_{h=1}^{n} (x_{n+h} \in x_i \leftrightarrow x_{n+h} \notin x_j) \Big) \tag{1}$$

  where $x_{n+1}, \ldots, x_{2 \cdot n}$ are new variables witnessing differences among the sets $x_0, \ldots, x_n$. As we are about to see, if r can be satisfied, then one can satisfy r' too, assigning different sets to the variables $x_{n+1}, \ldots, x_{2 \cdot n}$.

- By proceeding inside-out, we replace in r' each sub-formula of the form $(\forall\, y \in w)p$ by

$$\bigwedge_{i=0}^{2\cdot n}(x_i \in w \rightarrow p[y \mapsto x_i]) \tag{2}$$

  Let $\hat{r}$ be the formula resulting from elimination of all quantifiers from r'. Each disjunct D of a DNF-formula tautologically equivalent to $\hat{r}$ is tested for satisfiability as explained in Sec. 2.2.1. Satisfiability of $\hat{r}$ is then declared whenever one of such tests leads to success; when no disjunct is satisfiable, $\hat{r}$ is declared to be unsatisfiable.

The following lemma guarantees the correctness of this decision procedure.

Lemma 2.5 *Let r be the formula $(\forall y_1 \in w_1) \cdots (\forall y_m \in w_m)\ p$, where $y_1, \ldots, y_m$ are variables distinct from one another and distinct from the variables w_j, and p is a propositional combination of atomic formulas of the kinds $u = v$, $u \neq v$, $u \in v$, and $u \notin v$ (where each u and v stands for either a variable or $\emptyset$). Moreover, let $\hat{r}$ be obtained from r as described above. Then, r is satisfiable if and only if $\hat{r}$ is satisfiable.*

Proof: Consider the rewritings $r \rightsquigarrow r' \rightsquigarrow \hat{r}$ of the formula r as outlined above. The conjunct (1) is introduced in order to obtain a model of r which satisfies **(E)**. Observe that the selection of a suitable element z can always discriminate between two different sets; indeed, $x_i \neq x_j \leftrightarrow \exists\, z\,(z \in x_i \leftrightarrow z \notin x_j)$. More generally, by induction on k, it is easily shown that k distinct elements always suffice to distinguish between $k+1$ different sets (cf. [PPR97]). Hence, under **(E)**, r and r' are equisatisfiable.

Let us prove that the formula r' is satisfiable if and only if so is $\hat{r}$. Clearly, every model of r' is a model of $\hat{r}$. Conversely, assuming that some disjunct D of $\hat{r}$ is satisfiable, an assignment $\mathcal{A}$ satisfying D can be constructed as described before Lemma 2.4. In processing the nodes of $\mathcal{G}$ in order to obtain its enrichment $\mathcal{G}'$, we can treat the $\sim$-classes which contain *none* of the original symbols $x_0, \ldots, x_n$ before the remaining classes/nodes. It turns out easily that no new nodes will be added to the graph after the processing of the nodes in $\{[x_{n+1}], \ldots, [x_{2\cdot n}]\} \setminus \{[x_0], \ldots, [x_n]\}$. Indeed, for all x_{i_0}, x_{i_1} with $i_0, i_1 \in \{0, \ldots, n\}$ and $x_{i_0} \not\sim x_{i_1}$, there exists a $j \in \{n+1, \ldots, 2\cdot n\}$ such that both $x_j \in x_{i_b}$ and $x_j \notin x_{i_{1-b}}$ belong to D (with $b = 0$ or $b = 1$). Consequently, the edge $\langle [x_j], [x_{i_b}] \rangle$ appears in the graph $\mathcal{G}$ corresponding to D, while the edge $\langle [x_j], [x_{i_{1-b}}] \rangle$ does not (cf. Sec. 2.2.1, page 9). Since $\mathcal{G}'$ has no entering edges into any of $[x_0], \ldots, [x_n]$ which are new with respect to those in $\mathcal{G}$, it turns out that $\mathcal{A}(x_i) \subseteq \{\mathcal{A}(x_h)\ :\ h = 0, \cdots, 2\cdot n\}$ holds for each $i \in \{0, \cdots, n\}$. In conclusion, since w is one of $x_0, \ldots, x_n$ whenever we carry out the replacement of $(\forall\, y \in w)p$ by $\bigwedge_{i=0}^{2\cdot n}(x_i \in w \rightarrow p[y \mapsto x_i])$ during the rewriting, the truth values of the two formulas r' and $\hat{r}$ in $\mathcal{A}$ are the same; accordingly, $\mathcal{A}$ will be a model not only for $\hat{r}$ but also for r'. □

Example 2.6 Let r be the following quantified conjunction:

$$r \ \equiv\ (\forall y \in x_1)(y \in x_2)\ \wedge\ \emptyset \notin x_2\ \wedge\ x_1 \in x_1$$

Let us apply the procedure described above, in order to establish whether r is satisfiable. The following instance of formula (1), involving the two new variables x_3 and x_4, is conjoined with r:

$$\bigwedge_{i=0}^{1}\ \bigwedge_{j=i+1}^{2}\ \Big(x_i \neq x_j \rightarrow \bigvee_{h=1}^{2}(x_{h+2} \in x_i \leftrightarrow x_{h+2} \notin x_j)\Big)\,.$$

Since we have convened that $x_0 \equiv \emptyset$, and by virtue of **(N)**, the preceding conjunction becomes:

$$\begin{aligned} r' \ \equiv\ & \big(\,(\emptyset = x_1\ \vee\ \underline{x_3 \in x_1}\ \vee\ x_4 \in x_1) \\ & \wedge\,(\emptyset = x_2\ \vee\ x_3 \in x_2\ \vee\ \underline{x_4 \in x_2}) \\ & \wedge\,(x_1 = x_2 \vee x_3 \notin x_1 \vee x_3 \notin x_2 \vee \underline{x_4 \notin x_1} \vee x_4 \notin x_2) \\ & \wedge\,(x_1 = x_2 \vee x_3 \notin x_1 \vee x_3 \notin x_2 \vee x_4 \in x_1 \vee \underline{x_4 \in x_2}) \\ & \wedge\,(x_1 = x_2 \vee \underline{x_3 \in x_1} \vee x_3 \in x_2 \vee x_4 \notin x_1 \vee x_4 \notin x_2) \\ & \wedge\,(x_1 = x_2 \vee x_3 \in x_1 \vee \underline{x_3 \in x_2} \vee x_4 \in x_1 \vee x_4 \in x_2)\ \big). \end{aligned}$$

At this point we can transform r by replacing the sub-formula $(\forall y \in x_1)(y \in x_2)$ by the following instance of formula (2):

$$\begin{array}{rcl} r'' & \equiv & \big((\emptyset \in x_1 \rightarrow \emptyset \in x_2) \\ & & \wedge\ (x_1 \in x_1 \rightarrow x_1 \in x_2) \\ & & \wedge\ (x_2 \in x_1 \rightarrow x_2 \in x_2) \\ & & \wedge\ (x_3 \in x_1 \rightarrow x_3 \in x_2) \\ & & \wedge\ (x_4 \in x_1 \rightarrow x_4 \in x_2) \big) \end{array} \qquad \text{i.e.,} \qquad \begin{array}{rcl} r'' & \equiv & \big((\underline{\emptyset \notin x_1} \vee \emptyset \in x_2) \\ & & \wedge\ (x_1 \notin x_1 \vee \underline{x_1 \in x_2}) \\ & & \wedge\ (\underline{x_2 \notin x_1} \vee x_2 \in x_2) \\ & & \wedge\ (x_3 \notin x_1 \vee \underline{x_3 \in x_2}) \\ & & \wedge\ (\underline{x_4 \notin x_1} \vee x_4 \in x_2) \big). \end{array}$$

Establishing the satisfiability of r amounts now to deciding the satisfiability of the formula

$$\hat{r} \ \equiv\ r' \wedge r'' \wedge (\emptyset \notin x_2 \wedge x_1 \in x_1).$$

This is easily done by transforming it into DNF-form and by inspecting the resulting disjuncts. For instance, consider the disjunct D consisting of the conjunction of $(\emptyset \notin x_2 \wedge x_1 \in x_1)$ with the underlined literals in r' and r'':

$$\begin{array}{rcl} D & \equiv & x_3 \in x_1 \ \wedge\ x_4 \in x_2 \ \wedge\ x_4 \notin x_1 \ \wedge\ x_4 \in x_2 \ \wedge\ x_3 \in x_1 \ \wedge\ x_3 \in x_2 \\ & & \wedge\, \emptyset \notin x_1 \ \wedge\ x_1 \in x_2 \ \wedge\ x_2 \notin x_1 \ \wedge\ x_3 \in x_2 \ \wedge\ x_4 \notin x_1 \\ & & \wedge\, \emptyset \notin x_2 \ \wedge\ x_1 \in x_1 \end{array}$$

It turns out that $x_1 \sim x_3$. The graph $\mathcal{G}$ corresponding to this disjunct is depicted in Figure 5. As mentioned (see the proof of Lemma 2.5), the enrichment of $\mathcal{G}$ yielding $\mathcal{G}'$ is performed by processing first the $\sim$-classes (i.e., the nodes) which contain none of the original symbols x_0, x_1, x_2. Namely, the node $[x_4]$ is treated first and two new nodes, ν', ν'', are introduced. After that, no other node needs to be processed. From the graph $\mathcal{G}'$ we obtain the following values of $\Im$:

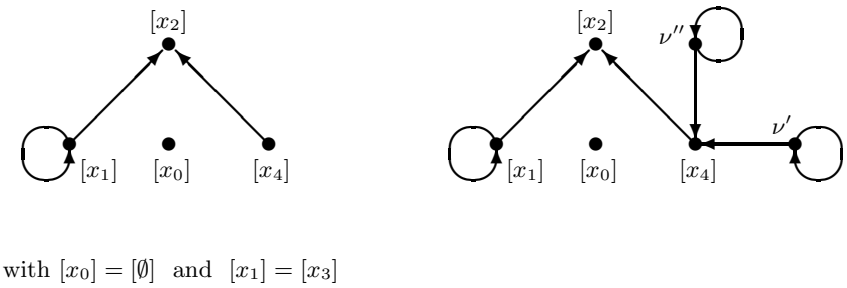
$$\begin{array}{lll} \Im([x_0]) = \emptyset, & \Im(\nu') = a_{\nu'}, & \Im(\nu'') = a_{\nu''}, \\ \Im([x_1]) = \Im([x_3]) = a_{[x_1]}, & \Im([x_4]) = \{a_{\nu'}, a_{\nu''}\}, & \Im([x_2]) = \{a_{[x_1]}, \{a_{\nu'}, a_{\nu''}\}\}. \end{array}$$

where $a_{[x_1]}, a_{\nu'}$, and $a_{\nu''}$ are distinct atoms. An assignment $\mathcal{A}$ satisfying D is then easily obtained. Hence, r is satisfiable. ■

We are now ready to state the main result of this section.

Theorem 2.7 *Under* **(E)**, **(N)**, **(W)**, **(L)**, **(R′)**, *and* **($\mathbf{D}_\in$)**, *the decision problem for* $\exists^*\forall$*-sentences is solvable.*

Proof: It is a plain consequence of Lemmas 2.4 and 2.5 that the $\exists^*\forall$-theorems of the said axiomatic theory constitute a decidable class of sentences. □



with $[x_0] = [\emptyset]$ and $[x_1] = [x_3]$

Figure 5: The graph $\mathcal{G}$ and its enrichment $\mathcal{G}'$ for the Example 2.6

3 A Goal-driven Unification Algorithm for Sets with Atoms

In this section we develop a unification algorithm for the theory considered in Sec. 1. We assume the standard notions of first-order term, substitution, etc.

Set unification is a major ingredient of {log}, a programming language akin to Prolog which incorporates sets. This language and some of its applications are discussed, among others, in [DOPR96]; revised versions and descendents of {log} can be downloaded from the web sites [setlog] or [SETS]. If, within {log}, the set unification algorithm were replaced by the novel one we will describe below, a language with the same versatility but with a neater semantics would result, as we have briefly argued in the Introduction. Potentially, this is a significant application of the present paper, although the details of the new algorithm, as we are about to see, turn out to be more complex (and might become even more as the algorithm gets refined).

Most commonly, semantic unification has been investigated in connection with *equational theories*, namely first-order theories whose axioms have the form $s = t$ for first-order terms s and t [BS98, BS01]. The unification problem in set-theoretic contexts can hardly be seen as a point in case. In fact, let us consider the set-theoretic axioms in Figure 2. We can easily drop the membership relator from the primitive symbols, in favor of a binary operation $\{\cdot \mid \cdot\}$ resulting from **(W)**: if x and y are terms denoting sets, then $\{x \mid y\}$ represents the set whose members are all elements of the set y plus the element x. The behavior of this construct can be rendered by simple equational properties [DPR98]:

$$\begin{array}{lrcl} (Ab) & \{X \mid \{X \mid Z\}\} & = & \{X \mid Z\}; \\ (C\ell) & \{X \mid \{Y \mid Z\}\} & = & \{Y \mid \{X \mid Z\}\}. \end{array}$$

Accordingly, one could express membership in terms of $\{\cdot \mid \cdot\}$, by putting: $X \in Y \leftrightarrow_{\text{Def}} Y = \{X \mid Y\}$. Our axiomatic assumptions would be then reformulated with the new construct and with additional Skolem symbols (new and δ) as displayed in Figure 6. Even thus, our theory would retain somewhat unconventional traits: its axioms would be clauses instead of simple atomic formulas, moreover we could not get rid of some negative literals (such as in $\mathbf{(N_2)}$, for instance).

$X \in Y \ \leftrightarrow_{\text{Def}} \ Y = \{X \mid Y\}$	$\{X\} =_{\text{Def}} \{X \mid \emptyset\}$	$\mathsf{ur}(X) \leftrightarrow_{\text{Def}} X = \{X\}$
$\mathbf{(E_0)}$	$\delta(X,X) \notin X \ \rightarrow \ X = \emptyset$	
$\mathbf{(E_1)}$	$V \in X \ \rightarrow \ \delta(X,Y) \in X$	
$\mathbf{(E_2)}$	$\delta(Y,X) \in X \wedge \delta(X,Y) \in Y \ \rightarrow \ X = Y$	
$\mathbf{(N_1)}$	$\delta(\emptyset, X) = \emptyset$	
$\mathbf{(N_2)}$	$\emptyset \neq \{\emptyset\}$	
$\mathbf{(W_1)}$	$V \in \{X \mid N\} \ \rightarrow \ V \in N \vee V = X$	
$\mathbf{(W_2)}$	$X \in \{X \mid N\}$	
$\mathbf{(W_3)}$	$V \in N \ \rightarrow \ V \in \{X \mid N\}$	
$\mathbf{(L_1)}$	$V \in X \,\mathsf{less}\, Y \ \rightarrow \ V \in X$	
$\mathbf{(L_2)}$	$Y \notin X \,\mathsf{less}\, Y$	
$\mathbf{(L_3)}$	$V \in X \ \rightarrow \ V \in X \,\mathsf{less}\, Y \vee V = Y$	
$\mathbf{(R'_1)}$	$Y \in X \wedge Y \in \delta(X,X) \ \rightarrow \ \mathsf{ur}(Y)$	
$\mathbf{(R'_2)}$	$\mathsf{ur}(\delta(X,X)) \wedge Y \in X \ \rightarrow \ \mathsf{ur}(Y)$	
$\mathbf{(D_{\in,1})}$	$\mathsf{ur}(\mathsf{new}(X))$	
$\mathbf{(D_{\in,2})}$	$\mathsf{new}(X) \in Y \ \rightarrow \ Y \notin X$	
$\mathbf{(C_1)}$	$\mathsf{ur}(\mathsf{c}_i)$	$i = 1, 2, 3, \ldots$
$\mathbf{(C_2)}$	$\mathsf{c}_i \neq \mathsf{c}_j$	$i, j = 1, 2, 3, \ldots$ and $i < j$

Figure 6: Weak set-theoretic axioms in clausal form.

Lemma 3.1 *The axiomatic theory $\mathbb{P}$ whose postulates are those in Figure 6 is a conservative extension of the theory $\mathbb{Q}$ whose axioms are those in Figure 2.*

Proof: (Sketch) The language underlying $\mathbb{P}$ is broader than the one underlying $\mathbb{Q}$, and our claim is that any sentence belonging to the narrower language is provable in $\mathbb{Q}$ if and only if it is provable in $\mathbb{P}$. On the one hand, we must check that every axiom of $\mathbb{Q}$ is provable in $\mathbb{P}$; on the other hand, that any construct of $\mathbb{P}$ which does not belong to the language of $\mathbb{Q}$ can be seen as resulting from the Skolemization of a sentence provable in $\mathbb{Q}$.

As regards the former of these issues, consider for example **(N)**. We can prove **(N)** within $\mathbb{P}$ as follows. Assuming by contradiction that $v \in \emptyset$, we get from $\mathbf{(E_1)}$ that $\delta(\emptyset,\emptyset) \in \emptyset$, and since $\delta(\emptyset,\emptyset) = \emptyset$ by $\mathbf{(N_1)}$ we get $\emptyset \in \emptyset \wedge \emptyset \in \delta(\emptyset,\emptyset)$, implying $\mathsf{ur}(\emptyset)$ by $\mathbf{(R'_1)}$. However, this conflicts with $\mathbf{(N_2)}$; and this contradiction, showing that $\emptyset$ must be devoid of elements, leads to the desired conclusion **(N)**. As another example, consider **(E)**, which we can prove within $\mathbb{P}$ as follows. Assuming by contradiction that distinct sets x, y have the same elements, within $\mathbb{P}$ we get from $\mathbf{(E_2)}$ that $\delta(y,x) \notin x \vee \delta(x,y) \notin y$, and consequently that $\delta(y,x) \notin y \vee \delta(x,y) \notin x$, entailing that x, y are both devoid of elements, by $\mathbf{(E_1)}$. This conflicts with $\mathbf{(E_0)}$, because $\mathbf{(E_0)}$ entails that $\emptyset$ is the only empty set; hence we conclude that **(E)** is provable in $\mathbb{P}$.

As regards the converse issue, namely to check that every construct of $\mathbb{P}$ makes sense in connection with $\mathbb{Q}$, we begin by observing that $\mathbf{(W_1)}$, $\mathbf{(W_2)}$, and $\mathbf{(W_3)}$ directly result from the Skolemization of **(W)**; and, likewise, $\mathbf{(L_1)}$, $\mathbf{(L_2)}$, and $\mathbf{(L_3)}$ originate from **(L)**.

Then we note that the function δ is intended to be a selection function able to discriminate between two different sets. In particular, it behaves as follows: for any non-empty set X, $\delta(X,Y)$ is a element of X possibly not belonging to Y (this is the case whenever $X \setminus Y$ is not empty). Hence, on the basis of **(E)**, $\delta(X,Y)$ and $\delta(Y,X)$ permit to distinguish between different sets X and Y. Moreover, δ is assumed to be such that "unless every element of X is a self-singleton, $\delta(X,X)$ is not a self-singleton and every element of $\delta(X,X) \cap X$ is self-singleton" (cf. regularity axiom $\mathbf{(R')}$).

As stated in $\mathbf{(D_{\in,1})}$ and $\mathbf{(D_{\in,2})}$, the function new associates to any set X a "new" self-singleton not belonging to any member of X (cf. plenitude axiom $\mathbf{(D_\in)}$). Finally, to justify the introduction of the infinitely many ur-constants c_i appearing in $\mathbf{(C_1)}$ and $\mathbf{(C_2)}$, we can exploit part $\mathbf{(Y_\in)}$ of the diagonalization lemma. □

To denote finite sets we fix a first-order signature $\Sigma = \{\emptyset, \{\cdot \,|\, \cdot\}, \mathsf{c}_1, \mathsf{c}_2, \dots\}$. As useful pieces of notation we will denote the term $\{t_1 \,|\, \{t_2 \,|\, \cdots \{t_n \,|\, t\}\}\}$ by $\{t_1, t_2, \dots, t_n \,|\, t\}$, and simply by $\{t_1, t_2, \dots, t_n\}$ when t is $\emptyset$.

As before, we make use of the monadic predicate symbol ur to state that certain terms designate atoms: thus, asserting $\mathsf{ur}(t)$ amounts to stating that $t = \{t\}$. Consequently, if X is a variable, then the formula $\mathsf{ur}(X)$ is satisfiable, whereas both $\mathsf{ur}(\emptyset)$ and $\mathsf{ur}(X) \wedge \mathsf{ur}(Y) \wedge X \neq Y \wedge \mathsf{ur}(\{X,Y\})$ are unsatisfiable. We assume that each term c_i denotes a (distinct) self-singleton, i.e., the following holds

$$\mathbf{(C)} \qquad \mathsf{ur}(\mathsf{c}_i) \;\wedge\; \mathsf{c}_i \neq \mathsf{c}_j \qquad \text{for } i, j = 1, 2, 3, \dots \text{ and } i < j.$$

In the context of axiomatic set theory, two terms s and t are said to be *unifiable* if there is a substitution σ such that $s\sigma$ and $t\sigma$ have the same designation, in which case one says that σ is a solution of the equation $s = t$. In our theory, which fulfills **(E)**, determining whether two terms s and t designate the same set amounts to establishing whether the existential closure of the formula $\forall y(y \in s \leftrightarrow y \in t)$ is valid. Although this is an $\exists^*\forall$-sentence, and hence could be submitted to the decision algorithm seen in Sec. 2, in what follows we propose a more specialized, goal-driven, unification algorithm. Our algorithm is actually aimed at solving a *system*, that is, a conjunction (represented by a *set*, as usual) $\mathcal{E}$ of equations. Of course, a substitution σ is a solution of $\mathcal{E}$ if and only if it simultaneously solves all equations in $\mathcal{E}$: needless to say, this is still a subproblem of $\exists^*\forall$-satisfiability checking.

Let us examine here what convenient format we can choose for representing the solutions σ we are after, which, without loss of generality, we may require to be *idempotent* in the sense that $t\sigma\sigma = t\sigma$ holds for all terms t. An equation of the form $X = \{t_0, \dots, t_n \,|\, X\}$, with $X \notin \mathit{vars}(t_0, \dots, t_n)$,

is said to be a *membership equation*: such an equation imposes that $\bigwedge_{i=0,\ldots,n} t_i \in X$ holds. Given a system $\mathcal{E}$, if a variable X occurs only once in $\mathcal{E}$, and it occurs in an equation of the form $X = t$, then X is said to be a *solved variable*. In this case, such an equation is said to be a *solved-form equation*. A system $\mathcal{E}$ is said to be in *pre-solved form* if it contains only membership or solved-form equations. $\mathcal{E}$ is said to be in *solved form* if it contains only solved-form equations. A system $\mathcal{E}$ in solved form uniquely identifies an idempotent substitution which, trivially, is a solution to $\mathcal{E}$.

In what follows, uppercase letters represent variables; N stands for a newly generated variable; r, s, t, possibly subscripted, stand for generic terms; and c, d represent constant symbols drawn from among the c_is. With $t[X]$ we denote a term having X as a subterm. In a term of the form $\{t_1, t_2, \ldots, t_n \,|\, t\}$ the subterm t is said to be the tail, provided that $\{\cdot \,|\, \cdot\}$ is not the main functor of t.

The unification algorithm Unify described in Figures 7 and 8 consists of three main parts. For any variable X, repeated applications of the procedure Simplify_memberships gather in a single equation all the membership equations related to X. The rationale of this is that the constraints represented by the conjunction of two equations $X = \{s_0, \ldots, s_m \,|\, X\}$ and $X = \{t_0, \ldots, t_n \,|\, X\}$ can be rendered by the single equation $X = \{s_0, \ldots, s_m, t_0, \ldots, t_n \,|\, X\}$.

The core of the algorithm is the procedure Rewriting_rule, which after selecting from $\mathcal{E}$ an equation which can trigger one (and only one) of several rewriting rules (cf. Figure 8), fires the corresponding action on $\mathcal{E}$. The various rewriting rules can be grouped together to point out their relationships between standard cases and names (Trivial, ..., Decomposition [BS01]). Some rules are non-deterministic, e.g. rules (7.1) and (7.2) that executes decomposition of terms denoting sets containing elements.

The combination of these two procedures is applied until $\mathcal{E}$ reaches pre-solved form, or fail is obtained. In the former case, the algorithm achieves a simplification of the membership equations providing a system in solved form as output. This is obtained by means of the procedure Eliminate_memberships.

procedure Unify($\mathcal{E}$);
    initialize $\mathcal{U}$ as the empty set;
    while $\mathcal{E}$ is not in pre-solved form **do**
        Simplify_memberships($\mathcal{E}$);
        Rewriting_rule($\mathcal{E}$);
    end_while;
    Eliminate_memberships($\mathcal{E}$);
end Unify;

procedure Simplify_memberships($\mathcal{E}$);
    while the following rule is applicable **do**
        $X = \{s_0, \ldots, s_m \,|\, X\} \wedge X = \{t_0, \ldots, t_n \,|\, X\} \wedge \mathcal{E} \rightsquigarrow$
                $X = \{s_0, \ldots, s_m, t_0, \ldots, t_n \,|\, X\} \wedge \mathcal{E}$
end Simplify_memberships;

procedure Eliminate_memberships($\mathcal{E}$);
    while the following rule is applicable **do**
        $X = \{s_0, \ldots, s_m \,|\, X\} \wedge \mathcal{E} \rightsquigarrow$
                $X = \{s_0, \ldots, s_m \,|\, N\} \wedge \mathcal{E}[X \mapsto \{s_0, \ldots, s_m \,|\, N\}]$
end Eliminate_memberships;

Figure 7: The structure of the algorithm Unify

We assume that there is a set $\mathcal{U}$, called *store*, which keeps facts of the form $\mathsf{ur}(X)$. At the beginning of the execution, $\mathcal{U}$ is initialized as the empty set. For any variable X, the algorithm

procedure Rewriting_rule($\mathcal{E}$);

apply one of the following rules:

Trivial	(1)	$t_1 = t_2 \wedge \mathcal{E}$ $t_1 \equiv t_2$	$\rightsquigarrow$	$\mathcal{E}$
Orient	(2.1)	$t = X \wedge \mathcal{E}$ t is not a variable	$\rightsquigarrow$	$X = t \wedge \mathcal{E}$
	(2.2)	$t = c \wedge \mathcal{E}$ t is neither a variable nor a constant d	$\rightsquigarrow$	$c = t \wedge \mathcal{E}$
Occur Check	(3.1)	$X = \{t_0, \ldots, t_i[X], \ldots, t_n \mid Y\} \wedge \mathcal{E}$ Y is a variable	$\rightsquigarrow$	$\bigwedge_{i=0}^{n} X = t_i \wedge \mathcal{E} \wedge$ $(i) Y = \emptyset \vee$ $(ii) Y = X$ Add ur(X) to $\mathcal{U}$
	(3.2)	$X = \{t_0, \ldots, t_i[X], \ldots, t_n\} \wedge \mathcal{E}$	$\rightsquigarrow$	$\bigwedge_{i=0}^{n} X = t_i \wedge \mathcal{E}$ Add ur(X) to $\mathcal{U}$
	(3.3)	$X = \{t_0, \ldots, t_n \mid Y\} \wedge \mathcal{E}$ ur$(X) \in \mathcal{U}$, Y is a variable, $X \notin vars(t_0, \ldots, t_n)$	$\rightsquigarrow$	$\bigwedge_{i=0}^{n} X = t_i \wedge \mathcal{E} \wedge$ $(i) Y = \emptyset \vee$ $(ii) Y = X$
	(3.4)	$X = \{t_0, \ldots, t_n\} \wedge \mathcal{E}$ ur$(X) \in \mathcal{U}, X \notin vars(t_0, \ldots, t_n)$	$\rightsquigarrow$	$\bigwedge_{i=0}^{n} X = t_i \wedge \mathcal{E}$
	(3.5)	$X = \{t_0, \ldots, t_n \mid c\} \wedge \mathcal{E}$ ur$(X) \in \mathcal{U}$	$\rightsquigarrow$	$\bigwedge_{i=0}^{n} X = t_i \wedge X = c \wedge \mathcal{E}$
Variable Elimination	(4.1)	$X = Y \wedge \mathcal{E}$ X occurs in $\mathcal{E}$, ur$(X) \in \mathcal{U}, X \not\equiv Y$	$\rightsquigarrow$	$\mathcal{E}[X \mapsto Y] \wedge X = Y$ Add ur(Y) to $\mathcal{U}$
	(4.2)	$X = c \wedge \mathcal{E}$ X occurs in $\mathcal{E}$	$\rightsquigarrow$	$\mathcal{E}[X \mapsto c] \wedge X = c$
	(4.3)	$X = t \wedge \mathcal{E}$ X occurs in $\mathcal{E}, X$ not in t and ur$(X) \notin \mathcal{U}$	$\rightsquigarrow$	$\mathcal{E}[X \mapsto t] \wedge X = t$
Symbol Clash	(5.1)	$c = \{t_1, \ldots, t_n \mid d\} \wedge \mathcal{E}$ $c \not\equiv d, n \geqslant 0$	$\rightsquigarrow$	fail
	(5.2)	$c = \emptyset \wedge \mathcal{E}$	$\rightsquigarrow$	fail
	(5.3)	$\emptyset = \{s \mid t\} \wedge \mathcal{E}$	$\rightsquigarrow$	fail
	(5.4)	$\{s \mid t\} = \emptyset \wedge \mathcal{E}$	$\rightsquigarrow$	fail
Symbol Clash	(6.1)	$c = \{t_0, \ldots, t_n \mid Y\} \wedge \mathcal{E}$	$\rightsquigarrow$	$\bigwedge_{i=0}^{n} c = t_i \wedge \mathcal{E} \wedge$ $(i) Y = \emptyset \vee$ $(ii) Y = c$
	(6.2)	$c = \{t_0, \ldots, t_n \mid t\} \wedge \mathcal{E}$ t is c or $\emptyset$	$\rightsquigarrow$	$\bigwedge_{i=0}^{n} c = t_i \wedge \mathcal{E}$
(Set) Decomposition	(7.1)	$\{t_0, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_n \mid X\} \wedge \mathcal{E}$	$\rightsquigarrow$	select arbitrarily i in $\{0, \ldots, n\}$; choose one among: (i) $\{t_1, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_{i-1}, t'_{i+1}, \ldots, t'_n \mid X\} \wedge t_0 = t'_i \wedge \mathcal{E}$ (ii) $\{t_0, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_{i-1}, t'_{i+1}, \ldots, t'_n \mid X\} \wedge t_0 = t'_i \wedge \mathcal{E}$ (iii) $\{t_1, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_n \mid X\} \wedge t_0 = t'_i \wedge \mathcal{E}$ (iv) $\{t_1, \ldots, t_m \mid X\} = \{t'_0, \ldots, t'_n \mid X\} \wedge X = \{t_0 \mid X\} \wedge \mathcal{E}$
	(7.2)	$\{t \mid s\} = \{t' \mid s'\} \wedge \mathcal{E}$ tail(s) and tail(s') are not the same variable	$\rightsquigarrow$	(i) $s = s' \wedge t = t' \wedge \mathcal{E}$ (ii) $\{t \mid s\} = s' \wedge t = t' \wedge \mathcal{E}$ (iii) $s = \{t' \mid s'\} \wedge t = t' \wedge \mathcal{E}$ (iv) $s = \{t' \mid N\} \wedge \{t \mid N\} = s' \wedge \mathcal{E}$

end Rewriting_rule;

Figure 8: Set-unification rewriting rules

checks whether $\mathsf{ur}(X)$ is in $\mathcal{U}$ before applying any substitution to X. Moreover, at the end of the unification process, the store is used to compute/filter out the solutions. More precisely, we say that a substitution σ *complies* with the store $\mathcal{U}$ if $\sigma(X) \equiv \mathsf{c}_i$ holds for a constant c_i, for each fact $\mathsf{ur}(X)$ in $\mathcal{U}$.

The following lemma states that Unify gives a correct answer whenever it terminates, provided its execution is fully non-deterministic: in order to generate a complete set of solutions, all possible choices for the selected action, when there are more than one, must be effected.

Lemma 3.2 *Let $\mathcal{E}$ be a system and $\mathcal{E}_1, \ldots, \mathcal{E}_k$ be the systems non-deterministically obtained after any number of iterations of the* **while***-statement within* Unify*. Moreover, let $\mathcal{U}_1, \ldots, \mathcal{U}_k$ be the corresponding values of the store. Then,*

soundness: *any solution σ to $\mathcal{E}_i$ which complies with the store $\mathcal{U}_i$ is also a solution to $\mathcal{E}$;*

exhaustiveness: *to any solution σ of $\mathcal{E}$ there corresponds, for some i in $\{1, \ldots, k\}$, a solution σ_i of $\mathcal{E}_i$ which complies with $\mathcal{U}_i$ and extends σ in the sense that $X\sigma_i \equiv X\sigma$ for all X in vars$(\mathcal{E})$.*

Proof: Correctness of the procedures Simplify_memberships and Eliminate_memberships are immediate to prove. Let us focus on the various possible steps of Rewriting_rule (cf. Figure 8). Soundness of actions (1) and (2) holds by the properties of equality. Let us analyze the other cases:

(3.1)–(3.2)**:** If $X = \{t_0, \ldots, t_i[X], \ldots, t_n \,|\, t\}$, then X must designate an atom. This fact is memorized by adding $\mathsf{ur}(X)$ to the store $\mathcal{U}$. Correctness follows immediately.

(3.3)–(3.5)**:** In these cases the fact $\mathsf{ur}(X)$ is already known (i.e., $\mathsf{ur}(X)$ has been inserted in $\mathcal{U}$ by earlier steps of the computation). Hence, the equations are rewritten accordingly.

(4)**:** The actions in these rules are substitution applications. Checking and updating the store guarantee correctness.

(5)**:** All the rules of this group generate failure actions. For instance, an atom c cannot equal $\{\ldots \,|\, d\}$ or d, for constant d distinct from c, by **(E)** and **(C)**. Similar considerations apply to the other three cases.

(6)**:** Similarly to what happens for the rules (3.3) and (3.4), since c denotes an atom, any solution satisfying the l.h.s. of a rule of this group must satisfy the equations on the corresponding r.h.s.

(7)**:** Rule (7.2) reflects the extensionality axiom. For a detailed proof of equivalence in the context of finite sets, see [DPR98]. Rule (7.2) cannot be applied when $\mathsf{tail}(s)$ and $\mathsf{tail}(s')$ are the same variable. This particular case is handled by rule (7.1), which is a variant introduced for ensuring termination.

□

What action selection strategy is adopted is immaterial, as just proved, from the standpoint of correctness; however, it may badly affect termination. Indeed, it is easy to find a sequence of actions leading to non-termination, as the following example brings into light.

Example 3.3 Consider the set of equations:

$$\{T_1 \,|\, S_1\} = \{T_2 \,|\, S_2\} \wedge \{T_3 \,|\, S_2\} = \{T_4 \,|\, S_1\}\,.$$

Apply action (7.2.*iv*) to both equations (in any order), obtaining:

$$\begin{array}{l} S_1 = \{T_2 \,|\, N_1\} \;\wedge\; \{T_1 \,|\, N_1\} = S_2 \wedge \\ S_2 = \{T_4 \,|\, N_2\} \;\wedge\; \{T_3 \,|\, N_2\} = S_1. \end{array}$$

Then (action (4.3)) apply the substitution obtained by the first and third equation, obtaining:

$$S_1 = \{T_2 \,|\, N_1\} \;\wedge\; \{T_1 \,|\, N_1\} = \{T_4 \,|\, N_2\} \wedge$$
$$S_2 = \{T_4 \,|\, N_2\} \;\wedge\; \{T_3 \,|\, N_2\} = \{T_2 \,|\, N_1\}.$$

The first and third equations are now in solved form; nevertheless, the second and fourth equations constitute a system of the same form as the initial one, on which the same sequence of actions can be applied. Of course, this can be repeated forever. ■

A simple deterministic strategy (similar to the one proposed in [DPPR00], in the context of the unification problem for finite sets) can be devised to ensure termination. The crucial point in such deterministic strategy is the introduction of a stack used to handle part of the equations produced by action (7.2). It turns out, in fact, that processing these equations as soon as they are generated avoids non-terminating computations. The proof of the following lemma gives the details on this strategy.

Lemma 3.4 *The algorithm* Unify *can be implemented so as to ensure termination.*

Proof: The algorithm proceeds by selecting an equation and by applying (non-deterministically) one of the rules of Figure 8. In order to ensure termination, we need to guide those selections which are performed after each application of rule (7.2). For doing this we introduce into the unification algorithm an auxiliary data-structure $\mathcal{S}$, a stack of equations. Whenever $\mathcal{S}$ is not empty, the equation at the top of $\mathcal{S}$ is always selected as the next to be processed. On the other hand, when $\mathcal{S}$ is empty, the algorithm selects any equation in $\mathcal{E}$ arbitrarily. Consider any application of rule (7.2). Among all equations generated by actions of this kind, only those of the form $t = t'$ are added to $\mathcal{E}$, while the remaining ones are pushed onto the stack $\mathcal{S}$.

Intuitively speaking, the key point in this strategy, ensuring termination, is the following: whenever, by rule (7.2), a new variable is introduced (namely, N in Figure 8), the algorithm exploits $\mathcal{S}$ in order to immediately process the newly generated equations. As a consequence of this strategy, two old variables become solved (within a finite number of steps), and the global number of non-solved variables does not increase.

To formally show termination, we introduce a measure of complexity for systems of equations. A computation of the unification algorithm can be seen as a sequence of rewriting stage. The termination of the algorithm will be proved by showing that with each stage the complexity of the system always decreases. Termination follows from well-foundedness of the measure.

In general, a rewriting stage corresponds to a single application of a rewriting action from (1)–(6) and (7.1) of Figure 8. Compound stages are those which start with action (7.2): these must comprise in full the sequence of subsequent actions that process the equations pushed on the stack by the application of rule (7.2).

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

- A is the number of variables in $\mathcal{E}$ that are not solved;
- B is the multiset $\{\![size(\ell) + size(r) : \ell = r \text{ in } \mathcal{E}, \ell = r \text{ is not solved}]\!\}$, where $size(t)$ is the number of occurrences of constants and function symbols in t;[2]
- C is the value $\displaystyle\sum_{\substack{\ell = r \in \mathcal{E} \\ \ell = r \text{ is not solved}}} size'(\ell)$ where $size'(t)$ is the number of occurrences of the set-constructor symbol $\{\cdot \,|\, \cdot\}$ in t.

First of all, observe that `Simplify_memberships(`$\mathcal{E}$`)` leaves the measure unchanged. However, it can be applied just a finite number of times for each non-solved variable and the number of non-solved variables cannot be increased with respect to the initial system.

Let us consider the procedure `Rewriting_rule(`$\mathcal{E}$`)`. Inspection reveals that

[2] A well-ordering between such multisets of integer numbers is the reflexive and transitive closure of the relation $\prec$ so defined: $\{\![a_1, \ldots, a_i, \ldots, a_n]\!\} \prec \{\![a_1, \ldots, a_{i-1}, b_1, \ldots, b_m, a_{i+1}, \ldots, a_n]\!\}$ if $b_1 < a_i \wedge \cdots \wedge b_m < a_i$.

- actions (1), (2.1), (3.1)—(3.5), (6.1) and (6.2) cannot increase A, while they always decrease B.
- Action (2.2) cannot increase A and B, while it always decreases C.
- Actions (4.1)—(4.3) decrease A (observe that after the application of this rule one membership equation can become a standard equation which will then be counted by the component B of the measure).
- Actions (5.1)—(5.4) lead to immediate termination of the algorithm by failure.
- Every branch of action (7.1) certainly decreases B, without increasing A.
- Each application of rule (7.2) must be considered together with the sequence of subsequent actions that process the equations pushed on the stack by (7.2). In particular, rule (7.2) is activated by the presence of an equation of the form:

$$\{t_0, \ldots, t_m \,|\, h\} \quad = \quad \{t'_0, \ldots, t'_n \,|\, k\} \tag{3}$$

  where both h, k can be atoms, $\emptyset$, or (distinct) variables.

  The stack-based selection strategy for the equations generates a sequence of applications of action (7.2), producing a conjunction of equations of the form $t_i = t'_j$ of smaller size than the equation (3), plus:

  1. an equation of the form $h = \{t'_{j_1}, \ldots, t'_{j_\ell} \,|\, k\}$, or
  2. an equation of the form $k = \{t_{i_1}, \ldots, t_{i_\ell} \,|\, h\}$, or
  3. two equations $h = \{t'_{j_1}, \ldots, t'_{j_\ell} \,|\, N\}$, and $k = \{t_{i_1}, \ldots, t_{i_\ell} \,|\, N\}$ (with N a new variable, the same for both equations).

  Moreover, such a sequence of actions produces a chain of solved equations $N_1 = \{\_ \,|\, N_2\}$, $N_2 = \{\_ \,|\, N_3\}$, …, $N_p = \{\_ \,|\, N\}$, one equation for each new variable introduced. By virtue of the stack-based strategy, all these equations are immediately processed by substitution application (by an action (4.i)). Consequently, they do not contribute to the complexity measure of the system obtained at the end of the rewriting stage.

  Let us analyze the form that the equations in the above listed cases 1.–3. might take. This depends on k and h:

  - h and k are both $\emptyset$. The equations generated are either $\emptyset = \emptyset$ (whose size is smaller than the equation (3)) or of the form $\emptyset = \{\cdot\}$. The latter leads to termination by failure with the next action.
  - h is a variable and k is $\emptyset$ (or vice versa). The only equations that do not lead to failure are those of the form $h = \{t'_{j_1}, \ldots, t'_{j_\ell} \,|\, \emptyset\}$. At the next step, substitution is applied and therefore the measure A of complexity decreases.
  - h and k are distinct variables. There are three possibilities:
    * an equation $h = \{t'_{j_1}, \ldots, t'_{j_\ell} \,|\, k\}$ is generated, or
    * an equation $k = \{t_{i_1}, \ldots, t_{i_\ell} \,|\, h\}$ is generated, or
    * two equations, $h = \{t'_{j_1}, \ldots, t'_{j_\ell} \,|\, N\}$ and $k = \{t_{i_1}, \ldots, t_{j_r} \,|\, N\}$, are generated ($N$ being a new variable).

    In the first two cases the subsequent substitution application causes A to decrease since the variable h (resp. k) becomes solved. In the third case two variables become solved and only one new variable is introduced. Again, A decreases.
  - h and k are atoms. The situation is analogous to the previous one. The only difference is that action (6.1) or (6.2) will now be applied immediately after rule (7.2). A remains unchanged but all introduced equations have size strictly smaller than equation (3). B therefore decreases.

– h is an atom and k is $\emptyset$ (or vice versa). Similar to the previous case: B decreases.

– k is an atom and h is a variable (or vice versa). Similarly to the above-treated cases, we have three possibilities:

  * an equation $h = \{t'_{j_1}, \ldots, t'_{j_\ell} \,|\, k\}$ is generated, or
  * an equation $k = \{t_{i_1}, \ldots, t_{i_\ell} \,|\, h\}$ is generated, or
  * two equations, $h = \{t'_{j_1}, \ldots, t'_{j_\ell} \,|\, N\}$ and $k = \{t_{i_1}, \ldots, t_{j_r} \,|\, N\}$, are generated ($N$ being a new variable).

  In the first case, the subsequent substitution application causes A to decrease since the variable h becomes solved. In the second case, action (6.1) is selected: then either the substitution $h \mapsto \emptyset$ or the substitution $h \mapsto k$ is applied: A decreases. In the third case the algorithm proceeds by applying the substitution for h, and A does not change. But then, action (6.1) is applied to the second equation. Consequently, either the substitution $N \mapsto \emptyset$ or the substitution $N \mapsto k$ is applied and A decreases with respect to the initial situation.

Termination of the procedure Eliminate_membership is a straightforward consequence. For any variable X occurring in an equation $X = \{\cdots \,|\, X\}$, we perform at most one rewriting and one substitution application. Equations in solved form remain in solved form. □

Then, we can state the following theorem:

Theorem 3.5 *Every branch of the non-deterministic computation of* Unify($\mathcal{E}$) *terminates either with* fail *or with a system $\mathcal{E}_i$ in solved form and its associated value $\mathcal{U}_i$ of the store. The whole set of solutions to $\mathcal{E}$ is spanned by the systems $\langle \mathcal{E}_1, \mathcal{U}_1 \rangle, \ldots, \langle \mathcal{E}_k, \mathcal{U}_k \rangle$ returned by successful computation branches. No solutions to $\mathcal{E}$ exist if and only if $k = 0$, i.e., iff* fail *is the only possible result.*

Proof: Immediate from Lemmas 3.2 and 3.4. Notice that, unlike in Lemma 3.2, each system $\mathcal{E}_i \cup \mathcal{U}_i$ admits a solution because we are referring to a *terminating* non-deterministic search. □

Remark 3.6 (Complexity Issues) The decision version of the above-discussed unification problem (which is, given s and t, to answer the question: *Is there a substitution σ such that $\sigma(s)$ and $\sigma(t)$ denote the same set?*) is NP-complete.

To show NP-hardness, we carry out a reduction of 3-SAT to the problem at hand in exactly the same straightforward way in which one handles other set-unification problems, cf. e.g. [DOPR96]. Let Φ be an instance

$$\Phi \equiv (\ell_1^{(1)} \vee \ell_2^{(1)} \vee \ell_3^{(1)}) \wedge \cdots \wedge (\ell_1^{(m)} \vee \ell_2^{(m)} \vee \ell_3^{(m)})$$

of 3-SAT, where each $\ell_i^{(j)}$ is either a propositional variable or the negation of a propositional variable. Let us assume that $V_1, V_2, \ldots, V_k$ are the propositional variables occurring in Φ.

We define a transformation function f, as

$$f(\ell) \;=\; \begin{cases} X_i & \text{if } \ell \equiv V_i, \\ Y_i & \text{if } \ell \equiv \neg V_i, \\ \{\, \emptyset\,,\; f(\ell_1), f(\ell_2), f(\ell_3) \,\} & \text{if } \ell \equiv \ell_1 \vee \ell_2 \vee \ell_3, \end{cases}$$

and translate Φ to the equation

$$\{\{X_1, Y_1\}, \ldots, \{X_k, Y_k\}, f(\ell_1^{(1)} \vee \ell_2^{(1)} \vee \ell_3^{(1)}), \ldots, f(\ell_1^{(m)} \vee \ell_2^{(m)} \vee \ell_3^{(m)})\} = \{\, \{\, \emptyset\,,\; \{\emptyset\}\} \,\}\,,$$

which requires each one of the sets $\{X_j, Y_j\}$ to be unified with $\{\, \emptyset\,,\; \{\emptyset\}\}$, and at least one among $f(\ell_1^{(i)})$, $f(\ell_2^{(i)})$, $f(\ell_3^{(i)})$ to be unified with $\{\emptyset\}$ (which encodes the truth value `true`, whereas $\emptyset$ encodes `false`) for each $i = 1, \ldots, m$.

To prove NP-completeness, it remains to show that if a system of equations admits a solution, then one can find a witness of its existence that can be checked in polynomial time. This can

be done by first guessing a graph which represents a potential solution to the system in the way described in Sec. 2.2.1, and then checking its adequacy against the given equations. In essence, the "model-graph technique" alluded to here is the one exploited in [COP90] and [OPP96] to show the NP-completeness of problems analogous to the $\exists^*\forall$-satisfiability problem treated in Sec. 2; in view of this analogy, it looks quite plausible that those NP-completeness results concerning the $\exists^*\forall$ class carry over to our own context, thus generalizing the rather specific case of set-unifiability. ■

3.1 Prolog implementation and examples

The unification algorithm has been implemented in SICStus Prolog 3.10.1 [SICStus] and a prototype is available at the URL: `http://www.dimi.uniud.it/~dovier/CST/cst.html`. The code consists in about one hundred Prolog clauses and its length is less than 600 lines, including comments. Actually, a relevant part of the code implements a minimal (textual) user interface. For the user convenience, it (optionally) performs a sort of tracing of the unification process: each firing of the rules of Figure 8 is echoed in order to allow one to trace the execution. The interface also takes care of variable names, by hiding SICStus' anonymous naming of internal variables.

The main Prolog predicate is `meta/0` that launches execution. Then the prompt `s-unify :-` appears. The user can introduce the input system as a sequence of equations separated by ",". To make the parsing simpler, the symbol "/" must be used instead of "|" within set-terms (even though, for ease of the reader, we have kept the usual notation in the examples which follow). If such a system is solvable, then the goal succeeds if E and U can be instantiated to a solved form of S and to the corresponding store value, respectively. For the sake of simplicity, here the store U is just the list of variables X such that $\mathsf{ur}(X)$ must hold. As usual, all possible solutions (solved-form-system plus store-value) are obtained through backtracking. Let us examine a few runs of the unification algorithm. (Time is expressed in msec and referred to an execution on a Pentium4 PC, 2.66GHz.)

Example 3.7

1. Prolog execution corresponding to the system

$$S \;\equiv\; X = \{X, Y\} \;\wedge\; Y = \{a\} \;\wedge\; Z = \{Z\}.$$

produces the following output (suitable `write` instructions enable one to track the rewriting actions):

```
s-unify :- X = {X,Y},Y = {a},Z = {Z}.
Selected equation: X = {X, Y}   Rule: 3.2-
Selected equation: X = Y        Rule: 4.1-
Selected equation: Y = Y        Rule: 1-
Selected equation: Y = {a}      Rule: 3.4-
Selected equation: Y = a        Rule: 4.2-
Selected equation: Z = {Z}      Rule: 3.2-
Selected equation: Z = Z        Rule: 1-

***********Found a Solution:
E: Y = a , X = a
U: Z , Y , X
Execution Time:         0

another solution (y./n.) ? y.
no more solutions
```

Notice that no substitution is computed for the variable Z. Since Z is in U, this means that $\mathsf{ur}(Z)$ must hold, namely, Z must be an atom. Thus, any substitution extending the substitution $[Y \mapsto a, X \mapsto a]$ by mapping Z to a constant (an atom) complies with the store and is also a solution to S.

2. Consider next the system

$$S \;\equiv\; Y = \{a|X\} \;\wedge\; \{a, b|X\} = \{a|Y\}.$$

Here are some of the solutions obtained by the implementation:

```
s-unify :- Y = {a|X},{a,b|X}={a|Y}.
Selected equation: Y = {a|X}    Rule: 4.2-
Selected equation: {a, b|X} = {a, a|X}  Rule: 7.1(iv)-
Selected equation: {b|X} = {a, a|X}     Rule: 7.1(iv)-
Selected equation: X = {a, b, a, a|_6646}       Rule: 4.2-

************Found a Solution:
E: X = {a, b, a, a|_6646} , Y = {a, a, b, a, a|_6646}
U:
Execution Time:         0

another solution (y./n.) ?

.....

************Found a Solution:
E: X = {b|_6149} , Y = {a, b|_6149}
U:
Execution Time:         10

another solution (y./n.) ?

.....
```

For this example, the Prolog program finds 13 solutions, each being equivalent to either one of the two listed above. This high number of solutions is mainly due to the insertion of multiple copies of `a` in the second set of the second equation after substitution application (i.e., the first application of rule (4.2) in the above run).

Notice that by changing the order of the two equations (i.e., by firing the goal $\{a, b \,|\, X\} = \{a \,|\, Y\}, Y = \{a \,|\, X\}$, one obtains 19 solutions. This different behavior of the Prolog program originates from a different order in rule applications. In fact, in the former case rule (7.1)(iv) is applied immediately after rule (4.2). In this manner, the tails of the two terms are unified in the generated equation $\{a, b \,|\, X\} = \{a, a \,|\, X\}$. This bounds the number of possible (alternative) subsequent instantiations. On the other hand, in the case of the other goal, action (7.2) is applied immediately to the equation $\{a, b \,|\, X\} = \{a \,|\, Y\}$; this event generates a number of non-deterministic choices (cf. Figure 8). Consequently, more redundancy is produced because each one of these choices has to be processed before applying the substitution rule.

3. Consider the system

$$S \;\equiv\; \{a, b|c\} = \{a|Y\} \;\wedge\; Y = \{a|X\} \;\wedge\; c = \{A, B\}$$

Here is an excerpt of the corresponding run:

```
s-unify :- {a,b|c}={a|Y},Y={a|X},c={A,B}.
Selected equation: {a,b|c} = {a|Y}   Rule: 7.2(i)-
Selected equation: {b|c} = Y  Rule: 2.1-
Selected equation: Y = {b|c}  Rule: 4.2-
Selected equation: a = a        Rule: 1-
Selected equation: {b|c} = {a|X}      Rule: 7.2(i)-
Selected equation: c = X        Rule: 2.1-
```

```
Selected equation: X = c        Rule: 4.2-
Selected equation: b = a        Rule: 5.1-fail
Rule: 7.2(ii)-
Selected equation: c = {a|X}    Rule: 6.1(i)-
Selected equation: X = {}       Rule: 4.2-
Selected equation: c = a        Rule: 5.1-fail
Rule: 6.1(ii)-
Selected equation: X = c        Rule: 4.2-
Selected equation: c = a        Rule: 5.1-fail
Rule: 7.2(iii)-
Selected equation: {b|c} = X  Rule: 2.1-
Selected equation: X = {b|c}  Rule: 4.2-
Selected equation: b = a        Rule: 5.1-fail
Rule: 7.2(iv)-
Selected equation: c = {a|_5686}        Rule: 6.1(i)-
Selected equation: _5686 = {}   Rule: 4.2-
Selected equation: c = a        Rule: 5.1-fail
Rule: 6.1(ii)-
Selected equation: _5686 = c    Rule: 4.2-
Selected equation: c = a        Rule: 5.1-fail
Rule: 7.2(ii)-
Selected equation: {b|c} = {a|Y}      Rule: 7.2(i)-
Selected equation: c = Y        Rule: 2.1-
Selected equation: Y = c        Rule: 4.2-
Selected equation: b = a        Rule: 5.1-fail
Rule: 7.2(ii)-
Selected equation: c = {a|Y}    Rule: 6.1(i)-
Selected equation: Y = {}       Rule: 4.2-
Selected equation: c = a        Rule: 5.1-fail
Rule: 6.1(ii)-
Selected equation: Y = c        Rule: 4.2-
Selected equation: c = a        Rule: 5.1-fail
Rule: 7.2(iii)-
Selected equation: {b|c} = Y  Rule: 2.1-
Selected equation: Y = {b|c}  Rule: 4.2-
Selected equation: b = a        Rule: 5.1-fail
Rule: 7.2(iv)-
Selected equation: c = {a|_3414}        Rule: 6.1(i)-
Selected equation: _3414 = {}   Rule: 4.2-
Selected equation: c = a        Rule: 5.1-fail
Rule: 6.1(ii)-
Selected equation: _3414 = c    Rule: 4.2-
Selected equation: c = a        Rule: 5.1-fail
Rule: 7.2(iii)-
Selected equation: {a, b|c} = Y       Rule: 2.1-
Selected equation: Y = {a,b|c}       Rule: 4.2-
Selected equation: a = a        Rule: 1-
Selected equation: {a,b|c} = {a|X}   Rule: 7.2(i)-
Selected equation: {b|c} = X  Rule: 2.1-
Selected equation: X = {b|c}  Rule: 4.2-
Selected equation: a = a        Rule: 1-
Selected equation: c = {A, B}   Rule: 6.2-
Selected equation: c = B        Rule: 2.1-
Selected equation: B = c        Rule: 4.2-
Selected equation: c = A        Rule: 2.1-
Selected equation: A = c        Rule: 4.2-

**********Found a Solution:
```

```
E: A = c , B = c , X = {b|c} , Y = {a,b|c}
U:
Execution Time:         10

another solution (y./n.) ?

....

************Found a Solution:
E: A = c , B = c , X = {a,b|c} , Y = {a,b|c}
U:
Execution Time:         10

....

no more solutions
```

Notice that the equations `A=c` and `B=c`, occurring in the solutions, force `A` and `B` to be self-singletons (namely, `c`) even if this fact is not entailed by the store `U`.

4. Let us recall the reduction of 3-SAT to the set-unification problem as described in Remark 3.6. In particular, consider an instance of 3-SAT of the form:

$$V_1 \wedge V_2 \wedge (V_3 \vee \neg V_1 \vee \neg V_2) \wedge \cdots \wedge (V_i \vee \neg V_1 \vee \neg V_2),$$

where $i \geqslant 3$. According to our reduction, such an instance translates to an equation of the form:

$$\{\{X_1, Y_1\}, \ldots, \{X_i, Y_i\}, \{X_1, \emptyset\}, \{X_2, \emptyset\}, \{X_3, Y_1, Y_2, \emptyset\}, \ldots, \{X_i, Y_1, Y_2, \emptyset\} = \{\{\emptyset, \{\emptyset\}\}\}.$$

Below we list some of the runs of our Prolog program corresponding to different values of i (for briefness, we switched-off the tracing facility). As expected, the execution times of such runs emphasize the exponential behaviour of the algorithm.

```
s-unify :- {{X1,Y1},{X2,Y2},{X3,Y3},{X1,{}},{X2,{}},{X3,Y1,Y2,{}}} = {{{},{{}}}}.

************Found a Solution:
E: Y3 = {}, X1 = {{}}, X2 = {{}}, X3 = {{}}, Y1 = {}, Y2 = {}
U:
Execution Time:         100


s-unify :- {{X1,Y1},{X2,Y2},{X3,Y3},{X4,Y4},{X1,{}},{X2,{}},{X3,Y1,Y2,{}},
           {X4,Y1,Y2,{}}} = {{{},{{}}}}.

************Found a Solution:
E: Y3 = {}, Y4 = {}, X1 = {{}}, X2 = {{}},  X3 = {{}}, X4 = {{}}, Y1 = {}, Y2 = {}
U:
Execution Time:         400


s-unify :- {{X1,Y1},{X2,Y2},{X3,Y3},{X4,Y4},{X5,Y5},{X1,{}},{X2,{}},
           {X3,Y1,Y2,{}},{X4,Y1,Y2,{}},{X5,Y1,Y2,{}}} = {{{},{{}}}}.

************Found a Solution:
E: Y3 = {}, Y4 = {}, Y5 = {}, X1 = {{}}, X2 = {{}}, X3 = {{}}, X4 = {{}}, X5 = {{}},
   Y1 = {}, Y2 = {}
U:
Execution Time:         1070
```

```
s-unify :- {{X1,Y1},{X2,Y2},{X3,Y3},{X4,Y4},{X5,Y5},{X6,Y6},{X1,{}},{X2,{}},
           {X3,Y1,Y2,{}},{X4,Y1,Y2,{}},{X5,Y1,Y2,{}},{X6,Y1,Y2,{}}} = {{{},{{}}}}.

************Found a Solution:
E: Y3 = {}, Y4 = {}, Y5 = {}, Y6 = {}, X1 = {{}}, X2 = {{}}, X3 = {{}}, X4 = {{}},
   X5 = {{}}, X6 = {{}}, Y1 = {}, Y2 = {}
U:
Execution Time:         2660


s-unify :- {{X1,Y1},{X2,Y2},{X3,Y3},{X4,Y4},{X5,Y5},{X6,Y6},{X7,Y7},{X1,{}},
           {X2,{}},{X3,Y1,Y2,{}},{X4,Y1,Y2,{}},{X5,Y1,Y2,{}},{X6,Y1,Y2,{}},
           {X7,Y1,Y2,{}}} = {{{},{{}}}}.

************Found a Solution:
E: Y3 = {}, Y4 = {}, Y5 = {}, Y6 = {}, Y7 = {}, X1 = {{}}, X2 = {{}}, X3 = {{}},
   X4 = {{}}, X5 = {{}}, X6 = {{}}, X7 = {{}}, Y1 = {}, Y2 = {}
U:
Execution Time:         6310
```

■

4 Conclusions

Various approaches can be adopted in axiomatizing and instrumenting hybrid set/multiset/hyperset theories, i.e., theories on sets or other kinds of aggregates ultimately based on atoms. On the one hand, as mentioned in the Introduction and detailed in [COP01, p. 263], reduction to the pure case constitutes the conceptually simplest (though naïve) approach. On the other hand, a treatment involving the notion of colored set may offer practical advantages in real implementations. Among others, multisets with colors are shown in [DPR01] to be well-suited to implement P-systems (the abstract model behind membrane computing [Pāu00]).

In this paper we introduced a third viable treatment, which models atoms as self-singletons following a proposal by Quine. One could have expected that the unification problem—as well as more general solvable cases of the decision problem—would call for significantly diversified solution techniques with the different approaches. On the opposite, the decision problem for $\exists^*\forall$-sentences, and also the more specific unification problem, have been solved by readjusting techniques previously designed for the cases of pure (or colored) sets and hypersets. In a sense, this shows that one can cope with each one of the three approaches to hybrid sets by exploiting essentially the same combinatorial ideas. In itself, this is a remarkable discovery: in the past we found a striking analogy, as regards the decision problem, between pure sets and hypersets [OP95], which however got disrupted, even for the special case of unification, when our focus shifted from the pure to the hybrid context [DOP99]. The uniformity of the results in this paper with earlier results, reassuring as it may be from an exclusively mathematical point of view, did not exempt us from having to carry out a new effort, which had its own rewards. Very much like earlier results paved the way to the new ones offered in this paper, certain new proofs which arose from our present context indicate how to improve earlier results, e.g. how to shorten the proof that goal-driven unification for colored sets always terminates.

We believe we have captured the most critical issues entering into the design of a unification algorithm for the third approach. However, refinements are certainly feasible and also desirable: it turns out, for example, that the instances

$$\{c_1, \ldots, c_n\} = \{c_n, \ldots, c_1\}$$

of unification with distinct constants c_i require rapidly growing execution times as n grows. This

problem would be circumvented if the four alternatives of rule (7.2) were not tried in the fixed order of Figure 8, but controlled by suitable heuristics.

Acknowledgments

We thank the anonymous referees for their useful and accurate comments, and Alberto Policriti for fruitful discussions.

References

[Ack37] W. Ackermann. Die Widerspruchsfreiheit der allgemeinen Mengenlhere. *Mathematische Annalen*, vol. 114, 1937.

[Acz88] P. Aczel. *Non-Well-Founded Sets*. CSLI Lecture Notes, vol. 14, Stanford, 1988.

[BS98] F. Baader and K. U. Schulz. Unification theory—an introduction. In W. Bibel and P. H. Schmitt, eds., *Automated Deduction. A basis for application*, vol. 1, Kluwer Academic Publishers, 1998.

[BS01] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*, vol. I, chapter 8, pages 445–532. Elsevier Science, 2001.

[BM96] J. Barwise and L. Moss. *Vicious Circles*. CSLI Publications, vol. 60, Stanford, 1996.

[Bel99] J. G. F. Belinfante. On Computer-Assisted Proofs in Ordinal Number Theory. *Journal of Automated Reasoning*, 22:341–378, 1999.

[BP96] D. Bellè and F. Parlamento. Decidability of the $\forall^*\exists^*$-class in the membership theory NWL. *Proceedings of Gödel'96-Logical Foundations of Mathematics, Computer Science and Physics-Kurt Gödel's Legacy*. Brno, Czech Republic. Lecture Notes in Logic, vol. 6, Springer-Verlag, 1996.

[BC*92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation* 98(2):142–170, 1992.

[CGO88] D. Cantone, S. Ghelfo, and E. G. Omodeo. The automation of syllogistic. I: Syllogistic normal forms. *Journal of Symbolic Computation*, 6(1):83–98, 1988.

[COP90] D. Cantone, E. G. Omodeo, and A. Policriti. The automation of syllogistic. II: Optimization and complexity issues. *Journal of Automated Reasoning*, 6(2):173–187, 1990.

[COP01] D. Cantone, E. G. Omodeo, and A. Policriti. *Set Theory for Computing – From decision procedures to declarative programming with sets*. Monographs in Computer Science, Springer-Verlag, 2001.

[COU02] D. Cantone, E. G. Omodeo, and P. Ursino. Formative processes with applications to the decision problem in set theory: I. Powerset and singleton operators. *Information and Computation*, 172:165–201, 2002.

[DOP99] A. Dovier, E. G. Omodeo, and A. Policriti. Solvable set/hyperset contexts: II. A goal-driven unification algorithm for the blended case. *Applicable algebra in engineering, communication and computing*, 9(4):293-332, 1999.

[DOPR96] A. Dovier, E. G. Omodeo, E. Pontelli, and G. Rossi. {log}: A Language for Programming in Logic with Finite Sets. *Journal of Logic Programming*, 28(1):1–44, 1996.

[DPR01] A. Dovier, C. Piazza, and G.-F. Rossi. Multiset rewriting by multiset constraint solving. *Romanian Journal of Information Science and Technology*, 4(1–2), 2001.

[DPR98] A. Dovier, A. Policriti, and G.-F. Rossi. A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms. *Fundamenta Informaticae*, 36(2/3):201–234, 1998.

[DPPR00] A. Dovier, C. Piazza, E. Pontelli, and G.-F. Rossi. Sets and constraint logic programming. *ACM Transactions on Programming Languages and Systems*, 22(5):861–931, 2000.

[DF89] E.-E. Doberkat and D. Fox. *Software Prototyping mit SETL*. B. G. Teubner, Stutgart, 1989.

[End77] H. B. Enderton. *Elements of set theory*. Academic Press, New York, 1977–1997.

[Far01] W. M. Farmer. STMM: A Set Theory for Mechanized Mathematics. *Journal of Automated Reasoning*, 26(3):269–289, 2001.

[FOT01] A. Formisano, E. G. Omodeo, and M. Temperini. Instructing equational set-reasoning with Otter. In R. Gore, A. Leitsch, and T. Nipkow, eds., *Automated Reasoning. Proceedings of International Joint Conference IJCAR 2001—(CADE + FTP + TABLEAUX)*, Lecture Notes in Computer Science 2083, pages 152–167, Springer-Verlag, Berlin, 2001.

[Gog78] D. Gogol. The $\forall_n\exists$-completeness of Zermelo-Fraenkel set theory. *Zeitschr. f. math. Logik und Grundlagen d. Math.*, 24(4):289-290, 1978.

[Gor96] M. Gordon. Set theory, higher order logic or both? *Lecture Notes in Computer Science*, 1125:191–202, Springer-Verlag, 1996.

[HL94] P. Hill and J. Lloyd. *The Gödel Programming Language*, MIT Press, Cambridge, MA, 1994.

[Jec79] T. Jech. *Set theory*, Number 79 in Pure and Applied Mathematics—A Series of Monographs and Textbooks. Academic Press, 1978.

[KP95] J. P. Keller and R. Paige. Program derivation with verified transformations - A case study. *Comm. Pure Appl. Math.*, 48(9-10):1053–1113, 1995. Special issue in honor of J. T. Schwartz.

[Lam02] L. Lamport. *Specifying Systems – The TLA+ Language and Tools for Hardware and Software Engineers*. Pearson Education, Inc., Addison-Wesley, 2002.

[Lev79] A. Levy. *Basic Set Theory*. Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1979.

[Mizar] Mizar Project Home Page. `http://www.mizar.org`.

[MS96] I. Meisels and M. Saaltink. The Z/EVES reference manual (for version 1.3). Technical Report TR-96-5493-03b, ORA Canada, November 1996.

[Noë93] P. A. J. Noël. Experimenting with Isabelle in ZF set theory. *Journal of Automated Reasoning*, 10(1):15–58, 1993.

[OPP93] E. G. Omodeo, F. Parlamento, and A. Policriti. A derived algorithm for evaluating ε-expressions over abstract sets. *Journal of Symbolic Computation*, 15:673–704, 1993.

[OPP96] E. G. Omodeo, F. Parlamento, and A. Policriti. Decidability of $\exists^*\forall$-sentences in Membership Theories. *Mathematical Logic Quarterly*, 42(1):41–58, 1996.

[OP95] E. G. Omodeo and A. Policriti. Solvable set/hyperset contexts: I. Some decision procedures for the pure, finite case. *Comm. Pure Appl. Math.*, 48(9-10):1123–1155, 1995. Special issue in honor of J. T. Schwartz.

[Pau95] L. C. Paulson. *Set Theory for Verification. II: Induction and Recursion. Journal of Automated Reasoning*, 15(2):167–215, 1995.

[Pau97] L. C. Paulson. Generic automatic proof tools. In R. Veroff, editor, *Automated Reasoning and Its Applications*, chapter 3. MIT Press, 1997.

[PG96] L. C. Paulson and K. Grabczewski. Mechanizing set theory. *Journal of Automated Reasoning*, 17(3):291–323, December 1996.

[Pāu00] G. Pāun. Computing with Membranes. *Journal of Computer and System Science*, 61(1):108–143, 2000.

[PPR97] F. Parlamento, A. Policriti, and K. P. S. B. Rao. Witnessing differences without redundancies. *Proceedings of the American Mathematical Society*, 125(2):587–594, 1997.

[PS95] A. Policriti and J. T. Schwartz. T-Theorem Proving I. *Journal of Symbolic Computation* 20(3):315–342, 1995.

[Qua92] A. Quaife. *Automated development of fundamental mathematical theories.* Kluwer Academic Publishers, 1992.

[Qui63] W. V. Quine. *Set theory and its logic.* The Belknap Press of Harvard University Press, Cambridge, Massachusetts, revised edition, 3^{rd} printing, 1971.

[Ras95] O. Rasmussen. A Ruby Proof System. Technical Report ID-TR-1995-161, Department of Computer Science, Technical University of Denmark, 1995.

[Ras96] O. Rasmussen. Ensuring Correctness of Ruby Transformations. In M. Sheeran and S. Singh, eds., *Designing Correct Circuits. Proceedings of the 3rd Workshop on Designing Correct Circuits (DCC96)*, Springer-Verlag, Berlin, 1996.

[Saa97] M. Saaltink. The Z/EVES system. In *ZUM '97: Z Formal Specification Notation. Proceedings of the 11th International Conference of Z Users*, pages 72–85, Berlin, Germany, 3–4 April 1997. Springer-Verlag.

[SDDS86] J. T. Schwartz, R. K. B. Dewar, E. Dubinsky, and E. Schonberg. *Programming with Sets: An introduction to SETL.* Texts and Monographs in Computer Science. Springer-Verlag, 1986.

[setlog] The {log} Project Home Page. `http://www.math.unipr.it/~gianfr/setlog.Home.html`.

[SETS] The Programming with {SETS} Home Page. `http://www.cs.nmsu.edu/~complog/sets`.

[SICStus] Swedish Institute for Computer Science. Sicstus Prolog Home Page. `http://www.sics.se/sicstus`.