UNIF 2001
15th International Workshop on Unification

Abstracts

June 18–19, 2001
Siena, Italy

Organizing Committee:
Franz Baader
Volker Diekert
Cesare Tinelli
Ralf Treinen

Università degli Studi di Siena
Dipartimento di Ingegneria dell'Informazione
Technical Report DII 09/01

The 15th International Workshop on Unification

UNIF 2001 is the 15th in a series of annual international workshops on unification, the previous ones having been mostly in Europe (France, Germany, Italy, and Spain) and twice in the United States.[1] The topic *unification* is understood in a rather broad sense at these workshops, also encompassing topics such as constraint solving and type checking. A (non-exclusive) list of typical UNIF topics includes:

- General E-unification and Calculi
- Special Unification Algorithms
- Higher-Order Unification
- Unification-Based Approaches to Grammar
- Constraint Solving
- Type Checking and Reconstruction
- Applications
- Narrowing
- Matching
- Combination Problems
- Disunification
- Unification Calculi
- Typed Unification
- Implementations

The 15th International Workshop on Unification was held June 18 – 19 in Siena, Italy, as part of the International Joint Conference on Automated Reasoning (IJCAR 2001).

UNIF 2001 was made possible in part by the kind support of the National Science Foundation through grant n. 0108548. The organizing committee wishes to thank Fabio Massacci, the IJCAR Conference Chair, and Dieter Hutter, the IJCAR Workshop Chair, for their assistance and cooperation.

Franz Baader, RWTH Aachen, Germany
Volker Diekert, Universität Stuttgart, Germany
Cesare Tinelli, University of Iowa, USA
Ralf Treinen, Université Paris-Sud, France

[1] See the UNIF web page at `http://www.lri.fr/~treinen/unif/`.

Contents

Timetable

Monday, June 18

10:30 *Coffee Starter*

11:00 *Invited Talk:* Extensions of AC, by *Paliath Narendran*
12:00 Uniform E-Semi-Unification, by *Alberto Oliart* and *Víctor Pérez*

12:30 *Lunch*

14:00 A Superposition Based Methodology to Design Satisfiability Decision Procedures, by *Alessandro Armando*, *Silvio Ranise* and *Michaël Rusinowitch*
14:30 Combining Decision Procedures for Positive Theories, by *Cesare Tinelli*
15:00 Solving Matching Problems in Unions of Non-Disjoint Equational Theories, by *Christophe Ringeissen*

15:30 *Coffee Break*

16:00 Unification in the Empty and Flat Theories with Sequence Variables and Flexible Arity Symbols, by *Temur Kutsia*
16:30 Unification and Projectivity in Propositional Logic, by *Silvio Ghilardi*
17:00 Solving Linear Equations over Regular Languages, by *Franz Baader* and *Ralf Kuesters*
17:30 Solving Knuth-Bendix ordering constraints, by *Konstantin Korovin* and *Andrei Voronkov*
18:00 Context Congruence Closure and Context Rewriting for Commutative Context-Free Grammars, by *Siva Anantharaman*

18:30 *End of First Day*

Tuesday, June 19

9:30 *Invited Talk:* Simultaneous Rigid E-unification and Formula Instantiation, by *Andrei Voronkov*

10:30 *Coffee Break*

11:00 Theories of Equations in Finitely Presented Groups, by *Markus Lohrey*
11:30 Parametrized Rewriting on Words, by *Benjamin Monate*
12:00 Second-Order Schema Matching Based on Projection Point Labeling, by *Keizo Yamada*, *Kouichi Hirata* and *Masateru Harao*

12:30 *Lunch*

14:00 *Invited Talk:* The Complexity of Word Equations, by *Claudio Gutiérrez*
15:00 Complexity of Unification and Matching Problems in the Varieties of Idempotent Semigroups, by *Ondřej Klíma*

15:30 *Coffee Break*

16:00 On the Complexity of Bounded Second Order Unification, by *Manfred Schmidt-Schauß*
16:30 Approximating E-Unification, by *Christopher Lynch* and *Barbara Morawska*
17:00 Non-Structural Subtype Entailment in Automata Theory, by *Joachim Niehren* and *Tim Priesnitz*

17:30 *End of Workshop*

Invited Talk: Extensions of AC

Paliath Narendran

Department of Computer Science, University at Albany — SUNY,
Albany, NY 12222, USA
http://www.cs.albany.edu/~dran/

Operators that satisfy the Associativity and Commutativity properties appear in many applications of equational unification. In most cases they also have additional properties such as the existence of identity and inverses (i.e., abelian groups). It was seen, early on, that such extensions of AC very quickly lead to undecidable E-unification problems—for instance, adding the distributivity axiom alone would enable a reduction from Hilbert's 10^{th} problem. In this talk, we survey attempts to "push the envelope" as far as possible; i.e., to obtain interesting extensions for which unification is decidable.

Uniform 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 $U_1 = \{\sigma_1^1 = \gamma_1,\ \sigma_2^1 = \gamma_2\}$. For the next cycle, we have the instances $\Gamma_1^2 = \{f(a,b) =^?_C f(a,b)\}$ 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

$$\mathcal{U}_C(\Gamma_1) = \{\gamma_1^1 = \{x := f(x_1,y_1), z_1 := z\},\ \gamma_1^2 = \{z := f(x_1,y_1), z_1 := x\}.$$

It is easy to see that $\sigma_1^2 = \gamma_1^2$ 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.

A Superposition Based Methodology to Design Satisfiability Decision Procedures

Alessandro Armando[1], Silvio Ranise[1,2], and Michaël Rusinowitch[2]

[1] DIST—Universitá di Genova, Genova, Italy, {armando, silvio}@dist.unige.it
[2] LORIA-INRIA-Lorraine, Nancy, France, {ranise, rusi}@loria.fr

1 Introduction

In verification with proof assistants (such as PVS, COQ, HOL, and Nqthm), decision procedures are typically used for eliminating trivial subgoals represented for instance as sequents modulo a background theory. These theories axiomatize standard data-types such as arrays, lists, bit-vectors and have proved to be quite useful for, e.g., hardware verification. Elimination of trivial sequents often reduces to the **problem of proving the unsatisfiability of conjunctions of literals modulo a background theory** T, which is the problem we shall consider here.

The rewriting approach permits us the uniform design of decision procedures for eliminating these subgoals and also offers an efficient alternative to congruence closure techniques. This approach was inspired by Greg Nelson's thesis [Nel81] where it is suggested to apply Knuth-Bendix completion to derive decision procedures. Here, instead of the Knuth-Bendix completion procedure, we apply a standard (refutation) complete superposition-based inference system for clausal equational logic (given for instance in [NR01]). This allows not only to handle pure equality but also several interesting axiomatic theories that were not handled previously that way such as lists, arrays, and extensional arrays. The proof that the decision procedures are correct is straightforward w.r.t. to other correctness proofs available in literature. To see this, consider the decision procedure for an extensional theory of arrays in [SDBL01]. The key ingredient is a modified congruence closure algorithm which is capable of handling "partial equations". The correctness proof is rather complex and it takes the main part of the paper; it is model-theoretic and rather *ad-hoc*. In [ARR01] (see Section 4 for some hints), we design a decision procedure for the same theory which is simpler to understand since it amounts to apply standard equality reasoning in contrast to handling partial equalities. Furthermore, our proof of correctness relies on basic properties of first order logic.

Finally, we notice that in our framework decision procedures for combinations of theories can be readily derived in a manner closely resembling the combination schema described in [NO78]. This is exemplified for a combination of the theory of lists and arrays in [ARR01]. (Notice that non-convex theories [NO78], such as the theory of arrays, are smoothly handled in our framework.)

2 Our two step methodology

Let T be a first-order theory. In the following, with literal we mean an expression of the form $l \bowtie r$, where l and r are two ground terms and $\bowtie$ is either $=$ or $\neq$ ($l = r$ and $l \neq r$ are called equality and disequality, respectively). We are concerned with the *problem of checking whether a set S of ground literals is T-satisfiable*, i.e. if $S \cup T$ is satisfiable.

If T is the quantifier-free theory of equality (which we denote with $\mathcal{E}$), there are many available solutions to solve this problem ranging from congruence closure algorithms to ground completion. Unfortunately, there are no general and uniform solutions when T is a theory extending $\mathcal{E}$. Nonetheless, some *ad-hoc* solutions for certain theories are available in literature. For example in [NO80], Nelson and Oppen describe a decision procedure for the "quantifier-free theory of LISP list structure". The procedure is obtained as an extension of a congruence closure algorithm with a mechanism which augments the graph by selected instances of the axioms of the theory. The proof of correctness is model theoretic and seems difficult to generalize. A discussion of the difficulties to derive a general method to obtain decision procedures by extending congruence closure algorithms as well as a decision procedure for the theory of arrays (without extensionality) can be found in [Nel81]. This discussion has motivated our work.

Let S be the input set of ground literals and $Ax(T)$ be a presentation of a theory T extending $\mathcal{E}$. Our **goal** is to derive a satisfiability decision procedure capable of establishing whether S is T-satisfiable or not. The **key idea** underlying our methodology is to tune a general *refutational complete superposition inference system* derived from [Rus91,BG94], which we call $\mathcal{SP}$, in order to obtain *termination* for the theory T. The proposed methodology consists of the following two steps.

Step 1: flattening. A flat literal is a literal of the form $l \bowtie r$, where l and r are terms of depth at most 1. The first step of our methodology is to flatten all the literals in S by introducing a finite number of "fresh" constants so to obtain a set S' of (ground) flat literals s.t. S' is T-satisfiable iff S is. As an example, consider the set of literals $\{f(c, c') = h(h(a)), h(h(h(a))) \neq a\}$. It can be flattened to the set $\{f(c, c') = h(c_1), c_3 \neq a\} \cup \{c_1 = h(a), c_3 = h(c_2), c_2 = h(c_1)\}$, where c_1, c_2, and c_3 are "fresh" constants.

Step 2: applying the superposition calculus $\mathcal{SP}$. We will make use of $\mathcal{SP}$ comprising the inference rules of Table 1 and the simplification rules of Table 2. $\mathcal{SP}$ is taken 'off-the-shelf' from [NR01]. It extends the system from [Rus91] by the *equality factoring rule* [BG94], so that more ordering restrictions are possible (in the non-Horn case). The binary relation $\succ$ is a reduction ordering [DJ90], which is total on ground terms. $\succ$ is extended to equational literals in the following way: $(a \bowtie b) \succ (c \bowtie d)$ if $\{a, b\} \succ\!\!\succ \{c, d\}$, where $\succ\!\!\succ$ is the multiset extension of $\succ$ and $\bowtie$ is either $=$ or $\neq$. Multisets of literals are compared using the multiset extension of $\succ$ on literals. The calculus $\mathcal{SP}$ is refutationally complete for general first-order equational logic [BG94,NR01]. A *saturation* of a set of clauses by $\mathcal{SP}$ is the final set of clauses

Name	Rule	Conditions
Superposition	$\dfrac{\Gamma \Rightarrow \Delta, l[u'] = r \quad \Pi \Rightarrow \Sigma, u = v}{\Gamma, \Pi \Rightarrow \Delta, \Sigma, l[v] = r}$	$u \not\preceq v$, $l[u'] \not\preceq r$, $*$
Paramodulation	$\dfrac{\Gamma, l[u'] = r \Rightarrow \Delta \quad \Pi \Rightarrow \Sigma, u = v}{l[v] = r, \Gamma, \Pi \Rightarrow \Delta, \Sigma}$	$u \not\preceq v$, $l[u'] \not\preceq r$, $*$
Reflection	$\dfrac{\Gamma, u' = u \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$	$(u' = u) \not\prec (\Gamma \cup \Delta)$
Factoring	$\dfrac{\Gamma \Rightarrow \Delta, u = t, u' = t'}{(\Gamma, t = t' \Rightarrow \Delta, u = t')}$	$u \not\preceq t$, $u \not\preceq \Gamma$, $(u = t) \not\prec \{u' = t'\} \cup \Delta$

Legenda:

- σ is the m.g.u. of the terms u and u' and it is applied to both the premises and the conclusion of each rule (it is omitted for simplicity);
- a *clause* is a disjunction of literals, i.e. an expression of the form $\neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$ (abbreviated with $A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_m$) where $A_1, \ldots, A_n$, $B_1, \ldots, B_m$ are equalities ($n \geq 0$ and $m \geq 0$);
- Γ, Δ, Π, and Σ denote multisets of equalities;
- $*$ stands for$(u = v) \not\preceq (\Pi \cup \Sigma)$, $(l[u'] = r) \not\prec (\Gamma \cup \Delta)$.

Fig. 1. Inference rules of $\mathcal{SP}$

Name	Rule	Conditions
Subsumption	$\dfrac{S \cup \{C, C'\}}{S \cup \{C\}}$	if for some θ, $\theta(C) = C'$, and for no ρ, $\rho(C') = C$
Simplification	$\dfrac{S \cup \{C[\theta(l)], l = r\}}{S \cup \{C[\theta(r)], l = r\}}$	$\theta(l) \succ \theta(r)$, $C[\theta(l)] \succ (\theta(l) = \theta(r))$
Deletion	$\dfrac{S \cup \{\Gamma \Rightarrow \Delta, t = t\}}{S}$	

Legenda:

- C and C' denote clauses whereas S denotes a set of clauses;
- θ and ρ denote substitutions.

Fig. 2. Simplification rules of $\mathcal{SP}$

generated by a fair derivation from S using rules in $\mathcal{SP}$ with higher priority given to the simplification rules. *If the saturation terminates for the union of T and any set of ground flat literals, then it is a decision procedure for T.* More precisely, if the final set of clauses contains the empty clause then the input set of literals is unsatisfiable; it is satisfiable, otherwise. (This is a direct consequence of the refutational completeness of $\mathcal{SP}$.) We shall call $\mathcal{SP}$ any fair application of the inference system with priority given to the simplification rules.

3 Our methodology at work: the theory of lists

Let $\mathcal{L}$ be the theory of lists whose language contains the binary function symbol cons, and the unary function symbols car and cdr. Let $Ax(\mathcal{L}) := \{\mathsf{car}(\mathsf{cons}(x, y)) =$

$x, \mathsf{cdr}(\mathsf{cons}(x,y)) = y\}$.[1] In the following, S denotes a set of ground literals already flattened. Then, the second step of our methodology amounts to analyzing the possible inferences in $\mathcal{SP}$ which are applicable to elements of $S \cup Ax(\mathcal{L})$.

Lemma 1. *The clauses occurring in the saturations of $S \cup Ax(\mathcal{L})$ by $\mathcal{SP}$ can only be the empty clause, ground flat literals, or the equalities in $Ax(\mathcal{L})$.*

Proof sketch. The proof is by induction on the length of the derivations. No inference between axioms in $Ax(\mathcal{L})$ is possible. Thus, by inspection of the rules in $\mathcal{SP}$, there are four cases to consider. The most interesting one is the following. A *Superposition* between an equality in $Ax(\mathcal{L})$ and a ground flat equality of the form $\mathsf{cons}(c_1, c_2) = c_3$ (where c_1, c_2, and c_3 are individual constants) only yields ground flat equalities. The remaining cases are obvious.

¿From the analysis of the possible inference, we can derive termination.

Lemma 2. *All the saturations of $S \cup Ax(\mathcal{L})$ by $\mathcal{SP}$ are finite.*

Proof sketch. It is sufficient to observe that the saturations of $S \cup Ax(\mathcal{L})$ by $\mathcal{SP}$ can only contain the empty clause or a finite number of ground flat literals for Lemma 1.

Finally, from termination, fairness, and the refutation completeness of $\mathcal{SP}$, we derive the following theorem.

Theorem 1. *$\mathcal{SP}$ is a decision procedure for $\mathcal{L}$.*

4 A decision procedure for arrays

We briefly overview how straightforward is to derive a decision procedure for the theory of arrays with extensionality in our framework. The key idea is to take into account the axiom of extensionality by a suitable transformation of the input set of literals and to apply the two step methodology described in Section 2 on the transformed set of input literals and the remaining axioms.

Let $\mathcal{A}_e^s$ be the many-sorted theory containing the sorts ELEM, INDEX, and ARRAY, the function symbols store and select of arity $\text{ARRAY}, \text{INDEX}, \text{ELEM} \longrightarrow \text{ARRAY}$ and $\text{ARRAY}, \text{INDEX} \longrightarrow \text{ELEM}$ respectively, and the set $Ax(\mathcal{A}_e^s)$ of axioms:

$$\mathsf{select}(\mathsf{store}(a,i,e),i) = e \quad (1)$$

$$i \neq j \Rightarrow \mathsf{select}(\mathsf{store}(a,i,e),j) = \mathsf{select}(a,j) \quad (2)$$

$$\forall i.(\mathsf{select}(a,i) = \mathsf{select}(b,i)) \Rightarrow a = b \quad (3)$$

where a and b are variables of sort ARRAY, i and j are variables of sort INDEX. Let $\Sigma_{\mathcal{A}_s^e}$ denote a signature containing the function symbols select, store, and a finite set of constants for each sort. Finally, let $Ax(\mathcal{A}^s)$ denote the set containing only axioms (1) and (2).

[1] Here and in the remainder of the paper, we omit the flattening step for simplicity.

Lemma 3. *Let S be a set of literals (built out of $\Sigma_{\mathcal{A}_s^e}$) and let S' be obtained from S by replacing all the inequalities of the form $t \neq t'$ with $\exists i.\mathsf{select}(t,i) \neq \mathsf{select}(t',i)$, where t and t' are terms of sort* ARRAY. *Then S is $\mathcal{A}_e^s$-satisfiable iff S' is $\mathcal{A}^s$-satisfiable.*

The proof of this Lemma is reported in [ARR01].

Theorem 2. *Let S be a set of literals (built out of $\Sigma_{\mathcal{A}_s^e}$) and let S' be obtained from S by replacing all the inequalities of the form $t \neq t'$ with $\mathsf{select}(t, sk(t,t')) \neq \mathsf{select}(t', sk(t,t'))$, where t and t' are terms of sort* ARRAY, *and sk is a Skolem function of arity* ARRAY, ARRAY $\longrightarrow$ INDEX. *Then S is $\mathcal{A}_e^s$-satisfiable iff S' is $\mathcal{A}^s$-satisfiable.*

This Theorem follows from Lemma 3 and basic properties of skolemization.

At this point, we are left with the problem of checking the satisfiability of a set of ground literals w.r.t. the theory presented by $Ax(\mathcal{A}^s)$. This can be done straightforwardly with a decision procedure obtained by applying the two step methodology described in Section 2 (see [ARR01] for details).

References

[ARR01] A. Armando, S. Ranise, and M. Rusinowitch. Uniform Derivation of Decision Procedures by Superposition. In *CSL'01*, 2001. To appear.

[BG94] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. of Logic and Comp.*, 4(3):217–247, 1994.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Hand. of Theoretical Comp. Science*, pages 243–320. 1990.

[Nel81] G. Nelson. Techniques for Program Verification. Technical Report CSL-81-10, Xerox Palo Alto Research Center, June 1981.

[NO78] G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. Technical Report STAN-CS-78-652, Stanford CS Dept., April 1978.

[NO80] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

[NR01] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Hand. of Automated Reasoning*. 2001.

[Rus91] M. Rusinowitch. Theorem-Proving with Resolution and Superposition. *J. of Symbolic Computation*, (1-2), 1991.

[SDBL01] A. Stump, D. L. Dill, C. W. Barrett, and J. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *LICS'01*, 2001. To appear.

Combining Decision Procedures for Positive Theories

Cesare Tinelli

Department of Computer Science
University of Iowa, USA
tinelli@cs.uiowa.edu

1 Introduction

In [BS95], Baader and Schulz present a non-deterministic procedure for combining decision procedure for the positive consequences of two atomic[1] theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with disjoint signatures into a decision procedure for the positive consequences of union of $\mathcal{T}_1$ and $\mathcal{T}_2$. This procedure is a generalization to the first-order case of a previous procedure by the same authors for the combination of decision procedures for E-unifiability [BS92].

We present a simpler version of the procedure in [BS95] that improves on the original one in the following ways: it is more deterministic, it has a simpler correctness proof and, although not shown here, it is extensible to theories sharing constructors in the sense of [BT00, BT01].

In the following, Σ_1 and Σ_2 will be two disjoint (first-order) signatures and for $i = 1, 2$, $\mathcal{T}_i$ will be a non-trivial atomic Σ_i-theory. We consider both theories as theories with equality, treating the equality symbol (denoted by $\equiv$ here) as a logical constant. We call the *positive theory of* $\mathcal{T}_i$ the set of positive closed Σ_i-formulae entailed by $\mathcal{T}_i$, where a formula is positive if it is in prenex normal form and its matrix's only logical connectives are conjunctions and disjunctions.

We will use the concept of *alien subterm*, defined formally in [BS95,BT01] among others. Informally, let us say that a term (atom) of signature $\Sigma_1 \cup \Sigma_2$ is an i-term (i-atom) if it is a variable or if its top symbol is in Σ_i for $i = 1$ or $i = 2$. Then, an alien subterm of an i-atom α is every maximal subterm of α that is not an i-term.

2 The Combination Procedure

The combination procedure takes as input a positive quantifier-free $(\Sigma_1 \cup \Sigma_2)$-formula φ and outputs a pair made of a positive Σ_1-sentence and a positive Σ_2-sentence. The procedure consists of the following steps.

[1] An *atomic* theory is a first-order theory axiomatized by a set of universally quantified atomic formulae. Atomic theories generalize equational theories to signatures with relation symbols.

Step 0: Convert into DNF. Convert the input φ into the disjunctive normal form $\psi_1 \vee \cdots \vee \psi_n$ and choose a disjunct ψ_j.

Step 1: Convert into Separate Form. Let S be the set of ψ_j's conjuncts. Purify each atom $\alpha \in S$ by abstracting every alien subterm t of α by a fresh variable v and adding the equation $v \equiv t$ to S. For the purification, if α is an equation of the form $t_1 \equiv t_2$, treat $\equiv$ as a Σ_i-symbol if t_1 and t_2 are both i-terms for $i = 1$ or $i = 2$; treat it as a Σ_1-symbol otherwise. Iterate the process on the new set S until no further purification is possible. For $i = 1, 2$, let φ_i be the conjunction of all Σ_i-atoms in S.[2]

Step 2: Identify Shared Variables. Choose a partition of all the variables occurring in both φ_1 and φ_2; pick a representative from each block in the partition and replace in φ_1 and φ_2 all variables in the same block by the chosen representative. For $i = 1, 2$, let γ_i be the formula obtained from φ_i this way.

Step 3: Partition Shared Variables. Let V be the set of all variables shared by γ_1 and γ_2. Then for $i = 1, 2$, let $\mathbf{u}_i$ be a tuple consisting of the variables of γ_i not in V. Choose a partition $\{V_1, V_2, V_3\}$ of V and for $i = 1, \ldots, 3$, let $\mathbf{v}_i$ be a tuple consisting of all the variables in V_i.

Step 4: Generate Output Pair. Output the pair of sentences

$$\langle\, \forall \mathbf{v}_1\, \exists \mathbf{v}_2\, \forall \mathbf{v}_3\, \exists \mathbf{u}_1\, \gamma_1(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{u}_1),\ \forall \mathbf{v}_1, \mathbf{v}_2\, \exists \mathbf{v}_3, \mathbf{u}_2\, \gamma_2(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{u}_2)\,\rangle.$$

Ignoring minor details, this procedure differs from Baader and Schulz's as follows. In the step corresponding to Step 3 above, their procedure generates an arbitrary partition $\mathbf{v}_1, \ldots, \mathbf{v}_{2n}$ of the shared variables V, whereas our procedure generates a partition with at most three blocks: $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$.[3] In the step corresponding to Step 4 above, Baader and Schulz's procedure generates the (in general, full) positive sentences

$$\forall \mathbf{v}_1\, \exists \mathbf{v}_2\, \cdots \forall \mathbf{v}_{2n-1}\, \exists \mathbf{v}_{2n}\, \exists \mathbf{u}_1\, \gamma_1 \quad \text{and} \quad \exists \mathbf{v}_1\, \forall \mathbf{v}_2\, \cdots \exists \mathbf{v}_{2n-1}\, \forall \mathbf{v}_{2n}\, \exists \mathbf{u}_2\, \gamma_2,$$

whereas our procedure always generates a $\forall\exists\forall\exists$-sentence and a $\forall\exists$-sentence.

The combination procedure above is sound and complete in the following sense.

Proposition 1 (Soundness and Completeness). *For every input $\varphi(\mathbf{v})$ to the combination procedure, $\mathcal{T}_1 \cup \mathcal{T}_2 \models \exists \mathbf{v}\, \varphi(\mathbf{v})$ iff there is a possible output (ψ_1, ψ_2) such that $\mathcal{T}_1 \models \psi_1$ and $\mathcal{T}_2 \models \psi_2$.*

The soundness and completeness proof that Baader and Schulz provide in [BS95] for their method relies on a fairly complex algebraic construction producing a free model of $\mathcal{T}_1 \cup \mathcal{T}_2$ as an amalgamation of the free model $\mathcal{A}_1$ of $\mathcal{T}_1$ with (countably)

[2] Equations between variables are considered arbitrarily as either Σ_1- or a Σ_2-atoms.

[3] We say *at most* three blocks because each of $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ can be the empty tuple.

infinitely-many generators and the free model $\mathcal{A}_2$ of $\mathcal{T}_2$ with (countably) infinitely-many generators. Our proof instead relies on a considerably simpler construction, similar to one described in [BT01]. This construction too yields a model $\mathcal{A}$ of $\mathcal{T}_1 \cup \mathcal{T}_2$ by combining $\mathcal{A}_1$ and $\mathcal{A}_2$. Although $\mathcal{A}$ is not a free model of $\mathcal{T}_1 \cup \mathcal{T}_2$ as in [BS95], it has a substructure that is so, which is enough for our purposes.

3 Decidability Results

Together with the fact that every step in the combination procedure is finitary (i.e., it has only finitely many possible executions) and effective, Proposition 1 yields the following decidability result.

Proposition 2. *If the positive $\forall\exists\forall\exists$-theory of $\mathcal{T}_1$ and the positive $\forall\exists$-theory of $\mathcal{T}_2$ are decidable, then the positive $\exists$-theory of $\mathcal{T}_1 \cup \mathcal{T}_2$ is decidable.*

As stated, the result above is asymmetric in $\mathcal{T}_1$ and $\mathcal{T}_2$: for $\mathcal{T}_1$, it requires that the positive $\forall\exists\forall\exists$-theory be decidable; for $\mathcal{T}_2$, it requires that just the positive $\forall\exists$-theory be decidable. This may be puzzling because our initial assumptions on $\mathcal{T}_1$ and $\mathcal{T}_2$ are the same. As it turns out, the asymmetry is only apparent. In fact, using the very result in Proposition 2 we can show that once the positive $\forall\exists$-theory of $\mathcal{T}_i$ $(i = 1, 2)$ is decidable, then the *full* positive theory of $\mathcal{T}_i$ is decidable. Here follows a sketch of the proof.

For every signature Δ, let $\emptyset^{\Delta}$ denote the empty Δ-theory. Using well-known results on the theory of finite trees [Mah88], it is possible to show that the positive theory of $\emptyset^{\Delta}$ is decidable if Δ is infinite and has no predicate symbols.

Lemma 1. *Let $i \in \{1, 2\}$ and let Δ be any infinite functional signature disjoint with Σ_i. If the positive $\forall\exists$-theory of $\mathcal{T}_i$ is decidable, then the (full) positive theory of $\mathcal{T}_i \cup \emptyset^{\Delta}$ is decidable.*

To prove this lemma one first observes that a $(\Sigma_i \cup \Delta)$-sentence φ belongs to the positive theory of $\mathcal{T}_i \cup \emptyset^{\Delta}$ iff the existential sentence φ_{s} obtained by Skolemizing the universal quantifiers in φ belongs to the positive theory of $\emptyset^{\Delta_{\mathrm{s}}} \cup \mathcal{T}_i$, where Δ_{s} is the signature obtained by adding the Skolem symbols in φ_{s} to Δ. Now, the positive $\exists$-theory of $\emptyset^{\Delta_{\mathrm{s}}} \cup \mathcal{T}_i$ is decidable by an application of Lemma 2 to $\emptyset^{\Delta_{\mathrm{s}}}$ and $\mathcal{T}_i$.

Lemma 1 readily leads to our claim.

Proposition 3. *Let $i \in \{1, 2\}$. The full positive theory of $\mathcal{T}_i$ is decidable iff if the positive $\forall\exists$-theory of $\mathcal{T}_i$ is decidable.*

The main consequence of Lemma 1 is the following.

Theorem 1. *If the positive $\forall\exists$-theory of $\mathcal{T}_1$ and the positive $\forall\exists$-theory of $\mathcal{T}_2$ are decidable, then the positive theory of $\mathcal{T}_1 \cup \mathcal{T}_2$ is decidable.*

To obtain the decidability of the positive theory of $\mathcal{T}_1 \cup \mathcal{T}_2$ with their method Baader and Schulz assume that the *full* positive theories of $\mathcal{T}_1$ and of $\mathcal{T}_2$ are decidable, which seems to entail that our results are more general. However, that is not the case. In fact, by Proposition 3, deciding the validity of positive $\forall\exists\forall\exists$- or $\forall\exists$-sentences in the component theories—which is necessary to use our procedure—is not "easier" than deciding the validity of arbitrary positive sentences—which is necessary to use Baader and Schulz's.

Nevertheless, our approach is better from a computational point of view, because it reduces the non-determinism of the combination procedure. In fact, recall that we can always limit ourselves to those partitions of the shared variables that consist of at most three blocks, whereas Baader and Schulz must in general consider all possible partitions.

3.1 Relation to Decision Procedures for Unification

An interesting by-product of our simplified combination method, pointed out by F. Baader, is a negative answer to an open question first raised in [BS92]: whether there exist equational theories in which unification with constants is decidable but general unification is not.

In logical terms, the decidability of unification with constants in an equational theory E corresponds to the decidability of the positive $\forall\exists$-theory of E. By contrast, the decidability of general unification in E corresponds to the decidability the full positive theory of E. Thanks to Proposition 3, we now know that the two decidability problems are equivalent for any such E.

4 Further Research

As mentioned in the introduction, the combination procedure, and its correctness proof, are extensible to component theories $\mathcal{T}_1$ and $\mathcal{T}_2$ that share constructors in the sense of [BT00, BT01]. In that case, if Σ is the signature shared by the two theories, it is enough to insert an *instantiation step* between the current steps 1 and 2 that arbitrarily replaces some of the variables shared by φ_1 and φ_2 by Σ-terms over fresh variables.

Since the instantiation step is not finitary when Σ contains function symbols of non-zero arity, the extended combination procedure does not lead to decidability results in general—except for limited cases in which the set of Σ-terms is finite modulo equivalence and renaming. However, the work in [DKR94] on the combination of unification algorithms for equational theories sharing constructors suggests that there might be cases in which the instantiation step can actually be executed deterministically.[4] We are currently looking for such cases and investigating the possibility of modifying the combination procedure to accommodate them.

[4] Although this would require the availability of unification algorithms for the theories, not just decidability procedures for their positive fragment.

References

[BS92] Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 50–65. Springer-Verlag, 1992.

[BS95] Franz Baader and Klaus U. Schulz. Combination of constraint solving techniques: An algebraic point of view. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications, RTA'95*, volume 914 of *Lecture Notes in Computer Science*, pages 50–65. Springer-Verlag, 1995.

[BT00] Franz Baader and Cesare Tinelli. Combining equational theories sharing non-collapse-free constructors. In H. Kirchner and Ch. Ringeissen, editors, *Proceedings of the 3rd International Workshop on Frontiers of Combining Systems, FroCoS'2000, Nancy (France)*, volume 1794 of *Lecture Notes in Artificial Intelligence*, pages 260–274. Springer-Verlag, March 2000.

[BT01] Franz Baader and Cesare Tinelli. Deciding the word problem in the union of equational theories. *Information and Computation*, 2001. (to appear).

[DKR94] Eric Domenjoud, Francis Klay, and Christophe Ringeissen. Combination techniques for non-disjoint equational theories. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction, Nancy (France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 267–281. Springer-Verlag, 1994.

[Mah88] M. J. Maher. Complete axiomatizations of finite, rational and infinite trees. In *LICS'88: Proceedings 3rd Symposium on Logic in Computer Science*, pages 348–357, Edinburgh, UK, June 1988.

Solving Matching Problems in Unions of Non-Disjoint Equational Theories

Christophe Ringeissen

LORIA — INRIA
615, rue du Jardin Botanique
BP 101, 54602 Villers-lès-Nancy Cedex France
E-mail: ringeiss@loria.fr Tel: +33 3 83 59 30 53 Fax: +33 3 83 27 83 19

Matching problems appear naturally in the definition of rewriting. Indeed, for performing a basic rewrite step, one has to solve an equation between the left-hand side of a rule (the *pattern*) and a term (*the subject*) in which the instantiation of variables is forbidden. This special case of equation, where one side can be considered as ground, is called match-equation. Many declarative programming languages use rewriting and equational rewriting as operational semantics. These languages enable the programmer to write rule-based specifications/programs involving operators which, for the sake of expressivity, may satisfy some structural axioms like associativity, commutativity, identity, idempotency, etc... Consequently, it is crucial to develop matching algorithms for equational theories. Moreover, since different equational theories must be implemented in the same framework, we face the problem of combining matching for unions of equational theories possibly sharing function symbols.

The combination problem for the union of non-disjoint equational theories has been already studied for an adequate notion of shared free constructors, and a result on the combination of matching algorithms is known [3]. More recently, some new results have been obtained for non-free constructors, in the context of combining decision procedures for the satisfiability problem [7] and for the word-problem [1, 2, 4]. In this talk, we study the possibility of extending results on the combination of matching algorithms [6], in order to allow shared non-free constructors but also shared non-constructor symbols.

The talk will be organized as follows. First, we present a particularly interesting equational theory T_E, which can be seen as a specific non-disjoint union of two given equational theories T and E. For this particular theory, a combined T_E-matching algorithm can be easily derived from matching algorithms known for T and E, whilst T_E-matching with additional free function symbols is as difficult as E-unification. This non-disjoint union T_E has been used to prove the existence of a theory for which matching is decidable whilst matching with additional free symbols is not [5]. Despite this negative result, we want now to generalize the techniques used for this particular case in order to obtain a way to construct unions of theories where matching problems can still be solved using matching algorithms available in component theories. Then, we compare this approach with the results we could expect by reusing, in the context of matching, the notion of non-free constructors introduced initially for the satisfiability problem and for the word-problem. Finally, as a related work, we report the approach investigated in [8] for solving matching problems thanks an extension of the Nelson-Oppen combination procedure for the satisfiability problem in unions of non-disjoint theories.

References

1. Franz Baader and Cesare Tinelli. Deciding the word problem in the union of equational theories sharing constructors. In P. Narendran and M. Rusinowitch, editors, *Proceedings of the 10th International Conference on Rewriting Techniques and Applications (Trento, Italy)*, volume 1631 of *Lecture Notes in Computer Science*, pages 175–189. Springer-Verlag, 1999.
2. Franz Baader and Cesare Tinelli. Combining equational theories sharing non-collapse-free constructors. In H. Kirchner and Ch. Ringeissen, editors, *Proceedings of the 3rd International Workshop on Frontiers of Combining Systems, FroCoS'2000, Nancy (France)*, volume 1794 of *Lecture Notes in Artificial Intelligence*, pages 260–274. Springer-Verlag, March 2000.
3. Eric Domenjoud, Francis Klay, and Christophe Ringeissen. Combination techniques for non-disjoint equational theories. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction, Nancy (France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 267–281. Springer-Verlag, 1994.
4. Camillo Fiorentini and Silvio Ghilardi. Path rewriting and combined word problems. Technical Report 250-00, Department of Computer Science, Università degli Studi di Milano, Milan, Italy, May 2000.
5. Ch. Ringeissen. Matching with Free Function Symbols — A Simple Extension of Matching? In *Rewriting Techniques and Applications*, 2001.
6. Christophe Ringeissen. Combining decision algorithms for matching in the union of disjoint equational theories. *Information and Computation*, 126(2):144–160, May 1996.
7. Cesare Tinelli and Christophe Ringeissen. Non-disjoint unions of theories and combinations of satisfiability procedures: First results. Technical Report UIUCDCS-R-98-2044, Department of Computer Science, University of Illinois at Urbana-Champaign, April 1998. (also available as INRIA research report no. RR-3402).
8. Cesare Tinelli and Christophe Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. Technical report n. 01-02, Department of Computer Science, University of Iowa, April 2001.

Unification in the Empty and Flat Theories with Sequence Variables and Flexible Arity Symbols*

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University Linz
A-4040, Linz, Austria
kutsia@risc.uni-linz.ac.at

Software Competence Center Hagenberg
Hauptstrasse, 99
A-4232, Hagenberg, Austria
teimuraz.kutsia@scch.at

The problem of studying equational theories with sequence variables and flexible arity symbols was posed by Bruno Buchberger and was inspired by a mathematical software system Mathematica [19], the programming language of which is a functional language [5] with an additional feature - sequence variables. In the usual variants of predicate logic sequence variables are not included. However, they are a convenient feature that often is used implicitly in every-day mathematics. In fact, sequence variables together with a natural induction principle for sequence variables (see [6]) expand the power of logic by an implicit formalization of natural numbers.

Sequence variables are used in the Theorema system [7] - an integrated environment for proving, solving and computing, built on the top of Mathematica. The proving context, where they occur in Theorema, makes it important to study unification problem with sequence variables. Sequence variables in the context of term rewriting were studied in [10]. In [8] they are used in inductive theorem proving over lists.

We use the term "flexible arity symbols" for symbols that can have an arbitrary arity. Such symbols occur in programming languages, term rewriting systems, etc. and sometimes are called "variable arity symbols". We build terms over two types of variables - individual and sequence variables, constants and two types of function symbols - of fixed and flexible arity. Sequence variables can be substituted by any (possibly empty) sequence of terms. We do not introduce an additional constructor for sequences in the theory. Presence of fixed arity symbols and sequence variables makes it necessary to solve a system of linear Diophantine equations over naturals to decide whether an expression is a term (or an equation).

We address the problem of solving equations (unification) in a theory with sequence variables and flexible arity symbols. This problem can be seen as a generalization of the problems like word equations [14–16], general A-unification [3],

* Supported by the Austrian Science Foundation (FWF) under Project SFB F1302 and by Software Competence Center Hagenberg (Austria) under MathSoft project.

equations over lists of atoms with concatenation [9] or Mathematica pattern matching.

We consider a general unification problem in empty and flat theories (a theory with a flat flexible arity symbol). A specific feature of the problem is that we require from unifiers to complete arity, which means that after applying the unifier on the terms to be unified no sequence variable occurs as an argument of a term with a fixed arity head. We answer two questions for these theories: a) is it decidable whether a unification problem is solvable and b) how to obtain a minimal unification procedure.

We show decidability of unification in empty and flat theories in two steps. First, with solvability preserving transformations we reduce the problem to a "simpler" unification problem. Second, we apply Baader-Schulz combination method [4] to the reduced problem and show its decidability. In fact, after these two steps the decidability problem for unification in the empty theory is reduced to the decidability problems of a word equation with linear constant restrictions, a word equation with individual variable restrictions on length (demanding that for a variable x in the word equation, which corresponds to an individual variable in the original unification problem, and for a unifier θ of the word equation, the length of $x\theta$ must be 1) and Robinson unification with linear constant restrictions. For the flat theory, in addition, we have decidability of word equations with linear constant restrictions among reduced decidability problems.

Unification procedure both in the empty and flat theories is designed as a tree generation process. We design three kinds of single steps of the process, called projection, arity completion and transformation. The idea of projection (see e.g. [1]) is to substitute some sequence variables with the empty sequence. Arity completion replaces sequence variables, which occur in the unification problem as arguments of terms with fixed arity heads, by sequences of new individual variables such that the obtained expressions are unification problems. Transformation for the empty theory is based on "widening" techniques and generalizes the basic algorithm for word unification [13, 15, 17, 18]. Transformation for the flat theory, in addition, applies special rules based on the "widening" techniques as well, to unify terms with a flat head.

We build the unification tree both for the empty and flat theories in a breadth first (level by level) manner. Each node of the tree is labeled either with a unification problem, $SUCCESS$ or $FAILURE$. Only the nodes labeled with $SUCCESS$ or $FAILURE$ are terminal nodes. The tree is generated as follows: If UP is a unification problem, we label the root of the tree with UP (zero level). First level nodes (the children of the root) of the tree are obtained from UP by projection if UP is solvable, otherwise we stop with failure. Second level nodes are obtained by arity completion, completing arity for each solvable unification problem attached to the first level nodes. Starting from the second level, we apply only a transformation step to a solvable unification problem of each node, thus getting new successor nodes. Note that for each node in the tree, if the unification problem attached to the node is not solvable we replace the label on the node by $FAILURE$. The branch with the leaf labeled by $SUCCESS$ is called a successful branch. For each node in the tree, the edge to a successor of the node is labeled with the substitution which was used to derive the successor. We compose substitutions (top-down) displayed on the edges of the successful branch, restrict the composed substitution on the variables of the unification problem and associate the obtained substitution to the leaf of the branch, which is labeled with $SUCCESS$. If $\Sigma(UP)$ is the set of all substitutions associated with successful leaves of the unification tree for UP, then $\Sigma(UP)$ is the minimal complete set of the unifiers of UP. Tree generation process terminates if the set is finite.

Example 1. Compute the minimal complete set of unifiers in the empty and flat theories ($\overline{x}, \overline{y}, \overline{u}, \overline{v}$ are sequence variables, z is an individual variable, a, b, c are constants, f and g are flexible arity function symbols, h is a flat flexible arity function symbol, s is a binary function symbol):

1. $MCU_{\emptyset}(f(\overline{x}, \overline{y}, z), f(a, b, c)) = \{\{\overline{x} \leftarrow, \ \overline{y} \leftarrow a, b, z \leftarrow c\}, \ \{\overline{x} \leftarrow a, \overline{y} \leftarrow b, z \leftarrow c\}, \ \{\overline{x} \leftarrow a, b, \overline{y} \leftarrow, z \leftarrow c\}\}$.
2. $MCU_{\emptyset}(f(g(a, \overline{x}), \ g(\overline{y}, c)), \ f(\overline{u}, \ g(b, \overline{v}))) = \{\{\overline{u} \leftarrow g(a, \overline{x}), \overline{y} \leftarrow b, \overline{v} \leftarrow c\}, \ \{\overline{x} \leftarrow, \overline{u} \leftarrow g(a), \overline{y} \leftarrow b, \overline{v} \leftarrow c\}, \ \{\overline{u} \leftarrow g(a, \overline{x}), \overline{y} \leftarrow b, \overline{y}, \ \overline{v} \leftarrow \overline{y}, c\}, \ \{\overline{x} \leftarrow, \overline{u} \leftarrow g(a), \overline{y} \leftarrow b, \overline{y}, \ \overline{v} \leftarrow \overline{y}, c\}\}$.
3. $MCU_{\emptyset}(f(s(a, \overline{x}), s(\overline{y}, \ c)), f(\overline{u}, s(b, \overline{v}))) = \{\{\overline{x} \leftarrow z, \ \overline{u} \leftarrow s(a, z), \ \overline{y} \leftarrow b, \ \overline{v} \leftarrow c\}\}$ (here z appears to complete arity of $s(a, \overline{x})$).
4. $MCU_{\emptyset}(f(\overline{x}), f(\overline{y})) = \{\{\overline{x} \leftarrow, \overline{y} \leftarrow\}, \{\overline{x} \leftarrow \overline{y}\}\}$. Note that the substitution $\{\overline{x} \leftarrow \overline{y}\}$ is not more general than $\{\overline{x} \leftarrow, \overline{y} \leftarrow\}$ because any substitution σ, such that $\{\overline{x} \leftarrow \overline{y}\} \circ \sigma = \{\overline{x} \leftarrow, \overline{y} \leftarrow\}$, replaces $\overline{y}$ with the empty sequence, while, by the definition of more general substitutions, it should not replace any sequence variables with the empty sequence.
5. $MCU_{\emptyset}(\overline{x}, \overline{y}) = \{\{\overline{x} \leftarrow \overline{y}\}\}$.
6. $MCU_{Flat}(h(\overline{x}, z), \ h(a, \ b, \ c)) = \{\{\overline{x} \leftarrow, \ z \leftarrow h(a, \ b, \ c)\}, \ \{\overline{x} \leftarrow a, \ z \leftarrow h(b, \ c)\}, \ \{\overline{x} \leftarrow h(a), \ z \leftarrow h(b, \ c)\}, \ \{\overline{x} \leftarrow a, b, \ z \leftarrow c\}, \ \{\overline{x} \leftarrow h(a), \ b, \ z \leftarrow c\}, \ \{\overline{x} \leftarrow a, h(b), \ z \leftarrow c\}, \ \{\overline{x} \leftarrow h(a), h(b), \ z \leftarrow c\}, \ \{\overline{x} \leftarrow h(a, b), \ z \leftarrow c\}, \ \{\overline{x} \leftarrow a, b, \ z \leftarrow h(c)\}, \ \{\overline{x} \leftarrow h(a), b, \ z \leftarrow h(c)\}, \ \{\overline{x} \leftarrow a, h(b), \ z \leftarrow h(c)\}, \ \{\overline{x} \leftarrow h(a), h(b), \ z \leftarrow h(c)\}, \ \{\overline{x} \leftarrow h(a, b), \ z \leftarrow h(c)\}\}$.

We have implemented the unification procedure for the empty theory in Mathematica 4.0. As a decision procedure for being a term as well as on the arity completion phase of the procedure we use the Omega package [2] to solve linear Diophantine equations over naturals. In the tree generation process for the efficiency reasons we did not implement the decision procedure for unifiability but put the bound on the depth of the tree which can be changed by the user - the idea which was used in the system for combining unification algorithms UniMoK [11].

As a future work, we intend to study unification in orderless and flat-orderless theories with sequence variables and flexible arity symbols. Orderless flexible arity symbols are counterparts of commutative fixed arity symbols.

The full version of this paper ([12]) is available from `http://www.sfb013.risc.uni-linz.ac.at/`.

References

1. H. Abdulrab and J.-P. Pécuchet. Solving word equations. *Journal of Symbolic Computation*, 8(5):499–522, 1990.
2. G.E. Andrews, P. Paule, and A. Riese. Macmahon's partition analysis III: the Omega package. SFB-Report 99-24, Johannes Kepler University Linz, 1999.
3. F. Baader and K. Schulz. General A- and AX-Unification via optimized combination procedures. In *Proceedings of the Second International Workshop on Word Equations and Related Topics*, volume 677 of *Lecture Notes in Artificial Intelligence*, pages 23–42. Springer-Verlag, 1993.
4. F. Baader and K. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21(2):211–244, 1996.
5. B. Buchberger. Mathematica as a rewrite language. In *Proceedings of Second Fuji International Workshop on Functional and Logic programming*, pages 1–13. Word Scientific, 1997.
6. B. Buchberger. The concept of sequence variables in logic languages. SFB-report, Johannes Kepler University Linz, April 2001.
7. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. Theorema: A progress report. In M. Kerber and M. Kohlhase, editors, *Proceedings of the 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, pages 100–115, St. Andrews, Scotland, August 6-7 2000.
8. B. Buchberger and D. Vasaru. Theorema: the induction prover over lists. In B. Buchberger, T. Ida, and D. Vasaru, editors, *Proceedings of First International Theorema Workshop*, number 97-20 in RISC Technical Report Series. Research Institute for Symbolic Computation, Johannes Kepler University Linz, 1997.
9. A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–91, 1990.
10. M. Hamana. Term rewriting with sequences. In B. Buchberger, T. Ida, and D. Vasaru, editors, *Proceedings of First International Theorema Workshop*, number 97-20 in RISC Technical Report Series. Research Institute for Symbolic Computation, Johannes Kepler University Linz, 1997.
11. S. Kepser and J. Richts. UniMoK: A system for combining equational unification algorithms. In P. Narendran and M. Rusinowitch, editors, *Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, pages 248–251. Springer-Verlag, 1999.
12. T. Kutsia. Unification in the empty and flat theories with sequence variables and flexible arity symbols. SFB-Report 01-13, Johannes Kepler University Linz, 2001.
13. A. Lentin. Equations in free monoids. In M. Nivat, editor, *Automata, Languages and Programming*, pages 67–85. North-Holland, Amsterdam, 1972.
14. G. Makanin. The problem of solvability of equations on a free semi-group. *Math. USSR Sbornik*, 32(2), 1977.
15. G. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.
16. K. Schulz. Word unification and transformation of generalized equations. *Journal of Automated Reasoning*, 11(2):149–184, 1993.
17. J. Siekmann. String unification. Technical report, Essex University, 1975.
18. J. Siekmann. *Unification and matching problems*. PhD thesis, Essex University, 1978.
19. S. Wolfram. *The Mathematica book*. Cambridge University Press and Wolfram Research, Inc., fourth edition, 1999.

Unification and Projectivity in Propositional Logic

Silvio Ghilardi

Dipartimento di Scienze dell'Informazione
Università degli Studi,
Milano, Italy
ghilardi@dsi.unimi.it

Classical propositional calculus enjoys the following remarkable property (corresponding to the unitarity of E-unification type for Boolean algebras): for every formula A, if there is a substitution σ such that $\sigma(A)$ is provable, then there is also 'the best' substitution with this property, i.e. there is a substitution σ such that $\sigma(A)$ is provable and any τ such that $\tau(A)$ is provable is, up to provable equivalence, an instantiation of σ.

We wonder whether the same property holds for other logical calculi, for instance for some modal or (super)intuitionistic calculi. The answer is negative for all logics L enjoying disjunction property: for instance, in the modal case, the formula $\Box x \vee \Box \neg x$ has unifiers (by this we mean substitutions making it a theorem in L)

$$\sigma_1 : x \mapsto 1 \qquad \sigma_2 : x \mapsto 0$$

(where 1 is 'syntactic truth' and 0 is 'syntactic false') and there is no unifier more general than both of them because if

$$\vdash_L \Box\, \sigma(x) \vee \Box\, \neg\sigma(x)$$

then, by disjunction property, either $\vdash_L \sigma(x)$ (so that σ is equivalent to σ_1) or $\vdash_L \neg\sigma(x)$ (so that σ is equivalent to σ_2). However we prove that unification is rather nice in intuitionistic logic and in many common modal systems like $K4$, $S4$, $S4Grz$, GL, etc. because there are finitely many 'best unifiers' for any formula admitting at least a unifier. In more explicit terms, this means that the unification type of the corresponding classes of (Heyting, modal) algebras is finitary; notice in fact that for these classes of algebras, an arbitrary (elementary) E-unification problem

$$A_1 =_E A_1', \ \ldots, \ A_n =_E A_n'$$

is equivalent to the (matching) problem

$$(A_1 \leftrightarrow A_1') \wedge \cdots \wedge (A_n \leftrightarrow A_n') =_E 1$$

and hence to the problem of making a single formula a theorem in the corresponding logical calculus.

The area of E-unification in propositional logic is nevertheless new, many problems still wait for a solution or for a better solution: these problems include extensions to unification with constants and to general unification, investigation of further natural systems like K, structural information concerning the relationship between unification types and the the lattice of varieties, improvements of the few existing algorithms from the computational point of view, complexity questions, etc.

In my contribution I shall make a survey of my recent work in the field. In the first part, basic definitions are recalled, moreover an equivalent conceptual algebraic approach [1] (quite useful for the determination of unification types) is introduced and many rather easy examples are provided (almost all belonging to propositional logic). This algebraic approach relates unification theory with the standard notion of projective and of finitely presented object in a category. In the second part, the case of Heyting algebras is closely investigated [2]: formulas giving rise to finitely presented projective algebras are characterized both from a syntactic and a semantic point of view, moreover an algorithm [4] (based on a combination of tableaux and resolution) is introduced in order to recognize them. Suitably iterated, the algorithm determines a projective approximation of a given formula, thus providing a complete conformal unification algorithm. The basic point concerning intuitionistic unification is in fact the following: any unifier for a formula A turns out to be a unifier for a projective formula P implying A (and having at most the same implicational degree as A) and is consequently less general than the mgu of P (which trivially is also a unifier for A). Moreover, the comparison order between mgus of projective formulas coincides with IPC-provability among the projective formulas themselves. From this it follows that a complete set of unifiers of unifiers for A is completely determined by (any complete) set of maximal projective formulas implying A (i.e. by what we called a projective approximation of A).

Intuitionistic unification theory has important (and partially unexpected) logical applications to exact formulas and to admissible inference rules. Problems arising from first implementation experiences [5] are also discussed: here ah hoc weak (incomplete) theorem provers seem to be indispensable (in principle, they should be strong enough to reduce redundancies in the output sets but not so strong to cause large inefficiency).

In the final part of my contribution, I show the suitable adaptation of the strategy used in the intuitionistic case to the above mentioned modal systems [3]. Here the proof of the semantic and the syntactic characterization of projective formulas becomes more difficult: bounded bisimulations and ranks in K. Fine's style should be used from the beginning, moreover building mgus of projective formulas seems to require very large iterations of basic kinds of substitutions. However the remaining part of the plan goes as expected, thus yeldying finitarity results for unification types in this case too.

References

[1] S. Ghilardi *Unification through projectivity*, Journal of Logic and Computation, 7, 6, pp.733-752 (1997).

[2] S. Ghilardi *Unification in intuitionistic logic*, Journal of Symbolic Logic, vol.64, n.2, pp.859-880 (1999).

[3] S. Ghilardi *Best soving modal equations*, Annals of Pure and Applied Logic, 102, pp.183-198 (2000).

[4] S. Ghilardi *A resolution/tableaux algorithm for projective approximations in IPC*, Journal of Logic and Computation, to appear.

[5] B. Bertacco *Unificazione e Proiettività nella Logica Intuizionista Proposizionale: Realizzazione e Studio di un Algoritmo*, Tesi di Laurea in Informatica, Università degli Studi di Milano, a.a. 2000/2001.

Solving Linear Equations over Regular Languages

Franz Baader[1] and Ralf Küsters[2]

[1] LuFG Theoretische Informatik, RWTH Aachen, Germany,
baader@informatik.rwth-aachen.de
[2] Institut für Informatik und Praktische Mathematik, CAU Kiel, Germany,
kuesters@ti.informatik.uni-kiel.de

1 Introduction

Unification of concept descriptions has been introduced by Baader and Narendran [5] as a new inference service for detecting and avoiding redundancies in Description Logic knowledge bases. The technical results in [5] are concerned with unification in the small Description Logic $\mathcal{FL}_0$, which allows for conjunction of concepts, value restriction, and the top concept. It is shown that unification of $\mathcal{FL}_0$-concept descriptions is equivalent to unification modulo the equational theory ACUIh, which is concerned with a binary associative-commutative-idempotent function symbol with a unit and finitely many unary function symbols that behave like endomorphisms for the binary symbol and the unit. Since ACUIh is a commutative/monoidal theory [6], unification modulo this theory can in turn be reduced to solving linear equations in a certain semiring. For the theory ACUIh, this semiring consists of finite sets of words over a finite alphabet Σ, with union as addition and concatenation as multiplication. Thus, the linear equations to be solved are of the following form:

$$S_0 \cup X_1 S_1 \cup \cdots \cup X_n S_n = T_0 \cup X_1 T_1 \cup \cdots \cup X_n T_n, \tag{1}$$

where the coefficients S_i, T_i are finite sets of words over Σ and the X_i are variables that stand for finite sets of words. In [5] it was shown that the problem of determining whether such an equation has a solution or not is ExpTime-complete.

Closely related to the problem of solving such linear language equations is the problem of solving positive set constraints [1] over unary and nullary function symbols where only union of sets is allowed. For solvability of positive set constraints over (at least two) unary and (at least one) nullary function symbols, ExpTime-completeness is shown in [1]. The main difference between this result and the one in [5] is that solutions of set constraints may be infinite, i.e., in the setting of linear language equations, the variables X_i may be replaced by (possibly) *infinite* sets of words.

In the present paper, we study the problem of solving equations of the form (1) over regular languages, i.e., both the coefficients S_i, T_i and the languages substituted for the variables X_i may be regular languages over Σ. Equations over regular languages have already been considered by Leiss [7,8]. However, he does not provide any decidability or complexity results for the case we are interested in. From the Description Logic point of view, the problem of solving such equations corresponds to unification in the Description Logic $\mathcal{FL}_{reg}$ [2], which extends $\mathcal{FL}_0$ by composition, union, and transitive closure of roles.

Our main contribution is that the problem is also ExpTime-complete. This result is obtained by reduction to/from decision problems for tree automata. Whereas for equations over finite languages automata on finite trees could be used, we now consider automata working on infinite trees. As a by-product of the reduction to tree automata, we also show that, if a system of linear equations has some (possibly irregular) solution, then it also has a regular one. Using this result, our exponential upper bound also follows from the complexity result in [1], if one takes into account that regular coefficients can be expressed using set constraints. The hardness result in [1] does not directly carry over since set constraints allow for more complex types of equations than the linear ones considered here.

2 Solving regular language equations

Let Σ be a finite alphabet. A *linear equation over regular languages* is an equation of the form (1) such that the coefficients S_i, T_i are regular languages over Σ. A *solution* θ of this equation is a substitution assigning to each variable X_i a language $L_i \subseteq \Sigma^*$ such that the equation holds, i.e., $S_0 \cup L_1S_1 \cup \cdots \cup L_nS_n = T_0 \cup L_1T_1 \cup \cdots \cup L_nT_n$ where for languages L, S the expression LS denotes their concatenation $\{vw \mid v \in L, w \in S\}$. This solution is called *regular* (*finite*) iff the languages L_i are regular (finite).

For example, let $\Sigma := \{a\}$ and consider the linear equation

$$\{a\} \cup X\{a\} = X.$$

The substitution that replaces X by $\{a\}\{a\}^*$ is a regular solution of this equation. Note that the coefficients are all finite, but that the equation does not have a finite solution.

Instead of only single equations, one can also look at *systems of regular language equations*, i.e., finite sets of equations. A substitution θ solves a system of regular language equations if it solves every equation in it simultaneously. However, a system of equations can easily (in linear time) be turned into a single equation with the same set of solutions by concatenating all constant languages in an equation with a symbol a (a new symbol for every equation), i.e., the languages S_i and T_i are replaced by $S_i\{a\}$ and $T_i\{a\}$. Then the different equations can be put together into a single equation without causing any interference (see [5] for details). Hence, for our complexity analysis we can focus on single equations.

The main theorem of this paper gives the exact complexity of solving a linear equation over regular languages.

Theorem 1. *Deciding whether a linear equation over regular languages has a (regular) solution is an ExpTime-complete problem.*

As an immediate consequence of the result for regular solvability, unification in $\mathcal{FL}_{reg}$ is ExpTime-complete as well.

In principle, to solve (1), we build a Büchi tree-automaton that accepts the trees representing i) sets of words obtained by instantiating the equation with its solutions (called *solution sets* in the following), and ii) the solution itself, i.e., the languages substituted for the variables.

The trees the Büchi-automaton is working on are total mappings from the set Σ^* into $\{0,1\}^{n+1}$. That is, a tree t is a $\{0,1\}^{n+1}$-labeled $|\Sigma|$-ary infinite tree. For $v \in \Sigma^*$, $t(v)_i$, $i = 0, \ldots, n$, denotes the ith component of $t(v)$.

Every path from the root of t to some node $v \in \Sigma^*$ corresponds to a (finite) word over Σ, namely v. The label $t(v) = b_0 b_1 \cdots b_n \in \{0,1\}^{n+1}$ of v should be read as follows: v belongs to the solution set of (1) iff $b_0 = 1$; and (the language substituted for) X_i, $i = 1, \ldots, n$, contains v iff $b_i = 1$. Thus, such a tree completely determines the solution set as well as the substitutions for the variables, provided that the substitution really solves the equation.

For a given tree t, we need to check whether the encoded solution set really corresponds to the result of inserting the encoded substitution into the right- and the left-hand side of equation (1). This can be done using a Büchi-automaton that checks whether the following conditions are equivalent for every $v \in \Sigma^*$:

1. $t(v)_0 = 1$.
2. $v \in S_0$, or there exists $i = 1, \ldots, n$ and w, w' such that $v = ww'$, $t(w)_i = 1$, and $w' \in S_i$.
3. $v \in T_0$, or there exists $i = 1, \ldots, n$ and w, w' such that $v = ww'$, $t(w)_i = 1$, and $w' \in T_i$.

It is fairly easy to construct a Büchi tree-automaton accepting exactly those trees satisfying the above equivalence. The size of the set of states of this automaton turns out to be exponential in the size of the equation, where the size of the regular sets S_i and T_i are measured by the size of non-deterministic finite automata accepting these sets. Moreover, all states of the Büchi-automaton are final. Thus, we are using a restricted form of Büchi-automata (sometimes called *looping tree-automata* in the literature [11]). Since the emptiness problem for Büchi tree-automata can be solved in polynomial time in the size of the automaton [10] (and actually in linear time for looping automata), this yields an exponential time algorithm deciding whether an equation of the form (1) has a solution. However, this does not a priori mean that there is also a regular one.

It is well known [10] that the set of trees accepted by a Büchi-automaton contains a regular (or rational) tree t if it is nonempty. In our setting this implies that the language $\{v \mid t(v) = b_0 \cdots b_n\}$ for some fixed label $b_0 \cdots b_n \in \{0,1\}^{n+1}$ is regular. In particular, the finite union

$$\bigcup_{b_0 b_1 \cdots b_n \in \{0,1\}^{n+1}, b_i = 1} \{v \mid t(v) = b_0 \cdots b_n\}$$

is regular. This set describes the language substituted for X_i. This shows the following proposition.

Proposition 1. *If a linear equation over regular languages has a solution, then it also has a regular one.*

As a direct consequence of this proposition we obtain the exponential upper bound claimed in Theorem 1 also for the regular case.

The hardness result can be shown similarly to the proof by Baader and Narendran [5] for systems of equations over finite languages. In their proof, the intersection problem of deterministic root-to-frontier automata on finite trees, which has been shown to be ExpTime-complete by Seidl [9], is reduced to the solvability of systems of equations over finite languages. One can use Seidl's result to show that the intersection problem for deterministic looping tree-automata is ExpTime-complete as well. A reduction of this problem to solvability of systems of linear equations over regular languages then establishes the exponential lower bound.

3 Conclusion and open problems

We have shown that deciding the solvability of (systems of) linear equations over regular languages, a problem equivalent (via linear-time reduction) to unification in the Description Logic $\mathcal{FL}_{reg}$, is ExpTime-complete.

In case an equation/unification problem is solvable, one is usually interested in obtaining an actual solution. It is easy to see that solutions can be derived from accepting runs of our Büchi-automata. In addition, if there is such a run, it is also possible to compute enough information about it to read off the solution it describes. However, since an equation may have several (even infinitely many) solutions, one must ask which solutions should be computed.

In the context of matching in Description Logics [3,4], it has been argued that least solutions, i.e., solutions where all variables are substituted by concept descriptions that are as specific as possible, are of interest. For our language equations this would mean that variables are replaced by the greatest (w.r.t. set inclusion) (regular) languages. Thus, the following questions arise. Do solvable linear equations over regular languages always have such greatest solutions? Are these solutions regular? (Only in this case can they be translated into $\mathcal{FL}_{reg}$-concept descriptions.) How hard is it to compute greatest solutions?

Another open problem is the complexity of matching in $\mathcal{FL}_{reg}$. This corresponds to solving linear equations over regular languages where the right-hand side does not contain variables (i.e., $T_1 = \ldots = T_n = \emptyset$). For linear equations over finite languages matching is polynomial. In the regular case, it is easy to see that the problem is at least PSpace-hard, but it is not clear whether it can be solved in PSpace.

Finally, for Description Logics one is also interested in unification in more expressive logics. On the language equation side, this could lead to other (probably more involved) kinds of equations, if the language-based approach is applicable at all. For matching, the language-based approach can also be applied to more expressive Description Logics (allowing for atomic negation and number restrictions) [4], but for some Description Logics (e.g., ones allowing for existential restrictions) other approaches are more appropriate [3]. For unification, nothing is known for these more expressive languages.

References

1. A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The Complexity of Set Constraints. In *Proc.CSL'93*, LNCS 832, pages 1–17, 1993.
2. F. Baader. Augmenting concept languages by transitive closure of roles: an alternative to terminological cycles. In *Proc. IJCAI-91*, pages 446–451, 1991.
3. F. Baader and R. Küsters. Matching in description logics with existential restrictions. In *Proc. KR2000*, pages 261–272, Breckenridge, CO, 2000.
4. F. Baader, R. Küsters, A. Borgida, and D. McGuinness. Matching in Description Logics. *Journal of Logic and Computation*, 9(3):411–447, 1999.
5. F. Baader and P. Narendran. Unification of concept terms in description logics. In *Proc. ECAI-98*, pages 331–335, 1998. An extended version has appeared in J. Symbolic Computation 31:277–305, 2001.
6. F. Baader and W. Nutt. Combination problems for commutative/monoidal theories: How algebra can help in equational reasoning. *J. Applicable Algebra in Engineering, Communication and Computing*, 7(4):309–337, 1996.
7. E. Leiss. Implicit language equations: Existence and uniqueness of solutions. *Theoretical Computer Science A*, 145:71–93, 1995.
8. E. Leiss. *Language Equations*. Springer-Verlag, 1999.
9. H. Seidl. Haskell overloading is DExpTime-complete. *Information Processing Letters*, 52(2), 1994.
10. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, Amsterdam, 1990.
11. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.

Solving Knuth-Bendix Ordering Constraints

Konstantin Korovin and Andrei Voronkov

Department of Computer Science
University of Manchester
{korovin|voronkov}@cs.man.ac.uk

We consider term algebras with simplification orders (which are monotonic, and well-founded). Solving constraints (quantifier-free formulas) in term algebras with this kind of orders has several important applications like pruning search space in automated deduction and proving termination of term rewriting systems. Two kinds of orders are normally used in automated deduction: the Knuth-Bendix order and various versions of recursive path orders. The Knuth-Bendix order is used in the state-of-the-art theorem provers, for example, E [12], SPASS [13], Vampire [11], and Waldmeister [3]. There is extensive literature on solving recursive path ordering constraints (e.g., [1,4,9,10]). The decidability of Knuth-Bendix ordering constraints was proved only recently in [6]. The algorithm described in that paper shows that the problem belongs to 2-NEXPTIME. It was also shown that the problem is NP-hard by reduction of the solvability of systems of linear Diophantine equations to the solvability of the Knuth-Bendix ordering constraints.

In our recent work we show that the existential theory of any term algebra with the Knuth-Bendix order is NP-complete [7]. Another question we consider is solvability of constraints consisting of a single inequality. For the lexicographic path orders this problem was shown to be NP-complete [2]. We show that solvability of the Knuth-Bendix ordering constraints consisting of a single inequality can be done in polynomial time. This algorithm can be consider as a part of a more complex algorithm for so called orientability problem: given a system of term rewrite rules decide whether there exists a Knuth-Bendix order which orients this system. We present a polynomial algorithm for the orientability problem for the Knuth-Bendix order [8].

Let us formulate our results more formally. We consider term algebras in a finite signature Σ with at least one constant, denoted $\mathrm{TA}(\Sigma)$. Let us now define Knuth-Bendix order on $\mathrm{TA}(\Sigma)$ [5]. The definition of Knuth-Bendix order is parametrized by a *weight function* on Σ, i.e., a function $w : \Sigma \to \mathbb{N}$, and a linear order $\gg$ on Σ. We require from the weight function the following: if $w(f) = 0$ and f is unary, then f must be the greatest w.r.t. $\gg$ in Σ, and weights of constants are positive. We define the weight of a ground term as a sum of weights of functors occurring in the term. Given a weight function w and a linear order $\gg$ on Σ, the *Knuth-Bendix order* on $\mathrm{TA}(\Sigma)$ is a binary relation $\succ$ defined as follows. For any ground terms $t = g(t_1, \ldots, t_n)$ and $s = h(s_1, \ldots, s_k)$ we have $t \succ s$ if one of the following conditions holds:

1. $w(t) > w(s)$;

2. $w(t) = w(s)$ and $g \gg h$;
3. $w(t) = w(s)$, $g = h$ and for some $1 \leq i \leq n$ we have $t_1 = s_1, \ldots, t_{i-1} = s_{i-1}$ and $t_i \succ s_i$.

Theorem 1. *The existential first-order theory of any term algebra with the Knuth-Bendix order is NP-complete.*

Let us sketch some steps of our proof, for full details see [7]. To prove the theorem we first extend our term algebra with the natural numbers with addition and the weight function on terms. We show how to transform arbitrary quantifier-free formula into an equivalent disjunction of conjunctions such that all occurring terms in the formula are variables. Then we introduce formulas which express that there exists at least n terms of the weight x, where n is a fixed parameter and show how to write them as linear Diophantine equations on weights of terms. Using obtained formulas we transform initial formula into an equivalent constraint which consists of linear Diophantine equations on weights of terms. Then we show that satisfiability of that constraint is equivalent to satisfiability of the systems of linear Diophantine equations over the natural numbers.

Our next theorem is about solving constraints consisting of a single inequality.

Theorem 2. *There exists a polynomial time algorithm for solving Knuth-Bendix ordering constraints consisting of a single inequality.*

For a proof see [8]. Also in that paper was presented a polynomial time algorithm for the orientability problem.

Theorem 3. *The problem of the existence of a Knuth-Bendix order which orients a given rewrite rule systems can be solved in polynomial time.*

References

1. H. Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.
2. H. Comon and R. Treinen. Ordering constraints on trees. In S. Tison, editor, *Trees in Algebra and Programming: CAAP'94*, volume 787 of *Lecture Notes in Computer Science*, pages 1–14. Springer Verlag, 1994.
3. Th. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. Waldmeister: High-performance equational deduction. *Journal of Automated Reasoning*, 18(2):265–270, 1997.
4. J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. In J.L. Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium, ICALP'91*, volume 510 of *Lecture Notes in Computer Science*, pages 455–468, Madrid, Spain, 1991. Springer Verlag.
5. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

6. K. Korovin and A. Voronkov. A decision procedure for the existential theory of term algebras with the Knuth-Bendix ordering. In *Proc. 15th Annual IEEE Symp. on Logic in Computer Science*, pages 291–302, Santa Barbara, California, June 2000.
7. K. Korovin and A. Voronkov. Knuth-Bendix constraint solving is NP-complete. Preprint CSPP-8, Department of Computer Science, University of Manchester, February 2000. to appear in ICALP'2001.
8. K. Korovin and A. Voronkov. Verifying orientability of rewrite rules using the Knuth-Bendix order. Preprint CSPP-11, Department of Computer Science, University of Manchester, March 2001. to appear in RTA'2001.
9. P. Narendran, M. Rusinowitch, and R. Verma. RPO constraint solving is in NP. In G. Gottlob, E. Grandjean, and K. Seyr, editors, *Computer Science Logic, 12th International Workshop, CSL'98*, volume 1584 of *Lecture Notes in Computer Science*, pages 385–398. Springer Verlag, 1999.
10. R. Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47:65–69, 1993.
11. A. Riazanov and A. Voronkov. Vampire. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 292–296, Trento, Italy, July 1999.
12. S. Schulz. System abstract: E 0.3. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence, pages 297–301, Trento, Italy, July 1999.
13. C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Topic. System description: SPASS version 1.0.0. In H. Ganzinger, editor, *Automated Deduction—CADE-16. 16th International Conference on Automated Deduction*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 378–382, Trento, Italy, July 1999.

Context Congruence Closure and Context Rewriting for Commutative Context-Free Grammars

Siva Anantharaman

LIFO, Université d'Orléans
F-45067 Orléans Cedex 02, France
siva@lifo.univ-orleans.fr

1 Introduction

Standard (or usual) context-free grammars are well-known for programmers. The concept of a *commutative* grammar arises when analyzing concurrent processes via Petri net models. In particular, for processes without synchronization the Petri net model can be assumed "communication-free" and with simple pre-conditions (cf. [1]); the transition rules are then of the form $X \to {Y_1}^{m_1} \dots {Y_p}^{m_p}$ where the X, Y_j are places in the net and the m_j are positive integers; these transition rules define the productions of a context-free commutative grammar. Over such grammars, the uniform word problem - whether or not any given commutative word w is accepted - turns out to be NP-complete (cf. [1,2]). Indeed, in the commutative case, if $\alpha_1\alpha_2$ derives into β, we cannot always expect to cut β into a *left*-part β_1 and a *right*-part β_2 such that α_1 (resp. α_2) derives into β_1 (resp. β_2); so a CYK-like approach needs often to backtrack. We propose here to circumvent this partially, via a rewrite approach on commutative strings with context, mixing top-down reductions with bottom-up completion steps, all with a goal-oriented strategy.

As in the standard case, we first transform our context-free commutative grammar G_0 into a Chomsky normal form G with some additional restrictions. This will allow us to view any given (commutative) word w over the terminal alphabet of G as a word over a fixed set $\mathbf{T}(G)$ of non-terminals of G, in a unique manner. We derive from G a set $\mathcal{R}_1$ of commutative reduction rules of the form $Y[\mathcal{C}]\,Z[\mathcal{D}] \Rightarrow X[\mathcal{C}\mathcal{D}]$ where X, Y, Z are non-terminals in G, and $\mathcal{C}, \mathcal{D}$ are context-variables to be instantiated to sub-words of w for the $\mathcal{R}_1$-rewrite steps. Most of the rules of $\mathcal{R}_1$ are essentially reversed productions of G, but some more are needed to take into account recursion. We also derive from G a set of context-congruence pairs, which store information on the basic divergence points of derivations. Under a (well-founded) rewrite ordering on commutative strings defined over a total precedence relation on the non-terminals of G, and compatible with the system $\mathcal{R}_1$, the pairs of $\mathcal{R}_2$ can be turned into congruence rewrite rules. In our analysis of any given commutative word w, the rules of $\mathcal{R}_2$ compensate for the absence of confluence of the rewrite system $\mathcal{R}_1$; to help them do that well enough, we will in general need to "saturate" $\mathcal{R}_2$, relative to w.

Given the word w to analyze, we first perform $\mathcal{R}_1$-reductions on w; this will be part of the *top-down* component of our approach. If this is inconclusive, we proceed with inferences of congruence closure or saturation on $\mathcal{R}_2$ *à la KB*, so as to render further top-down inferences possible; such saturation steps constitute the *bottom-up* component of the approach; saturation can be "lazy". The approach in its entirety can be formulated in terms a non-deterministic automaton $\mathcal{A} = \mathcal{A}(G, w)$. Let w be seen as the commutative word $w = {X_1}^{n_1} \ldots {X_r}^{n_r}$ over $\mathbf{T}(G)$. The states of the automaton are (as a first approximation) commutative monomials over the non-terminals of G each labeled with a sub-word of w (in short: **labeled words** over w). The initial state of $\mathcal{A}$ will be the monomial $X_1[-]^{n_1} \ldots X_r[-]^{n_r}$, i.e., the word w itself whose non-terminals are labeled with the empty context; the acceptance state is the monomial $A[{X_1}^{n_1} \ldots {X_r}^{n_r}]$, where A is the axiom of G. The transitions of $\mathcal{A}$ are 'top-down' if they correspond to $\mathcal{R}_1$-reductions or to $\mathcal{R}_2$-rewrite steps; or 'bottom-up' if they are saturation steps on $\mathcal{R}_2$. In more formal terms, the states of our automaton will be couples formed by labeled words over w, and "current" sets of rules $\mathcal{R}_2'$ obtained via lazy saturation of $\mathcal{R}_2$. The saturation steps do part of what a *CYK*-like approach would have tried with backtrackings; but they will do that in a very limited manner and more intelligently, with a rewrite-based and goal-oriented strategy.

2 The Approach: Basic Definitions, Examples

Notational convention: since we will also be manipulating rewrite rules, we will employ the symbol "$\longrightarrow\!\!\!\!\!+$" for grammatical productions, reserve the symbols "$\Rightarrow$" and "$\rightarrow$" for rewrite rules, and the corresponding long symbols "$\Longrightarrow$, $\longrightarrow$" for the respective rewrite steps using these rules. Composite reductions via a "combination" of these two sets of rules will be denoted "$\succ\!\!\!\longrightarrow$". **cfcg** will be short for "context-free commutative grammar".

We look at a **cfcg** G essentially as a set of string rewrite rules on labeled words (the set of reversed productions of G, plus a few more), and a set of "congruence rules": (oriented) pairs of labeled words defining the basic bridges between different derivations. We do not want too many congruences, nor the contexts to be too big; and we want to express any (commutative) word over the terminal alphabet of G uniquely over a fixed set of non-terminals of G.

Definition 1 *A* **cfcg** $G = (\Sigma, N, A, \mathcal{P})$ *is said to be in* restricted Chomsky normal form *(RCN-*form *for short) iff the following conditions are satisfied:*

i) All productions in G *are either* standard, *i.e. of the form* $X \longrightarrow\!\!\!\!\!+\ YZ$ *for non-terminals* X, Y, Z*; or* unit, *i.e. of the form* $X \longrightarrow\!\!\!\!\!+\ V$*, for non-terminals* X, V*; or* scalar, *i.e. of the form* $V \longrightarrow\!\!\!\!\!+\ v$*, with* $v \in \Sigma$*.* (We'll say in these last two cases: X produces a unit, resp. V produces a scalar.)

ii) if $V \longrightarrow\!\!\!\!\!+\ v$ *is a scalar production in* G *with* $v \in \Sigma$*, then there is no* **other** *scalar production in* G *with* v *on the right-hand side.*

iii) the unit productions of G appear only 'coupled' with recursive productions, that is to say under the form $X \twoheadrightarrow UX \mid V \mid \dots$, *where the non-terminal* V *must produce a scalar.* []

The terminal words on such grammars are generated symbol by symbol: productions of the form $V \twoheadrightarrow ab$ are not allowed in our RCN-forms.

Example 1 If G_0 is given (in usual Chomsky normal form) as $A \twoheadrightarrow PA \mid QB \mid c$, $B \longrightarrow AD \mid QB \mid c$, $D \longrightarrow PB \mid c$, $P \longrightarrow a$, $Q \longrightarrow a \mid b$, then an equivalent RCN-form is the **cfcg** G: $P \longrightarrow a$, $Q \longrightarrow b$, $R \longrightarrow c$, $A \twoheadrightarrow PA \mid QB \mid R$, $B \twoheadrightarrow AD \mid AR \mid QB \mid PB \mid R$, $D \twoheadrightarrow PB$ □

Contexts are commutative words over the non-terminals in G, inside square brackets. A *labeled non-terminal* is a non-terminal with an attached context; notation: $N[\mathcal{C}]$; we say that the context is *"empty"*, when it is the non-terminal itself; we write then more simply N, or $N[-]$. Monomials over the labeled non-terminals will be referred to as *labeled words*.

Definition 2 *Let* G *be a given* **cfcg** *in* RCN*-form.*

i) We associate with G *a commutative string rewrite system, denoted* $\mathcal{R}_1(G)$, *whose elements are obtained as follows:*

- if $X \twoheadrightarrow YZ$ (resp. $X \twoheadrightarrow Z$) is a standard (resp. unit) production in G, then $Y[\mathcal{C}]\,Z[\mathcal{D}] \Rightarrow X[\mathcal{C}\mathcal{D}]$ (resp. $Z[\mathcal{C}] \Rightarrow X[\mathcal{C}]$), where $\mathcal{C}, \mathcal{D}$ are context-variables, is a rewrite rule in $\mathcal{R}_1(G)$;
- if $Y \twoheadrightarrow YV$ is a recursive production in G, then for any non-terminals X, Z of G, context-variables $\mathcal{C}, \mathcal{D}$, the rewrite rules $X[\mathcal{C}\,Y]\,Z[YV] \Rightarrow X[\mathcal{C}\,Y^2V]$, as well as $X[\mathcal{C}\,Y]\,V[\mathcal{D}] \Rightarrow X[\mathcal{C}\,Y\mathcal{D}]$ are elements of $\mathcal{R}_1(G)$;
- if $X \twoheadrightarrow UX \mid Z \mid \ldots$, are productions in G with $Z \in \mathbf{T}(G)$, i.e. produces a scalar, then for any context variable $\mathcal{C}$, $U[\mathcal{C}]\,Z \Rightarrow X[\mathcal{C}Z]$ is in $\mathcal{R}_1(G)$.

ii) Also associated with G *is a set of context-congruence pairs, denoted* $\mathcal{R}_2(G)$, *whose elements are obtained as follows:*

- if $X \twoheadrightarrow PQ$ and $Y \twoheadrightarrow PQ$ are two standard productions of G, then $X[PQ] \equiv Y[PQ]$ is an element of $\mathcal{R}_2(G)$;
- if $X \twoheadrightarrow PQ$ and $Y \twoheadrightarrow PR$ are two standard productions of G, then $X[PQ]\,R \equiv Y[PR]\,Q$ is an element of $\mathcal{R}_2(G)$;
- if $X \twoheadrightarrow Z$, $Y \twoheadrightarrow Z$ are unit productions in G with $Z \in \mathbf{T}(G)$, i.e. Z produces a scalar, then $X[Z] \equiv Y[Z]$ is in $\mathcal{R}_2(G)$;
- if $X \twoheadrightarrow UX \mid Z \mid ..$, and $Y \twoheadrightarrow VY \mid Z \mid ..$, are productions in G with Z producing a scalar, then $X[UZ] \equiv Y[UZ]$ is in $\mathcal{R}_2(G)$. []

We turn the congruence pairs into rewrite rules under a rewrite ordering $\prec$ (compatible with $\mathcal{R}_1$), via a suitable total precedence $<$ over the non-terminals of G, by setting: $w_1 \prec w_2$ iff **either** $|w_1| < |w_2|$, **or** $|w_1| = |w_2|$ and $w_1 \ll w_2$ as a multiset. A *labeled word* is *over* w if its contexts are sub-words of w. The set of labeled words over w gets ordered under $\prec$, by ignoring the contexts attached to the non-terminals in the words.

A labeled word is *w-relevant* if its *non-empty* contexts concatenated together give a sub-word of w up to commutativity. A congruence pair (rule) of $\mathcal{R}_2(G)$ is said to be *w-relevant* iff its left-hand side (right-hand side) is w-relevant.

Example 2 Let G be the following **cfcg** in RCN-form:

$$A \twoheadrightarrow PY \mid QB, \quad B \twoheadrightarrow XR \mid WR, \quad X \twoheadrightarrow PQ, \quad Y \twoheadrightarrow QR, \quad W \twoheadrightarrow WW$$

$$P \twoheadrightarrow a, \quad Q \twoheadrightarrow b, \quad R \twoheadrightarrow c, \quad W \twoheadrightarrow d$$

and suppose we are given three words to analyze: $w_1 = abc$, $w_2 = bd^3c$, $w_3 = cd^2$. Assume that our rewrite ordering $\prec$ is defined over the total precedence:

$$A < B < X < Y < P < Q < R < W$$

The word w_1 is transformed uniquely into PQR, w_2 into QW^3R, and w_3 into RW^2; the following congruence rules in $\mathcal{R}_2$:

$$X[PQ]\,R \longrightarrow Y[QR]\,P, \quad W[WW]\,R \longrightarrow B[WR]\,W, \quad Y[QR]\,W \longrightarrow B[WR]\,Q$$

will be relevant to their analysis.

For the analysis of $w_1 = PQR$: from the starting state PQR on $\mathcal{A}$, suppose we have gone along the following transitions via $\mathcal{R}_1$-reductions:

$$PQR \implies X[PQ]\,R \implies B[PQR]$$

The last state is not of acceptance. However, the first of the congruence rules mentioned above is right simplifiable by $\mathcal{R}_1$ into the rule $X[PQ]\,R \longrightarrow A[PQR]$, then left simplifiable into the rule $B[PQR] \longrightarrow A[PQR]$. (This is a *saturation* step, cf. below.) This last congruence rule defines an $\mathcal{R}_2$-reduction from the state reached above, to the acceptance state $A[PQR]$; so w_1 is accepted by G.

For the analysis of $w_2 = QW^3R$: consider first the situation where we have gone along the chain of $\mathcal{R}_1$-transitions

$$QW^3R \implies QW^2\,B[WR] \implies A[QWR]W[WW];$$

A rewrite rule in $\mathcal{R}_1$, defined via recursion on W, leads us subsequently to the state $A[QW^3R]$ of acceptance. Consider next the situation where we apply the following $\mathcal{R}_1$-transitions: $QW^3R \implies W^3\,Y[QR]$. Here an $\mathcal{R}_2$-rewrite step via the congruence rule $Y[QR]\,W \longrightarrow B[WR]\,Q$ can be applied, to lead us to the state $W^2Q\,B[WR]$; we are back to case considered first.

Finally, the word $w_3 = RW^2$: we have the following two possible chains of $\mathcal{R}_1$-transitions, both leading to the same state:

$$RW^2 \implies B[RW]W[-] \implies B[RW^2], \quad \text{or}$$
$$RW^2 \implies W[WW]\,R \implies B[RW^2]$$

From the last state reached no more transitions are possible under $\mathcal{R}_1$, or under $\mathcal{R}_2$ even after saturation; so the word w_3 is rejected. □

If N is a non-terminal in w, then an N*-translate* is defined for a w-relevant congruence rule $\mathcal{W}_1 \longrightarrow \mathcal{W}_2$ iff : i) at least one of the two labeled words $N\mathcal{W}_i, i = 1, 2$, is *reducible under* $\mathcal{R}_1$, and ii) if $\mathcal{U}_i$ are $\mathcal{R}_1$-normal forms of $N\mathcal{W}_i, i = 1, 2$, then the congruence pair $\mathcal{U}_1 \equiv \mathcal{U}_2$ obtained after $\mathcal{R}_1$-normalization is also w-relevant. By definition, a *saturation*, or a *congruence closure* step on $\mathcal{R}_2(G)$ w.r.t. w, is the process of augmenting the set $\mathcal{R}_2(G)$ with such w*-relevant translates*, or via right-simplification by rules in $\mathcal{R}_1$, or via superposition with rules of $\mathcal{R}_1$; such a step is denoted by: $\mathcal{R}_2(G) \vdash \mathcal{R}_2'(G)$.

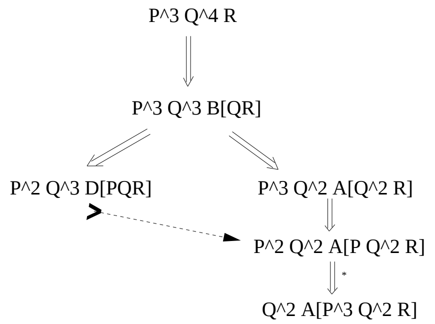
Let $\mathcal{C}$ be a labeled word, and $\mathcal{D}$ labeled word over w. Then $\mathcal{D}$ is a (w, G)*-instance* of $\mathcal{C}$ iff $\mathcal{D}$ can be reduced to $\mathcal{C}$ by using the productions of G in the reverse direction.

Example 3 *(Example* 1 - *contd.)* Let G be the following **cfcg**:

$$A \twoheadrightarrow PA \mid R \mid QB, \quad B \twoheadrightarrow AD \mid AR \mid QB \mid PB \mid R,$$

$$D \twoheadrightarrow PB, \quad P \twoheadrightarrow a, \quad Q \twoheadrightarrow b, \quad R \twoheadrightarrow c$$

Suppose $\prec$ is defined over the total precedence $A < B < D < P < Q < R$, and $w = a^3b^4c$ is the word to analyze; w is represented over $T(w)$ by P^3Q^4R, which admits more than one reduction path w.r.t. $\mathcal{R}_1$; for instance, we have:



Both branches seem to lead to dead-ends. Let us first look at the final state reached (via $\mathcal{R}_1$) along the right branch. We have by definition, a congruence rule $B[QR] \longrightarrow A[QR]$ in $\mathcal{R}_2(w)$; from which we derive, via P-translations, the rules $B[PQR] \longrightarrow A[PQR]$, $B[P^2QR] \longrightarrow A[P^2QR]$, .. etc. And then, via Q-translations, the rules $QA[PQR] \longrightarrow A[PQ^2R]$, $Q^2A[PQR] \longrightarrow A[PQ^3R]$, .. etc.; these w-relevant translations suffice to lead from the state mentioned, to the acceptance state $A[P^3Q^4R]$ (the word w is therefore accepted by G).

From the final state reached (via $\mathcal{R}_1$) along the left branch no such translations seem available. However, we have the following congruence rule in $\mathcal{R}_2(G)$: $D[PB]Q \longrightarrow A[QB]P$. It suffices then to see that PQR is a (w, G)-instance of PB in the sense defined above; so we derive the congruence rule $D[PQR]Q \longrightarrow A[Q^2R]P$, which defines a "composite" reduction step: $P^2Q^3D[PQR] \rightarrowtail P^2Q^2A[PQ^2R]$, from this 'final' state on the left, to a state on the right at a lower level, i.e. where a bigger sub-word of w is covered. □

Remark: Guessing the appropriate (w, G)-instances is easy if we "remember" the sub-words of w covered along the top-down inferences; e.g. the sub-word QR covered by B gave us the clue in the above example.

Theorem 1 *The word w of size m is accepted by the* **cfcg** *G if and only if* every *chain of length $p \leq m$ of top-down transitions $t_1 \ldots t_p$ from the starting state on $\mathcal{A}(G, w)$, can be completed by a sequence of transitions $u_{p+1}t_{p+1} \ldots u_m t_m$ towards an acceptance state, where each t_j is top-down, and u_j a (possibly empty) chain of bottom-up transitions.*

References

1. J. Esparza, "*Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes*", Fundamenta Informaticae, Vol. 30, 1997, pp. 23-41.
2. D.T. Huynh, "*Commutative Grammars: The Complexity of the Uniform Word Problems*", Information and Control, Vol. 57, 1983, pp. 21-39.

Invited Talk: Simultaneous Rigid E-Unification and Formula Instantiation

Andrei Voronkov

University of Manchester
http://www.cs.man.ac.uk/~voronkov/

We give an overview of results on simultaneous rigid E-unification and the related problem of formula instantiation. We show how these problems arise in automated reasoning by the tableau method and in query answering for first-order formulas. Then we will explain techniques used to prove the known results on simultaneous rigid E-unification and formulate some open problems.

Theories of Equations in Finitely Presented Groups

Markus Lohrey

Universität Stuttgart, Institut für Informatik
Breitwiesenstr. 20–22, 70565 Stuttgart, Germany

This abstract reports some ongoing work on theories of equations in some finitely presented groups. The question whether an equation has a solution in some finitely presented group corresponds to a particular unification problem.

We start with a short historical overview on the topic. In 1977 Makanin has shown that for a free monoid the existential theory of equations is decidable [11]. Due to the technical difficulty of Makanin's proof it is not easy to give precise bounds on the complexity of the algorithm developed by Makanin. A first estimate by several towers of exponentials was finally be reduced to exponential space [7], which is currently the best known upper bound on Makanin's algorithm. A very good exposition of Makanin's algorithm can be found in [1]. In 1983 Makanin extended his decidability result to the existential theory of equations of a free group [12]. In this case Kościelski and Pacholsky [10] were able to prove that the scheme of Makanin is not primitive recursive for free groups. Based on results of Merzlyakov [14], Makanin has also shown that for a free group the positive theory of equations is decidable [13]. In 1999 Plandowski has developed a new method for solving word equations which allowed him to prove that for a free monoid the existential theory of equations is in PSPACE [15]. Plandowski's result was extended by Gutiérrez to the case of free groups [8]. Recently these decidability results were further extended to wider classes of finitely presented groups, namely free partially commutative groups [4] and plain groups [3]. Finally let us mention that the many of the decidability results mentioned above have been generalized to include the possibility of specifying constraints for variables, which typically have the form of recognizable sets [2, 16].

Currently we are trying to extend the decidability results mentioned above to a larger class of groups, which we call *partially commutative plain groups*, briefly *pcp groups*. These groups generalize both plain groups [9] and free partially commutative groups, which in algebra are also known as graph groups, see e.g [5]. Let us first define these groups. Let (V, E) be a finite undirected graph (i.e. E is a symmetric and irreflexive relation on V), where the set of nodes is partitioned into two sets V_{free} and V_{fin}. Each node $v \in V$ is labeled with a group G_v. In the case $v \in V_{\text{free}}$, G_v is the free group of rank 1, generated by a symbol a_v, and in the case $v \in V_{\text{fin}}$, G_v is a finite group. For $v \in V_{\text{fin}}$ define $\Gamma_v = G_i \backslash \{1\}$ and let $\overline{a} \in \Gamma_v$ be the inverse of $a \in \Gamma_v$ in G_v. For $v \in V_{\text{free}}$ define $\Gamma_v = \{a_v, \overline{a_v}\}$ and let $\overline{\overline{a_v}} = a_v$. We assume that the sets Γ_v for $v \in V$ are pairwise disjoint. Finally the pcp group $G(V, E)$ is the group generated (as a monoid) by the set $\Gamma = \bigcup_{v \in V} \Gamma_v$ and that satisfies the following defining relations:

- $a\overline{a} = 1$ for all $a \in \Gamma$

- $ab = c$ if there exists a $v \in V_{\text{fin}}$ such that $a, b, c \in \Gamma_v$ and $ab = c$ in the finite group G_v
- $ab = ba$ if there exists a $(u, v) \in E$ such that $a \in \Gamma_u$ and $b \in \Gamma_v$

Thus the group $G(V, E)$ results from the free product of the groups G_v $(v \in V)$ by allowing elements of E-adjacent groups to commute. This construction is a special case of the graph product investigated in [6]. If $E = \emptyset$ then $G(V, E)$ is also called a *plain group* [9] and if $V_{\text{fin}} = \emptyset$ then $G(V, E)$ is called a *free partially commutative group* or a *graph group*. For the following consideration let us fix a pcp group $G = G(V, E)$.

In the following Ω denotes a finite set of variables (or unknowns) and we let $^{-} : \Omega \to \Omega$ be an involution without fixed points. First we will define the existential theory of equations of the pcp group G. Atomic formulas are of type $U = V$, where $U, V \in (\Gamma \cup \Omega)^*$. A propositional formula is build up by atomic formulas using negations, conjunctions, and disjunctions. The evaluation of such a formula in G is completely straightforward, but of course if a variable X is interpreted by $x \in G$ then $\overline{X}$ is interpreted by x^{-1}. The existential theory of equations of the pcp group G consists of all closed existentially quantified propositional formulas which evaluate to *true* in G.

Conjecture 1. For every pcp group G the existential theory of equations is decidable.

The above conjecture was proven in [4] for the case of a free partially commutative group and in [3] for the case of a plain group.

A positive formula is build up by equations using conjunctions, disjunctions, existential quantifiers, and universal quantifiers, but does not contain negations. The positive theory of equations of the pcp group G consists of all closed positive formulas that evaluate to *true* in G. As already mentioned, Makanin has shown that for a free group the positive theory of equations is decidable [13]. Makanin used a result of Merzlyakov [14] in order to reduce the positive theory of equations of a free group to the existential fragment. We conjecture that we can extend this reduction to the case of general pcp groups, which together with Conjecture 1 leads to the following conjecture.

Conjecture 2. For every pcp group G the positive theory of equations is decidable.

Other interesting topics for further research include the search for good upper complexity bounds and the inclusion of constraints (e.g. recognizable constraints or even larger classes) for the variables.

References

1. V. Diekert. Makanin's Algorithm. In J. Berstel and D. Perrin, editors, *Algebraic Combinatorics on Words.* Cambridge University Press. A preliminary version is on the web: `http://www-igm.univ-mlv.fr/~berstel/Lothaire/index.html`.

2. V. Diekert, C. Gutiérrez, and C. Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. In A. Ferreira and H. Reichel, editors, *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS 01), Dresden (Germany) 2001*, number 2010 in Lecture Notes in Computer Science, pages 170–182. Springer, 2001.
3. V. Diekert and M. Lohrey. A note on the existential theory of equations in plain groups. *International Journal of Algebra and Computation*, 2001. to apear.
4. V. Diekert and A. Muscholl. Solvability of equations in free partially commutative groups is decidable. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 01), Crete (Greece) 2001*, Lecture Notes in Computer Science. Springer, 2001. to appear.
5. C. Droms. Isomorphisms of graph groups. *Proceedings of the American Mathematical Society*, 100:407–408, 1987.
6. E. R. Green. *Graph Products of Groups*. PhD thesis, The University of Leeds, 1990.
7. C. Gutiérrez. Satisfiability of word equations with constants is in exponential space. In *Proc. of the 39th Ann. Symp. on Foundations of Computer Science, FOCS 98*, pages 112–119, Los Alamitos, California, 1998. IEEE Computer Society Press.
8. C. Gutiérrez. Satisfiability of equations in free groups is in PSPACE. In *32nd Ann. ACM Symposium on Theory of Computing (STOC'2000)*. ACM Press, 2000.
9. R. H. Haring-Smith. Groups and simple languages. *Transactions of the American Mathematical Society*, 279:337–356, 1983.
10. A. Kościelski and L. Pacholski. Complexity of unification in free groups and free semigroups. In *Proceedings of the31st Annual Symposium on Foundations of Computer Science*, volume II, pages 824–829, Los Alamitos, 1990. IEEE Computer Society Press.
11. G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik*, 103:147–236, 1977. English transl. in Math. USSR Sbornik 32 (1977).
12. G. S. Makanin. Equations in a free group. *Izv. Akad. Nauk SSR*, Ser. Math. 46:1199–1273, 1983. English transl. in Math. USSR Izv. 21 (1983).
13. G. S. Makanin. Decidability of the universal and positive theories of a free group. *Izv. Akad. Nauk SSSR*, Ser. Mat. 48:735–749, 1984. In Russian; English translation in: *Math. USSR Izvestija, 25*, 75–88, 1985.
14. Y. I. Merzlyakov. Positive formulas on free groups. *Algebra i Logika Sem.*, 5(4):25–42, 1966. (Russian).
15. W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 99)*, pages 495–500. IEEE Computer Society Press, 1999.
16. K. U. Schulz. Makanin's algorithm for word equations — Two improvements and a generalization. In K. U. Schulz, editor, *Word Equations and Related Topics*, number 572 in Lecture Notes in Computer Science, pages 85–150, Berlin-Heidelberg-New York, 1991. Springer.

Parameterized Rewriting on Words

Benjamin Monate

LRI, CNRS UMR 8623, Bât. 490 - Université Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France
monate@lri.fr

1 Introduction

Basic rewriting techniques are widely used to perform automatic proofs about equational specifications. For example the symmetric groups $\mathcal{S}_n$ are studied as the quotient of a free structure by some well chosen equations. The Knuth-Bendix completion automatically computes a convergent finite rewriting that leads to an efficient decision algorithm for equality : the normalization. One can exhibit a convergent rewriting system for *each* $\mathcal{S}_n$, n being in $\mathbb{N}$. The completion cannot be done in practice for very large symmetric groups *i.e.* for large values of n, because its complexity is exponential in the number of equations. Our goal in this paper is to perform proofs about this kind of equational specification without instantiating the parameters. Therefore the system $\mathcal{S}_n$ is directly presented as a parameterized word rewriting system :

$$\begin{aligned}\mathcal{S}_n \ = \{ & a_i^2 = \epsilon \mid 1 \leq i \leq n-1 \\ & a_i a_j = a_j a_i \mid i - j \geq 2 \wedge 1 \leq i, j \leq n-1 \\ & (a_i a_{i+1})^3 = \epsilon \mid 1 \leq i \leq n-2 \}\end{aligned}$$

In Section 1 of the paper we define the syntax of parameterized rewriting. In Section 2 give a critical pair theorem in this setting and conclude with an extension to some other kind of parameterization.

An extended version of this abstract with complete proofs and more precise explanations (in French) is available at http://www.lri.fr/~monate/ftp/prw.html

2 Basic notions of parameterized rewriting

We define a formal syntax for parameterized signatures. For $\mathcal{S}_n$ we want to define the signature as $\Sigma = \{a_i|_I 1 \leq i \leq N-1\}_{\{N\}}$ where I is an infinite countable set of variables (containing i) and N is a parameter. The general definition is

Definition 1 *A parameterized signature is a* 4*-tuple* $\Sigma = (a, I, \varphi, \mathcal{N})$ *denoted by* $\Sigma = \{a_i|_I\, \varphi\}_{\mathcal{N}}$ *such that*

- *a is a generic arbitrary symbol*

- *$\mathcal{N}$ is a countable set of parameters.*
- *I is a countable set of variables such that $\mathcal{N} \cap I = \emptyset$ and $i \in I$.*
- *φ is a formula of Presburger's arithmetic [2] whose variables are in $I \cup \mathcal{N}$.*

We need to give an interpretation to a parameterized signature that links the notion of parameterized signatures to the notion of alphabets. We define a function $[\![\,]\!]$ which associates an infinite family of alphabets with any parameterized signature. For the example of $\mathcal{S}_n$ the interpretation has to satisfy

$$[\![\Sigma]\!] = (\emptyset; \{a_1\}; \{a_1; a_2\}; \cdots; \{a_1; a_2; \cdots; a_k\} \cdots)$$

Definition 2 *For any substitution $\nu : N \to \mathbb{N}$ the function $[\![.]\!]_\nu$ is defined by*

$$[\![\{a_i|_I\varphi\}_{\mathcal{N}}]\!]_\nu = \bigcup_{\substack{\mu\,:\,I \to \mathbb{N} \\ \nu(\mu(\varphi)) \text{ is true}}} \{a_{\nu(\mu(i))}\}$$

This function is called ν-interpretation *and associates an alphabet that may be infinite, empty or finite with every parameterized signature. One can define the interpretation of a signature Σ by the countable family $([\![\Sigma]\!]_\nu)_{\nu:N\to\mathbb{N}}$.*

From now on we will implicitly refer to the signature $\Sigma = \{a_i|_I\varphi\}_{\mathcal{N}}$.

Parameterized words are intuitively words build over a parameterized signature. In the case of $\mathcal{S}_n$ one has to handle words like a_1 (which exists only when $n \geq 2$) or $a_i a_j | i - j \geq 2$. The formal definition handles these cases.

Definition 3 *A* parameterized word *is either $a_{i_0} \ldots a_{i_k}|\psi$ or $\epsilon|\psi$ such that*

- *ψ is a formula of Presburger's arithmetic whose variables are in $I \cup \mathcal{N}$*
- *k is a non-negative integer*
- *$i_0 \cdots i_k$ are elements of I*

The ν-interpretation of a parameterized word is a set of words built over the alphabet $[\![\Sigma]\!]_\nu$.

Definition 4 *The ν-interpretation of a parameterized word is defined by*

$$[\![a_{i_0} \cdots a_{i_k}|\psi]\!]_\nu = \bigcup_{\substack{\mu\,:\,I \to \mathbb{N} \\ \nu(\mu(\psi)) \text{ is true} \\ j \in [0;k] a_{\nu(\mu(i_j))} \in [\![\Sigma]\!]_\nu}} \{a_{\nu(\mu(i_0))} \cdots a_{\nu(\mu(i_k))}\}$$

and $[\![\epsilon|\psi]\!]_\nu = \begin{cases} \{\epsilon\} & \text{if } \nu(\psi) \text{ is satisfiable} \\ \emptyset & \text{if } \nu(\psi) \text{ is not satisfiable} \end{cases}$.

The family of all ν-interpretations of a word is called the interpretation *of the word.*

For example we have $[a_1]_{n\to 2} = \{a_1\}$, $[a_1]_{n\to 0} = \emptyset$ and $[a_j a_i | i-j \geq 2]_{n\to 5} = \{a_1a_3; a_1a_4; a_2a_4\}$. The ν-interpretation of a parameterized word can be infinite if the ν-interpretation of the signature is an infinite alphabet.

We introduce a syntactical definition that let us write rules like $u_iu_j \to u_ju_i | i - j \geq 2$. Note that such a rule represents the set of rules $u_iu_j \to u_ju_i$ for all integers i and j satisfying the constraint.

Definition 5 *A parameterized rule is a triplet (u, v, ψ) denoted $(u \to v)|\psi$ such that*

- *u and v are two parameterized words without constraint*
- *ψ is formula of Presburger's arithmetic*

A parameterized rewriting system is a finite set of parameterized rules. As usual u is the left member of the rule and r the right member. ψ is the constraint of the rule.

The ν-interpretation of a rule is a set of rules built over the ν-interpretation of the signature. For example $[u_iu_j \to u_ju_i | i-j \geq 2]_{N\to 4} = \{u_3u_1 \to u_1u_3\}$.

Definition 6 *The* ν-interpretation *of a rule is defined by*

$$[\![(u = v)|\psi]\!]_\nu = \bigcup_{\substack{\mu : I \to \mathbb{N} \\ \nu(\mu(\psi)) \text{ is true}}} \{\nu(\mu(u)) \to \nu(\mu(v))\}$$

The ν-interpretation of a rewriting system is the union of the ν-interpretation of its rules.

The main property of the parameterized rewriting is that it is preserved by interpretation in the sense that if $u \to_R v$ for parameterized rewriting then for all $u_0 \in [u]_\nu$ and $v_0 \in [\![v]\!]_\nu$ the rewriting system $R_0 = [\![R]\!]_\nu$ reduces u_0 into v_0. Moreover every v_0 is obtained by the reduction from an u_0. The following definition for the rewriting relation has this property.

Definition 7 *In the signature $\Sigma = \{a_i|_I\, \varphi\}_N$, the parameterized word $u|\psi$ rewrites into $v|\chi$ by the rule $l \to r|\xi$ at position $p \in \mathbb{N}$ with the substitution $\sigma : I \to Expr(I, N)$ and we denote*

$$u|\psi \xrightarrow[p,\sigma]{l\to r|\xi} v|\chi \quad \textit{if and only if}$$

1. *Matching conditions:*
   (a) *u has a subword of length $|l|$ at position p* i.e.
   $$\begin{array}{ll} u = a_{i_1} \cdots a_{i_{p-1}} & a_{i_p} \cdots a_{i_{p+|l|-1}}\; a_{i_{p+|l|}} \cdots a_{i_m} \\ l = & a_{i'_1} \cdots a_{i'_{|l|}} \end{array}$$
   (b) *$\psi \Rightarrow i_p = \sigma(i'_1)$, ... , $\psi \Rightarrow i_{p+|l|-1} = \sigma(i'_{|l|})$ are valid formulae of Presburger's arithmetic.*

(c) *Denoting* $\{x_1; \ldots ; x_n\} = Var(\sigma(\xi)) \setminus Var(\psi)$, *the formula* $\psi \Rightarrow \exists x_1 \ldots x_n \sigma(\xi)$ *is valid.*

2. *Production conditions:*
   (a) $v = a_{\sigma(i_1)} \cdots a_{\sigma(i_{p-1})} \sigma(r) a_{\sigma(i_{p+|l|})} \cdots a_{\sigma(i_m)}$
   (b) $\sigma(\chi) \Leftrightarrow \psi \wedge \sigma(\xi)$ *is valid.*

This definition has been proved to have the previously mentioned preservation property. For example the term $a_{i+3}a_i|i \geq 1 \wedge i+3 \leq n-1$ reduces into $a_i a_{i+3}|i \geq 1 \wedge i+3 \leq n-1$. The latter is not reducible.

3 Completion for parameterized rewriting

We can now define the notion of critical pair of rules by enumerating all possible overlaps between the left members of rules and prove a critical pair lemma.

Theorem 1 *A parameterized rewriting system is locally confluent if and only if its critical pairs are locally confluent.*

This theorem is interesting because local confluence is preserved by ν-interpretation. But on the contrary even all ν-interpretation can be confluent whereas the parameterized system is not.

We have an algorithm to perform completion on parameterized rewriting systems. As always one needs to find suitable orderings on parameterized words. An implementation is currently under development in the software C*i*ME [1]. We have also extended the syntax of words in order to allow *exponents* in words in order to describe systems like $a_i^n \to \epsilon$ and perform completion on such systems. This is a very promising extension as presentations of algebraic structures often use exponents. For example the presentation of $\mathcal{S}_n$ with a transposition and an n-cycle contains $a^n = \epsilon$.

References

1. Evelyne Contejean, Claude Marché, Benjamin Monate, and Xavier Urbain. Cime version 2, 2000. Prerelease available at `http://www.lri.fr/~demons/cime.html`.
2. M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du premier Congrès des Mathématiciens des Pays slaves*, Warszawa, 1929.

Second-Order Schema Matching Based on Projection Point Labeling*

Keizo Yamada, Kouichi Hirata, and Masateru Harao

Department of Artificial Intelligence
Kyushu Institute of Technology
Kawazu 680-4, Iizuka 820-8502, Japan
{yamada,hirata,harao}@ai.kyutech.ac.jp

Summary. In this paper, we propose a second-order schema matching algorithm based on a projection point labeling. This algorithm provides us to construct some suitable matchers efficiently.

1 Introduction

An expression which exhibits abstracted or generalized knowledge is called a *schema*, and the knowledge processing which deals with such a schema as a meta knowledge is called *schema-guided* [2]. The schema-guided knowledge processing is known to be useful for automated program synthesis [2], program transformation [7], analogical reasoning [3] and so on. It is important for the schema-guided knowledge processing to design the matching procedure between a schema and a given problem, which is called a *schema matching*. In this paper, we regard a schema as a second-order formula without quantifiers on predicate and function variables.

Since we can formulate both a schema and a first-order closed formula representing a problem as second-order λ-terms, the schema matching is regarded as a kind of a *second-order matching*, and is known to be intractable in general [1]. On the other hand, our previous work [5] has given a sharp characterization between a tractable and an intractable second-order matching. In particular, we have shown that the schema matching without free individual variables is tractable, and designed a schema matching algorithm by extending the transformations introduced by Huet and Lang [6, 7] and by dividing into two stages, a *predicate* and a *term* matching [4, 8].

In this paper, we divide the schema matching procedure into two processes, a *pre-process* and a *projection position labeling*, instead of the above division. The pre-process consists of a simplification [6, 7]. For an expression E as a set of pairs of schemata and first-order closed formulas, the result applying a pre-process to E is an expression of the form $\{\langle P_i(t_1^i, \ldots, t_{n_i}^i), \varphi_i\rangle \mid i \in I\}$, where P_i is a predicate variable and φ_i is a first-order closed formula. The projection position labeling labels to the symbols of a formula φ_i to which a projection is applicable [6, 7]. Then, we determine

* This work is partially supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (B) 13558036 and 11558040.

whether or not E is matchable, by extracting a common part of labeled formulas for each $i \in I$. The present schema matching is more suitable for the implementation by procedural programming languages such as Java than the previous one [4,8].

2 Preliminaries

Let IC, IV, FC, PC, and PV be a set of *individual constants* (denoted by $a, b, c, \ldots$), a set of *individual variables* (denoted by $x, y, x, \ldots$), a set of *function constants* (denoted by $f, g, h, \ldots$), a set of *predicate constants* (denoted by $p, q, r, \ldots$), and a set of *predicate variables* (denoted by $P, Q, R, \ldots$), respectively. We deal with no function variables explicitly, by the definition of schemata denoted below. Throughout of this paper, we assume a set of elementary types containing the Boolean type o. Furthermore, we assume that each $d \in IC \cup IV \cup FC \cup PC \cup PV$ has a type denoted by $\tau(d)$. Here, each $d \in IC \cup IV$ has an elementary type not equal to o, each $d \in FC$ has a type $\mu_1 \times \cdots \times \mu_n \rightarrow \mu$ where neither μ_i nor μ is o, and each $d \in PC \cup PV$ has a type $\mu_1 \times \cdots \times \mu_n \rightarrow o$.

Terms are defined as usual [1]. Here, we assume that each term contains no λ *abstraction*. The *size* of a term t is the number of occurrences of all symbols in t and is denoted by $|t|$.

In this paper, we deal with the logical connectives $\wedge, \vee, \supset$ as predicate constants satisfying that $\tau(\wedge) = \tau(\vee) = \tau(\supset) = o \times o \rightarrow o$ and $\tau(\neg) = o \rightarrow o$. For a quantifier Q ($Q \in \{\forall, \exists\}$) we also deal with $Qx.$ as the predicate constant satisfying that $\tau(Qx.) = o \rightarrow o$. If $\tau(\varphi) = o$ and $\tau(x) \neq o$, then $Qx.\varphi$ has the type o.

We assume that a *schema* is a term containing predicate variables and neither function nor free individual variables, a *first-order formula* is a term containing neither predicate variables nor function variables, and a *first-order closed formula* is a first-order formula containing no individual free variables (or a *formula* in short). Here, *bound* and *free* variables (with respect to quantifiers) are defined as usual. In the following, we denote schemata and formulas by $\Phi, \Psi, \ldots$ and $\varphi, \psi, \ldots$, respectively. A *head* of a schema Φ is a left-most symbol of Φ and denoted by $\mathrm{hd}(\Phi)$.

In contrast to the terms, a *substitution-term* is just a λ-term [6, 7]. For a substitution-term $\lambda v_1 \cdots v_n.t$ and a term t_i ($1 \leq i \leq n$), $t[v_1 := t_1, \ldots, v_n := t_n]$ denotes the substitution-term replacing a variable v_i in t to t_i simultaneously, which is the same expression $(\lambda v_1 \cdots v_n.t)t_1 \cdots t_n$ in λ-calculus. For a substitution-term t_i and an individual variable v_i ($1 \leq i \leq m$) such that t_i does not contain v_i as free variables, $\{t_1/v_1, \ldots, t_m/v_m\}$ is called a *substitution*.

For a term t and a substitution θ, $t\theta$ is defined inductively as follows:

1. If $t = c$, then $t\theta = c$.
2. If $t = x$ and $t'/x \in \theta$, then $t\theta = t'$; otherwise $t\theta = x$.
3. If $t = f(\overline{t_n})$ and $f \in FC \cup PC$, then $t\theta = f(\overline{t_n\theta})$.
4. If $t = P(\overline{t_n})$, $P \in PV$ and $\lambda\overline{v_n}.t'/P \in \theta$, then $t\theta = t'[v_1 := t_1\theta, \ldots, v_n := t_n\theta]$; otherwise $t\theta = P(\overline{t_n\theta})$.
5. If $t = Qx.t'$ and $Q \in \{\forall, \exists\}$, then $t\theta = Qy.((t'\{y/x\})\theta)$, where y is a new variable.

A finite set of pairs of schemata and formulas is called an *expression*. An expression of the form $\{\langle P_i(t_1^i, \dots, t_{n_i}^i), \varphi_i \rangle \mid i \in I\}$ is called a *reduced expression*, where P_i is a predicate variable and φ_i is a formula. For an expression $E = \{\langle \Phi_i, \varphi_i \rangle \mid i \in I\}$, the *size* of E, denoted by $|E|$, is defined as $\sum_{i \in I}(|\Phi_i| + |\varphi_i|)$.

Let E be an expression $\{\langle \Phi_i, \varphi_i \rangle \mid i \in I\}$. A substitution θ such that $\Phi_i\theta = \varphi_i$ for all $i \in I$ is called a *matcher* of E. The *schema matching* for E is a procedure to find a matcher of E. If there is a matcher of E, then E is called *matchable*.

3 Schema Matching Based on Projection Point Labeling

For an expression E, our schema matching first applies the following *pre-process* to E until the reduced expression is obtained:

1. $\{\langle p(\overline{s_n}), p(\overline{t_n}) \rangle\} \cup E \Rightarrow \{\langle s_i, t_i \rangle \mid 1 \le i \le n\} \cup E$, where $p \in IC \cup FC \cup PC \cup \{\neg, \wedge, \vee, \supset\}$.
2. $\{\langle Qx.\Phi, Qy.\varphi \rangle\} \cup E \Rightarrow (E - \{\langle Qx.\Phi, Qy.\varphi \rangle\}) \cup \{\langle \Phi\{w/x\}, \varphi\{w/y\} \rangle\}$, where $Q \in \{\forall, \exists\}$.

Here, w is a special individual constant (possibly subscribed) representing the bound variables in an expression. The pre-process is corresponding to a *simplification* [6,7]. Then, we can show the following theorem.

Theorem 1. *For an expression E, let E' be a reduced expression obtained by applying the pre-process to E. Then, E is matchable iff E' is matchable.*

Let E be a reduced expression $\{\langle P_i(\Phi_1^i, \dots, \Phi_{n_i}^i), \varphi_i \rangle \mid i \in I\}$ obtained by applying the pre-process. Then, our schema matching applies the *projection position labeling* to E. Intuitively, each label presents the positions to which a projection can be applicable. Note that a projection cannot be applicable to all individual variables in a reduced expression, because they are always bound.

Definition 1. Let Φ be a schema $P(\Phi_1, \dots, \Phi_n)$, φ a formula, and $\rho(\varphi)$ a set $\{i \mid \varphi = \Phi_i, 1 \le i \le n\}$. Then, a *judgment term* $T(\Phi, \varphi)$ of φ for Φ is a labeled term defined inductively as follows:

1. $T(\Phi, w) = *^{\rho(w)}$, $T(\Phi, c) = c^{\rho(c)}$ $(c \in IC)$, and $T(\Phi, x) = x^{\emptyset}$ $(x \in IV)$.
2. If $\varphi = f(\varphi_1, \dots, \varphi_m)$, $f \in FC$ and $T(\Phi, \varphi_i)$ is a judgment term of φ_i for Φ $(1 \le i \le m)$, then $T(\Phi, \varphi) = f^{\rho(t)}(T(\Phi, \varphi_1), \dots, T(\Phi, \varphi_m))$.
3. If $t = p(\varphi_1, \dots, \varphi_m)$, $p \in PC \cup \{\neg, \wedge, \vee, \supset\}$ and $T(\Phi, \varphi_i)$ is a judgment term of φ_i for Φ $(1 \le i \le m)$, then $T(\Phi, \varphi) = p^{\emptyset}(T(\Phi, \varphi_1), \dots, T(\Phi, \varphi_m))$.
4. If $t = Qx.\varphi_1$ $(Q \in \{\forall, \exists\})$ and $T(\Phi, \varphi_1)$ is a judgment term of φ_1 for Φ, then $T(\Phi, \varphi) = Qx^{\emptyset}.T(\Phi, \varphi_1)$.

Here, $*$ is a new symbol to which no imitations can be applicable.

Furthermore, we introduce a *common judgment term* as a common part of judgment terms.

Definition 2. For a reduced expression $\{\langle \Phi_i, \varphi_i \rangle \mid i \in I\}$ such that $\mathrm{hd}(\Phi_i) = P$, let $T(\Phi_i, \varphi_i)$ be a judgment term of φ_i for Φ_i. Then, a *common judgment term* $\sqcap_{i \in I} T(\Phi_i, \varphi_i)$ of $\{T(\Phi_i, \varphi_i) \mid i \in I\}$ is defined inductively as follows:

1. If $T(\Phi_i, \varphi_i) = c^{\rho_i}$ and $c \in IC \cup IV \cup \{*\}$ for each $i \in I$, then:
$$\sqcap_{i \in I} T(\Phi_i, \varphi_i) = c^{\cap_{i \in I} \rho_i}.$$
2. If $T(\Phi_i, \varphi_i) = f^{\rho_i}(\varphi_1^i, \dots, \varphi_m^i)$ and $f \in FC$ for each $i \in I$, then:
$$\sqcap_{i \in I} T(\Phi_i, \varphi_i) = f^{\cap_{i \in I} \rho_i}(\sqcap_{i \in I} T(\Phi_i, \varphi_1^i), \dots, \sqcap_{i \in I} T(\Phi_i, \varphi_m^i)).$$
3. If $T(\Phi_i, \varphi_i) = p^{\emptyset}(\varphi_1^i, \dots, \varphi_m^i)$ and $p \in PC \cup \{\neg, \wedge, \vee, \supset\}$ for each $i \in I$, then:
$$\sqcap_{i \in I} T(\Phi_i, \varphi_i) = p^{\emptyset}(\sqcap_{i \in I} T(\Phi_i, \varphi_1^i), \dots, \sqcap_{i \in I} T(\Phi_i, \varphi_m^i)).$$
4. If $T(\Phi_i, \varphi_i) = Qx_i^{\emptyset}.\varphi_i'$ and $Q \in \{\forall, \exists\}$ for each $i \in I$, then:
$$\sqcap_{i \in I} T(\Phi_i, \varphi_i) = Qx^{\emptyset}. \sqcap_{i \in I} (T(\Phi_i, \varphi_i')\{x/x_i\}).$$
5. If there exist $k, j \in I$ such that $\mathrm{hd}(T(\Phi_k, \varphi_k)) \neq \mathrm{hd}(T(\Phi_j, \varphi_j))$, then:
$$\sqcap_{i \in I} T(\Phi_i, \varphi_i) = *^{\cap_{i \in I} \rho_i}.$$

Finally, we introduce a *reduced judgment term*.

Definition 3. For a reduced expression $E = \{\langle \Phi_i, \varphi_i \rangle \mid i \in I\}$, a *reduced judgment term* $\sqcap E$ of E is a labeled term obtained by applying the following rule to $\sqcap_{i \in I} T(\Phi_i, \varphi_i)$ as possible:

> For a term t in $\sqcap_{i \in I} T(\Phi_i, \varphi_i)$, if there exists a j ($1 \leq j \leq m$) such that $t = f^{\emptyset}(t_1, \dots, t_m)$ and $t_j = *^{\emptyset}$, then replace t with $*^{\emptyset}$ in $\sqcap_{i \in I} T(\Phi_i, \varphi_i)$.

As similar as our previous work [8], the following theorem holds.

Theorem 2. *Let E be a reduced expression. Then, $\sqcap E$ can be constructed in $O(|E|^2)$ time. Furthermore, E is matchable iff $\sqcap E \neq *^{\emptyset}$.*

Example 1. Consider the following expression E_1:

$$E_1 = \left\{ \begin{array}{l} \langle \forall x_1.((P(x_1, f(x_1)) \wedge P(x_1, g(x_1))) \supset P(f(x_1), x_1)), \\ \forall x_2.((\exists z_1.p(z_1, f(x_2)) \wedge \exists z_2.p(z_2, g(x_2))) \supset \exists z_3.p(z_3, f(x_2)))\rangle \end{array} \right\}.$$

At first, we apply the pre-process to E_1 as follows.

$$\begin{array}{rl} E_1 \Rightarrow & \left\{ \begin{array}{l} \langle (P(w, f(w)) \wedge P(w, g(w))) \supset P(f(w), w), \\ \quad (\exists z_1.p(z_1, f(w)) \wedge \exists z_2.p(z_2, g(w))) \supset \exists z_3.p(z_3, f(w)))\rangle \end{array} \right\} \\ \Rightarrow & \left\{ \begin{array}{l} \langle P(w, f(w)) \wedge P(w, g(w)),\ \exists z_1.p(z_1, f(w)) \wedge \exists z_2.p(z_2, g(w))\rangle, \\ \langle P(f(w), w),\ \exists z_3.p(z_3, f(w))\rangle \end{array} \right\} \\ \Rightarrow & \left\{ \begin{array}{l} \langle P(w, f(w)),\ \exists z_1.p(z_1, f(w))\rangle, \\ \langle P(w, g(w)),\ \exists z_2.p(z_2, g(w))\rangle, \\ \langle P(f(w), w),\ \exists z_3.p(z_3, f(w))\rangle \end{array} \right\} \quad \left(= \left\{ \begin{array}{l} \langle \Phi_1, \varphi_1 \rangle, \\ \langle \Phi_2, \varphi_2 \rangle, \\ \langle \Phi_3, \varphi_3 \rangle \end{array} \right\} = E_1' \right). \end{array}$$

Next, we apply the projection point labeling to E_1'. The judgment terms are:

$$T(\Phi_1,\varphi_1) = \exists z_1^{\emptyset}.p^{\emptyset}(z_1^{\emptyset}, f^{\{2\}}(w^{\{1\}})), T(\Phi_2,\varphi_2) = \exists z_2^{\emptyset}.p^{\emptyset}(z_2^{\emptyset}, g^{\{2\}}(w^{\{1\}})),$$
$$T(\Phi_3,\varphi_3) = \exists z_3^{\emptyset}.p^{\emptyset}(z_3^{\emptyset}, f^{\{1\}}(w^{\{2\}})).$$

By Definition 2, it holds that $\sqcap_{i\in\{1,2,3\}} T(\Phi_i,\varphi_i) = \exists z^{\emptyset}.p^{\emptyset}(z^{\emptyset}, *^{\emptyset})$. By Definition 3, it holds that $\sqcap E_1' = *^{\emptyset}$, which implies that E is not matchable by Theorem 1 and 2. On the other hand, consider the following expression E_2:

$$E_2 = \left\{ \begin{array}{l} \langle \forall x_1(P(x_1,f(x_1)) \wedge P(x_1,g(x_1)), \\ \ \forall x_2(\exists z_1.p(z_1,f(x_2)) \wedge \exists z_2.p(z_2,g(x_2)))\rangle \end{array} \right\}.$$

By the pre-process for E_2, we obtain the following reduced expression E_2'.

$$E_2' = \{\langle \Phi_1,\varphi_1\rangle, \langle \Phi_2,\varphi_2\rangle\}.$$

It holds that $\sqcap_{i\in\{1,2\}} T(\Phi_i,\varphi_i) = \exists z^{\emptyset}.p^{\emptyset}(z^{\emptyset}, *^{\{2\}}) = \sqcap E_2' \neq *^{\emptyset}$, which implies that E_2 is matchable by Theorem 1 and 2. Furthermore, we can extract the matcher of E_2 as $\{\lambda xy.\exists z.p(z,y)/P\}$, by using $\sqcap E_2'$

4 Discussion

We have proposed in this paper a second-order schema matching algorithm which decides matchability in $O(|E|^2)$ time. From the viewpoint of applications, it is important to design an algorithm which does not only decide matchability but also produces matchers efficiently. Since the expression by projection point labeling holds all the information for the applicability of the rules, we are able to design an algorithm which extracts some optimal matcher, by introducing moderate heuristics as for the application of rules. We have also shown that some efficient matcher extracting algorithms are available [8].

References

1. L. D. Baxter, *The complexity of unification*, Doctoral Thesis, Department of Computer Science, University of Waterloo, 1977.
2. P. Flener, *Logic program synthesis from incomplete information* (Kluwer Academic Press, 1995).
3. M. Harao, *Proof discovery in LK system by analogy*, in: Proc. 3rd Asian Computing Science Conf., LNCS **1345** (Springer, 1997) 197–211.
4. M. Harao, K. Yamada, K. Hirata, *Efficient second order predicate matching algorithm*, in: Proc. Korea-Japan Joint Workshop on Algorithm and Computations, 31–39, 1999.
5. K. Hirata, K. Yamada, M. Harao, *Tractable and intractable second-order matching problems*, in: Proc. 5th Annual Intern. Computing and Combinatorics Conf., LNCS **1627** (Springer, 1999) 432–442.
6. G. P. Huet, *A unification algorithm for typed λ-calculus*, Theor. Comput. Sci. **1** (1975) 27–57.
7. G. P. Huet, B. Lang, *Proving and applying program transformations expressed with second-order patterns*, Acta Inform. **11** (1978) 31–55.
8. K. Yamada, K. Hirata, M. Harao, *Schema matching and its complexity*, Trans. IEICE **J82-D-I** (1999) 1307–1316 (in Japanese).

Invited Talk: The Complexity of Word Equations

Claudio Gutiérrez

Computer Science Department
Universidad de Chile

A *word* over an alphabet A is a finite sequence of elements of A. The problem this talk surveys has an almost trivial statement: Given alphabets $C = \{c_1, \ldots, c_n\}$ (of constants) and $V = \{v_1, \ldots, v_m\}$ (of variables) and two words w_1, w_2 over the alphabet $C \cup V$, are there words $u_1, \ldots, u_m$ over C such that after replacing each v_i by u_i in w_1, w_2 we get two identical words? This kind of equations have received several names, like *word equations* (A. A. Markov), *string diophantine equations* (H. Wang), *equations in free semigroups* (in the mathematical community), *associative unification* (in the computer science community).

The problem gained attention in the early sixties in relation to the problem of solving Diophantine equations in the work of A. A. Markov. In the early seventies the problem attracted the interest of the people in Computer Science in relation to formal languages, automata, variable length codes, unification.

Several partial results were obtained before G. S. Makanin, in 1977, solved the problem in its complete generality given an algorithm which when inputing an arbitrary word equation answer yes or no depending if it has a solution or not. Makanin's procedure gained fame as one of the most complex decision procedures in the literature. After this breakthrough, the problem spread in several directions, the computational complexity of it playing a central role. Using Makanin's algorithm it was shown that the problem of satisfiability of word equations is in EXPSPACE. Recently, in the late 90's, Rytter and Plandowski proved that minimal solutions of word equations are highly compressible. Form here, by purely word combinatorial methods, in 1999 Plandowski devised an algorithm which gives a PSPACE bound for the general problem, which is the best known so far. Proving lower bounds has shown to be much simpler, and it is known that the problem is NP-hard. Today it is conjectured that the problem is NP-complete.

In this survey we are going to restrict ourselves to complexity issues. We will take a look at Lentin's Pig-pug method (published independently by Plotkin), Makanin's algorithm, Rytter and Plandowski's result about compression and Plandowski's algorithm. We will discuss the role played by the variables. There are two paradigmatic cases, both known: if we restrict to equations which have at most two distinct variables (any number of occurrences) the satisfiability problem can be solved in polynomial time. On the contrary, for equations in which every variable occurs at most twice, the problem was shown to be NP-hard. We will finish with some open problems.

Complexity of Unification and Matching Problems in the Varieties of Idempotent Semigroups

Ondřej Klíma

Dept. of Mathematics, Masaryk University, Janáčkovo nám. 2a,
662 95 Brno, Czech Republic
klima@math.muni.cz

Abstract

The complexity of the Matching problem in the theory of one associative, idempotent and commutative binary function symbol is known to be polynomial. On the other hand, it was proved that the Matching problem is NP-complete in the theory consisting of one binary function symbol which is only associative and idempotent (see [8]). We describe the complexity of the Matching problem for all equational theories of a single binary function symbol consisting of associativity, idempotency together with any set of additional identities. We also describe the complexity of the Unification problem for these theories.

Overview

From algebraic point of view every equational theory corresponds to a variety of algebras of a given type and equational unification is solving of equations in the free algebra of this variety.

An idempotent semigroup (also called a band) is a semigroup where the identity $x^2 = x$ is satisfied. The variety of all bands is denoted as $\mathcal{B}$. The lattice of all varieties of bands (see Figure 1) was described independently by Birjukov [3], Fennemore [4] and Gerhard [5].

Some varieties of bands from the bottom of the lattice are assigned certain names in literature:

$\mathcal{LSNB} = \mathrm{Mod}(x^2 = x, xyzxz = xyz)$ – all left seminormal bands,
$\mathcal{RSNB} = \mathrm{Mod}(x^2 = x, xzxyz = xyz)$ – all right seminormal bands,
$\mathcal{RB} = \mathrm{Mod}(x^2 = x, xyxzx = xyzx)$ – all regular bands,
$\mathcal{LRB} = \mathrm{Mod}(x^2 = x, xyx = xy)$ – all left regular bands,
$\mathcal{RRB} = \mathrm{Mod}(x^2 = x, xyx = yx)$ – all right regular bands,
$\mathcal{NB} = \mathrm{Mod}(x^2 = x, xyzx = xzyx)$ – all normal bands,
$\mathcal{RCB} = \mathrm{Mod}(x^2 = x, xyx = x)$ – all rectangular bands,
$\mathcal{SL} = \mathrm{Mod}(x^2 = x, xy = yx)$ – all semilattices,
$\mathcal{T}$ – all trivial semigroups.

Baader [1] and Schmidt-Schauss [9] proved that the Unification problem in the variety of all bands is of type zero. Then Baader obtained the following characterization of unification type for all varieties of bands.

Proposition 1 ([2]). *The Unification problem for a variety of bands $\mathcal{V}$ is of unitary type if $\mathcal{V} \in [\mathcal{T}, \mathcal{RCB}]$, finitary type if $\mathcal{V} \in [\mathcal{SL}, \mathcal{NB}]$ and type zero otherwise.*

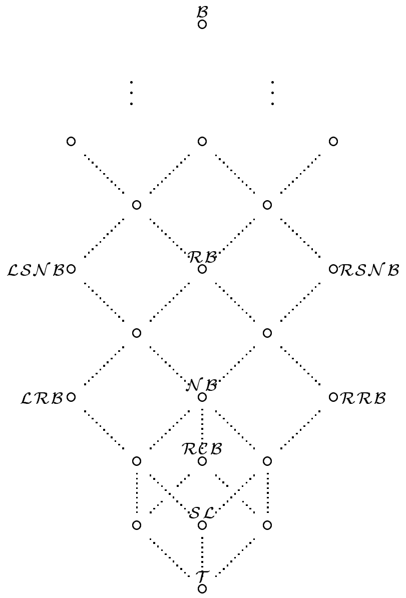


Fig. 1. Lattice of varieties of bands

As we are interested only in the existence of solutions, we concentrate on the Unification problem with constants, because every system of equations without constants is trivially solvable in an arbitrary variety of bands.

The Matching problem is a special case of the Unification problem where only systems of equations with variable-free right-hand sides (so called matching systems) are considered. Recall a result about complexity of the Matching problem in $\mathcal{B}$.

Proposition 2 **([8]).** *The Matching problem in the variety of all bands $\mathcal{B}$ is NP-complete.*

In contrast with [8] we assume that our function symbol satisfies some additional non-trivial identities. We describe the borderline between varieties with the Matching (resp. Unification) problems decidable in polynomial time and varieties with the NP-complete Matching (resp. Unification) problems (under the assumption P$\neq$NP).

Our Results[1]

One can prove that the Unification problem in every proper subvariety of the variety $\mathcal{B}$ is in NP. The proof of such a proposition is based on the fact that the validity of identities for every variety of bands is decidable in polynomial time and on the following lemma.

Lemma 1. *Let $\mathcal{V}$ be a proper subvariety of the variety $\mathcal{B}$ which has exactly k subvarieties. Then for an arbitrary n-element set Σ and a word $u \in \Sigma^+$ there is a word $w \in \Sigma^+$ such that $|w| < 4n^k$ and $u = w$ is an identity for the variety $\mathcal{V}$.*

[1] For the proofs of the results see [7].

Most of the Unification and Matching problems in the varieties of bands are NP-hard.

Proposition 3. *Let $\mathcal{V}$ be a subvariety of the variety of all bands $\mathcal{B}$ which contains some non-regular band, i.e. $\mathcal{V} \not\subseteq \mathcal{RB}$. Then the Matching problem in the variety $\mathcal{V}$ is NP-hard.*

Proposition 4. *Let $\mathcal{V}$ be a proper subvariety of the variety of all bands $\mathcal{B}$ which contains some non-normal band, i.e. $\mathcal{V} \not\subseteq \mathcal{NB}$. Then the Unification problem in the variety $\mathcal{V}$ is NP-hard.*

We use a reduction from the NP-complete problem 3-SAT to prove NP-hardness of our problems. For a formula Φ we construct a system of equations S such that S is satisfiable in the varieties from the interval $[\mathcal{LSNB}, \mathcal{B}]$ (resp. $[\mathcal{LRB}, \mathcal{B}]$) if and only if Φ is satisfiable. The system for the proof of Proposition 3 can be taken from [8] and the system for the proof of Proposition 4 is similar.

We are able to find polynomial-time algorithms solving the Unification and Matching problems in certain varieties of bands.

Proposition 5 ([6]). *The Unification problem in the variety $\mathcal{SL}$ is decidable in polynomial time.*

An easy modification of the original proof of Proposition 5 gives the same result for all varieties $\mathcal{V} \subseteq \mathcal{NB}$.

It is not difficult to see that every satisfiable matching system in the variety $\mathcal{SL}$ has the maximal solution α in the sense that for each variable x, $\alpha(x)$ contains all letters occurring in $\beta(x)$ for some solution β. A similar result can be proved for the other subvarieties of the variety $\mathcal{RB}$. Moreover, such a maximal solution can be computed in polynomial time if it exists and if it is not the case then we are able to discover it also in polynomial time.

Conclusion

Theorem 1. *The Unification problem is decidable in polynomial time for all subvarieties of the variety $\mathcal{NB}$ and it is NP-complete for the other proper subvarieties of $\mathcal{B}$. The Matching problem is decidable in polynomial time for all subvarieties of the variety $\mathcal{RB}$ and it is NP-complete for the other subvarieties of $\mathcal{B}$. (See Figure 2.)*

In the languague of equational theories, this result can be formulated in the following way. The Unification problem in a theory of a single associative and idempotent binary function symbol satisfying some additional non-trivial identity is decidable in polynomial time if the function symbol satisfies the identity $xyzx = xzyx$ and it is NP-complete otherwise. An analogous result for the Matching problem and the identity $xyxzx = xyzx$ can be formulated.

For a complete description of the complexity of the Unification and Matching problems in the varieties of idempotent semigroups it remains to determine the complexity of the Unification problem in the variety of all idempotent semigroups $\mathcal{B}$. Up to our knowledge, it is an open problem.

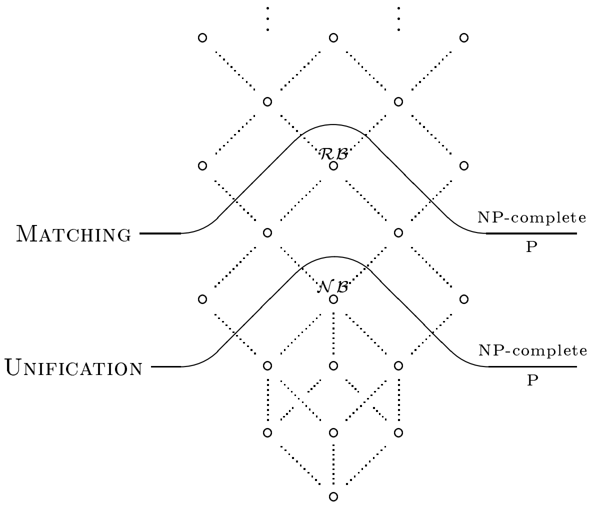


Fig. 2. Complexity of the Unification and Matching problems in the proper subvarie of the variety of all bands $\mathcal{B}$

References

1. Baader F.: The Theory of Idempotent Semigroups is of Unification Type Zero, J. of Automated Reasoning **2** (1986) 283–286.
2. Baader F.: Unification in Varieties of Idempotent Semigroups, Semigroup Forum **36** (1987) 127–145.
3. Birjukov A.P.: Varieties of Idempotent Semigroups, Algebra i Logika **9** (1970) 255–273 (Russian).
4. Fennemore C. F.: All Varieties of Bands I, II, Math. Nachr. **48** (1971) 237–252, 253–262.
5. Gerhard J. A.: The Lattice of Equational Classes of Idempotent Semigroups, J. Algebra **15** (1970) 195–224.
6. Kapur D., Narendran P.: Complexity of Unification Problems with Associative-commutative Operators, J. of Automated Reasoning **9**, No.2, (1992) 261–288.
7. Klíma O.: Complexity of Unification and Matching Problems in the Varieties of Idempotent Semigroups, `http://www.math.muni.cz/~klima/Math/publications.html`.
8. Klíma O., Srba J.: Matching Modulo Associativity and Idempotency is NP-complete, In Proceedings of MFCS'00, Vol. 1893 of LNCS (2000) 456–466.
9. Schmidt-Schauss M.: Unification under Associativity and Idempotence is of Type Nullary, J. of Automated Reasoning **2** (1986) 277–281.

On the Complexity of Bounded Second Order Unification

Manfred Schmidt-Schauß

Fachbereich Informatik, Johann Wolfgang Goethe-Universität,
Postfach 11 19 32, D-60054 Frankfurt,Germany
Tel: (+49)69-798-28597, Fax: (+49)69-798-28919,
E-mail: `schauss@ki.informatik.uni-frankfurt.de`
URL: `www.ki.informatik.uni-frankfurt.de/`

1 Overview

It is well-known that first order unification is decidable in linear time (for further information see the overview article [BS94]) whereas second order (and higher-order) unification is undecidable [Gol81]. This result was sharpened in [Far91] for a restricted signature and in [Lev98, LV00] for severe restrictions on occurrences of second order variables.

Bounded second order unification (BSOU) is second order unification under the restriction that only a bounded number of holes in the instantiating terms for second order variables is permitted, however, the size of the instantiation is not restricted.

We report on work in progress and argue that the decision algorithm in [SS99] for bounded second order unification can be modified such that it runs in nondeterministic polynomial time. Together with the result that it is $\mathcal{NP}$-hard, this show that bounded second order unification is $\mathcal{NP}$-complete.

This also shows that monadic second order unification (MSOU), which was also treated in [Far88] is $\mathcal{NP}$-complete. Since MSOU can be easily encoded as word unification, this compares favorably to the best known upper bounds NEXPTIME [Gut98], and PSPACE [Pla99] for word unification.

This supports the claim that bounded second order unification is easier than context unification, whose decidability is currently an open question.

There are two key results to obtain this sharp bound for the complexity of bounded second order unification:
One is the result on the exponential upper bound of the exponent of periodicity for a minimal unifier of a word-unification problem and of a context unification problem, respectively [Mak77, KP96, SSS98]. This upper bound can be represented in polynomial space.
The other key result is in a paper of Plandowski [Pla94], where it is shown that the following equality test is polynomial: Given two context free grammars G_1, G_2 where $L(G_1) = \{t_1\}, L(G_2) = \{t_2\}$. The question whether $t_1 = t_2$ can be answered in polynomial time depending on the size of the grammar (not the words). This for example shows that the equality test of two words with nested natural exponents can be done in polynomial time. Note that the represented words may be of exponential size.

For example, this allows a polynomial test for checking the validity of

$$((ab)^{1001}a(ba)^{2001}b)^3 = ((aba(ba)^3b(ab)^4)^7)^{143}.$$

The idea for applying Plandowskis theorem is to encode the powers as a series of non-terminals. E.g. a^{15} would be encoded as $E_1 = a, E_2 = E_1E_1, E_3 = E_2E_2, E_4 = E_3E_3, E = E_4E_3E_2E_1$.

The exploitation of similar ideas led to a PSPACE upper bound for stratified context unification [SS01b].

2 Modifications of the Algorithm and Complexity

Bounded second order unification is the following question: Given equations $s_i \doteq t_i, i = 1, \ldots, m$ where s_i, t_i are second order terms, and an integer n, are the equations $s_i \doteq t_i, i = 1, \ldots, m$ unifiable, under the restriction that for every second order variable X, the sum of the occurrences of arguments used in the instantiations of X in a unifier is bounded by n.

As shown in [SS99], this can easily be transformed in non-deterministic polynomial time into a similar question, where the condition is sharpened such that every second order variable has arity one and the argument is used at most once in the instantiations. The second order variables under this restriction are called *Z-context variables* (zero or one instantiation).

The main modification of the algorithm in [SS99] is to permit a kind of nested powers: like $(f([], a)^4 g(b, []))^3)^2$. This avoids the explicit generation of large terms and in particular an exponential explosion of space usage. Instead it represents large terms implicitly. The price to pay is an algorithm with more rules which, unfortunately, are also more complex.

Furthermore there are two kinds of equations, term equations, and context equations $U := S$, where U is a context variable, and the context equations are only used for sharing purposes. There is also sharing on the term level, such that every subterm is represented by a first order variable. Instead of simple powers we permit expressions $pow(U, n_1, n_2)$, where $n_2 \geq n_1$ are nonnegative integers and U is a context variable for which there is a sharing equation. The reason is that several rules are easier to express using the generalized power expressions.

There are basically three situations for the non-deterministic algorithm in [SS99]:

1. Every term in an equation is either a variable or the top symbol is a Z-context variable. This is a presolved form, and the set of equations is unifiable. This situation is the same for the new algorithm.
2. There are so-called surface cycles like $X(a) \doteq y_1, y_1 \doteq f(x, y), y \doteq X(b)$. In this case one possibility of the algorithm is to replace X by $f(x, [])^k$, where k is smaller than the upper bound for the exponent of periodicity for the input problem. The modified algorithm adds $U_1 := f(x, []), U_2 := pow(U_1, 0, k)$. The equations above are replaced by $pow(U_1, 0, k)(a) \doteq y_1, y_1 \doteq f(x, y), y = pow(U_1, 0, k)(b)$.
   Note that there may be longer surface cycles and that there are further selection possibilities.
3. There are no surface cycles. Then it is necessary e.g. to make a generalized decomposition of terms, in particular those that have power-expressions at the top. The theorem of Plandowski is of help here to show that the macro decomposition steps can be done in non-deterministic polynomial time. This means, there is a guessing step, and a subsequent test that requires polynomial time. If we only used a top-down, single-step decomposition as in [SS99], then the number of steps would be exponential in the worst case.

Let us make an example for this decomposition macro step. The macro step may require a variable identification before application. Assume there are equations

$$
\begin{aligned}
&y_1 \doteq pow(U_1, 0, 1000)(x_1), y_1 \doteq pow(U_3, 1, 2000)(x_2), y_1 \doteq X(x_3),\\
&\ldots,\\
&U_1 := pow(U_2, 2, 100), U_2 := f(x_5, [])(g([], x_6)\\
&U_3 := pow(U_4, 0, 200), U_4 := g([], x_6)f(x_5, [])
\end{aligned}
$$

Then a guessing step consists of guessing the depth up to which the same path can be used, and then of guessing the different continuations of the path. The representation of the guess may be as follows:

$$X \to pow(U_3, 1, 500)f([], x_7)X'$$

Then the following algorithmic steps have to be performed:
A test whether the paths are indeed the same. I.e., whether $pow(U_1, 0, 499)$ and $pow(U_3, 1, 500)$ are the same contexts. Here the polynomial upper bound on the time complexity can be used.
The second step is to throw away the start of the path, and to compute the tail. The interesting part of the resulting system is:

$$
\begin{aligned}
&y_2 \doteq pow(U_1, 499, 1000)(x_1), y_2 \doteq pow(U_3, 500, 2000)(x_2),\\
&y_2 \doteq f(x_8, x_7), x_8 \doteq X'(x_3), \ldots
\end{aligned}
$$

The next decomposition step will reveal that $x_8 = x_5$.

Further changes to the algorithm are more or less straightforward adaptions to the modified data structures.

3 Conclusion

The claim is that bounded second order unification is $\mathcal{NP}$-complete based on extending the decision algorithm by nested power expressions and using Plandowskis result.

The author is sure that the same method of modifying the decision algorithm in [SS01a] will prove that stratified context unification is in $\mathcal{NP}$.

References

[BS94] Franz Baader and Jörg Siekmann. Unification theory. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 41–125. Oxford University Press, 1994.

[Far88] W.A. Farmer. A unification algorithm for second order monadic terms. *Annals of Pure and Applied Logic*, 39:131–174, 1988.

[Far91] W.A. Farmer. Simple second-order languages for which unification is undecidable. *J. Theoretical Computer Science*, 87:173–214, 1991.

[Gol81] W.D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

[Gut98] C. Gutierrez. Satisfiability of word equations with constants is in exponential space. In *Proceedings FOCS'98*, pages 112–119, Palo Alto, California, 1998. IEEE Computer Society Press.

[KP96] Antoni Kościelski and Leszek Pacholski. Complexity of Makanin's algorithms. *Journal of the Association for Computing Machinery*, 43:670–684, 1996.

[Lev98] Jordi Levy. Decidable and undecidable second order unification problems. In *Proceedings of the 9th Int. Conf. on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 47–60, 1998.

[LV00] Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Information and Computation*, 159:125–150, 2000.

[Mak77] G.S. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, 32(2):129–198, 1977.

[Pla94] W. Plandowski. Testing equivalence of morphisms in context-free languages. In *ESA 94*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470, 1994.

[Pla99] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *FOCS 99*, pages 495–500, 1999.

[SS99] Manfred Schmidt-Schauß. Decidability of bounded second order unification. Technical Report Frank-report-11, FB Informatik, J.W. Goethe-Universität Frankfurt am Main, 1999. submitted for publication.

[SS01a] Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. *Journal of Logic and Computation*, 2001. accepted for publication.

[SS01b] Manfred Schmidt-Schauß. Stratified context unification is in pspace. In *Proceedings of CSL'01*, 2001. to appear.

[SSS98] Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *Proceedings of the 9th Int. Conf. on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 61–75, 1998.

Approximating E-Unification

Christopher Lynch and Barbara Morawska

Department of Mathematics and Computer Science Box 5815,
Clarkson University, Potsdam, NY 13699-5815, USA,
E-mail: clynch@clarkson.edu, morawskb@clarkson.edu **

Equational logic commonly arises in deductive problems. A particularly important problem is to decide whether there is a substitution that makes a given goal true in an equational theory, or to find a set of substitutions that will generate all possible solutions. This it the problem of *E-unification* [1]. The problem arises in automated deduction, formal verification, type inference and many other areas of computer science. Often deductions are performed modulo a given equational theory, and calls to an E-unification procedure are made constantly through a deductive process.

Unfortunately, E-unification is undecidable in general, even for the problem of only deciding whether a solution exists. However, even when it is decidable, it may have very high complexity. Therefore, it would be useful to have some way of reducing the number of calls to the procedure. One way of doing this would be to approximate E-unification. By this we mean to use an efficient algorithm for deciding E-unification that is complete but not sound. This means that if it says that the E-unification problem is not satisfiable, then we can be sure of the answer. If it says it is satisfiable, we must then run a sound E-unification procedure to determine if that answer is correct. A deductive procedure could then call the approximating algorithm before calling the real algorithm, and avoid many unnecessary calls to the real procedure.

In this paper, we give such an approximating E-unification algorithm. The running time of the algorithm is linear if the equational theory is considered constant, and cubic if the equational theory is considered as part of the input. Our algorithm will approximate any E-unification problem for any equational theory.

What we exactly do in this paper is to show that E-unification is decidable for equational theories such that each equation has no repeated variables and the goal has no repeated variables, with the complexity given above. This class of theories contains all ground equations, but it is larger than that. It is a somewhat restrictive class of theories and has not been studied. However, it is especially interesting for the purpose of approximating E-unification.

The way the approximation works is to take each equation in the goal and the equational theory and rename the variables so that no variable occurs more than once. If this new E-unification problem is unsatisfiable, then the original one is also. In many cases this gives a helpful result, but not in all cases, such as collapse theories.

In order to create the decision procedure, we first create a sound and complete goal-directed inference procedure for the case where the equational theory is linear and the goal contains no repeated variables. The inference rules given for this procedure are similar to the inference rules for Syntactic Theories [4, 6, 7], but in our case we do not require that the theory is syntactic. An interesting feature of this inference system is that we can prove the completeness with Eager Merging. The problem of proving completeness with Eager Merging (or Eager Variable Elimination) is an open problem for many inference

** This work was supported by NSF grant number CCR-9712388 .

systems [2, 3, 5, 8]. Since so little is known about it, it is interesting to have a completeness result with Eager merging, even for such a restricted inference system. The only procedure known to us where Eager Merging has been proved to preserve completeness is given in [11]. Eager Merging is important in practice, because it gives some determinism to a don't-know nondeterministic procedure.

In the conclusion of the paper, we show that our approximation technique is not only useful for E-unification. It is also useful for Logic Programming, where we must decide if there is a substitution that can be applied to the goal so that the goal follows from the given Horn Clauses. We get similar nice complexity results for this case. Our techniques have some similarities to what is done for Local Theories [10], but they are only concerned with universal goals, and we can cover existential goals.

The full version of the paper is at [9].

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge, 1998.
2. J. Gallier and W. Snyder. A general complete E-unification procedure. In *RTA 2*, ed. P. Lescanne, LNCS vol. 256, 216-227, 1987.
3. J. Gallier and W. Snyder. Complete sets of transformations for general E-unification. In *TCS*, vol. 67, 203-260, 1989.
4. C. Kirchner. Computing unification algorithms. In *Proceedings of the First Symposium on Logic in Computer Science*, Boston, 200-216, 1990.
5. C. Kirchner and H. Kirchner. *Rewriting, Solving, Proving.* `http://www.loria.fr/~ckirchne/` , 2000.
6. C. Kirchner and F. Klay. Syntactic Theories and Unification. In *LICS 5*, 270-277, 1990.
7. F. Klay. Undecidable Properties in Syntactic Theories. In *RTA 4*,ed. R. V. Book, LNCS vol. 488, 136-149, 1991.
8. C. Lynch and B. Morawska. Goal Directed E-Unification. To appear at RTA 2001.
9. C. Lynch and B. Morawska. Approximating E-Unification `http://www.clarkson.edu/~clynch/papers/approx_full.ps/`, 2000.
10. D. McAllester. Automated Recognition of Tractability in Inference Relations. In *Journal of the ACM*, vol.40(2) , pp. 284-303, 1993.
11. A. Middeldorp, S. Okui, T. Ida. Lazy Narrowing: Strong Completeness and Eager Variable Elimination. In *Theoretical Computer Science* 167(1,2), pp. 95-130, 1996.

Non-Structural Subtype Entailment in Automata Theory

Joachim Niehren and Tim Priesnitz

Programming Systems Lab, Universität des Saarlandes, Saarbrücken, Germany
www.ps.uni-sb.de/~{niehren,tim}

Summary. Decidability of non-structural subtype entailment is a long standing open problem in programming language theory. In this paper, we apply automata theoretic methods to characterize the problem equivalently by using regular expressions and word equations. This characterization induces new results on non-structural subtype entailment, constitutes a promising starting point for further investigations on decidability, and explains for the first time why the problem is so difficult. The difficulty is caused by implicit word equations that we make explicit.

1 Introduction

Subtyping is a common concept of many programming languages (including C++ and Java). A subtype relation $\tau' \leq \tau$ means that all functions in a program that expect an argument of type τ are sufficiently polymorphic so that they can also be applied to values of the subtype τ'. Thus, one can safely replace values of a type τ by values of subtype τ'.

Subtype constraints are systems of inequations $t \leq t'$ that talk about the subtype relation. Terms t and t' in subtype constraints are built from type variables and type constructors. Two logical operations on subtype constraints were investigated: satisfiability and entailment [1, 4, 6, 11, 16, 18]. *Subtype satisfiability* can be checked in cubic time for many type languages. A quadratic time algorithm for the variable free case is presented in [9].

Interest in *subtype entailment* was first raised by practical questions on type inference engines with subtyping [5, 17, 19]. The efficiency of such systems relies on the existence of powerful simplification algorithms for typings. Such operations can be formulated on the basis of subtype entailment algorithms.

It then turned out that subtype entailment is a quite complex problem, even for unexpressive type languages where types are ordinary trees. Rehof and Henglein [20] clarified the situation for structural subtyping. This is a tree ordering that relates trees of the same shape only. It is induced by lifting an ordering on constants. If trees are built over the signature $\{int, real, \times, \rightarrow\}$, for instance, then structural subtyping is induced by the usual axiom $int \leq real$ which says that every integer is a real number. Rehof and Henglein showed that structural subtype entailment is coNP-complete [7] for finite trees (simple types) and PSPACE-complete [8] for possibly infinite trees (recursive types).

Subtyping becomes *non-structural* if the constants $\bot$ and $\top$ are admitted that stand for the least and greatest type. Now, trees of different shapes can be related since all trees τ satisfy $\bot \leq \tau \leq \top$. Several cases are to be distinguished: one can consider only *finite* trees or admit *infinite* trees, or one may assume that all function symbols are *co-variant* (such as $\times$) or that some are *contra-variant* (as the function type constructor $\rightarrow$ in its first argument). One can also vary the number of type constructors of each *arity*.

Decidability of non-structural subtype entailment (NSSE) is a prominent open problem in programming language theory. Only a PSPACE lower bound is known which holds in both cases, for finite trees and for infinite trees [8]. The signature $\{\bot, f, \top\}$ is enough to prove PSPACE hardness if f is a type constructor of arity at least 2. But this result does not explain why finding a decision procedure for NSSE is so difficult[1]. On the other hand, only a fragment of NSSE could be proved decidable [14] (and PSPACE-complete).

The idea behind the approach is to first reformulate Rehof and Henglein's approach for structural subtype entailment in automata theory and second to lift it to the non-structural case. We have carried out both steps successfully but report only on the second step. It leads us to a new characterization of NSSE that uses regular expressions and word equations [10, 15]. Word equations raise the real difficulty behind NSSE since they spoil the usual pumping arguments from automata theory. They also clarify why NSSE differs so significantly from seemingly similar entailment problems [12, 13].

We present the new characterization of NSSE in the remainder of this introduction. Let R_1 and R_2 range over regular expressions over a finite alphabet $\mathbf{A}$. We next introduce a new class of set expressions E over $\mathbf{A}$ that we call *cap set expressions.*

$$E ::= R_1 R_2^\circ \mid E_1 \cup E_2$$

Cap set expressions denote sets of words over $\mathbf{A}$ (that we call *cap sets*). They subsume regular expressions R through $R\varepsilon^\circ$. To interpret cap set expression, we must define a set S° for sets $S \subseteq \mathbf{A}^*$. Let pr be the prefix operator lifted to sets of words. We then set:

$$S^\circ \;=\; \{\pi \mid \pi \in pr(\mu^*),\ \mu \in S\}$$

A word π belongs to S° if π is a prefix of a power $\mu \dots \mu$ of some word $\mu \in S$. If S is infinite, then the $^\circ$ operator becomes problematic: there exists a regular expression R for which the cap set described by R° is neither regular nor context free. Consider for instance $(21^*)^\circ$ which denotes the set of all prefixes of words $21^n\, 21^n \dots 21^n$ where $n \geq 0$. Clearly this set is not context-free. We need to derive appropriate *restrictions* on cap set expressions such that following theorem becomes true.

Theorem 1 (Characterization). *The decidability of NSSE for a signature $\{\bot, f, \top\}$ with a single function symbol of arity $n \geq 1$ is equivalent to the decidability of the universality problem for the class of restricted cap set expressions over the alphabet $\{1, \dots, n\}$.*

The theorem implies that all variants of NSSE with a single function symbol of arity $n \geq 2$ are equivalent. It does not even matter whether finite or infinite trees are considered.

The theorem can also be used to relate NSSE to word equations. The idea is to express membership in cap sets in the positive existential fragment of word equations with regular constraints [21]. The reduction can easily be based on the following lemma that is well known in the field of string unification.

Lemma 1. *For all words $\pi \in \mathbf{A}^*$ and nonempty words $\mu \in \mathbf{A}^+$ it holds that $\pi \in pr(\mu^*)$ if and only if $\pi \in pr(\mu\pi)$. Thus, all sets $S \subseteq \mathbf{A}^+$ of nonempty words satisfy:*

$$\pi \in S^\circ \leftrightarrow \exists\mu\exists\nu\ (\mu \in S \wedge \pi\nu{=}\mu\pi)$$

[1] A recent approach accepted for publication at the 2001 conference on Principles of Programming Languages failed and was retracted before publication (personal communication with the authors).

We can thus express the universality problem of cap set expressions E in the positive $\forall\exists^*$ fragment of the first-order theory of word equations with regular constraints. Unfortunately, even the positive $\forall\exists^3$ fragment of a single word equation is undecidable [3] except if the alphabet is infinite [2] or a singleton [22]. Therefore, it remains open whether NSSE is decidable or not. But it becomes clear that the difficulty is raised by word equations hidden behind cap set expressions R°, i.e. the equation $\pi\nu{=}\mu\pi$ in Lemma 1.

Theorem 1 constitutes a promising starting point to further investigate decidability of NSSE. For instance, we can infer a new decidability result for a special case where $n = 1$: NSSE is decidable for the signature $\{\bot, f, \top\}$ if f is a unary type constructor.

An extended version of this abstract is available at `www.ps.uni-sb.de/papers`.

2 Non-Structural Subtype Constraints

We investigate non-structural subtype constraints over signatures of function symbols $\Sigma = \{\bot, f, \top\}$ with a single non-constant function symbol f that is co-variant. We write ar_g for the *arity* of a function symbol $g \in \Sigma$, i.e $\mathsf{ar}_\bot = \mathsf{ar}_\top = 0$ and $\mathsf{ar}_f \geq 1$.

The choice of such signatures imposes two restrictions: first, we do not allow for contravariant type constructors. These could easily be covered in our framework but would impose an additional burden to the presentation. Second, we do not treat larger signature with more than one non-constant function symbol. This is a true restriction that cannot easily be circumvented.

2.1 Words, Trees, and Non-Structural Subtyping

A *word* over an alphabet $\mathbf{A}$ is a finite sequence of letters in $\mathbf{A}$. We denote words by π, ν, or μ and the set of words over $\mathbf{A}$ with $\mathbf{A}^*$. The *empty word* is written as ε and the free-monoid *concatenation* of words π and μ by juxtaposition $\pi\mu$, with the property that $\varepsilon\pi = \pi\varepsilon = \pi$. A *prefix* of a word π is a word μ for which there exists a word ν such that $\pi = \mu\nu$. If μ is a prefix of π then we write $\mu \leq \pi$ and if μ is a proper prefix of π then we write $\mu < \pi$.

We next define finite and infinite trees over Σ. We consider trees as partial functions $\tau : \mathbb{N}^* \rightsquigarrow \Sigma$ which map words over natural numbers to function symbols. A tree τ is *finite* if its domain D_τ is finite and otherwise *infinite*. The words in $D_\tau \subseteq \mathbb{N}^*$ are called the *nodes* or *paths* of the tree. The idea is to identify a node with the path that addresses it relative to the root. We require that every tree has a root $\varepsilon \in D_\tau$ and that tree domains D_τ are always prefix closed and arity-consistent. The latter means for all trees τ, nodes $\pi \in D_\tau$, and naturals $i \in \mathbb{N}$ that $\pi i \in D_\tau$ if and only if $1 \leq i \leq \mathsf{ar}_{\tau(\pi)}$.

We will freely interpret function symbols in Σ as tree constructors. To make clear distinctions, we will write $=_\Sigma$ for equality of symbols in Σ and $=$ for equality of trees over Σ. Given $g \in \Sigma$ and trees $\tau_1, \ldots, \tau_{\mathsf{ar}_g}$ we define $\tau = g(\tau_1, \ldots, \tau_{\mathsf{ar}_g})$ by $\tau(\varepsilon) =_\Sigma g$ and $\tau(i\pi) =_\Sigma \tau_i(\pi)$ for all $\pi \in D_{\tau_i}$ and $1 \leq i \leq \mathsf{ar}_g$. We thus consider ground terms over Σ as (finite) trees, for instance $f(\bot, \top)$ or $\bot$. Thereby, we have overloaded our notation since a constant $a \in \Sigma$ can also be seen as tree $\varepsilon \mapsto a$. But this should never lead to confusion.

Let $<_\Sigma$ be the irreflexive partial order on Σ that satisfies $\bot <_\Sigma f <_\Sigma \top$ and $\leq_\Sigma$ its reflexive counterpart. We define *non-structural subtyping* to be the unique partial order on trees which satisfies for all trees τ_1, τ_2 over Σ:

$$\tau_1 \leq \tau_2 \quad \text{iff} \quad \tau_1(\pi) \leq_\Sigma \tau_2(\pi) \text{ for all } \pi \in D_{\tau_1} \cap D_{\tau_2}$$

2.2 Constraint Language

We assume an infinite set of tree valued variables that we denote by x, y, z, u, v, w. A *subtype constraint* φ is a conjunction of literals with the following abstract syntax where $n = \mathsf{ar}_f$:

$$\varphi, \varphi' ::= x \leq f(y_1, \ldots, y_n) \mid f(y_1, \ldots, y_n) \leq x \mid x = \bot \mid x = \top \mid \varphi \wedge \varphi'$$

We interpret constraints φ in the structure of trees over Σ with non-structural subtyping. We distinguish two cases, the structure of finite trees or else of possibly infinite trees. We interpret function symbols in both cases as tree constructors and the predicate symbol $\leq$ by the non-structural subtype relation. Again, this overloads notation: we use the same symbol $\leq$ for the subtype relation on trees and the predicate symbol denoting the subtype relation in constraints. Again, this should not raise confusion.

Note that we do not allow for formulas $x \leq y$ in our constraint language. This choice will help us to simplify our presentation essentially. It is, however, irrelevant from the point of view of expressiveness. We can still express $x \leq y$ by using existential quantifiers:

$$x \leq y \leftrightarrow \exists z \exists u \; (f(x, u, \ldots, u) \leq z \wedge z \leq f(y, u, \ldots, u))$$

As in this equivalence, we will sometimes use first-order formulas Φ built from constraints and the usual first-order connectives. We will write V_Φ for the set of free variables occurring in Φ. A solution of Φ is a variable assignment α into the set of finite (resp. possibly infinite) trees which satisfies the required subtype relations; we write $\alpha \models \Phi$ if α solves Φ and say that Φ is *satisfiable*.

Example 1. The constraint $x \leq f(x)$ is satisfiable, even when interpreted over finite trees. We can solve it by mapping x to $\bot$. In contrast, the equality constraint $x \leq f(x) \wedge f(x) \leq x$ is unsatisfiable over finite trees. It can however be solved by mapping x to the infinite tree $f(f(f(\ldots)))$.

A formula Φ_1 *entails* Φ_2 (we write $\Phi_1 \models \Phi_2$) if all solutions $\alpha \models \Phi_1$ satisfy $\alpha \models \Phi_2$. We will consider entailment judgments that are triples of the form (φ, x, y) that we write as $\varphi \models^? x \leq y$. *Non-structural subtype entailment* (NSSE) for Σ is the problem to check whether entailment $\varphi \models x \leq y$ holds for a given entailment judgment $\varphi \models^? x \leq y$.

Example 2. The prototypical example where NSSE holds somehow surprisingly is:

$$x \leq f(y) \wedge f(x) \leq y \models^? x \leq y \qquad \text{(yes)}$$

To see this, note that all finite trees in the unary case are of the form $f \ldots f(\bot)$ or $f \ldots f(\top)$. Thus, $x \leq y \vee y < x$ is valid in this case. Next let us contradict the assumption that there is a solution $\alpha \models y < x \wedge x \leq f(y) \wedge f(x) \leq y$. Transitivity yields $\alpha(y) \leq f(\alpha(y))$ and then also $f(\alpha(x)) \leq f(\alpha(y))$. Hence $\alpha(x) \leq \alpha(y)$ which contradicts $\alpha(y) < \alpha(x)$.

Note that entailment judgments of the simple form $\varphi \models^? x \leq y$ can express more general entailment judgments, where both sides are conjunctions of inequations $t_1 \leq t_2$ between nested terms or variables. The main trick is to replace a judgment $\varphi \models^? t_1 \leq t_2$ with terms t_1 and t_2 by $\varphi \wedge x = t_1 \wedge y = t_2 \models^? x \leq y$ where x and y are fresh variables. Note also that the omission of formulas $u \leq v$ on the left hand side does not restrict the problem.

References

1. R. M. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, September 1993.
2. F. Baader and K. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In *Journal of Symbolic Computation*, volume 21, pages 211–243, 1996.
3. V. Durnev. Unsolvability of positive $\forall\exists^3$-theory of free groups. In *Sibirsky mathematichesky jurnal*, volume 36(5), pages 1067–1080, 1995. In Russian, also exists in English translation.
4. J. Eifrig, S. Smith, and V. Trifonow. Sound polymorphic type inference for objects. In *ACM Conference on Object-Oriented Programming: Systems, Languages, and Applications*, 1995.
5. J. Eifrig, S. Smith, and V. Trifonow. Type inference for recursively constrained types and its application to object-oriented programming. *Elec. Notes in Theoretical Comp. Science*, 1, 1995.
6. Y. Fuh and P. Mishra. Type inference with subtypes. *Theoretical Computer Science*, 73, 1990.
7. F. Henglein and J. Rehof. The complexity of subtype entailment for simple types. In *Proceedings of the 12^{th} IEEE Symposium on Logic in Computer Science*, pages 362–372, 1997.
8. F. Henglein and J. Rehof. Constraint automata and the complexity of recursive subtype entailment. In *Proc. of the25^{th} Int. Conf. on Automata, Languages, & Programming*, LNCS, 1998.
9. D. Kozen, J. Palsberg, and M. I. Schwartzbach. Efficient recursive subtyping. In *In Proc. of the 20^{th} ACM Symp. on Principles of Programming Languages*, pages 419–428. ACM Press, 1993.
10. G. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, 32, 1977. (English translation).
11. J. C. Mitchell. Type inference with simple subtypes. *The Journal of Functional Programming*, 1(3):245–285, July 1991.
12. M. Müller, J. Niehren, and R. Treinen. The first-order theory of ordering constraints over feature trees. In *IEEE Symposium on Logic in Computer Science*, pages 432–443, 21-24 June 1998.
13. J. Niehren, M. Müller, and J.-M. Talbot. Entailment of atomic set constraints is PSPACE-complete. In *14^{th} IEEE Symposium on Logic in Computer Sience*, pages 285–294, 1999.
14. J. Niehren and T. Priesnitz. Entailment of non-structural subtype constraints. In *Asian Computing Science Conference*, LNCS, pages 251–265, Thailand, Dec. 1999. Springer-Verlag, Berlin.
15. W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proc. of the 40^{th} IEEE Symp. on Found. of Comp. Science*, pages 495–500. IEEE Comp. Society Press, 1999.
16. F. Pottier. Simplifying subtyping constraints. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, pages 122–133. ACM Press, New York, May 1996.
17. F. Pottier. A framework for type inference with subtyping. In *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming*, pages 228–238, Sept. 1998.

18. F. Pottier. *Type inference in the presence of subtyping: from theory to practice.* PhD thesis, Institut de Recherche d'Informatique et d'Automatique, 1998.
19. J. Rehof. Minimal typings in atomic subtyping. In *ACM Symposium on Principles of Programming Languages.* ACM Press, 1997.
20. J. Rehof. *The Complexity of Simple Subtyping Systems.* PhD thesis, DIKU, Copenhagen, 1998.
21. K. U. Schulz. Makanin's algorithm for word equations – two improvements and a generalization. In *Word Equations and Related Topics*, LNCS 572, pages 85–150. Springer-Verlag, 1991.
22. Y. Vazhenin and B. Rozenblat. Decidability of the positive theory of a free countably generated semigroup. In *Math. USSR Sbornik*, volume 44, pages 109–116, 1983.