Uniform $\boldsymbol{E}$-Semi-Unification

Alberto Oliart* and Víctor Pérez

Laboratorio Nacional de Informática Avanzada A.C.
aoliart@lania.mx,vperez@lania.mx

1 Introduction

We want to study the problem of Semi-Unification under ecuational theories, or E-Semi-Unification. The Semi-Unification problem has applications in areas of computer science like programming languages, term rewriting, and automated deduction [2,4–6]. The general form of the syntactic semi-unification problem (see below for definitions) has been shown to be undecidable [7], but there are a number of restrictions andor generalizations that can be shown to be decidable. One such restriction is the so called uniform case, which is decidable with a polinomial time algorithm [8].

Definition 1. A *semi-unification instance* is a set $\Gamma = \{t_1 \leq^? u_1, \ldots, t_n \leq^? u_n\}$ of pairs of terms. We say that a semi-unification instance Γ has a solution if and only if there is a substitution σ, called the semi-unifier, and substitutions $\rho_1, \rho_2, \ldots, \rho_n$ such that $t_1\sigma\rho_1 = u_1\sigma, \ldots, t_n\sigma\rho_n = u_n\sigma$.

The *Semi-Unification Problem* (SUP for short) is: Given an arbitrary semi-unification instance Γ decide whether Γ has a semi-unifier. If Γ contains a single pair, then we call it the *Uniform Semi-Unification Problem* (USUP). It has been shown in [7] that SUP is undecidable. USUP, on the other hand, is decidable [2,3,9], with a time complexity of $O(n^2\alpha(n)^2)$, where $\alpha(n)$ is the inverse of Ackerman's function [8].

Definition 2. A substitution σ is the *principal solution* of a semi-unification instance if and only if for any other solution σ', $\sigma \leq \sigma'$.

Definition 3. Let E be a set of identities. An E-Semi-Unification instance is a set of pairs of terms $\Gamma = \{t_1 \leq^?_E u_1, \ldots, t_n \leq^?_E u_n\}$. A E-semi-unification instance has a solution if and only if there is a substitution σ such that

$$t_1\sigma \leq_E u_1\sigma, \ldots, t_n\sigma \leq_E u_n\sigma.$$

We denote the set of all E-Semi-Unifiers of a given Γ as $\mathcal{S}_E(\Gamma)$

* Supported by CONACyT grant 32029-A

Let Γ be a uniform E-Semi-Unification instance, and let X be the set of variables occurring in Γ. A *complete set of E-Semi-unifiers* for Γ is a set CSU such that $CSU \subseteq \mathcal{S}_E(\Gamma)$ and for each $\sigma_1 \in \mathcal{S}_E(\Gamma)$ there is $\sigma_2 \in CU$ such that $\sigma_2 \leq_E \sigma_1$ over X. We say that CU is a *minimal complete set of E-Semi-unifiers* if and only if for any two elements of CU σ_1, σ_2 we have that σ_1 and σ_2 are not comparable with respect to $\leq_E$.

We have experimented with an algorithm that can be used to solve the syntactic uniform case, and that can be easily extended to the equational problem. The application of this algorithm first appeared in [11], but it was used to solve just a particular case, and in the context of adding dimensions to a typing system for a particular programming language.

2 A Procedure for Uniform *E*-Semi-Unification.

In [1], Mathias Baaz introduced a partial procedure for SUP. Since SUP is not decidable, the procedure may never stop if there is no solution to the given instance. We have a version of Baaz's algorithm that solves the syntactic Uniform Semi-Unification Problem, which we don't show for space reasons. We present here a modification of Baaz's algorithm, based on an idea by Rittri in [10,11].

We need to "extend" the set of variables X to a set $\overline{X}$, and define a set of functions $[]_w : \overline{X} \to \overline{X}$. The elements of $\overline{X}$ are of the form x_w, where $w \in \mathbb{N}^*$. Let $\mathcal{T}_{\Sigma,\overline{X}}$ be the set of all finite terms over $\overline{X}$ and Σ. The variable $x_\varepsilon \in \overline{X}$, where ε is the empty string, is identified with the variable $x \in X$. This way we can think of X as a subset of $\overline{X}$, and also of $\mathcal{T}$ as a subset of $\overline{\mathcal{T}}$. For $w \in \mathbb{N}^*$ let $[\,]_w : \overline{X} \to \overline{X}$ be the function defined by:

$$[x_v]_w = x_{vw}$$

for every $x_v \in \overline{X}$.

We now show a generalized version of the Baaz procedure. The procedure presented next is an extension to Baaz's procedure as presented in [1], but in our case with the restriction that an instance can only have one inequality.

1. Let $U_0 = \{\Sigma_0 = id\}$ and $i = 0$.
2. Let $U_{i+1} = \emptyset$. For each $\sigma_i^j \in U_i$ do
   (a) Let $\Gamma_i^j = \{[\sigma_i^j(t)]_i =_E^? \sigma_i^j(u)\}$ be an instance of E-Unification.
   (b) Let $\mathcal{U}_E(\Gamma_i^j)\}$ be the minimal complete set of E-Unifiers. For each $\gamma \in \mathcal{U}_E(\Gamma_i^j)\}$ do:
      i. If γ is a renaming on $\sigma_i^j(u)$, then σ_i^j is a solution, add it to $\mathcal{SU}_E(\Gamma)$, if there is no $\sigma \in \mathcal{SU}_E(\Gamma)$ such that $\sigma \leq_E \sigma_i^j$.
      ii. Else, $U_{i+1} = U_{i+1} \cup \{\gamma \circ \sigma_i^j\}$.
3. If $U_{i+1} = \emptyset$ then STOP. If $\mathcal{SU}_E(\Gamma)$ is empty, then Γ has no solutions, else return $\mathcal{SU}_E(\Gamma)$.
4. Goto 2.

This algorithm builds a set of E-Semi-unifiers. If the corresponding unification type of E is 1, this is, it has the most general unifier property, then this algorithm becomes very similar to the one presented by Rittri in [11]. In order to have a total procedure when this general procedure does not give us one, we need to have some "external" check, or modify the procedure in some way.

3 An Example: Commutativity

Since C-Unification is finitary, we can apply the above algorithm to C-Semi-Unification instances. There are no known algorithms for C-Unification that return the complete set of unifiers, but we can easily solve the problem by eliminating the redundant ones.

Example 1. $\Gamma = \{f(a,b) \leq^?_C f(x,y)\}$ We consider now the case of commutativity. In the example above, the application of the renamings have no effect, so $\Gamma_0 = \{f(a,b) =^?_C f(x,y)\}$. The C-Unification algorithm produces the minimal set of C-unifiers: $\mathcal{U}_C(\Gamma_1) = \{\gamma^1_1 = \{x := a, y := b\},\ \gamma_2 = \{x := b, y := a\}$. With this we get the set $U_1 = \{\sigma^1_1 = \gamma_1,\ \sigma^1_2 = \gamma_2\}$. For the next cycle, we have the instances $\Gamma^2_1 = \{f(a,b) =^?_C f(a,b)\}$ and $\Gamma^2_2 = \{f(b,a) =^?_C f(a,b)\}$ both having the identity as unique solution, and with this the algorithm ends.

Example 2. $\Gamma = \{f(f(x,y),z) \leq^?_C f(x,z)\}$ In this case, we get as the first instance of C-Unification is

$$\Gamma_1 = \{f(f(x_1,y_1),z_1) =^?_C f(x,z)\}$$

which has the minimal set of solutions

$$\mathcal{U}_C(\Gamma_1) = \{\gamma^1_1 = \{x := f(x_1,y_1), z_1 := z\},\ \gamma^2_1 = \{z := f(x_1,y_1), z_1 := x\}.$$

It is easy to see that $\sigma^2_1 = \gamma^2_1$ actually leads to a solution of the original problem, while the other will produce a cycle. In this case, the algorithm never stops, but it actually finds a solution.

To get a total algorithm in this case, we need a way to check for cycles, more or less in the same way that it is done for the syntactic case for Unification. This check can be done counting the lenght of the strings of natural numbers on the term variables, which coincides with the number of times a funtion $[]_w$ has been applied to a variable. In the second example, we have that $x := f(x_1,y_1)$, which means that the length of the subscript for the left hand side x is zero, while the length of the subscript of x_1 is 1, and this is an occues check condition. The algorithm should stop considering this C-Unifier. In this example, the check works, and we can stop reporting the final set of solutions.

4 Conclusions and Future Work

We conjecture that the "occurs check" described above gives a total algorithm for the C case. As mentioned, we have used the same check on the uniform syntactic case. We are now using the general form of the algorithm, as proposed by Rittri, trying to find a more general way of finding procedures to solve E-Semi-unification problems for some equational theories.

We have worked with some theories that are of type 1, as is the case of Boolean Rings, with only partial success. We are trying to embed into the Baaz algorithm some of the techniques that have given good results in E-Unification, as is the case of narrowing.

References

1. Matthias Baaz. Note on the existenec of most general unifiers. In P.Clote and J. Krajicek, editors, *Arithmetic, Proof Theory and Logical Complexity*, pages 20–29. Oxford University Press, 1993.
2. P. Narendran D. Kapur, D. Musser and J. Stillman. Semi-unification. *Theoretical Computer Science*, 81(2):169–188, April 1991.
3. F. Henglein. Type inference with polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2), April 1993.
4. S. Jahama and A.J. Kfoury. A general theory of semi-unification. Technical Report 93-018, Boston University, December 1993.
5. Said Jahama and A.J. Kfoury. Type reconstruction in the presence of polymorphic recursion and recursive types. Technical Report BU-CS 93-019, Boston University, 1993.
6. A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. Type-reconstruction in the presence of polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):290–311, April 1993.
7. A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. The undecidability of the semi-unification problem. *Information and Computation*, 102(1):83–101, January 1993.
8. A. Oliart and W. Snyder. A fast algorithm for semi-unification. In C. Kircher and H. Kircher, editors, *Proceedings of 15th Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 239–253. Springer-Verlag, July 1998.
9. P. Pudlác. On a unification problem related to kreisel's conjecture. *Commentationes Mathematicae Universitatis Carolinae*, 1988. Prague Czechoslovakia.
10. M. Rittri. Semi-unification of two terms in abelian groups. *Info. Processing Letters*, (52):61–68, 1994.
11. M. Rittri. Dimensions inference under polymorphic recursion. In *7th ACM Conference on Functional Programming Languages and Computer Architecture*, pages 147–159. ACM, ACM Press, 1995.