Minimal Set Unification*

Puri Arenas-Sánchez¹ and Agostino Dovier²

1 Dpto. Informática y Automática, Fac. C.C. Matemáticas, (U.C.M.).
Avda Complutense s/n, 28040 MADRID. puri@dia.ucm.es
² Dip. di Informatica, Univ. di Pisa. Corso Italia 40, 56100 PISA (I). dovier@di.unipi.it

Abstract. A unification algorithm is said to be minimal for a unification problem if it generates exactly a complete set of minimal unifiers, without instances, without repetitions. Aim of this paper is to describe a new set unification algorithm minimal for a significant collection of sample problems that can be used as benchmarks for testing any set unification algorithm. To this end, a deep combinatorial study for such problems has been realized. Moreover, an existing naïve set unification algorithm has been also tested in order to show its bad behavior for most of the sample problems.

Keywords: Logic Programming with Sets, CLP, Unification.

1 Introduction

The drawing up of many papers concerning Constraint Logic Programming with Sets (see e.g. [5, 6, 12]) has pointed out that the complexity of the (NP complete—see e.g. [9]) set unification problem is the real bottleneck of any attempt to extend Logic Programming with set entities.

The loss of the uniqueness of the most general unifier property forces any set unification algorithm to compute a complete set of unifiers (i.e., as it will be explained in Sect. 2, a set of unifiers which covers all possible solutions for the problem at hand) for any satisfiable input. Since the cardinality of a complete set of unifiers grows exponentially w.r.t. the size of the unification problem, it seems inconvenient to return all the unifiers at once. Rather, the non-determinism lying inside any Logic Programming interpreter suggests the use of a non-deterministic unification algorithm which returns exactly one unifier for each non-deterministic branch: in this way any non-deterministic computation can run without undergoing the unpleasant effects of the non-uniqueness of the most general unifier.

In [2] it has been shown that if a representation for sets comprising also a constant symbol for the universal set and the set-minus operator is adopted, the unique most general unifier theorem can be recovered. However, this approach is more properly described as boolean unification rather than set unification.

* P. Arenas-Sánchez is partially supported by the Spanish National Project TIC92-0793-C02-01 "PDR"and the Esprit BRA Working Group Nr. 6028 "CCL". The work is partially supported by C.N.R. grant 94.00472.CT12, "Logic Programming with Sets".
Nested sets are not allowed, and the answer to a unification problem contains a large amount of information, becoming scarcely readable.

In [12] any set unification problem is delayed until it is transformed into a simple ground 'test'. This improves efficiency, however, if the two terms do not become ground, obscure answers such as \( \{X_1, f(X_1, X_3)\} = \{Y_1, f(\{Y_3\}, X_1), X_2\} \) are returned.

There are different ways to represent a finite set. Among them, the union of singletons representation which depicts \( \{s_1, \ldots, s_m\} \) as \( \{s_1\} \cup \ldots \cup \{s_m\} \), and the list representation which uses the term \( \{s_1 | \{s_2 | \cdots \{s_m | \} \} \cdots \} \) for denoting the same set. The former representation (which is associated with an ACI equational theory—cf. [13]) is more expressive than the latter (which is associated to the equational theory described in Sect. 2). For instance, the problem \( X_1 \cup \cdots \cup X_m = \{a_1\} \cup \cdots \cup \{a_n\} \), where \( X_i \)'s are pairwise distinct variables and \( a_j \)'s are pairwise distinct constant symbols, admits \( (2^m - 1)^n \) independent solutions. Since the semantics of \( \{t | s\} \) is \( s \cup \{t\} \), if \( m > 1 \) such problem cannot be expressed by a list representation.

Since the minimum cardinality of a complete set of unifiers expressible with the list representation is itself conspicuous (cf. Sect. 3), we prefer to deal with such representation, avoiding further problems that the "union of singletons" approach would open. The same choice has been performed in [5, 12].

In this paper we present a new Set Unification Algorithm SUA which is minimal (it computes a complete set of minimal unifiers) for a significant collection of sample problems. In order to prove the minimality of SUA, a deep combinatorial study has been necessary, in order to compute the minimal number of solutions for our sample unification problems. In our opinion, the selected problems can be used as benchmarks for testing any set unification algorithm. Two reasons justify our choice: their simplicity (which reflects into a simplification of the analysis) and the fact that they maximize the number of solutions for unification problems of given size (the presence of distinct variables as elements of the sets to be unified guarantees the maximum number of solutions).

The aim of getting minimal algorithms for set unification has already been treated in literature. For instance in [14] three set unification algorithms are proposed. The most efficient seems to be the third one, however, comparing its results with our minimality study, it is possible to conclude that such algorithm is not minimal for problems (6), (7), and (8) described in Sect. 3. In [3], the presented na"ive set unification algorithm (based on [8]) has a minimal behavior only for the first of such benchmarks, as it will be proved in Sect. 4.

The paper is organized as follows: in Sect. 2 we comment briefly on some preliminary concepts needed in the rest of the paper. Sect. 3 presents eight sample unification problems with their corresponding minimality studies and the recursive functions computing the number of solutions. Numerical values for such functions are reported in appendix. The behavior of a na"ive set unification algorithm—based on [8]—on such benchmarks is analyzed in Sect. 4. In Sect. 5 a new set unification algorithm, named SUA, is presented, proving its termination and minimality for all sample problems suggested. Some conclusions are finally drawn up in Sect. 6.