Unif 2002
16th International Workshop on Unification

Abstracts

July 25–26, 2002
Copenhagen, Denmark
Affiliated with RTA2002, Part of FLoC'02.

Organizing Committee:
Christophe Ringeissen
Cesare Tinelli
Ralf Treinen
Rakesh M. Verma

Technical Report 02-05
Department of Computer Science
University of Iowa
Iowa City, Iowa 52242

Contents

The 16th International Workshop on Unification

UNIF 2002 is the 16th 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	Narrowing
Special Unification Algorithms	Matching
Higher-Order Unification	Combination Problems
Unification-Based Approaches to Grammar	Disunification
Constraint Solving	Unification Calculi
Type Checking and Reconstruction	Typed Unification
Applications	Implementations

The 16th International Workshop on Unification was held June 25 – 26, 2002, in Copenhagen, Denmark, as an affiliated workshop of the Conference on Rewriting Techniques and Applications (RTA 2002), and as part of the Federated Logic Conference (FLoC'02).

This technical report collects extended abstracts of the talks presented at UNIF 2002 and the abstracts of three invited talks given by Claude Kirchner, on "Abstract Canonical Inference Systems", Michael Kohlhase, on "Sorted Higher-Order Unification", and Christopher Lynch, on "Decidability and Complexity of E-unification for some classes of equational theories."

The organizing committee wishes to thank the invited speakers, the RTA steering committee and the RTA 2002 program committee for hosting UNIF 2002, and to thank the FLoC'02 organizing committee for their cooperation and their excellent support throughout the preparation of this meeting.

Christophe Ringeissen, LORIA Nancy, France
Cesare Tinelli, University of Iowa, USA
Ralf Treinen, Université Paris-Sud, France
Rakesh M. Verma, University of Houston, USA

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

Sorted Higher-Order Unification

Michael Kohlhase

School of Computer Science Carnegie Mellon University
E-mail: `Michael_Kohlhase@asuka.mt.cs.cmu.edu`

Abstract

In recent years, many theorem proving systems have been extended to higher-order logic (in particular based on variants of the simply typed lambda calculus), not the least, since the built-in notion of beta-eta reduction gives a convenient and transparent definition mechanism. The drawback of this approach is that the main computational device (higher-order unification, which is undecidable in general), becomes a major bottleneck.

The main assumption in this talk is that the introduction of additional knowledge into higher-order unification will make the search for higher-order unifiers (and thus for proofs in higher-order logic) more guided and ultimately yield more efficient higher-order theorem provers. The source of knowledge that we want to explore in this talk is taxonomic information given in a concept hierarchy.

Concretely, we will introduce a sorted lambda-calculus where the universe is subdivided into various subsets (called sorts), and the signature is augmented by sort information. We introduce the technical device of term declarations, which give a very expressive mechanism for specifying (and using) sort information.

The central result will be a unification algorithm for this logic and some decidability results for the higher-order pattern fragment.

We will conclude the talk with a brief presentation of two applications of sorted higher-order unification outside theorem proving: natural language semantics and formula retrieval from knowledge bases.

Decidability of Bounded Higher-Order Unification

Manfred Schmidt-Schauß[1] & Klaus U. Schulz[2]

[1] Institut für Informatik, J.-W.-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

[2] CIS, University of Munich, Oettingenstr 67, D-80538 München, Germany
Tel: (+49)89-2178-2700, Fax: (+49)89-2178-2701,
E-mail: schulz@cis.uni-muenchen.de

Abstract. This is a sketch of work to prove the following result: unifiability of terms in the simply typed lambda calculus with β and η rules becomes decidable if there is a bound on the number of bound variables and lambdas in a unifier in η-long β-normal form.

1 Introduction

First-order unification [BS94] is a fundamental operation in several areas of computer science, e.g. automated deduction, term rewriting, logic programming and type-checking. The generalization to higher-order unification increases the expressiveness, the applicability and improves the level of abstraction. This explains the interest in various kinds of higher-order systems (e.g., [And86,Pau94,Pfe01,Bar90,Bir98,Mil91,HKMN95], [Nip91,Klo92,DJ90,Hue75], [Dow01]).

It is well-known that second-order unification — hence higher-order unification — is undecidable ([Gol81,Far91,LV00]).

In [SS99,SS01] it was shown that second-order unification becomes decidable if an upper bound on the number of occurrences of bound variables in the substitution terms is fixed, which has as a corollary the well-known result that second-order unification with monadic function symbols is decidable [Hue75,Zhe79,Far88].

We generalize the latter result to higher-order unification in the simply typed lambda calculus with β and η rules [Bar84,HS86]. We show that solvability of higher-order unification problems is decidable if for any variable a bound on the number of lambda-binders and occurrences of bound variables in the image of the variable under a unifier is given. Each term $\sigma(x)$ for a unifier σ is assumed to be in η-expanded β-normal form. Note that the bound does not imply a bound on the size of a unifier.

The result implies that undecidability proofs for higher-order unification require an unbounded number of lambda-bound variables or lambdas in a unifier in η-expanded β-normal form. It can be used to define a semi-decision procedure for ordinary higher-order unification where we start with a given bound b for the variables and lambdas in the unifier and increase b as long as we have an unsolvable problem. From a practical point of view, the bound E on the exponent of periodicity used in the decision algorithm also has to be increased iteratively, since E depends nonelementary on b.

The result obtained is a new and non-trivial decidability result for higher-order unification without any syntactic restrictions on the input problems. It is a generalization of the result in the second order case [SS01], where the bound on the lambdas can be omitted.

This result can also be seen as a parameterized decidability of higher-order unification in the sense of [DF99].

All details can be found in a technical report [SSS01].

There is recent result on higher-order matching and unification [Wie02], where a condition on the unifiers is used, which implies that there is a comutable upper bound on the size of unifiers.

2 A Sketch of the Ideas

Given a multi-set of higher order equations, instead of all unifiers, a size-mininmal unifier in η-long β-normal form is demanded. The input to the decision algorithm is a multi-set S of equations and a function $b : FV(S) \to \mathbb{N} \setminus \{0\}$. A ground substitution σ is a *unifier* of (S, b), iff it is in η-long β-normal form, and for every $x \in FV(S)$, the sum of the number of lambdas and occurrences of bound variables in $\sigma(x)$ is not greater than $b(x)$.

Similar to associative unification, minimality of a unifier implies a structural constraint. The so-called exponent of periodicity of a minimal unifier is bounded by a computable function of the size of the multi-set of equations. This means, there is a bound on the number n of repetitions C^n where $C \neq [\cdot]$ is a ground first-order context; where a first order context is built from function symbols that have only arguments of elementary type.

The algorithm for bounded second order unification in [SS01] can be used as a plan to construct the decision algorithm for bounded higher-order unification.

*Surface positions*are defined as positions in terms that are neither in the body of a lambda nor within an argument of a (flattened) application whose head is a (free) variable.

The main structure that guides the algorithm are cycles:

Definition 2.1. *A* cycle *in (S, b) is a sequence $s_1 \doteq t_1, \dots, s_h \doteq t_h$ of length $h \geq 1$ of equations, such that for all $1 \leq i \leq h$: $s_i \equiv x_i\ r_{i,1}\ \dots\ r_{i,m_i}$, and x_i occurs on the surface of $t_{i-1 \;\mathtt{mod*}\; h}$. There should be at least one term t_i of the form $f(t_{i,1}, \dots, t_{i,n})$ and at least one term s_i of the form $x_i\ r_{i,1}\ \dots\ r_{i,m_i}$ with* $\mathrm{ar}(x_i) \geq 1$.

The decision algorithm is based on a careful analysis and transformation of cycles in the multi-set of equations (S, b).

- If all equations in (S, b) are of the form $x \ldots \doteq y \ldots$, then the equations are trivially unifiable.
- If there is no cycle in (S, b), then imitation and decomposition are sufficient to make progress.
- If there is a cycle, and if "the minimal cycle" is ambiguous (there are different paths in the cycle), then there is a set of rules that make progress.
- If there is a cycle and if "the minimal cycle" is unique (there is a single path of the variables in the cycle), then either several rules make progress, or the bound on the exponent of periodicity helps to stop a nonterminating computation possibilitiy

The given algorithm is non-elementary, since the algorithm has to guess the exponent of periodicity, which is non-elementary in the size of the input. The problem itself is also non-elementary, since β-normalization can be encoded in a similar way as in [Wie99].

References

[And86] Peter Andrews. *An introduction to mathematical logic and type theory: to truth through proof.* Academic Press, 1986.

[Bar84] Henk P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* North-Holland, Amsterdam, New York, 1984.

[Bar90] Henk P. Barendregt. Functional programming and lambda calculus. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, chapter 7, pages 321–363. Elsevier, 1990.

[Bir98] Richard Bird. *Introduction to Functional Programming using Haskell.* Prentice Hall, 1998.

[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.

[DF99] Rodney.G. Downey and Michael R. Fellows. *Parametrized Complexity.* Springer, 1999.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, chapter 6, pages 243–320. Elsevier, 1990.

[Dow01] Gilles Dowek. Higher-order unification and matching. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 16, pages 1009–1062. North-Holland, 2001.

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

[HKMN95] M. Hanus, H. Kuchen, and J.J. Moreno-Navarro. Curry: A truly functional logic language. In *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, pages 95–107, 1995.

[HS86] J.Roger Hindley and Jonathan P. Seldin. *Introduction to combinators and λ-calculus*. Cambridge University Press, 1986.

[Hue75] Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[Klo92] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E.Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 2–116. Oxford University Press, 1992.

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

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables and simple unification. *J. of Logic and Computation*, 1(4):497–536, 1991.

[Nip91] Tobias Nipkow. Higher-order critical pairs. In *Proc. 6th IEEE Symp. LICS*, pages 342–349, 1991.

[Pau94] Lawrence C. Paulson. *Isabelle*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[Pfe01] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 17, pages 1063–1147. North-Holland, 2001.

[SS99] Manfred Schmidt-Schauß. Decidability of bounded second order unification. Frank report 11, FB Informatik, J.W. Goethe-Universität Frankfurt am Main, 1999. available at `http://www.ki.informatik.uni-frankfurt.de/papers/articles.html`.

[SS01] Manfred Schmidt-Schauß. Decidability of bounded second order unification, 2001. submitted for publication.

[SSS01] Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher order unification. Frank report 15, Institut für Informatik, J.W. Goethe-Universität Frankfurt am Main, 2001. also appeared as: Forschungsbericht, Centrum für Informations- und Sprachverarbeitung, Universitt München. The paper is available at `http://www.ki.informatik.uni-frankfurt.de/papers/articles.html`.

[Wie99] ToMasz Wierzbicki. Complexity of the higher-order matching. In *Proc. 16^{th} CADE*, Lecture Notes in Computer Science, pages 82–96. Springer-Verlag, 1999.

[Wie02] ToMasz Wierzbicki. A decidable variant of the higher order matching. In *Proc. RTA'02*, 2002. to appear.

[Zhe79] A.P Zhezherun. Decidability of the unification problem for second order languages with unary function symbols. *Kibernetika (Kiev)*, 5:120–125, 1979. Translated as *Cybernetics* 15(5): 735-741,1980.

Simple β_0-Unification for Terms with Context Holes

Makoto Hamana

Department of Computer Science, University of Gunma, Japan
hamana@cs.gunma-u.ac.jp

1 Introduction

Contexts are widely used notion in the lambda calculus and its application to computer science. Contexts are terms with some holes to be filled, and the difference between hole-filling and usual substitution is that filling a hole with a term can *capture* variable in the term by bound variable when the system has binding operators.

In this note we present a unification algorithm for terms involving both usual variables and holes. This algorithm is slightly simplified version of the one presented in the paper [Ham01]. An example of what we want to do is the following: consider the unification problem between lambda terms

$$\lambda x.M =^? \lambda x.f(x)$$

where M is a free variable. An intuitive answer $M \mapsto f(x)$ is not a unifier of the usual higher-order unification because of the properties of substitution avoiding capture of bound variable.

But we want to take this answer $M \mapsto f(x)$ as a unifier of the problem. If we regard M as *a hole of a context* filling by $f(x)$, capturing x by the binder is no problem. We consider this kind of a variant of higher-order unification between terms with holes modulo Miller's β_0-axiom [Mil91]. The β_0-axiom is defined by $(\lambda x.M)y = M[x := y]$, i.e. only an application with a variable causes substitution. This axiom is the basis of pattern unification [Mil91,Nip93].

The main benefit of our algorithm is decidability and simplicity. The unification problems we consider have variable (and hole) binding, whereas the algorithm is very similar to the first-order unification. The imitation and projection rules used in higher-order unification is no longer needed, the same effect can be done by a more simple rule taking just *inverse* of an intermediate substitution in a unification step.

The unification algorithm presented here have been used for a logic programming with variable binding [Ham01]. It is also hopeful to be a unification in the *calculi for contexts* [Mas99,HO01,SSK01] considered extensively in recent years, which are the lambda calculi extended with hole binding operator.

We should notice that the problem we consider is not *context unification* [Com92,SS94]. Context unification is a unification for terms with context variables (not holes) which are second-order variables only substituted by lambda terms with exactly one occurrence of the bound variable (meaning a hole), where holes themselves are not aimed to be unified.

2 Term Language

We will work with own term language to express the system having variable/hole-binding precisely. Terms are defined by

$$t ::= f(t_1, \ldots, t_n) \mid t@a \mid [a]t \mid \xi x \mid \mathsf{nam}(a)$$
$$\xi ::= [a_1 := a'_1, \ldots, a_n := a'_n]$$

where f is a function symbol, a's are *names* which are used for bindings. A term $t@a$ denotes an application, $[a]t$ an abstraction, $\mathsf{nam}(a)$ a name, and ξx a *variable with renamer* where ξ is an *renamer*. Types are defined by $\sigma ::= \iota \mid [\mathsf{N}]\sigma$ where ι is a base type and $[\mathsf{N}]\sigma$ is a type abstracted by a name. We use additional information of types, called stages, to control bound and free names occurring in a term. A type judgment is of the form $\Gamma \vdash t : \sigma : A$ where a typing context Γ is a finite set of variable declarations $\{x_1 : \sigma_1 : A_1, \ldots, x_n : \sigma_n : A_n\}$, σ is a *type*, and a *stage* A is a finite set of names expressing a possible free names occurring in a term. Well-typed terms are defined by the typing rules below.

$$\frac{A = \{\boldsymbol{a}\} \text{ and } B \supseteq \{\boldsymbol{b}\}}{\Gamma_1, x : \sigma : A, \Gamma_2 \vdash [\boldsymbol{a} := \boldsymbol{b}]x : \sigma : B} \qquad \frac{a \in A}{\Gamma \vdash \mathsf{nam}(a) : \iota : A}$$

$$\frac{\Gamma \vdash t_1 : \sigma_1 : A, \ldots, \Gamma \vdash t_n : \sigma_n : A}{\Gamma \vdash f(t_1, \ldots, t_n) : \sigma : A}$$

$$\frac{\Gamma \vdash t : \sigma : A}{\Gamma \vdash [a]t : [\mathsf{N}]\sigma : A - \{a\}} \qquad \frac{\Gamma \vdash t : [\mathsf{N}]\sigma : A}{\Gamma \vdash t@a : \sigma : A \cup \{a\}} \; a \notin \mathsf{BN}(t)$$

A *renaming* $\rho : A \to B$ is a function between sets of names presented by the notation $[a_1 \mapsto b_1, \ldots, a_n \mapsto b_n]$. A renaming ρ is extended to functions $\rho^\sharp$ and $\rho^\natural$ on terms from $T(\Gamma)$ to $T(\Gamma)$.

3 Unification

Next, we consider unification of terms modulo the equational theory generated by the following axioms:

$$(\alpha) \;\; \Gamma \vdash [a]t = [a'][a \mapsto a']^\sharp t : [\mathsf{N}]\sigma : A$$
$$(\beta_0) \;\; \Gamma \vdash ([a]t)@b = [a \mapsto b]^\sharp t : \sigma : A$$
$$(\eta) \;\; \Gamma \vdash [a](t@a) = t : [\mathsf{N}]\sigma : A.$$

and inference rules:

Rules for congruence

$$(\mathrm{Ax})\frac{\Gamma \vdash s = t : \sigma : A \in \{(\alpha), (\beta_0), (\eta)\} \cup E}{\Gamma \vdash s = t : \sigma : A}$$

$$(\text{Congr-}f)\frac{\Gamma \vdash t_i = t_i' : \sigma_i : A \quad f : \sigma_1, \ldots, \sigma_n \to \sigma \in \Sigma}{\Gamma \vdash f(t_1, \ldots, t_i, \ldots, t_n) = f(t_1, \ldots, t_i', \ldots, t_n) : \sigma : A}$$

$$(\text{Congr-}[])\frac{\Gamma \vdash s = t : \sigma : A}{\Gamma \vdash [a]s = [a]t : [\mathsf{N}]\sigma : A - \{a\}}\ a \in A$$

$$(\text{Congr-}@)\frac{\Gamma \vdash s = t : [\mathsf{N}]\sigma : A}{\Gamma \vdash s@a = t@a : \sigma : A \cup \{a\}}\ a \notin A$$

$$(\mathrm{Subst})\frac{\Gamma \vdash s : \sigma : A \quad \Gamma, x : \sigma : A \vdash t = t' : \sigma : B}{\Gamma \vdash [x \mapsto s]t = [x \mapsto s]t' : \sigma : B}$$

Rules for equivalence relation

$$(\mathrm{Ref})\frac{\Gamma \vdash t : \sigma : A}{\Gamma \vdash t = t : \sigma : A} \qquad (\mathrm{Sym})\frac{\Gamma \vdash s = t : \sigma : A}{\Gamma \vdash t = s : \sigma : A}$$

$$(\mathrm{Tr})\frac{\Gamma \vdash s = t : \sigma : A \quad \Gamma \vdash t = u : \sigma : A}{\Gamma \vdash s = u : \sigma : A}$$

Rules for variation

$$(\natural)\frac{\Gamma \vdash s = t : \sigma : A}{\rho\Gamma \vdash \rho^\natural s = \rho^\natural t : \sigma : \rho A}\ \rho \text{ is a 1-1 renaming}$$

Structural rules

$$(\mathrm{Wk})\frac{\Gamma \vdash s = t : \sigma : B}{\Gamma,\, x{:}\sigma{:}A \vdash s = t : \sigma : B} \qquad (\text{Wk-r})\frac{\Gamma \vdash s = t : \sigma : A}{\Gamma \vdash s = t : \sigma : A \cup \{a\}}\ a \notin \mathsf{BN}(s) \cup \mathsf{BN}(t)$$

$$(\text{Str-r})\frac{\Gamma \vdash s = t : \sigma : A}{\Gamma \vdash s = t : \sigma : A - \{a\}}\ a \notin \mathsf{FN}(s) \cup \mathsf{FN}(t)$$

Here, $\mathsf{FN}(t)$ and $\mathsf{BN}(t)$ denote sets of free and bound names respectively.

A *unification problem* is a pair $(\Gamma \vdash t : \tau : A,\ \Gamma' \vdash t' : \tau : A')$ of terms. A *unifier* to the unification problem is a pair (θ, ρ) satisfying $\Gamma'' \vdash \theta\rho^\natural t = \theta t' : \tau : A'$ where $\theta : \rho\Gamma \cup \Gamma' \to T(\Gamma'')$ is a substitution, ρ is a 1-1 renaming between the sets of names in the pair of terms.

A *standardized unification problem* is a unification problem satisfying the following properties: (i) The names in two terms are mutually disjoint. (ii) Γ and Γ' are mutually disjoint. (iii) $|A| = |A'|$. (iv) t and t' has no β_0, η-redexes. (v) The bound variables in t and t' are numbered by the method of de Bruijn level. (vi) The types of all variables are ι.

We can show that any unification problem can be a standardized unification problem, so it is sufficient to consider a unification algorithm for standardized unification problem. Also we can easily recover a unifier of the original problem from the unifier of standardized one. See detail [Ham01].

Unification algorithm BUA. The unification problem is extended to a pair of sequences of terms and a unifier is similarly defined. For a unification problem, a *system* is of the form $t_1{:}A_1 =^? t'_1{:}A'_1, \ldots, t_n{:}A_n =^? t'_n{:}A'_n$. The unification algorithm BUA on systems given in Fig. 1 computes unifiers for a given unification problem. This is based on Martelli and Montanari's unification algorithm for first-order terms by the set of transformations. In the rules (v-elim1) and

$$(f\text{-elim})\frac{G_1, f(s_1,\ldots,s_n) : A =^? f(t_1,\ldots,t_n) : B, G_2}{G_1, s_1 : A =^? t_1 : B, \ldots, s_n : A =^? t_n : B, G_2}$$

$$(\text{triv})\frac{G_1, \xi x : A =^? \xi x : A, G_2}{G_1, G_2} \qquad ([]\text{-elim})\frac{G_1, [k]s : A =^? [k]t : B, G_2}{G_1, s : A \cup \{k\} =^? t : B \cup \{k\}, G_2}$$

$$(\text{ov-elim})\frac{G_1, \mathsf{nam}(a) : A =^? \mathsf{nam}(b) : B, G_2}{\rho(G_1, s : A =^? t : B, G_2)} \qquad (@\text{-elim})\frac{G_1, s@a : A =^? t@b : B, G_2}{\rho(G_1, s : A =^? t : B, G_2)}$$

$$(\text{v-elim1})\frac{G_1, \xi x : B =^? t : A, G_2}{\theta\rho(G_1, G_2)} \qquad (\text{v-elim2})\frac{G_1, t : A =^? \xi x : B, G_2}{\theta\rho(G_1, G_2)}$$

In (v-elim1) and (v-elim2), $x \notin \mathsf{VAR}(t)$.

Fig. 1. Unification algorithm BUA

(v-elim2) of BUA, the following data is used:

- $x : \tau : D \in \Gamma$ ($\in \Gamma'$ in (v-elim2)),
- a bijective renaming $\rho : B \to A$, where $A = \langle a_1, \ldots, a_n \rangle$, $B = \langle b_1, \ldots, b_n \rangle$, is defined by $\rho(i) \triangleq i$.
- an renaming $\xi^\diamond : D \to B$,
- a substitution θ is arbitrarily taken from the set $\Theta = \{[x{:}\tau{:}D \mapsto s{:}\tau{:}D] \mid \langle s, E \rangle \in S,\ E \subseteq D\}$ (i.e. there are $\#\Theta$-different θ's) where

$$\begin{aligned} S \triangleq \{\langle t[p_1 := \xi'_1 \circ \rho^{-1}(t_{|p_1}), \ldots, p_n := \xi'_n \circ \rho^{-1}(t_{|p_n})], \bigcup_i \xi'_i B \rangle \\ \mid \{p_1, \ldots, p_n\} = \mathsf{FNPOS}(t) \text{ and } p_1 \neq \ldots \neq p_n, \\ \xi'_1, \ldots, \xi'_n \in \{\xi' : B \to D \mid \xi^\diamond \circ \xi' = \mathsf{id}_B\} \cup \{\mathsf{id}_B\}\}. \end{aligned}$$

Note that $\mathsf{FNPOS}(t)$ denotes the set of all positions of free names occurring in t, $t[p := s]$ is a replacement of a term t at the position p with a term s, and $t_{\restriction p}$ denotes a subterm of t at the position p. Also (v-elim1) and (v-elim2) produce $\#\Theta$-different branches of derivation.

Theorem 1. *(***[Ham01]***) The algorithm BUA is sound, complete and decidable, and the unifiers are $\beta_0\eta$-normalized.*

Not all pattern unification problems can be solved by BUA because our term language only admits a restricted form of application (i.e. $([a]t)@b$, b is a name). However, the syntactic restriction of unification problems for BUA is imposed by just well-typedness, namely BUA can solve *non-pattern* unification problems if they are well-typed. The following is an example of such a problem and solving steps by BUA.

Example. Consider a non-pattern unification problem $\lambda a.Faa =^? \lambda d.gd$ where F is a free variable. The right-hand side is not a higher-order pattern because the bound variable d is applied twice to the free variable. Corresponding unification problem in our syntax is

$$(F{:}[\mathsf{N}][\mathsf{N}]\iota{:}\varnothing \vdash [a]F@a@a : [\mathsf{N}]\iota : \varnothing, \quad \vdash [d]g(\mathsf{nam}(d)) : [\mathsf{N}]\iota : \varnothing).$$

The first step is standardization. The higher-order variable F is replaced with $[b][c]F'$ using the first-order variable F' and the outermost bound name is de Bruijn indexed: $[1]([b][c]F')@1@1 : \varnothing =^? [1](g(\mathsf{nam}(1))) : \varnothing$. β_0-normalization to the left-hand side gives

$$[1]([b := 1, c := 1]F') : \varnothing =^? [1](g(\mathsf{nam}(1))) : \varnothing.$$

Stripping the binder [1], we can apply (v-elim1)-rule:

$$\frac{([b := 1, c := 1]F') : \{1\} =^? (g(\mathsf{nam}(1))) : \{1\}}{F' \mapsto g(\mathsf{nam}(b)), \quad F' \mapsto g(\mathsf{nam}(c)).}$$

So the unifiers are $F \mapsto [b][c]g(\mathsf{nam}(b))$ and $F \mapsto [b][c]g(\mathsf{nam}(c))$.

In this (v-elim) step, these answers are obtained by applying all possible *inverse* substitutions computed in the set S of BUA (this case is $[1 \mapsto b]$ and $[1 \mapsto c]$ for $[b := 1, c := 1]$) to the right-hand side. This is the crucial point why BUA can solve non-pattern unification problem by very simple way.

References

[Com92] H. Comon. Completion of rewrite systems with membership constraints. In *Coll. on Automata, Languages and Programming, LNCS 623*, 1992.

[Ham01] M. Hamana. A logic programming language based on binding algebras. In N. Kobayashi and B. C. Pierce, editors, *4th International Symposium on Theoretical Aspects of Computer Software, TACS 2001*, volume 2215 of *Lecture Notes in Computer Science*, pages 243–262. Springer, 2001.

[HO01] M. Hashimoto and A. Ohori. A typed context calculus. *Theoretical Computer Science*, 266:249–271, 2001.

[Mas99] I. Mason. Computing with contexts. *Higher-Order and Symbolic Computation 12*, pages 171–201, 1999.

[Mil91] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[Nip93] T. Nipkow. Functional unification of higher-order patterns. In *Proc. 8th IEEE Symp. Logic in Computer Science,*, pages 64–74, 1993.

[SS94] M. Schmidt-Schauß. Unification of stratified second-order terms. Technical Report 12/94, J. W. Goethe Universität, Frankfurt, 1994.

[SSK01] M. Sato, T. Sakurai, and Y. Kameyama. A simply typed context calculus with first-class environments. In *5th International Symposium on Functional and Logic Programming*, volume LNCS 2024, pages 359–374, 2001.

Parallelism and Tree Regular Constraints

(Extended Abstract)

Joachim Niehren[1] and Mateu Villaret[2]

[1] Programming Systems Lab, Universität des Saarlandes, Saarbrücken, Germany.
[2] IMA, Universitat de Girona, Campus de Montilivi, Girona, Spain.

Abstract. Parallelism constraints are logical descriptions of trees. Parallelism constraints subsume dominance constraints and are equally expressive to context unification. Parallelism constraints belong to the constraint language for lambda structures (CLLS) which serves for modeling natural language semantics. In this paper, we investigate the extension of parallelism constraints by tree regular constraints. This canonical extension is subsumed by the monadic second-order logic over parallelism constraints. We analyze the precise expressiveness of this extension on basis of a new relationship between tree automata and logic. Our result is relevant for classifying different extensions of parallelism constraints, as in CLLS. Finally, we prove that parallelism constraints and context unification remain equivalent when extended with tree regular constraints.

Keywords: computational logic, tree automata, unification.

Parallelism constraints [10, 12] are logical descriptions of trees, i.e., of ground terms such as $f(f(a,b),a)$. Parallelism constraints constitute a purely conjunctive language. They can talk about the mother, ancestor, and parallelism relation of a tree:

1. The parallelism relation $\pi_1/\pi_2{\sim}\pi_3/\pi_4$ holds for nodes $\pi_1, \ldots, \pi_4$ of some tree (see Fig. 1) if π_1 is an ancestor of π_2, π_3 an ancestor of π_4, while tree segment between π_1 and π_2 is structurally equal to the segment between π_3 and π_4.
2. The labeling relation $\pi{:}f(\pi_1, \ldots, \pi_n)$ requires that node π is labeled with f and has the children $\pi_1, \ldots, \pi_n$ in this order.

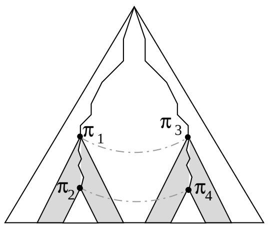


Fig. 1. Parallelism

Parallelism constraints subsume *dominance constraints* [16, 2] for which efficient satisfiability tests exists [1]. Dominance constraints are widely used throughout computational linguistics (see e.g. [20, 8]). They can express the ancestor relation $\pi_1 \triangleleft^* \pi_2$ between nodes π_1 and π_2 of some tree (which is equivalent to $\pi_1/\pi_2{\sim}\pi_1/\pi_2$).

Parallelism constraints are equal in expressive power to the equational language of *context unification* (CU) [5, 21, 22, 15] as proved in [17]. Whether CU is decidable (and thus the satisfiability of parallelism constraints) is a prominent open problem in unification theory. So far, only fragments could be proved decidable.

Parallelism constraints belong to the constraint language for lambda structures (CLLS) which serves for modeling natural language semantics [10, 9, 11]. CLLS extends parallelism constraints in several directions: there are lambda binding and beta reduction constraints [3, 4], but also anaphoric binding and group parallelism constraints. These extensions are used in applications but their expressiveness has never been studied.

In the talk, we investigate the canonical extension P+R of parallelism with tree regular constraints. The formulas of P+R are conjunctions of parallelism constraints P with regular constraints R. Let $\mathcal{A}$ be a tree automaton:

- A tree regular restriction $\mathsf{tree}(\pi) \in \mathcal{L}(\mathcal{A})$ is valid in a tree with node π if the subtree rooted by π belongs to the language recognized by tree automaton $\mathcal{A}$.

The extended language P+R is sufficiently restricted so that processing methods for pure parallelism constraints still apply: Given an extended constraint of P+R, we can first enumerate the *minimal solved forms* (i.e., the most general unifiers) of the pure parallelism part by saturation [12] and then test all minimal solved forms for compatibility with the tree regular part. We obtain a semi-decision procedure for P+R; even if we could decide the satisfiability of parallelism constraints we might still have to check infinitely many minimal solved forms for compatibility.

The language P+R is obviously subsumed by the monadic second-order logic over parallelism constraints. But it is less clear to which precise logical fragment P+R corresponds. This is the question, we will answer in this talk.

The basic idea is to exploit the classical relationship between tree automata and the weak monadic second-order logic of the binary tree (WS2S) [24, 7], which states that regular constraints R and formulas of WS2S have the same expressive power. But unfortunately, this result cannot be lifted to extensions of parallelism constraints as the languages P+R and WS2S have different models. We propose to consider the monadic second-order logic over dominance constraints (SDom) instead of WS2S. Both languages talk about the ancestor relation of trees. But WS2S is interpreted over the infinite binary tree while SDom models ground terms.

We establish a new relationship between tree automata and the logic SDom on basis of the old techniques for WS2S: We show that tree regular constraints R are equal in expressiveness to formulas of SDom.We then lift our new result to the respective extensions of parallelism constraints: We prove that the languages P+R and P+SDom can be inter-translated.This answers the question raised above. Our result also shows that the languages SDom and WS2S have equal expressiveness. We thereby generalize and complement an earlier insight of Rogers [2, 13] who noticed that the first-order theory of dominance constraints can be expressed WS2S.

Finally, we reconsider the relationship between CU and parallelism constraints (which have the same expressiveness [17]). We show that P+R is equal in expressive power to CU with tree regular constraints.This language in turn is equivalent to linear second order unification with tree regular constraints [14]. It is open whether CU+R is decidable (even if we freely assume that CU is decidable). But the situation is better for the special case of string unification with regular constraints [23] which can be decided in PSPACE [6].

Our contributions are relevant for classifying extensions of parallelism constraint, as for instance provided by CLLS. A forthcoming paper [18] proves, for instance, that the monadic second-order dominance logic SDom can express lambda binding constraints. The results of this paper thus imply that the extension of parallelism with lambda binding constraints (as provided by CLLS) can be expressed in both languages: P+R and CU+R.

A complete version of this extended abstract is available on the web [19].

References

1. Ernst Althaus, Denys Duchier, Alexander Koller, Kurt Mehlhorn, Joachim Niehren, and Sven Thiel. An efficient algorithm for the configuration problem of dominance graphs. In *12th ACM-SIAM Symposium on Discrete Algorithms*, pages 815–824, 2001.
2. R. Backofen, J. Rogers, and K. Vijay-Shanker. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language, and Information*, 4:5–39, 1995.
3. Manuel Bodirsky, Katrin Erk, Alexander Koller, and Joachim Niehren. Beta reduction constraints. In *RTA'01*, volume 2051 of *LNCS*, pages 31–46, 2001.
4. Manuel Bodirsky, Katrin Erk, Alexander Koller, and Joachim Niehren. Underspecified beta reduction. In *ACL'01*, pages 74–81, 2001.
5. Hubert Comon. Completion of rewrite systems with membership constraints. *Symbolic Computation*, 25(4):397–453, 1998. Extends on a paper at ICALP'92.
6. Volker Diekert, Claudio Guterrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is pspace-complete. In *STACS 2001*, volume 2010 of *LNCS*, pages 170–182, 2001.
7. John Doner. Tree acceptors and some of their applications. *Journal of Computer System Science*, 4:406– 451, 1970. Received December 1967, Revised May 1970.
8. Denys Duchier and Claire Gardent. Tree descriptions, constraints and incrementality. In Harry Bunt, Reinhard Muskens, and Elias Thijsse, editors, *Computing Meaning, Volume 2*, volume 77 of *Studies In Linguistics And Philosophy*, pages 205–227. Kluwer Academic Publishers, Dordrecht, December 2001.
9. Markus Egg, Alexander Koller, and Joachim Niehren. The constraint language for lambda structures. *Logic, Language, and Information*, 10:457–485, 2001.
10. Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. Constraints over lambda-structures in semantic underspecification. In *COLING/ACL'98*, pages 353–359, 1998.
11. Katrin Erk, Alexander Koller, and Joachim Niehren. Processing underspecified semantic representations in the constraint language for lambda structures. *Journal of Language and Computation*, 2002. To appear.
12. Katrin Erk and Joachim Niehren. Parallelism constraints. In *RTA'00*, volume 1833 of *LNCS*, pages 110–126, 2000.

13. Alexander Koller, Joachim Niehren, and Ralf Treinen. Dominance constraints: Algorithms and complexity. In *LACL'98*, volume 2014 of *LNAI*, 2001.
14. Jordi Levy and Mateu Villaret. Linear second-order unification and context unification with tree-regular constraints. In *RTA'00*, volume 1833 of *LNCS*, pages 156–171, 2000.
15. Jordi Levy and Mateu Villaret. Context unification and traversal equations. In *RTA'01*, volume 2051 of *LNCS*, pages 167–184, 2001.
16. Mitchell P. Marcus, Donald Hindle, and Margaret M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st ACL*, pages 129–136, 1983.
17. Joachim Niehren and Alexander Koller. Dominance constraints in context unification. In *Third International Conference on Logical Aspects of Computational Linguistics (Dec. 1998, Grenoble, France)*, volume 2014 of *LNAI*, 2001.
18. Joachim Niehren and Mateu Villaret. On lambda binding, parallelism constraints and context unification, 2002. Available at `http://www.ps.uni-sb.de/Papers`.
19. Joachim Niehren and Mateu Villaret. Parallelism and tree regular constraints, 2002. Long version. Available at `http://www.ps.uni-sb.de/Papers`.
20. J. Rogers and K. Vijay-Shanker. Obtaining trees from their descriptions: An application to tree-adjoining grammars. *Computational Intelligence*, 10:401–421, 1994.
21. M. Schmidt-Schauß. A decision algorithm for stratified context unification. Technical Report F-rep.-12, FB Informatik, J.W. Goethe Universität Frankfurt, 1999.
22. Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. In *CADE-16*, LNAI, pages 67–81, 1999.
23. Klaus U. Schulz. Makanin's algorithm for word equations- two improvements and a generalization. In *Proceedings of the First International Workshop of Word Equations and Related Topics (IWWERT'90)*, volume 572 of *LNCS*, Tübingen, Germany, 1992.
24. J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1967.

Parameterized String Rewriting Systems

Benjamin Monate

LRI, CNRS UMR 8623, Bâtiment 490
Université Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France
`Benjamin.Monate@lri.fr`

Abstract. In this paper we propose a new formalism for rewriting in order to describe infinite families of constrained string rewriting systems[3]: parameterized string rewriting. Moreover, these systems may include parametric exponents or finite products. Our main result is a critical pair lemma allowing decision of local confluence. We apply this procedure to check automatically local confluence of parameterized rewriting presentations of some well-known families of monoids and groups.

Standard rewriting techniques are widely used to perform automated proofs about equational specifications. For example symmetric groups $\mathcal{S}_N$ are studied as quotients of a free structure by some well chosen equations. The Knuth-Bendix completion automatically computes a convergent finite rewriting system which leads to an efficient decision algorithm for equality: normalization. One can exhibit a convergent rewriting system presenting $\mathcal{S}_N$ for *each* value of N in $\mathbb{N}$. In this paper we show how one can deal with the whole $\mathcal{S}_N$ family at once, without giving a value to the *parameter* N. To achieve this goal, we define an extension of constrained string rewriting: *parameterized string rewriting*. We propose a decision algorithm for local confluence of such systems through a critical pair lemma.

This is well-known folklore that $\mathcal{S}_N$ is generated by $N-1$ transpositions $\tau_1, \ldots \tau_{N-1}$ and the following equations:

$$\begin{aligned} \mathcal{S}_N \;&= <(\tau_1, \ldots, \tau_{N-1})/\{\tau_i^2 = \epsilon \mid 1 \leq i \leq N-1 \\ &\quad \tau_i\tau_j = \tau_j\tau_i \mid i-j \geq 2 \wedge 1 \leq i,j \leq N-1 \\ &\quad (\tau_i\tau_{i+1})^3 = \epsilon \mid 1 \leq i \leq N-2\} > \end{aligned}$$

$\mathcal{S}_N$ can be presented as a (non confluent) parameterized string rewriting system using our formalism:

$$\begin{aligned} \mathcal{S}_N \;&= \{\tau_i^2 \to \epsilon \mid 1 \leq i \leq N-1 \\ &\quad \tau_i\tau_j \to \tau_j\tau_i \mid i-j \geq 2 \wedge 1 \leq i,j \leq N-1 \\ &\quad (\tau_i\tau_{i+1})^3 \to \epsilon \mid 1 \leq i \leq N-2\} \end{aligned}$$

Another typical parameterized string rewriting system describes the dihedral group of order N generated by rotation r and symmetry s. One can prove that

$$D_N =< (r,s)/\{s^2 = r^N = (s.r)^2 = \epsilon\} >$$

is a presentation for the dihedral group of order N. One may conjecture the following complete presentation in parameterized rewriting for D_N:

$$R_{2N} = \{\ s^2 \to \epsilon \\ sr \to r^{N-1}s \\ r^N \to \epsilon\}$$

Using the test for local confluence of parameterized rewriting, this presentation is automatically proved to be convergent (termination is trivial). This test has been fully implemented in the software C*i*ME[2]. One may need to add some rules to encode parameterized steps of transitivity. Using the C*i*ME syntax:

```
let N = parameters "N";
let S = pword_signature N "s | {N>=2}; r | {N>=2}" ;
let R = psrs S "s s -> | {N>=2};
                s r -> r^{N-1} s | {N>=2} ;
                r^{N} -> | {N>=2};"
        ;
let Rnorm = psrs S "s r^{k} -> r^{N-k} s | {N>=2 /\ 1 <= k <= N-1 };";
pconfluent_ext R Rnorm;
```

The rule in `Rnorm` is an iteration of second rule.

As one can see in the previous examples parameterized rewriting deals with many kinds of parameterization:

- parameterization of the number of generators ($\mathcal{S}_N$)
- parameterization of the structure of the rules (D_N)

The second kind of parameterization is addressed with two syntactic constructs: exponents (like in $r^N = \epsilon$) and finite products (like in $\Pi_{i=0}^{N} a_i a_{i+1}$). The set of constraints we may use in rules is some kind of decidable fragment of arithmetic. We use Presburger's arithmetic.

The main result of parameterized rewriting is: *if a parameterized system is locally confluent then every instance of the system is locally confluent.* The converse does not hold, as one can only perform "uniform proofs" with parameterized rewriting.

In order to apply rewriting rules we provide an algorithm which computes all possible decompositions in 3 parts of a parameterized word, and an algorithm testing if two parameterized words are "equal". Those algorithms are really technically involved. The case of words with exponents is somewhat similar to ρ-terms [1].

A full description of parameterized rewriting with complete proofs is available in [4].

References

1. Hubert Comon. On unification of terms with integer exponents. *Math. Systems Theory*, 28:67–88, 1995.

2. Evelyne Contejean, Claude Marché, Benjamin Monate, and Xavier Urbain. Cime version 2, 2000. Prerelease available at `http://cime.lri.fr/`.
3. Claude Kirchner, Helene Kirchner, and Michaël Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automatic deduction.
4. Benjamin Monate. *Propriétés uniformes de familles de systèmes de réécriture de mots paramétrées par des entiers.* Thèse de doctorat, Université Paris-Sud, Orsay, France, January 2002.

A Tree Automata Theory for Unification Modulo Equational Rewriting

Hitoshi Ohsaki and Toshinori Takai

National Institute of Advanced Industrial Science and Technology (AIST)
Nakoji 3–11–46, Amagasaki 661–0974, Japan

{ ohsaki , takai }@ni.aist.go.jp

Abstract. An extension of tree automata framework, called equational tree automata, is presented. This theory is useful to deal with unification modulo equational rewriting. In the manuscript, we demonstrate how equational tree automata can be applied to unification problems.

1 Equational Tree Languages

Unification modulo equational theory is a central topic in automated reasoning. Tree automata are the powerful technique for handling unification modulo rewriting [2]. On the other hand, to model some network security problems like Diffie-Hellman key exchange algorithm, rewrite rules and equations (e.g. associativity and commutativity axioms) have to be separately dealt with in the underlying theory, but it causes the situation where the standard tree automata technique is useless. In our recent papers [5, 6], we have proposed an extension of tree automata, which is called equational tree automata. This framework subsumes Petri nets (Example 1). In a practical example, equational tree automata can be used to verify a security problem of Diffie-Hellman protocol (Example 2).

We start this section with basics of tree automata and the equational extension. A *tree automaton* (TA for short) $\mathcal{A}$ is defined by the 4-tuple $(\mathcal{F}, \mathcal{Q}, \mathcal{Q}_{\mathit{fin}}, \Delta)$: each of those components is a signature $\mathcal{F}$ (a finite set of function symbols with fixed arities), a finite set $\mathcal{Q}$ of states (special constants with $\mathcal{F} \cap \mathcal{Q} = \varnothing$), a subset $\mathcal{Q}_{\mathit{fin}}$ of $\mathcal{Q}$ consisting of so-called *final states* and a finite set Δ of transition rules in the following form:

– $f(p_1, \ldots, p_n) \to t$

for some $f \in \mathcal{F}$ with $\mathsf{arity}(f) = n$ and $p_1, \ldots, p_n \in \mathcal{Q}$ and $t \in \mathcal{T}(\{f\} \cup \mathcal{Q})$. Note that function symbols, except state symbols, in the right-hand side must be the same as one in the left-hand side. An *equational tree automaton* (ETA for short) $\mathcal{A}/\mathcal{E}$ is the combination of a TA $\mathcal{A}$ and an equational system (a set of equations; ES for short) $\mathcal{E}$ over the same signature $\mathcal{F}$. An ETA $\mathcal{A}/\mathcal{E}$ is called

– *regular* if the right-hand side t is a single state q,
– *monotone* if the right-hand side t is a single state q or a term $f(q_1, \ldots, q_n)$

for every transition rule $f(p_1, \ldots, p_n) \to t$ in Δ. Note that equational tree automata defined in [5, 6] are in the above monotone case. A term t is *accepted* by

$\mathcal{A}/\mathcal{E}$ if $t \in \mathcal{T}(\mathcal{F})$ and $t \rightarrow^*_{\mathcal{A}/\mathcal{E}} q$ for some $q \in \mathcal{Q}_{fin}$. Elements in the set $\mathcal{L}(\mathcal{A}/\mathcal{E})$ are ground terms accepted by $\mathcal{A}/\mathcal{E}$.

A *tree language* (TL for short) L over $\mathcal{F}$ is some subset of $\mathcal{T}(\mathcal{F})$. A TL L is $\mathcal{E}$*-recognizable* if there exists $\mathcal{A}/\mathcal{E}$ such that $L = \mathcal{L}(\mathcal{A}/\mathcal{E})$. Similarly, L is called $\mathcal{E}$-monotone ($\mathcal{E}$-regular) if $\mathcal{A}/\mathcal{E}$ is monotone (regular). If L is $\mathcal{E}$-recognizable with $\mathcal{E} = \varnothing$, we say L is recognizable. Likewise, we say L is monotone (regular) if L is $\varnothing$-monotone ($\varnothing$-regular). We say $\mathcal{A}/\mathcal{E}$ is a C-TA (A-TA, AC-TA) if $\mathcal{E} = \mathsf{C}$ ($\mathcal{E} = \mathsf{A}$, $\mathcal{E} = \mathsf{AC}$, respectively).

Lemma 1. *Every* C*-recognizable tree language is regular.*

Proof. Let $\mathcal{F}$ be the signature. Suppose L is a tree language recognizable with a C-TA $\mathcal{A}/\mathsf{C}$, where $\mathcal{A} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_{fin}, \Delta)$. We define $\mathcal{B} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_{fin}, \Delta')$, where

$$\Delta' = \left\{ f(p_1, \ldots, p_n) \rightarrow q \;\middle|\; \begin{array}{l} f(q_1, \ldots, q_n) \rightarrow r \in \Delta \text{ such that} \\ f(p_1, \ldots, p_n) \sim_{\mathsf{C}} f(q_1, \ldots, q_n) \text{ and } r \rightarrow^*_{\mathcal{A}/\mathsf{C}} q \end{array} \right\}.$$

Then it can be proved that the regular TA $\mathcal{B}$ recognizes L. □

Lemma 2. *The following language hierarchy holds if* $\mathcal{E} = \mathsf{A}$*:*

regular TL $\subsetneq$ $\mathcal{E}$*-regular TL* $\subsetneq$ $\mathcal{E}$*-monotone TL* $\subsetneq$ $\mathcal{E}$*-recognizable TL*

Proof. For the first inclusion, we take the tree language $L = \{t \mid |t|_{\mathsf{a}} = |t|_{\mathsf{b}}\}$ over the signature $\mathcal{F} = \{\mathsf{f}, \mathsf{a}, \mathsf{b}\}$, where $\mathsf{arity}(\mathsf{f}) = 2$ and a, b are constant symbols. If $\mathcal{F}_{\mathrm{A}} = \{\mathsf{f}\}$ then L is A-regular (Lemma 8, [5]), but is not regular. The second inclusion is proved in [6]. For the third inclusion we suppose $\mathcal{F} = \mathcal{F}_0 \cup \{\mathsf{f}\}$ with $\mathcal{F}_{\mathrm{A}} = \{\mathsf{f}\}$. Here $\mathcal{F}_0$ denotes a set of constant symbols. Then, a (word) language W over $\mathcal{F}_0$ is context-sensitive if and only if an A-monotone tree language is *maximal* for W. A tree language L is called maximal for a language W if for all terms t in $\mathcal{T}(\mathcal{F})$, $\mathsf{leaf}(t) \in W$ if and only if $t \in L$. Similarly, it holds that a language W is recursively enumerable if and only if an A-recognizable tree language is maximal for W. It is known that recursively enumerable languages strictly include context-sensitive languages. □

Question 1. Does the above hierarchy hold also for $\mathcal{E} = \mathsf{AC}$? Note that the first inclusion is trivial. Kudlek and Mitrana [3] showed in word case that multiset regular languages and multiset arbitrary languages are closed under homomorphisms and substitutions but multiset monotone languages are not. However, it does not imply the strict hierarchy of tree case, because of different definitions.

2 AC-Tree Automata for Unification Problems

In this section we discuss the applications of equational tree automata for unification problems. Our examples rely on the following decidability result.

Theorem 1 (Reachable property problem). *Given a ground AC-TRS* $\mathcal{R}/\mathsf{AC}$ *and tree languages* L_1, L_2 *over* $\mathcal{F}$ *with* $\mathcal{F}_{\mathrm{AC}}$*. If* L_1 *and* L_2 *are* AC*-recognizable tree languages, it is decidable whether there exist some* s *in* L_1 *and* t *in* L_2 *such that* $s \rightarrow^*_{\mathcal{R}/\mathsf{AC}} t$*, i.e.* $(\rightarrow^*_{\mathcal{R}/\mathsf{AC}})[\, L_1 \,] \cap L_2 \neq \varnothing$ *is a computable question.*

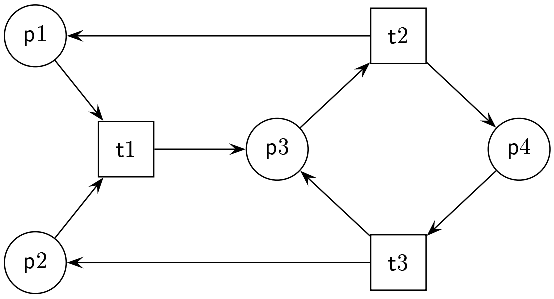


Fig. 1. A Petri net example: $\mathcal{P}$

Proof. For a singleton $\mathcal{F}_{\mathrm{AC}}$, the proof proceeds in the way of Lemma 4 in [5]. To extend $\mathcal{F}_{\mathrm{AC}}$ by allowing to have arbitrary many AC-symbols, we apply the similar argument of Section 3 in [5]. □

Example 1. Petri nets are known to be a special class of ground AC-TRSs. A Petri net is a triple (P, T, W), where P is a finite set of places, T is a finite set of transitions and W is a weight-function $(P \times T) \cup (T \times P) \to \mathbb{N}$. For instance, the Petri net $\mathcal{P}$ illustrated in Fig. 1 has W with $W(\mathsf{p1}, \mathsf{t1}) = 1$, $W(\mathsf{p2}, \mathsf{t1}) = 1$, $W(\mathsf{p3}, \mathsf{t2}) = 1$, $W(\mathsf{p4}, \mathsf{t3}) = 1$, $W(\mathsf{t1}, \mathsf{p3}) = 1$, $W(\mathsf{t2}, \mathsf{p1}) = 1$, $W(\mathsf{t2}, \mathsf{p4}) = 1$, $W(\mathsf{t3}, \mathsf{p2}) = 1$, $W(\mathsf{t3}, \mathsf{p3}) = 1$. In the figure, places are denoted by circles, and transitions are squares. The value of W determines the weight of directed arcs between places and transitions. Then, the associated ground AC-TRS $(\mathcal{F}, \mathcal{R}/\mathsf{AC})$ is defined by $\mathcal{F} = \{+\} \cup \{\epsilon, \mathsf{p1}, \ldots, \mathsf{p4}\}$, $\mathcal{F}_{\mathrm{AC}} = \{+\}$ and $\mathcal{R} = \{\mathsf{p1}+\mathsf{p2} \to \mathsf{p3},\ \mathsf{p3} \to \mathsf{p1}+\mathsf{p4},\ \mathsf{p4} \to \mathsf{p2}+\mathsf{p3}\} \cup \{\epsilon + \mathsf{p}i \to \mathsf{p}i,\ \mathsf{p}i \to \mathsf{p}i + \epsilon \mid 1 \leqslant i \leqslant 4\}$. In this setting, a state of a Petri net (the number of *tokens* on each place) is encoded by a multiset of place symbols. The empty multiset is represented by ϵ.

Given two sets L_1, L_2 of states of $\mathcal{P}$. According to Theorem 1, it is decidable whether there exist states $m_1 \in L_1$ and $m_2 \in L_2$ such that $m_1 \to^*_{\mathcal{P}} m_2$, provided L_1, L_2 are leaf-languages of AC-recognizable tree languages over $\mathcal{F}$. The binary relation $\to^*_{\mathcal{P}}$ is the reflexive-transitive closure of one-step transition relation. This decidability property generalizes the result of Mayr [4].

Using the above property, for instance, we can solve *coverability* problem, which is a question of whether there exists m_3 such that $m_1 \to^*_{\mathcal{P}} m_3$ and $m_2 \subseteq m_3$. Actually, it is verified by solving the following question, which is decidable:

$$\exists\sigma?.\ \ t_1 \to^*_{\mathcal{R}/\mathsf{AC}} t_2 + x\sigma.$$

Here t_1, t_2 are terms over $\mathcal{F}$ such that $\mathsf{leaf}(t_1) = m_1$ and $\mathsf{leaf}(t_2) = m_2$.

Example 2. We consider a simple network protocol. The protocol illustrated in Fig. 2 is called Diffie-Hellman key exchange algorithm (e.g., Section 22.1, [7]). In the protocol, a principal A chooses a prime number N and sends to B together with an integer $k(A) \circ N$ that is generated with a random number $k(A)$. Here we suppose that nobody else can guess $k(A)$ from $k(A) \circ N$. Then B returns $k(B) \circ N$

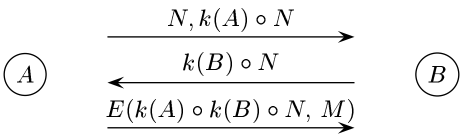


Fig. 2. Diffie-Hellman key exchange algorithm

to A. By assuming $\circ$ to be associative and commutative, $k(A) \circ k(B) \circ N$ can be used as a common secret key for A and B. It enables A to send only B a secret message M encrypted with this key. A security problem for this protocol is whether or not someone else can retrieve a secret message M by listing on the channel.

In term rewriting, the axiom of encryption and decryption and the property of keys are specified by the AC-rewrite system $\mathcal{R} = \{D(x, E(x,y)) \to y\}$ and $\mathsf{AC} = \{x \circ y \approx y \circ x,\ (x \circ y) \circ z \approx x \circ (y \circ z)\}$. On the other hand, a principal C wiretapping the channel can obtain N, $k(A) \circ N$, $k(B) \circ N$ and $E(k(A) \circ k(B) \circ N,\ M)$. Moreover, C is supposed to have personal data C, $k(C)$ and to be able to use function symbols $D, E, \circ$. So C's knowledge is the set L of terms constructible from these components. Then, the security problem is verified by solving the following unification problem:

$$\exists \sigma ?.\ \ x\sigma \to^*_{\mathcal{R}/\mathsf{AC}} M \ \text{ for some } \ x\sigma \in L.$$

In this setting, $(\to^*_{\mathcal{R}/\mathsf{AC}})[\,L\,]$ is an AC-monotone tree language. One should notice that in order to compute $(\to^*_{\mathcal{R}/\mathsf{AC}})[\,L\,]$ by using a modified algorithm of Kaji *et al.* [2], *intersection-emptiness* problem for AC-monotone tree languages must be decidable. Obviously a membership problem $M \in (\to^*_{\mathcal{R}/\mathsf{AC}})[\,L\,]$ is decidable.

Closure properties for equational tree languages are stated below.

Theorem 2. *$\mathcal{E}$-monotone tree languages are closed under union and intersection if $\mathcal{E} \in \{\,\mathsf{C},\ \mathsf{A},\ \mathsf{AC}\,\}$. The same closure property holds for $\mathcal{E}$-recognizable tree languages.*

Proof. If $\mathcal{E} = \mathsf{C}$, it is trivial from Lemma 1. Moreover, $\mathcal{E}$-monotone case is proved in [6]. For $\mathcal{E}$-recognizable tree languages, we take $\mathcal{A}$ and $\mathcal{B}$ where $\mathcal{A} = (\mathcal{F}, \mathcal{Q}_1, \mathcal{Q}_{1\mathit{fin}}, \Delta_1)$ and $\mathcal{B} = (\mathcal{F}, \mathcal{Q}_2, \mathcal{Q}_{2\mathit{fin}}, \Delta_2)$. Here we suppose $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \varnothing$. We define $\mathcal{C} = (\mathcal{F}, \mathcal{Q}, \mathcal{Q}_{\mathit{fin}}, \Delta_\cup)$ where $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$, $\mathcal{Q}_{\mathit{fin}} = \mathcal{Q}_{1\mathit{fin}} \cup \mathcal{Q}_{2\mathit{fin}}$ and $\Delta_\cup = \Delta_1 \cup \Delta_2$. By construction, $\mathcal{L}(\mathcal{A}/\mathcal{E}) \cup \mathcal{L}(\mathcal{B}/\mathcal{E}) \subseteq \mathcal{L}(\mathcal{C}/\mathcal{E})$ is obvious. And the reverse can be proved easily, because if $t \to^*_{\mathcal{C}/\mathcal{E}} q \in \mathcal{Q}_i$ then $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}_i)$. Note that for all transition rules $l \to r \in \Delta_\cup$, if $|r|_\mathcal{Q} = 0$ then $l = r$ (l, r are the same constant symbols), because $r \in \mathcal{T}(\{\mathsf{root}(l)\} \cup \mathcal{Q})$. Moreover, $s \to_{\mathcal{C}/\mathcal{E}} s'$ with $|s|_{\mathcal{Q}_i} \geqslant 1$ implies $|s'|_{\mathcal{Q}_i} \geqslant 1$ for each $1 \leqslant i \leqslant 2$.

For the intersection of $\mathcal{L}(\mathcal{A}/\mathcal{E})$ and $\mathcal{L}(\mathcal{B}/\mathcal{E})$, we define $\mathcal{D} = (\mathcal{F}, \mathcal{P}, \mathcal{P}_{\mathit{fin}}, \Delta_\cap)$:
- $\mathcal{P} = \{\langle p\rangle,\ \langle q\rangle,\ \langle p, q\rangle \mid p \in \mathcal{Q}_1 \text{ and } q \in \mathcal{Q}_2\}$,
- $\mathcal{P}_{\mathit{fin}} = \{\langle p, q\rangle \mid p \in \mathcal{Q}_{1\mathit{fin}} \text{ and } q \in \mathcal{Q}_{2\mathit{fin}}\}$,

– $\Delta_\cap = \Delta_\times \cup \Delta_{\overline{1}} \cup \Delta_{\overline{2}} \cup \Delta_\leftrightarrow$, where

$$\Delta_\times: \quad f(\langle p_1, q_1\rangle, \ldots, \langle p_n, q_n\rangle) \to \langle p, q\rangle$$
$$\text{if } f(p_1, \ldots, p_n) \to^*_{\mathcal{A}/\mathcal{E}} p \text{ and } f(q_1, \ldots, q_n) \to^*_{\mathcal{B}/\mathcal{E}} q \text{ with } f \notin \mathsf{fun}(\mathcal{E}),$$

$$\Delta_{\overline{1}}: \quad f(\langle p_1, q_1\rangle, \langle p_2, q_2\rangle) \to C_f[\![\langle p'_1, q_1\rangle, \langle p'_2, q_2\rangle, \langle p'_3\rangle \ldots, \langle p'_m\rangle]\!]$$
$$f(\langle p_1, q_1\rangle, \langle p_2\rangle) \to C_f[\![\langle p'_1, q_1\rangle, \langle p'_2\rangle, \ldots, \langle p'_m\rangle]\!]$$
$$\text{if } f(p_1, p_2) \to C_f[\![p'_1, \ldots, p'_m]\!] \in \Delta_1 \text{ with } f \in \mathsf{fun}(\mathcal{E});$$
$$f(\langle p_1, q_1\rangle, \langle p_2, q_2\rangle) \to f(\langle p'_1, q_1\rangle, \langle q_2\rangle)$$
$$f(\langle p_1, q_1\rangle, \langle p_2\rangle) \to \langle p'_1, q_1\rangle$$
$$\text{if } f(p_1, p_2) \to p'_1 \in \Delta_1 \text{ with } f \in \mathsf{fun}(\mathcal{E}),$$

$$\Delta_{\overline{2}}: \quad f(\langle p_1, q_1\rangle, \langle q_2\rangle) \to C_f[\![\langle p_1, q'_1\rangle, \langle q'_2\rangle, \ldots, \langle q'_m\rangle]\!]$$
$$f(\langle q_1\rangle, \langle q_2\rangle) \to C_f[\![\langle q'_1\rangle, \ldots, \langle q'_m\rangle]\!]$$
$$\text{if } f(q_1, q_2) \to C_f[\![q'_1, \ldots, q'_m]\!] \in \Delta_2 \text{ with } f \in \mathsf{fun}(\mathcal{E});$$
$$f(\langle p_1, q_1\rangle, \langle q_2\rangle) \to \langle p_1, q'_1\rangle$$
$$f(\langle q_1\rangle, \langle q_2\rangle) \to \langle q'_1\rangle$$
$$\text{if } f(q_1, q_2) \to q'_1 \in \Delta_2 \text{ with } f \in \mathsf{fun}(\mathcal{E}),$$

$$\Delta_\leftrightarrow: \quad f(\langle p_1, q_1\rangle, \langle p_2\rangle) \to f(\langle p_1\rangle, \langle p_2, q_1\rangle)$$
$$f(\langle p_1\rangle, \langle p_2, q_2\rangle) \to f(\langle p_1, q_2\rangle, \langle p_2\rangle)$$
$$f(\langle p_1, q_1\rangle, \langle q_2\rangle) \to f(\langle q_1\rangle, \langle p_1, q_2\rangle)$$
$$f(\langle q_1\rangle, \langle p_2, q_2\rangle) \to f(\langle p_2, q_1\rangle, \langle q_2\rangle)$$
$$\text{if } f \in \mathsf{fun}(\mathcal{E})$$

for all $f \in \mathcal{F}$, $p_1, \ldots, p_n, p'_1, \ldots, p'_m, p \in \mathcal{Q}_1$ and $q_1, \ldots, q_n, q'_1, \ldots, q'_m, q \in \mathcal{Q}_2$. In the above definition, C_f denotes a non-empty context consisting of f. (If $\mathsf{arity}(f) = n$, C_f has at least n holes.) The ETA $\mathcal{D}/\mathcal{E}$ satisfies $\mathcal{L}(\mathcal{D}/\mathcal{E}) = \mathcal{L}(\mathcal{A}/\mathcal{E}) \cap \mathcal{L}(\mathcal{B}/\mathcal{E})$ if $\mathcal{E} = \mathsf{A}$ ($\mathcal{E} = \mathsf{AC}$). □

Remark 1. $\mathcal{E}$-regular tree languages are closed under union. Moreover, C-regular TL are closed under intersection, but A-regular TL are not closed intersection. On the other hand, closure under intersection of AC-regular TL is open [5].

References

1. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi: *Tree Automata Techniques and Applications*, draft, 1999. Available on `http://www.grappa.univ-lille3.fr/tata/`
2. Y. Kaji, T. Fujiwara and T. Kasami: *Solving a Unification Problem under Constrained Substitutions Using Tree Automata*, JSC 23, pp. 79–117, 1997.
3. M. Kudlek and V. Mitrana: *Normal Forms of Grammars, Finite Automata, Abstract Families, and Closure Properties of Macrosets*, Multiset Processing, LNCS 2235, pp. 135–146, 2001.
4. E.W. Mayr: *An Algorithm for the General Petri Net Reachability Problem*, SIAM J. Comput. 13(3), pp. 441–460, 1984.
5. H. Ohsaki and T. Takai: *Reachability and Closure Properties of Equational Tree Languages*, Proc. of 13th RTA, 2002. To appear in LNCS.
6. H. Ohsaki: *Beyond Regularity: Equational Tree Automata for Associative and Commutative Theories*, Proc. of 15th CSL, LNCS 2142, pp. 539–553, 2001.
7. B. Schneier: *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Second Edition, John Wiley & Sons, 1996.

Filtering Unification: the Case of Modal Logic

Silvio Ghilardi and Lorenzo Sacchetti

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

1 Introduction

In this paper we deal with the case in which E-unification is *filtering*: this means that, given two solutions to a unification problem, there is always another one which is better than both of them. In this quite curious situation, unification problems can either be very nice (i.e. they admit a most general unifier), or extremely bad (no bases of unifiers exist). Examples are well-known: we have the commutative/monoidal theories (see [1],[3], [12] and also the survey paper [4]). Commutative/monoidal theories include the $\Box, \wedge, \top$-fragment of the (multimodal) system K, see [2]. We shall show in this paper that filtering unification is quite common indeed in *full* modal systems. First of all, we prove that (under the mild hypothesis that finitely presented algebras are closed under binary products) filtering unification is characterized by the fact that finitely presented *projective* algebras are closed under binary products (Theorem 1 below). Secondly, we apply this characterization to the case of normal extensions E of the modal system $K4$ and show that *E has filtering unification iff it extends* $K4.2^+$ (Theorem 2 below).

2 Unification and Projectivity

We recall here the algebraic approach to E-unification given in [6]. Given an equational theory E, V_E is the variety of models of E, i.e. it is the category of algebras satisfying the equations in E with related morphisms. Among algebras in V_E, we are expecially interested in those which are finitely presented and projectives. An E-algebra $\mathcal{P}$ is said to be *projective* iff it is a retract of a free algebra iff the covariant representable functor $V_E(\mathcal{P}, -)$ from V_E into the category of sets preserves regular epimorphisms. An E-algebra $\mathcal{A}$ is said to be *finitely presented* (fp for short) iff it is isomorphic to a finitely generated free algebra divided by a finitely generated congruence iff [5] the representable functor $V_E(\mathcal{A}, -)$ preserves filtered colimits.

We now introduce the relevant definition for E-unification from an algebraic point of view. In this context an *E-unification problem* is simply a finitely presented algebra $\mathcal{A}$ and a solution for it (also called a *unifier* for $\mathcal{A}$) is a pair given by a projective fp algebra $\mathcal{P}$ and a morphism $u : \mathcal{A} \longrightarrow \mathcal{P}$. Given two unifiers for $\mathcal{A}$, say $u_1 : \mathcal{A} \longrightarrow \mathcal{P}_1$ and $u_2 : \mathcal{A} \longrightarrow \mathcal{P}_2$, we say that u_1 is *more general than* u_2

($u_1 \geq u_2$) iff there exists a morphism $f : \mathcal{P}_1 \longrightarrow \mathcal{P}_2$ such that $f \circ u_1 = u_2$. A set S of unifiers for $\mathcal{A}$ is said to be a *complete set* of unifiers for $\mathcal{A}$ if every unifier for $\mathcal{A}$ is less general than a member of S. A complete set of unifiers for $\mathcal{A}$ is said to be a *basis* of unifiers for $\mathcal{A}$ iff its members are pairwise incomparable with respect to the preorder $\geq$. A unifier σ for $\mathcal{A}$ is said to be a *most general unifier* (mgu) for $\mathcal{A}$ iff $\{\sigma\}$ is a complete set of unifiers for $\mathcal{A}$. The definition of *unification type* is the expected one, namely we say that a solvable unification problem has type $1, \omega, \infty$ iff it has an mgu, a finite basis or an infinite basis of unifiers, respectively (type is 0 otherwise, i.e. iff there are no bases at all). The unification type of E is the 'worst case' type among solvable unification problems.

It can be shown [6] that this definition is equivalent (as far as unification type is concerned) to the standard one. It is indeed quite efficient as it makes the determination of unification type very trivial in concrete cases: e.g. Boolean algebras, Post algebras of order n, n-valued Lukasiewicz algebras, $S5$-modal algebras, Brouwverian semilattices, intermediate Dummett algebras, etc. all have unitary unification type because being unifiable and being projective coincide for fp algebras (thus identity morphisms act as mgus, which does not mean at all that unification problems are solved by identical substitutions!).

We say that *unification is filtering* in E iff given two solutions u_1, u_2 to a unification problem there is a third one (possibly coincident with u_1 or u_2) which is more general than both of them. It is immediately seen that if unification is filtering, than unification type can only be 0 or 1. The following (easy, but crucial) theorem gives a characterization of filtering unification:

Theorem 1. *If finitely presented projective E-algebras are closed under binary products, then unification in E is filtering. In case finitely presented E-algebras are closed under binary products, then the converse holds too.*

Proof. If fp projective algebras are closed under binary products, unification is filtering because given two unifiers $u_1 : \mathcal{A} \longrightarrow \mathcal{P}_1$ and $u_1 : \mathcal{A} \longrightarrow \mathcal{P}_2$ for $\mathcal{A}$, we clearly have that $\langle u_1, u_2 \rangle : \mathcal{A} \longrightarrow \mathcal{P}_1 \times \mathcal{P}_2$ is more general than both of them (the comparison morphisms being the two projections). Conversely, suppose that unification is filtering and that fp algebras are closed under products. Given fp projective algebras $\mathcal{P}_1, \mathcal{P}_2$, the unification problem $\mathcal{P}_1 \times \mathcal{P}_2$ has the two projections π_i as solutions; consequently there is a better unifier $u : \mathcal{P}_1 \times \mathcal{P}_2 \longrightarrow \mathcal{P}$ and for $i = 1, 2$ there are comparison morphisms h_i such that $h_i \circ u = \pi_i$. As the composite morphism

$$\mathcal{P}_1 \times \mathcal{P}_2 \xrightarrow{u} \mathcal{P} \xrightarrow{\langle h_1, h_2 \rangle} \mathcal{P}_1 \times \mathcal{P}_2$$

is the identity, $\mathcal{P}_1 \times \mathcal{P}_2$ is projective being a retract of $\mathcal{P}$. ⊣

Closure of finitely generated *free* algebras under binary products is sufficient for filtering unification (as it trivially implies the same closure property for finitely presented projective algebras): thus unification is filtering in commutative/monoidal categories where products and coproducts coincide. However closure of finitely generated free algebras is not necessary for filtering unification (even if fp algebras are closed under products): a counterexample comes easily from the results of next Section (the product of two finitely generated free $K4.2^+$ algebras is projective but not free - to see it, just count the terminal points in the canonical models over a finite propositional language).

3 Filtering unification in normal extensions of $K4$

It is well-known that normal modal logics (over $K4$) are in bijective correspondence with *varieties* (i.e. equational classes) of $K4$-modal algebras: a $K4$-modal algebra is a tuple $\mathcal{A} = \langle A, \top, \wedge, \vee, \bot, \neg, \Box \rangle$, where $\langle A, \top, \wedge, \vee, \bot, \neg \rangle$ is a Boolean algebra, $\Box$ is a hemimorphism (i.e. a unary operator preserving $\wedge, \top$) such that $\Box x \leq \Box\Box x$ holds for all $x \in A$.

We fix from now on a variety V_E of $K4$-modal algebras. We first have that the condition mentioned in Theorem 1 is satisfied:

Lemma 1. *The product of two finitely presented E-algebras is finitely presented as well.*

To get concrete information from Theorem 1 above, we need to introduce the duality machinery. We recall that a *transitive Kripke frame* is a pair $\langle P, R \rangle$ where P is a set and R is a binary transitive relation on P (called the accessibility relation on P). A *p-morphism* $f : \langle P, R \rangle \longrightarrow \langle P', R' \rangle$ between Kripke frames is a function satisfying the two conditions: a) $pRq \Rightarrow f(p)R'f(q)$, for all $p, q \in P$; b) $f(p)R'p' \Rightarrow \exists q \in P$ (pRq and $f(q) = p'$), for all $p \in P, p' \in P'$.

We recall that a *Boolean space* is a compact Hausdorff topological space $\langle P, \tau \rangle$ for which clopen subsets form a basis. A relation R on a topological space $\langle P, \tau \rangle$ is said to be *point-closed* iff for all $p \in P$ the subset $R(p) = \{q \mid pRq\}$ is closed. A *transitive descriptive frame* [11] is a triple $T = \langle P, R, \tau \rangle$, where: 1) $\langle P, \tau \rangle$ is a Stone space; 2) $\langle P, R \rangle$ is a transitive Kripke frame; 3) for every clopen Z, we have that $\Box Z = \{p \mid \forall q(pRq \Rightarrow q \in Z)\}$ is also clopen; 4) R is point-closed.

Transitive descriptive frames and $K4$-modal algebras are related each other by the *duality theorem* (see e.g. [13]), saying that the category of $K4$-modal algebras with related morphisms is dual to the category of transitive descriptive frames and continuous p-morphisms.

A *descriptive generated subframe* of a transitive descriptive frame like $T = \langle P, R, \tau \rangle$ is any $T_C = \langle C, R_C, \tau_C \rangle$ such that C is a closed subset of P which is also *generated* (meaning that $p \in C$ and pRq imply $q \in C$). The topology τ_C and the relation R_C are both obtained by restriction (consequently, clopens in T_C will be exactly subsets of the kind $Z \cap C$, where Z is a clopen of T). Sometimes we shall indicate T_C directly with C. The inclusion maps $i : T_C \hookrightarrow T$ are exacly (up to isomorphisms) the regular monomorphisms (i.e. the equalizers) in the category of descriptive frames and continuous p-morphisms.

Notice that, due to the fact that we only consider *transitive* descriptive frames, we have that given a point $p \in P$ in $T = \langle P, R, \tau \rangle$, the *cone over p*, namely the subset $T_p = \{q \in P \mid pR^+q\}$ (here R^+ is the relation $R \cup id$), always identifies a descriptive generated subframe: it is in fact closed because points are closed and because R is point-closed, it is also generated because R is transitive.

Given a variety V_E, a descriptive frame T is an *E-descriptive frame* iff its dual algebra $T^\star$ belongs to V_E. We clearly have a (restricted) duality between V_E and the full subcategory formed by E-descriptive frames. Here is our main result:

Theorem 2. *Unification in E is filtering iff $V_E \subseteq V_{K4.2^+}$, where $K4.2^+$ is axiomatized by the equation*[1]

$$\Diamond^+\Box^+x \to \Box^+\Diamond^+x = \top.$$

Proof. (Rough sketch). Left-to-right side: first of all observe that $K4.2^+$ can be equivalently axiomatized by the equation $\Box^+(\Box^+\Diamond^+\Box^+x \to \Box^+x) \to (\Box^+x \vee \Box^+\Diamond^+\neg x) = \top$. If this equation does not hold in V_E, by a standard fact (which is ultimately due to compactness), there is a Kripke model based on an E-descriptive frame T where the left hand modal formula (term) fails in a 'maximal' point p. The failure of the formula implies that we can tripartite T_p into three disjoint clopens, one of which is the cluster of p and the other two are generated subframes. By a topological argument (based essentially on the fact that our spaces are compact Hausdorff) it is possible to map T_p by a continuous p-morphism *onto* a descriptive frame of the kind $(\mathcal{F}_E(\emptyset)^\star + \mathcal{F}_E(\emptyset)^\star)(C_1 \cup C_2)^\circ_*$ or of the kind $(\mathcal{F}_E(\emptyset)^\star + \mathcal{F}_E(\emptyset)^\star)(C_1 \cup C_2)^\bullet_*$.[2] However these cannot be E-descriptive frames, essentially because of the closure of fp projective algebras under binary coproducts.

Right-to-Left Side: in a $K4.2^+$ descriptive frame T, we can define the relation $\exists r\ (pR^+r\ \&\ qR^+r)$ which turns out to be an equivalence relation. The equivalence classes (called *cohesive components*) are finitely many, clopen and generated in case T is the dual of a finitely generated algebra. If $\mathcal{P}_1, \mathcal{P}_2$ are fp projectives algebras, the proof of Lemma 1 represents $\mathcal{P}_1^\star, \mathcal{P}_2^\star$ as two disjoint generated clopens of a frame of the kind $\mathcal{F}_E(\underline{x})^\star$, hence to show that $\mathcal{P}_1 \times \mathcal{P}_2$ is projective it is sufficient to exhibit a continuous p-morphism $f : \mathcal{F}_E(\underline{x})^\star \longrightarrow \mathcal{P}_1^\star \cup \mathcal{P}_2^\star$ acting identically on elements belonging to $\mathcal{P}_1^\star$ and $\mathcal{P}_2^\star$. This can easily be built using projectivity of $\mathcal{P}_1, \mathcal{P}_2$ and the fact that no cohesive component intersects both $\mathcal{P}_1^\star$ and $\mathcal{P}_2^\star$. ⊣

To complete the picture we just mention that there are extensions of $K4.2^+$ for which most general unifiers do not exist [9](and hence unification is very bad because it is filtering); however [10], most general unifiers for all solvable unification problems exist for systems like $K4.2^+$, $S4.2$, $GL.2^+$, $Grz.2$, etc. (we cannot enter into details for these results here, we only mention that they are

[1] We use the abbreviations $\Box^+y$ for $\Box y \wedge y$ and $\Diamond^+y$ for $\neg\Box^+\neg y$, respectively.

[2] These are cones of frames obtained from $\mathcal{F}_E(\emptyset)^\star + \mathcal{F}_E(\emptyset)^\star$ by attaching it a new point $*$ seeing (possibly itself) and the elements belonging to two closed non empty generated subframes C_1, C_2. Notice that we use $(-)^\star$ to indicate also the descriptive frame dual to a modal algebra. The notation $\mathcal{F}_E(\underline{x})$ is used for the free E-algebra on the finite set $\underline{x}$.

established by using either finite dualities or techniques similar to those e.g. in [7],[8]).

References

1. F. Baader "Unification in Commutative Theories", Journal of Symbolic Computation, 8(5), pp. 479-497 (1989).
2. F. Baader, P. Narendran "Unification of Concept Terms in Description Logics", Journal of Symbolic Computation, 31(3), pp. 277-305 (2001).
3. F. Baader, W. Nutt "Combining problems for commutative/monoidal theories: How algebra can help in equational reasoning", J. of Applicable Algebra in Engineering, Communication and Computimg, 7(4), pp. 309-337 (1996).
4. F. Baader, W. Snyder "Unification Theory", in A. Robinson, A. Voronkov (eds.) "Handbook of Automated Reasoning", Elsevier/MIT, vol. I, pp. 445-533 (2001).
5. F. Borceux, "Handbook of Categorical Algebra", 3voll., Cambridge University Press (1994).
6. S. Ghilardi, " Unification through projectivity", Journal of Logic and Computation, 7, 6, pp.733-752 (1997).
7. S. Ghilardi, "Unification in Intuitionistic Logic", Journal of Symbolic Logic, vol 64 no.2 (1999), pp.859-880.
8. S. Ghilardi, "Best Solving Modal Equations", Annals of Pure and Applied Logic, 102, pp.183-198 (2000).
9. S. Ghilardi, "Unification, Finite Duality and Projectivity in Varieties of Heyting Algebras" (submitted).
10. S. Ghilardi, L. Sacchetti, "Most General Unifiers in Modal Logic", in preparation.
11. R.I. Goldblatt "Metamathematics of modal logic", Reports on Mathematical Logic 6, pp.41-77 (1976) and 7, pp.21-52 (1977).
12. W. Nutt "Unification in Monoidal Theories", in M.E. Stickel (ed.) 'Proceedings of the 10th International Conference on Automated Deduction', Springer LNAI 449, pp. 618-632 (1990).
13. G. Sambin, V. Vaccaro "Topology and duality in modal Logic", Annals of Pure and Applied Logic, vol.37, pp.249-296 (1988).

Abstract Canonical Inference Systems

Nachum Dershowitz[1] and Claude Kirchner[2]

[1] School of Computer Science Tel Aviv University Ramat Aviv, Tel Aviv 69978 Israel
E-mail: nachumd@tau.ac.il
[2] LORIA and INRIA BP 101, 54602 Villers-les-Nancy Cedex France
E-mail: Claude.Kirchner@loria.fr

Abstract. We provide a general proof theoretical setting under which the so-called "completion processes" (as used for equational reasoning) can be modeled, understood, studied, proved and generalized. This framework—based on a well-founded ordering on proofs—allows us to derive saturation processes and redundancy criteria abstractly.

A Superposition Decision Procedure for the Logic of Equality with Interpreted and Uninterpreted Functions*

Silvio Ranise

UHP–Nancy 2 & INRIA-Lorraine, Nancy, France, ranise@loria.fr
U. degli Studi di Genova–DIST, Genova, Italy, silvio@dist.unige.it

1 Introduction

The logic of equality with uninterpreted functions (EUF, for short) [4] has been proved very useful in many verification contexts such as the proof of correctness of microprocessors [7, 6] and of compilers [13]. For this kind of verification efforts, the proof obligations amounts to proving the equivalence between an abstract model (e.g. a specification) and a concrete model (e.g. an implementation). Since both models are expressed as formulae of the logic of EUF, the verification problem amounts to checking the equivalence of two formulae in EUF. Uninterpreted functions play a key rôle to make the problem amenable to actual verification techniques by abstracting away unimportant details. Unfortunately, the abstraction process does not preserve validity, e.g. if we replace the + operation over integers in $x + y = y + x$ with an uninterpreted function symbol f, we obtain the formula $f(x,y) = f(y,x)$ which is not valid for a function f which is not commutative. In this context, some *ad hoc* solutions to the problem of giving an interpretation to functions have been proposed in literature, although, to the best of our knowledge, no general solution is known. In [7], a validity checker for the logic of EUF is augmented with an heuristic to handle the interpreted functions used to model reading from and writing to memory elements. In [6, 5], extensions to the technique of [7] capable of handling memory components and transitive constraints (respectively) are proposed. The verification problem is eventually encoded into propositional logic so that efficient boolean satisfiability checkers can be used. In [13], an SMV-like model checker is extended with an abstraction hierarchy designed according to the specific optimisations the compiler performs in order to give an incremental interpretation of certain (arithmetic) functions.

In this paper, we describe a generic technique to build decision procedures for the logic of EUF extended with interpreted functions. Our technique is based on a simple extension of the results in [2, 3] to check the satisfiability of a class of clauses—which we call the EUF class—modulo a background theory $\mathcal{T}$, which is

* The author would like to thank Maria Paola Bonacina for her comments on a preliminary version of this abstract; Michaël Rusinowitch and Alessandro Armando for many useful discussions on related issues.

assumed to interpret some of the functions occurring in the verification problem. We show that if the saturation w.r.t. a superposition calculus of finite sets of ground literals terminates, then also the saturation of finite sets of clauses in the EUF class terminates. This simple modularity result allows us to lift the satisfiability procedures for lists, arrays, and sets [3] to decision procedures for EUF clauses modulo the theories of lists, arrays, and sets, respectively.

One of the main advantages of superposition over model theoretic decision procedures is that the former are proved correct more easily than the latter. As an example, compare the satisfiability procedures for an extensional theory of arrays of [15] with our procedure in [2]. Furthermore, when having a flexible saturation theorem prover at hand, such as E [14], it suffices to appropriately tune its parameters in order to efficiently implement the procedure. Some preliminary experimental results which give evidence that procedures built in this way perform well in practice can be found in [1].

2 Preliminaries

We assume a finite signature Σ s.t. $\Sigma := \Sigma_U \cup \Sigma_I$ where Σ_U and Σ_I are disjoint sets. By Σ_P we denote a finite set of predicate symbols of arity 0 disjoint from Σ containing two distinct constant symbol $\bot$ and $\top$ denoting the truth values "true" and "false", respectively. We assume the usual inductive definition of first-order (ground) terms over Σ. A flat term is a first-order term of depth at most 1. A first-order literal is either an expression of the form p $(\neg p)$ for p in Σ_P or an expression of the form $l \bowtie r$, where l, r are first-order ground terms over Σ and $\bowtie$ is either $=$ or $\neq$. A flat first-order literal is a literal of the form $l \bowtie r$, where l is a flat term, r is a constant, and $\bowtie$ is $=$ or l, r are constants and $\bowtie$ is $\neq$. A clause is a set of literals. We assume that $Ax(\mathcal{T})$ is a finite set of clauses over $\Sigma_I \cup \mathcal{X}$, where $\mathcal{X}$ is a finite set of variable symbols. We say that $Ax(\mathcal{T})$ presents the theory $\mathcal{T}$ and that the function symbols in Σ_I are interpreted. We also say that the function symbols in Σ_U are uninterpreted.

The set of EUF terms is the smallest set containing the first-order ground terms over Σ, expressions of the form $\mathsf{if}\,\phi\,\mathsf{then}\,\tau_1\,\mathsf{else}\,\tau_2$ (where τ_i (for $i = 1, 2$) is an EUF term and ϕ is an EUF formula), and expressions of the form $\mathsf{let}\,\lambda = \tau_0\,\mathsf{in}\,\tau_1$ (where λ is a constant in Σ_U, and τ_i (for $i = 0, 1$) is an EUF term). An EUF formula is either a first-order literal, a boolean combination of EUF formulae, or an expression of the form $\mathsf{if}\ \phi_0\ \mathsf{then}\ \phi_1\ \mathsf{else}\ \phi_2$ where ϕ_i (for $i = 0, 1, 2$) is an EUF formula. For example,

$$
\begin{array}{l}
\mathsf{let}\ \mathtt{reg}\ =\ \mathsf{if}\ \mathtt{bubble\text{-}wb}\ \mathsf{then}\ \mathtt{regfile}\ \mathsf{else}\ \mathsf{store}(\mathtt{regfile}, \mathtt{dest\text{-}wb}, \mathtt{result})\ \mathsf{in} \\
\mathtt{arg1}' = \mathsf{if}\quad (\mathtt{bubble\text{-}ex} \vee (\mathtt{src1} \neq \mathtt{dest\text{-}ex})) \\
\qquad\qquad \mathsf{then}\ \mathsf{select}(\mathtt{reg}, \mathtt{src1}) \\
\qquad\qquad \mathsf{else}\ \ alu(\mathtt{op\text{-}ex}, \mathtt{arg1}, \mathtt{arg2})
\end{array}
$$

is an EUF formula encoding part of the transition function of the pipelined ALU described in [7], where `bubble-wb` and `bubble-ex` are two (boolean) control registers of the pipeline, `regfile` is the register file, `src1` and `src2` are the address of the arguments in `regfile`, `op-ex` is the ALU operation to be performed, `dest-ex` is the address of `regfile` where the output has to be written, *alu* is an uninterpreted function abstracting the combinatorial part of the ALU, and select and store are interpreted (in the theory of arrays) function symbols which model reading from and writing to `regfile`, respectively.

We will make use of the superposition calculus $\mathcal{SP}$ of [9]. The calculus is parameterised w.r.t. a reduction ordering which we assume total over ground terms. $\mathcal{SP}$ is refutationally complete, i.e. any fair derivation of the rules of $\mathcal{SP}$ to an unsatisfiable set of clauses will derive the empty clause. A saturation of a set of clauses C by $\mathcal{SP}$ is the final set of clauses generated by a fair derivation from C using the rules of $\mathcal{SP}$ with higher priority given to simplification rules.

In the following, we study the *problem of checking whether the EUF formula $l = r$ is valid modulo the background theory $\mathcal{T}$*, where l, r are ground EUF terms; in symbols, $\mathcal{T} \models l = r$ (where $\models$ is the first-order relation of logical consequence, see e.g. [8]).

3 The Decision Procedure

We define a notion of EUF clauses and show how EUF formulae can be translated into EUF clauses. We will obtain a superposition decision procedure by defining a reduction ordering that allows us to lift the satisfiability procedures for ground sets of literals of [2, 3] to satisfiability procedures for EUF clauses in a modular way.

Our procedure works by refutation. In order to show that $\mathcal{T} \models l = r$ we prove the unsatisfiability of $Ax(\mathcal{T}) \cup E$, where $E := \{l = c_l, r = c_r, c_l \neq c_r\}$, c_l and c_r are two constants not occurring in Σ.

Let Elimination. In E, we replace each expression of the form $\mathsf{let}\, \lambda = \tau_0 \,\mathsf{in}\, \tau_1$ with τ_1 and we add the formula $\lambda = \tau_0$ to E. Let E_1 be the set of formulae so obtained. It is easy to see that the transformation preserves satisfiability and avoids the blow up of the size of E_1 by using the constants introduced by the let expressions to name subexpressions with multiple occurrences. The let construct is the key to cut down the dimension of the formulae to be verified.

Clausal Normal Form Translation. We rely on a specific clausal normal-form (CNF) transformation for formulae of the logic of EUF. The key observation is that expressions built out of if-then-else only are a logical basis, see e.g. [11] for a proof. It is easy to write a function *euf2ite* which maps formulae of the logic of EUF into formulae whose only propositional connective is the if-then-else. For example, one can replace the disjunction $\phi \vee \psi$ with $\mathsf{if}\, \phi' \,\mathsf{then}\, \top \,\mathsf{else}\, \psi'$, where ϕ' (ψ') is the result of (recursively) applying *euf2ite* to ϕ (ψ, respectively), see again [11] for details. Afterwards, we normalise the expression by replacing subexpressions of the form $\mathsf{if}\,(\mathsf{if}\, \phi_u \,\mathsf{then}\, \phi_v \,\mathsf{else}\, \phi_w)\,\mathsf{then}\, \phi_y \,\mathsf{else}\, \phi_z$ by $\mathsf{if}\, \phi_u \,\mathsf{then}\,(\mathsf{if}\, \phi_v \,\mathsf{then}\, \phi_y \,\mathsf{else}\, \phi_z)\,\mathsf{then}\,(\mathsf{if}\, \phi_w \,\mathsf{then}\, \phi_y \,\mathsf{else}\, \phi_z)$. It is easy to see that this transformation preserves satisfiability, it eliminates all if-then-else expression occurring as the first argument of another such expression, and the bottom leaves of the transformed expression will be just $\bot$ or $\top$. At this point, it is possible to read off the CNF of the formula, see [10] for more details.

Let E_2 be the set of clauses obtained by applying the above transformations to E_1. The clauses in E_2 are of the form $\bigvee_{p \in P} p \vee \bigvee_{i=1}^{n} l_i \bowtie r_i$, where P is a finite set of literals built out of Σ_P and $l_1, r_1, ..., l_n, r_n$ (for $n \geq 0$) are ground first-order terms.

Transformation to EUF clauses. Let Σ_S be a finite set of predicate symbols of arity 0 disjoint from Σ_P. The set of EUF clauses is the smallest set containing clauses of the form $\bigvee_{p \in P} p \vee l \bowtie r$ or of the form $\bigvee_{p \in P} p$, where P is a (possibly empty) finite set of literals over $\Sigma_P \cup \Sigma_S$ and l, r are ground first-order flat terms.

We now transform the set E_2 of clauses into a set E_3 of EUF clauses. This is done in two phases. First, we perform the flattening of the literals of the form $l \bowtie r$ of the clauses in E_2 by introducing fresh constants not occurring in Σ (see [2, 3] for details) and let E'_2 be the resulting set of clauses. This transformation conservatively extends the theory in which we are working, thereby preserving the satisfiability. Second, for each clause in E'_2 containing more than one equational literal, we replace each literal $l \bowtie r$ by a new 0-ary predicate symbol s in Σ_S and we add the clause $\neg s \vee l \bowtie r$. We repeat this until all the clauses contain at most one literal of the form $l \bowtie r$. Also this transformation preserves satisfiability (see [12] for a proof). It is easy to see that E_3 is a set of EUF clauses.

Saturation of EUF clauses. A preliminary step is necessary to apply $\mathcal{SP}$ to non-equational literals. We replace literals of the form p or $\neg p$ with the (equational) literals $c_p = \mathit{true}$ or $c_p \neq \mathit{true}$, respectively, where c_p is a constant symbol not occurring in Σ and true is a new special symbol distinct from all symbols in Σ. This transformation preserves satisfiability (see, e.g. [9] for a proof).

In the following, let Σ'_P be the set of constants introduced to replace the 0-ary predicate symbols s.t. Σ'_P and Σ are disjoint. We assume that the reduction ordering $\succ$ used for $\mathcal{SP}$ is such that $t \succ c \succ d \succ \mathit{true}$ for each constant c in Σ, for each constant d in Σ'_P, and for every ground term t that contains a symbol of arity greater than 0. This requirement is easily satisfied when using RPO by defining a suitable precedence relation over the function symbols and it is compatible with the requirements on $\succ$ in [2, 3]. Now, consider the background theory $\mathcal{T}$ and assume that the saturation of $Ax(\mathcal{T}) \cup L$ is finite, where L is a finite set of ground flat literals. We show that the saturation of $Ax(\mathcal{T}) \cup C$ is also finite, where C is a finite set of EUF clauses. There are two cases to consider. First, the only literal of a clause $\bigvee_{p \in P} c_p \bowtie \mathit{true} \vee l \bowtie r$ which can be involved in an inference of $\mathcal{SP}$ is $l \bowtie r$ by the requirement on $\succ$. So, the assumption of finite saturation for ground flat literals straightforwardly yields the finiteness of the saturation for these clauses. Second, inferences involving EUF clauses containing only literals over Σ'_P are also finite since from the finite set of constants Σ'_P we can generate a finite set of distinct clauses only. We conclude that saturation of sets of EUF clauses is finite.

4 Discussion

We have applied our methodology to the verification of the pipelined ALU described in [7]. We use the E prover [14] to perform saturation. On this example, using the automatic mode of E, our procedure terminates in about half a second. We are in the process of tackling the verification of a pipelined microprocessor also described in [7]. We expect that some tuning of the settings of E will be necessary for better performances.

We think that the main advantage of the decision procedure described in this abstract is the fact that it exploits (almost) off-the-shelf standard techniques of the rewriting approach to automated theorem proving (such as saturation and CNF translation) to flexibly and modularly prototype/build verification tools.

References

1. A. Armando, M.P. Bonacina, S. Ranise, and M. Rusinowitch. High-Performance Deduction for Verification: A Case Study in the Theory of Arrays. In VERIFY'02 (FLoC'02 Affiliated Workshop), 2002. To appear.
2. A. Armando, S. Ranise, and M. Rusinowitch. Uniform Derivation of Decision Procedures by Superposition. In *Computer Science Logic (CSL01), Paris, France*, 2001.
3. A. Armando, S. Ranise, and M. Rusinowitch. A Rewriting Approach to Satisfiability Procedures. *Information and Computation*, 2002. To appear.
4. R. E. Bryant, S. German, and M. N. Velev. Processor Verification Using Efficient Decision Procedures for a Logic of Uninterpreted Functions. In *TABLEAUX'99*, number 1617 in LNAI, pages 1–13. Springer-Verlag, 1999.
5. R. E. Bryant and M. N. Velev. Boolean Satisfiability with Transitivity Constraints. *ACM Trans. on Comp. Logic*, 2002. To appear.
6. R. E. Bryant and M. N. Velev. Processor Verification Using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic. *ACM Trans. on Comp. Logic*, 2002. To appear.
7. J. R. Burch and D. L. Dill. Automatic verification of pipelined microprocessors control. In David L. Dill, editor, *Proc. of the 6th Intl. Conf. on Computer-Aided Verification CAV*, vol. 818, pages 68–80, California, USA, 1994.
8. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Pr., 1972.
9. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Hand. of Automated Reasoning*. 2001.
10. A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In A. Robinson and A. Voronkov, editors, *Hand. of Automated Reasoning*. 2001.
11. J. Posegga and P. H. Schmitt. Automated Deduction with Shannon Graphs. *J. of Logic and Computation*, 5(6):697–729, 1995.
12. A. Riazanov and A. Voronkov. Splitting Without Backtracking. Technical Report CSPP-10, University of Manchester, Manchester, UK, January 2001.
13. Y. Rodeh and O. Shtrichman. Finite Instantiations in Equivalence Logic with Uninterpreted Function. In *13th Intl. Conf. on Computer Aided Verification (CAV'01)*, pages 144–154, 2001.
14. S. Schulz. System Abstract: E 0.61. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. of the 1st IJCAR, Siena*, number 2083 in LNAI, pages 370–375. Springer, 2001.
15. A. Stump, D. L. Dill, C. W. Barrett, and J. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *LICS'01*, 2001.

Direct Combination of Completion and Congruence Closure

Leo Bachmair[1] and Christelle Scharff[2]

[1] State University of New York at Stony Brook, Department of Computer Science, Stony Brook, NY, USA, leo@cs.sunysb.edu
[2] Pace University, Department of Computer Science, New York, NY, USA, cscharff@pace.edu

Theories presented by finite sets of ground (i.e., variable-free) equations are known to be decidable. There are different approaches to dealing with ground equational theories. Researchers interested in term rewriting, for instance, have applied *completion*, a method that transforms a given set of equations into a set of directed rules that is both terminating and confluent; see [3] for an overview. Such a convergent rewrite system defines unique normal forms for equal terms and hence provides a decision procedure for the word problem of the underlying equational theory. Completion itself is a semi-decision procedure but under certain reasonable assumptions about the strategy used to transform equations, is guaranteed to terminate if the input is a set of ground equations.

A rather distinct approach to solving word problems for ground equational theories is based on the computation of the so-called *congruence closure* of a relation. Various efficient congruence closure algorithms have been proposed. They typically use a compact representation of the given terms by a directed acyclic graph; cf., [6, 4, 8, 10]. Completion methods are usually not as efficient, though one efficient ground completion method has been described by [11], who obtains an $O(n\ logn)$ algorithm that cleverly uses congruence closure to transform a given set of ground equations into a convergent ground rewrite system. The standard completion approach is quadratic in the worst case [9].

In this work we bridge the gap between the two approaches and provide a better understanding of the connection between completion and congruence closure by defining a new, efficient congruence closure method that integrates key ideas of completion and graph-based methods in a novel, direct and natural way. Our method takes its origin in the abstract congruence closure of [1] and the SOUR-graph method of [7]. It may be viewed as a modified version of the SOUR-graph approach, but with simpler graphs. In contrast to SOUR graphs it is not always possible to extract a (convergent) rewrite system from these graphs, but one always obtains an (partially constructed) abstract congruence closure, that is, a convergent rewrite system over an *extended* signature.

In a SOUR graphs the vertices represent terms and the edges carry information about subterm relations between terms (S), rewrite rules (R), unifiability of terms (U) and order relations between terms (O). We consider only ground terms and hence do not need unification edges. We also dispense with order edges and manipulate rewrite edges in a different way, based on the explicit use of edges (E) representing unordered equations. (In short, our modifications amount to what might be called "SER graphs.") The deletion of the order edges is crucial for reasons of efficiency: It is a characteristic advantage of our method—and, of course, of (abstract) congruence closure algorithms in general—that no explicit term ordering needs to be computed. (The corresponding disadvantage is that we do not obtain a convergent rewrite system on the original signature, but only over an extended signature.) The graph transformation rules we use are simpler versions of SOUR graph rules and essentially represent (combinations of) critical pair computations and term simplifications. We use fewer rules and also explicitly formulate a "merge rule" that is well-known from congruence closure algorithms, but only implicitly used in SOUR graphs.

Specifically, we represent terms and equations by a directed acyclic graph (DAG), where each vertex is labeled by a symbol from the initial term signature Σ and edges are classified as E (equation), R (rewrite), or S (subterm). In addition, each vertex has a name, which we view as a constant; the set K of all such constants is required to be disjoint from Σ. This shared representation is inherently more efficient than any other representation of terms. Each vertex represents a term over the extended signature $\Sigma \cup K$. It also represents a term over the initial signature Σ, but the representation schema is not as direct as in the case of SOUR graphs and, due to the absence of a term ordering, it may make no sense to interpret rewrite rules over the original signature. The graph transformation rules implement a version of abstract congruence closure. We may (i) *orient* an equation, (ii) compute critical pairs (RR rule), (iii) simplify terms (SR rule), or (iv) *merge* nodes to ensure closure under congruence. Exhaustive application of these transformation rules terminates, is sound in that the equational theory represented over Σ-terms does not change, and complete in the sense that the final rewrite system over the extended signature is convergent.

This "congruence closure phase" does not produce a convergent rewrite system over the original signature, but we may obtain such a system by further transforming the graph in a way reminiscent of the compression and selection rules of [2]. These rules eliminate vertices from the graph (and constants from K) and redirect R edges. Their exhaustive application produces a graph representing a convergent rewrite system over the initial signature.

There is a link between unification and congruence closure [5]; unification closure can be reduced to a special form of congruence closure . We plan to explore this connection between unification and congruence closure in the graph-based framework of our method.

A long version of the paper is available at:

http://www.csis.pace.edu/~scharff.

References

[1] Bachmair, L. and Ramakrishnan, I.V. and Tiwari, A. and Vigneron, L. Congruence Closure Modulo Associativity and Commutativity. In Kirchner, H. and Ringeissen, C. editors, *Frontiers of Combining Systems, 3rd Intl Workshop FroCoS 2000*, Volume 1794 of *Lecture Notes in Artificial Intelligence*, pages 245-259, Nancy, France, March 2000, Springer-Verlag.

[2] Bachmair, L. and Tiwari, A. and Vigneron, L. Abstract Congruence Closure. *To appear in Journal of Automated Reasoning*, 2002.

[3] Dershowitz, N. and Jouannaud, J.-P. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244-320, Elsevier Science Publishers B.V. (North Holland), 1990, also as Research report 478, LRI.

[4] Downey, P. J. and Sethi, R. and Tarjan, R.E. Variations on the common subexpressions problems. *JACM*, 27(4), pages 758-771, 1980.

[5] Kanellakis, P. and Revesz, P. On the Relationship of Congruence Closure and Unification. *Journal of Symbolic Computation*, 7(3& 4):427-444, 1989, Special issue on unification, part one.

[6] Kozen, D. Complexity of finitely presented algebras. In *Proceedings of 9th Symposium of Theory of Computation*, pages 164-177, May, 1977.

[7] Lynch, C. and Strogova, P. SOUR Graphs for Efficient Completion. *Discrete Mathematics and Theoretical Computer Science*, volume 2, pages 1-25, 1998.

[8] Nelson, G. and Oppen, D.C. Fast Decision Procedures Based on Congruence Closure. *JACM*, 27(2):356-364, 1980.

[9] Plaisted, D. A. and Sattler-Klein, A. Proof Lengths for Equational Completion. *Information and Computation*, 125(2):154-170, 1996.

[10] Shostak, R.E. Deciding Combinations of Theories. *JACM*, 31(1):1-12, 1984.

[11] Snyder, W. A Fast Algorithm for Generating Reduced Ground Rewriting Systems from a Set of Ground Equations. *Journal of Symbolic Computation*, 15(4):415-450, 1993.

New Completeness Results for Lazy Conditional Narrowing

Mircea Marin and Aart Middeldorp

Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba 305-8573, Japan
{mmarin,ami}@score.is.tsukuba.ac.jp

1 Lazy Conditional Narrowing

Narrowing was originally invented as a general method for solving unification problems in equational theories that are presented by confluent term rewriting systems (TRSs for short). More recently, narrowing was proposed as the computational mechanism of functional-logic programming languages and several new completeness results concerning the completeness of various narrowing strategies and calculi have been obtained in the past few years. Here completeness means that for every solution to a given goal a solution that is at least as general is computed by the narrowing strategy. Since narrowing is a complicated operation, numerous calculi consisting of a small number of more elementary inference rules that simulate narrowing have been proposed.

Completeness issues for the lazy narrowing calculus LNC have been extensively studied in [3] and [2]. The main result of [3] is the completeness of LNC with leftmost selection for arbitrary confluent TRSs and normalized solutions. In [2] restrictions on the participating TRSs and solutions are presented which guarantee that all non-determinism due to the choice of inference rule of LNC is removed.

In this paper we consider the lazy conditional narrowing calculus LCNC of [4]. LCNC is the extension of LNC to conditional term rewrite systems (CTRSs for short). The extension is motivated by the observation that CTRSs are much more expressive than unconditional TRSs for describing interesting problems in a natural and concise way. However, the additional expressive power raises two problems: (1) confluence is insufficient to guarantee the completeness with respect to normalized solutions, and (2) the search space increases dramatically because the conditions of the applied rewrite rule are added to the current goal.

In [4] several completeness results for LCNC are presented. The only result which does not assume some kind of termination assumption does not permit extra variables in the conditions and right-hand sides of the rewrite rules. In this paper we show the completeness of LCNC with leftmost selection for the class of (confluent) *deterministic* oriented CTRSs. Determinism was introduced by Ganzinger [1] and has proved to be very useful for the study of the (unique) termination behavior of well-moded Horn clause programs (cf. [5]).

We consider goals G consisting of *unoriented* equations $s \approx t$ and *oriented* equations $s \rhd t$. An *oriented* CTRS $\mathcal{R}$ is a set of rewrite rules of the form

$l \to r \Leftarrow c$ where c is a goal consisting of oriented equations. The induced rewrite relation is denoted by $\to_{\mathcal{R}}$. A substitution θ is a solution of a goal $G = e_1, \ldots, e_n$ if $s\theta \leftrightarrow^*_{\mathcal{R}} t\theta$ if $e_i = s \approx t$ and $s\theta \to^*_{\mathcal{R}} t\theta$ if $e_i = s \rhd t$, for all $i \in \{1, \ldots, n\}$. For confluent CTRSs $\mathcal{R}$ the relations $\leftrightarrow^*_{\mathcal{R}}$ and $\downarrow^*_{\mathcal{R}}$ coincide and hence unoriented equations can be viewed as *joinability* statements.

The lazy conditional narrowing calculus LCNC_ℓ consist of the following inference rules:

[o] *outermost narrowing*

$$\frac{f(s_1, \ldots, s_n) \simeq t, G}{s_1 \rhd l_1, \ldots, s_n \rhd l_n, c, r \simeq t, G} \qquad \simeq\, \in \{\approx, \rhd, \backsimeq\}$$

if $f(l_1, \ldots, l_n) \to r \Leftarrow c$ is a fresh variant of a rewrite rule in $\mathcal{R}$,

[i] *imitation*

$$\frac{f(s_1, \ldots, s_n) \simeq x, G}{(s_1 \simeq x_1, \ldots, s_n \simeq x_n, G)\theta} \qquad \simeq\, \in \{\approx, \rhd, \backsimeq, \lhd\}$$

if $\theta = \{x \mapsto f(x_1, \ldots, x_n)\}$ with $x_1, \ldots, x_n$ fresh variables,

[d] *decomposition*

$$\frac{f(s_1, \ldots, s_n) \simeq f(t_1, \ldots, t_n), G}{s_1 \simeq t_1, \ldots, s_n \simeq t_n, G} \qquad \simeq\, \in \{\approx, \rhd\},$$

[v] *variable elimination*

$$\frac{s \simeq x, G}{G\theta} \qquad \simeq\, \in \{\approx, \rhd, \backsimeq, \lhd\}$$

if $x \notin \mathcal{V}\text{ar}(s)$ and $\theta = \{x \mapsto s\}$,

[t] *removal of trivial equations*

$$\frac{s \simeq s, G}{G} \qquad \simeq\, \in \{\approx, \rhd\}.$$

In the above rules we use $s \backsimeq t$ to denote the equation $t \approx s$ and $s \lhd t$ to denote the equation $t \rhd s$.

Compared to the definition of LCNC in [4], in LCNC_ℓ leftmost selection is built-in. Another difference is that the *parameter-passing equations* $s_1 \rhd l_1, \ldots, s_n \rhd l_n$ created by the outermost narrowing rule [o] are regarded as oriented equations. This is in line with the earlier completeness proofs for LNC and LCNC [2–4]. Furthermore, the conditions c are placed before $r \simeq t$ whereas in [4] they are placed after $r \simeq t$. The relative locations of c and $r \simeq t$ are irrelevant for the completeness results reported in that paper, but our new completeness proofs do not go through if we stick to the order in [4]. Finally, the removal of trivial equations rule [t] is usually restricted to equations between identical variables. The variation adopted here allows us to shorten some trivial derivations.

If G_1 and G_2 are the upper and lower goal in the inference rule $[\alpha]$, we write $G_1 \Rightarrow_{[\alpha]} G_2$. The applied rewrite rule or substitution may be supplied as subscript, that is, we write things like $G_1 \Rightarrow_{[\mathtt{o}],\, l \to r \Leftarrow c} G_2$ and $G_1 \Rightarrow_{[\mathtt{i}],\, \theta} G_2$. A finite LCNC_ℓ-derivation $G_1 \Rightarrow_{\theta_1} \cdots \Rightarrow_{\theta_{n-1}} G_n$ may be abbreviated to $G_1 \Rightarrow^*_\theta G_n$ where θ is the composition $\theta_1 \cdots \theta_{n-1}$ of the substitutions $\theta_1, \ldots, \theta_{n-1}$ computed along its steps. An LCNC_ℓ-refutation is an LCNC_ℓ-derivation ending in the empty goal $\square$.

2 Soundness and Completeness

Soundness of LCNC_ℓ is not difficult to prove.

Theorem 1. *Let $\mathcal{R}$ be an arbitrary CTRS and G a goal. If $G \Rightarrow^*_\theta \square$ then $\theta\lceil_{\mathcal{V}\text{ar}(G)}$ is a solution of G.* $\square$

Let X be a finite set of variables. A goal $G = e_1, \ldots, e_n$ is called *X-deterministic* if $\mathcal{V}\text{ar}(s_i) \subseteq X \cup \bigcup_{j=1}^{i-1} \mathcal{V}\text{ar}(e_j)$ if $e_i = s_i \rhd t_i$ and $\mathcal{V}\text{ar}(e_i) \subseteq X \cup \bigcup_{j=1}^{i-1} \mathcal{V}\text{ar}(e_j)$ if $e_i = s_i \approx t_i$, for all $1 \leqslant i \leqslant n$. An oriented CTRS $\mathcal{R}$ is called deterministic if and only if $\mathcal{V}\text{ar}(r) \subseteq \mathcal{V}\text{ar}(l, c)$ and the conditions c of every rewrite rule $l \to r \Leftarrow c$ in $\mathcal{R}$ are $\mathcal{V}\text{ar}(l)$-deterministic.

The following oriented CTRS $\mathcal{R}$ is a natural encoding of the efficient computation of Fibonacci numbers:

$$\begin{array}{rcl@{\qquad}rcl}
0 + y & \to & y & \mathsf{nth}(0, \langle y, z\rangle) & \to & y \\
\mathsf{s}(x) + y & \to & \mathsf{s}(x + y) & \mathsf{nth}(s(x), \langle y, z\rangle) & \to & \mathsf{nth}(x, z) \\
\mathsf{fib}(0) & \to & \langle 0, \mathsf{s}(0)\rangle & & & \\
\mathsf{fib}(\mathsf{s}(x)) & \to & \langle z, y + z\rangle \;\Leftarrow\; \mathsf{fib}(x) \rhd \langle y, z\rangle & & &
\end{array}$$

Note that $\mathcal{R}$ is deterministic. It is also terminating. However, when we change the oriented condition $\mathsf{fib}(x) \rhd \langle y, z\rangle$ into an unoriented condition $\mathsf{fib}(x) \approx \langle y, z\rangle$ then we lose termination: Because $\mathsf{fib}(\mathsf{s}(0)) \to^+ \langle \mathsf{s}(0), \mathsf{s}(0)\rangle$ we have $\mathsf{fib}(0) \downarrow \langle 0, \mathsf{nth}(0, \mathsf{fib}(\mathsf{s}(0)))\rangle$ and thus

$$\mathsf{fib}(\mathsf{s}(0)) \to \langle \mathsf{nth}(0, \mathsf{fib}(\mathsf{s}(0))), 0 + \mathsf{nth}(0, \mathsf{fib}(\mathsf{s}(0)))\rangle$$

which gives rise to an infinite rewrite sequence. As a consequence, the completeness results for LCNC presented in [4], which deal only with join CTRSs, do not apply to this CTRS. The completeness result presented below does not have this problem.

Theorem 2. *Let $\mathcal{R}$ be a confluent oriented deterministic CTRS and let G be an X-deterministic goal. If θ is a solution of G such that $\theta\lceil_X$ is normalized then $G \Rightarrow^*_{\theta'} \square$ for some substitution θ' with $\theta' \leqslant \theta$ $[\mathcal{V}\text{ar}(G)]$.* $\square$

The proof is based on a transformation of rewrite proofs of $G\theta$. The crucial insight is that determinism permits one to identify suitable sets of variables X_1, X_2, … such that each intermediate goal G_i in the refutation $G \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \cdots \Rightarrow \square$ that is being constructed is X_i-deterministic.

3 Refinements

Completeness with respect to normalized solutions does not imply that all solutions computed by LCNC_ℓ are normalized. This is illustrated in the following example.

Example 1. Consider the confluent TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{b}, \mathsf{g}(\mathsf{b}) \to \mathsf{c}\}$ and the goal $G = x \approx y, \mathsf{g}(x) \approx \mathsf{c}, \mathsf{f}(y) \approx \mathsf{b}$. The following LCNC_ℓ-refutation computes the non-normalized solution $\theta = \{x \mapsto \mathsf{f}(\mathsf{a}), y \mapsto \mathsf{a}\}$:

$$\begin{aligned} G &\Rightarrow_{[\mathtt{v}],\,\{x \mapsto \mathsf{f}(y)\}} \mathsf{g}(\mathsf{f}(y)) \approx \mathsf{c}, \mathsf{f}(y) \approx \mathsf{b} \Rightarrow_{[\mathtt{o}],\,\mathsf{g}(\mathsf{b}) \to \mathsf{c}} \mathsf{f}(y) \rhd \mathsf{b}, \mathsf{c} \approx \mathsf{c}, \mathsf{f}(y) \approx \mathsf{b} \\ &\Rightarrow_{[\mathtt{o}],\,\mathsf{f}(\mathsf{a}) \to \mathsf{b}} y \rhd \mathsf{a}, \mathsf{b} \rhd \mathsf{b}, \mathsf{c} \approx \mathsf{c}, \mathsf{f}(y) \approx \mathsf{b} \Rightarrow_{[\mathtt{v}],\,\{y \mapsto \mathsf{a}\}} \mathsf{b} \rhd \mathsf{b}, \mathsf{c} \approx \mathsf{c}, \mathsf{f}(\mathsf{a}) \approx \mathsf{b} \\ &\Rightarrow^2_{[\mathtt{t}]} \mathsf{f}(\mathsf{a}) \approx \mathsf{b} \Rightarrow_{[\mathtt{o}],\,\mathsf{f}(\mathsf{a}) \to \mathsf{b}} \mathsf{a} \rhd \mathsf{a}, \mathsf{b} \approx \mathsf{b} \Rightarrow^2_{[\mathtt{t}]} \square \end{aligned}$$

Substitution θ can be normalized to $\theta' = \{x \mapsto \mathsf{b}, y \mapsto \mathsf{a}\}$. Since θ' is also a computed answer substitution of G:

$$\begin{aligned} G &\Rightarrow_{[\mathtt{o}],\,\mathsf{f}(\mathsf{a}) \to \mathsf{b}} y \rhd \mathsf{a}, x \approx \mathsf{b}, \mathsf{g}(x) \approx \mathsf{c}, \mathsf{f}(y) \approx \mathsf{b} \\ &\Rightarrow_{[\mathtt{v}],\,\{y \mapsto \mathsf{a}\}} x \approx \mathsf{b}, \mathsf{g}(x) \approx \mathsf{c}, \mathsf{f}(\mathsf{a}) \approx \mathsf{b} \Rightarrow_{[\mathtt{v}],\,\{x \mapsto \mathsf{b}\}} \mathsf{g}(\mathsf{b}) \approx \mathsf{c}, \mathsf{f}(\mathsf{a}) \approx \mathsf{b} \\ &\Rightarrow_{[\mathtt{o}],\,\mathsf{g}(\mathsf{b}) \to \mathsf{c}} \mathsf{b} \rhd \mathsf{b}, \mathsf{c} \approx \mathsf{c}, \mathsf{f}(\mathsf{a}) \approx \mathsf{b} \Rightarrow^2_{[\mathtt{t}]} \mathsf{f}(\mathsf{a}) \approx \mathsf{b} \\ &\Rightarrow_{[\mathtt{o}],\,\mathsf{f}(\mathsf{a}) \to \mathsf{b}} \mathsf{a} \rhd \mathsf{a}, \mathsf{b} \approx \mathsf{b} \Rightarrow^2_{[\mathtt{t}]} \square, \end{aligned}$$

there is no need to compute the non-normalized solution θ.

We propose a simple marking strategy to avoid computing non-normalized solutions. Before presenting the formal details, we illustrate how this strategy avoids the first refutation in the previous example. We want to compute normalized bindings $x\theta$ and $y\theta$ for the variables x and y. Therefore, we mark the occurrences of x and y in G with the marker $\dagger$. Whenever a marked variable is instantiated we mark the root symbol of the instantiation. So the first step becomes

$$x^\dagger \approx \mathsf{f}(y^\dagger), \mathsf{g}(x^\dagger) \approx \mathsf{c}, \mathsf{f}(y^\dagger) \approx \mathsf{b} \Rightarrow_{[\mathtt{v}],\,\{x \mapsto \mathsf{f}^\dagger(y^\dagger)\}} \mathsf{g}(\mathsf{f}^\dagger(y^\dagger)) \approx \mathsf{c}, \mathsf{f}(y^\dagger) \approx \mathsf{b}$$

Note that only the first occurrence of f in the new goal is marked. The next step is the same as before since the marker is not involved:

$$\mathsf{g}(\mathsf{f}^\dagger(y^\dagger)) \approx \mathsf{c}, \mathsf{f}(y^\dagger) \approx \mathsf{b} \Rightarrow_{[\mathtt{o}],\,\mathsf{g}(\mathsf{b}) \to \mathsf{c}} \mathsf{f}^\dagger(y^\dagger) \rhd \mathsf{b}, \mathsf{c} \approx \mathsf{c}, \mathsf{f}(y^\dagger) \approx \mathsf{b}$$

At this point we do *not* perform the [o] step since the marker on f signals that any instantiation $\mathsf{f}(y\theta)$ of $\mathsf{f}(y)$ should be normalized, but then $\mathsf{f}(y\theta)$ cannot be rewritten to b. Since there are no other applicable inference rules, the derivation ends in failure after just two steps.

The introduction and propagation of the markers is achieved by the modifying the inference rules [i], [d], [v], and [t] of LCNC_ℓ. In the following $t^\dagger$ denotes the term obtained from t by marking its root symbol, provided the root symbol of t unmarked. Otherwise $t^\dagger = t$.

[i] *imitation*

$$\frac{f(s_1,\dots,s_n) \simeq x, G}{(s_1 \simeq x_1,\dots,s_n \simeq x_n, G)\theta} \qquad \frac{f(s_1,\dots,s_n) \simeq x^\dagger, G}{(s_1\theta')^\dagger \simeq x_1^\dagger,\dots,(s_n\theta')^\dagger \simeq x_n^\dagger, G\theta'}$$

with $\simeq \in \{\approx, \rhd, \backsimeq, \lhd\}$, $\theta = \{x \mapsto f(x_1,\dots,x_n), x^\dagger \mapsto f^\dagger(x_1,\dots,x_n)\}$, $\theta' = \{x, x^\dagger \mapsto f^\dagger(x_1,\dots,x_n)\}$, and $x_1,\dots,x_n$ fresh variables,

[d] *decomposition*

$$\frac{f(s_1,\dots,s_n) \simeq f(t_1,\dots,t_n), G}{s_1 \simeq t_1,\dots,s_n \simeq t_n, G} \qquad \frac{f^\dagger(s_1,\dots,s_n) \simeq f(t_1,\dots,t_n), G}{s_1^\dagger \simeq t_1,\dots,s_n^\dagger \simeq t_n, G}$$

$$\frac{f(s_1,\dots,s_n) \simeq f^\dagger(t_1,\dots,t_n), G}{s_1 \simeq t_1^\dagger,\dots,s_n \simeq t_n^\dagger, G} \qquad \frac{f^\dagger(s_1,\dots,s_n) \simeq f^\dagger(t_1,\dots,t_n), G}{s_1^\dagger \simeq t_1^\dagger,\dots,s_n^\dagger \simeq t_n^\dagger, G}$$

for $\simeq \in \{\approx, \rhd\}$,

[v] *variable elimination*

$$\frac{s \rhd x, G}{G\theta} \quad \frac{s \rhd x^\dagger, G}{G\theta'} \quad \frac{s \simeq x}{G\theta'} \quad \frac{s \simeq x^\dagger}{G\theta'} \qquad \simeq \in \{\approx, \backsimeq, \lhd\} \qquad x \notin \mathcal{V}\mathrm{ar}(s)$$

with $\theta = \{x \mapsto s, x^\dagger \mapsto s^\dagger\}$ and $\theta' = \{x, x^\dagger \mapsto s^\dagger\}$,

[t] *removal of trivial equations*

$$\frac{s \simeq t, G}{G} \qquad \simeq \in \{\approx, \rhd\},$$

if s and t are identical after removing all marks.

Note that the imitation rule [i] is not applicable if the root symbol of the non-variable side of the equation is marked.

¿From the completeness proof of LCNC$_\ell$ we obtain several insights to make the calculus more deterministic. Furthermore, and similar to the refinements developed in [2] for the unconditional case, we succeeded in removing almost all non-determinism due to the choice of the inference rule by imposing further syntactic conditions on the participating CTRSs and restricting the set of solutions for which completeness needs to be established.

References

1. H. Ganzinger. Order-sorted completion: The many-sorted way. *Theoretical Computer Science*, 89:3–32, 1991.
2. A. Middeldorp and S. Okui. A deterministic lazy narrowing calculus. *Journal of Symbolic Computation*, 25(6):733–757, 1998.
3. A. Middeldorp, S. Okui, and T. Ida. Lazy narrowing: Strong completeness and eager variable elimination. *Theoretical Computer Science*, 167(1,2):95–130, 1996.
4. A. Middeldorp, T. Suzuki, and M. Hamada. Complete selection functions for a lazy conditional narrowing calculus. *Journal of Functional and Logic Programming*, 2002(3), March 2002.
5. E. Ohlebusch. Termination of logic programs: Transformational methods revisited. *Applicable Algebra in Engineering, Communication and Computing*, 12:73–116, 2001.

The decidability of the first-order theory of the Knuth-Bendix orders in the case of unary signatures

Konstantin Korovin and Andrei Voronkov

University of Manchester
{korovin,voronkov}@cs.man.ac.uk

Introduction of ordering constraints has been one of the main breakthroughs in the saturation based theorem proving. Using solvability of ordering constraints we can dramatically reduce the number of redundant inferences in a resolution-based prover. As a consequence, the problem of solving ordering constraints for the known simplification orders is one of the important problems in the area. A simplification order is a total monotonic order on ground terms. Given such an order we can consider ordering constraints which are quantifier-free formulas in the language of the term algebra with equality and order. Two kinds of orders are mainly used in automated deduction: the Knuth-Bendix orders [5] and various versions of the recursive path orders [3]. Because of its importance, the decision problem for ordering constraints has been well-studied. For the recursive path orders decidability and complexity issues were considered in [4, 1, 11, 12, 10, 9]. For the Knuth-Bendix orders we have the following results: the decidability of constraints [6], a nondeterministic polynomial-time algorithm for constraint solving [7], a polynomial-time algorithm for solving constraints consisting of a single inequality [8].

In resolution-based theorem proving there are important simplifications which allow us to remove clauses form the search space (for example subsumption). It turns out that in order to express applicability conditions for these simplifications, we need to consider constraints which involve first-order quantifiers. Unfortunately the first-order theory of the recursive path orders is undecidable [13, 2]. Only recently the decidability of the first-order theory of recursive path orders in the case of unary signatures has been proven [9]. A signature is called unary if it consists of unary function symbols and constants.

Continuing our work on the Knuth-Bendix orders we prove the decidability of the first-order theory of the Knuth-Bendix orders in the case of unary signatures.

Theorem 1. *The first order theory of any Knuth-Bendix order for a unary signature is decidable.*

Our decision procedure uses interpretation of unary terms as trees and uses decidability of the weak monadic second-order theory of binary trees.

Let us note that the decidability problem for the first-order theory of the Knuth-Bendix orders in the the case of arbitrary signatures remains open.

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. The first-order theory of lexicographic path orderings is undecidable. *Theoretical Computer Science*, 176(1-2):67–87, 1997.
3. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
4. J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. *Lecture Notes in Computer Science*, 510:455–468, 1991.
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 ordering constraint solving is NP-complete. In F. Orejas, P.G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 979–992. Springer Verlag, 2001.
8. K. Korovin and A. Voronkov. Verifying orientability of rewrite rules using the Knuth-Bendix order. In A. Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference, RTA 2001*, volume 2051 of *Lecture Notes in Computer Science*, pages 137–153. Springer Verlag, 2001. An extended version will appear in Information and Computation.
9. Narendran and Rusinowitch. The theory of total unary rpo is decidable. In *First International Conference on Computational Logic*, 2000.
10. Narendran, Rusinowitch, and Verma. RPO constraint solving is in NP. In *CSL: 12th Workshop on Computer Science Logic*. LNCS, Springer-Verlag, 1998.
11. R. Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47:65–69, 1993.
12. R. Nieuwenhuis and J.M. Rivero. Solved forms for path ordering constraints. In *In Proc. 10th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1631 of *Lecture Notes in Computer Science*, pages 1–15, Trento, Italy, 1999.
13. Treinen. A new method for undecidability proofs of first order theories. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 10, 1990.

Decidability and Complexity of *E*-Unifications for some classes of Equational Theories

Christopher Lynch

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

1 Introduction

Most of the work on E-unification has been of two types: (1) Give a general procedure which computes a complete set of unifiers for any equational theory E, or (2) Give a halting procedure that computes a complete set of unifiers for a particular theory E, and analyze its complexity.

The advantage of the first type of result is that the procedure works for every theory. If the procedure is proved to be complete, then we can be guaranteed that if it is given a theory E and a goal $s \approx t$ which is E-unifiable, then it will eventually find a unifier which is more general, modulo E, than any given solution. However, since E-unification is undecidable, there are theories E and goals $s \approx t$ which are not E-unifiable, such that the procedure never halts, and the procedure does not tell us if this goal is E-unifiable.

The advantages and disadvantages of the second type of result are the opposite. We can be guaranteed that the E-unification procedure will halt for any goal. However, this technique only works for one particular theory E. If one examines the recent survey paper on E-unification for example[1], all of the results are of one of these two types.

Our work over the past couple of years has been to pursue a third type of result, which is in between the other two types of results. We aim to find classes of equational theories such that the E-unification problem for every theory in that class in decidable, and then we will analyze the complexity of the procedure.

This type of result should be useful in the area of verification, where a verification procedure calls an E-unification algorithm as a sub-procedure. If we use a general E-unification procedure, then we cannot be assured that the sub-procedure will halt. But we may not want to have a specific procedure, since there may be many theories we need to solve. It may be the case that we do not know the theory ahead of time, but we can be assured that it is in a particular class. In this case, our type of result is what is needed.

Our results are similar to some fairly recent results in the theory of Resolution inference systems. Resolution is a complete procedure for theorem proving in first order logic. For many years, the focus had been on finding ways of making Resolution more efficient, but still making it applicable to all theories, therefore it still will not always halt, as in our type 1 results above. The more recent

** This work was supported by NSF grant number CCR-0098270

results on Resolution decision procedures [2] are of the same type as the results that we are pursuing in the area of E-unification.

Completion-based inference systems are a typical approach to E-unification. These procedures convert the theory E into a convergent set of equations, and apply those equations to the goal equation. Orderings are crucial in completion-based inference systems.

In our research, we decided to investigate a goal-directed inference system for E-unification, instead of a completion-based approach. This is an inference systems where the E-unification problem is given as a goal equation, and the inference system breaks the goal up into subgoals, and the subgoals into further subgoals, etc. A successful result will convert the goal equation into a *normal form* that represents an E-unifier of that equation. Orderings are not used in our goal-directed inference systems.

We will briefly define completion-based and goal-directed inference systems. We will talk about different methods of finding decidable classes of equational theories. Then we will show how to use those methods for both inference systems. Mainly, we are interested in goal-directed inference systems.

2 Inference Systems

First, let us define the problems we are trying to solve. Let E be a finite set of equations.. Let $u \approx v$ be an equation, called the *goal equation*. We write $E \models u \approx v$ if $u \approx v$ is true in any equational model that makes all equations in E true. A substitution θ is an *E-unifier* of $u \approx v$ if $E \models u\theta \approx v\theta$. A set of substitutions Θ is a *complete set of E-unifiers* of $u \approx v$ ($\Theta = CSU_E(u \approx v)$) if every θ in Θ is an E-unifier of $u \approx v$, and for all E-unifiers, σ of $u \approx v$ there is a θ in Θ such that $\theta \leq_E \sigma$.

If a procedure can decide, given E and ground $u \approx v$, if $\models u \approx v$, we say that the procedure *decides the word problem*. If it can decide if $u \approx v$ has an E-unifier, we say it *decides the unification problem*. If the procedure can produces a complete set of E-unifiers of $u \approx v$, we say that it *solves the unification problem*. Notice that an algorithm that solves the unification problem can also decide the unification problem, and an algorithm that decides the unification problem can decide the word problem.

Many of the equational theories E we talk about in this paper are *linear*, i.e., if t is a term appearing in an equation in E then no variable occurs more than once in t. We also consider a set of equations G to be of *varity 1* if no variable appears more than once anywhere in G. For example, $f(x) \approx g(x)$ is linear but not of varity 1.

The three inference rules for Completion-based inference systems are presented below. The first premise in each inference is the equational theory, and the second premise is the goal. The first rule modifies the equational theory, and the other rules modify the goal.

Critical Pair

$$\frac{\{u[s'] \approx v, s \approx t\} \cup E \quad G}{\{u[t]\sigma \approx v\sigma\} \cup E' \quad G}$$

where $\sigma = mgu(s, s')$, $u[s]\sigma \not\preceq v\sigma$, $s\sigma \not\preceq t\sigma$, s' is not a variable. and $E' = \{u[s'] \approx v, s \approx t\} \cup E$.

Narrowing:

$$\frac{\{s \approx t\} \cup E \quad \{u[s'] \approx v\}}{\{s \approx t\} \cup E \quad \{u[t]\sigma \approx v\sigma\}}$$

where $u \not\preceq v$, $s \not\preceq t$, $\sigma = mgu(s, s')$, and s' is not a variable.

Equality Resolution:

$$\frac{E \quad \{u \approx v\}}{E \quad \{\top\}}$$

where $\sigma = mgu(u, v)$.

Any time $\top$ is generated in G, then the composition $\theta_\top$ of all the unifiers leading to it is an E-unifier of the original goal. Let Θ be the set of all such $\theta_\top$. Then Θ is a complete set of E-unifiers for the original goal.

Completion-based inference systems use the Critical Pair rule to modify E. In fact, this rule in itself may not halt. Goal-directed inference systems work directly on the goal, and never modify E. Goal directed inference systems are an extension of the inference rules for standard unification.[1]

Decomposition

$$\frac{\{f(s_1, \cdots, s_n) \approx f(t_1, \cdots, t_n)\} \cup G}{\{s_1 \approx t_1, \cdots, s_n \approx t_n\} \cup G}$$

Variable Elimination:

$$\frac{\{x \approx t\} \cup G}{\{x \approx t\} \cup G[x \mapsto t]}$$

where $x \in G$ and $x \notin t$, and t is not a variable.

Decomposition and Variable Elimination will convert every goal $u \approx v$ into a representation of its most general unifier, if it has one. [2] If $u \approx v$ is not unifiable, then the procedure will halt and that fact will be detected.

To extend these inference rules to deal with an equational theory E, there must be an additional inference rule.

Mutate

$$\frac{\{u \approx v\} \cup G}{\{u \approx s, t \approx v\} \cup G}$$

where $s \approx t \in E$. [3]

[1] Since E is not modified, it is not included in the inference rules.

[2] Note that this normal form is not the standard one, since it allows variable-variable pairs that are not solved [4].

[3] For simplicity, we assume that E is closed under symmetry.

This Mutate rule is also called Lazy Paramodulation and Root Rewriting in places. It is not known whether the above set of inference rules is complete. This is a variant of a famous open question called the Eager Variable Elimination problem[1]. To ensure completeness, we must replace Variable Elimination with the following rule.

Variable Decomposition

$$\frac{\{x \approx f(t_1, \cdots, t_n)\} \cup G}{\{x \approx f(x_1, \cdots, x_n)\} \cup (\{x_1 \approx t_1, \cdots, x_n \approx t_n\} \cup G)[x \mapsto f(x_1, \cdots, x_n)]}$$

where x is a variable.

This rule is also called Imitation and Root Imitation. We give a variant of these inference rules in [4], but this idea is not original to us. See [1] for more references. The goal-directed inference systems referred to later in this paper are all variants of this set.

3 Decidable Equational Classes

To turn these inference systems into decision procedures, we must give a modification of the inference system that is complete, and show that it halts for certain types of theories. We consider three ways of proving that a procedure halts.

1. Give a well-founded ordering, and show that the inference steps reduce this ordering
2. Define a finite set of terms, and show that every term that is generated by the procedure is in this set.
3. After each inference step, mark some positions in the newly generated goal. Forbid inferences at the marked spot. Show that in some finite time, all the positions in the goal will be marked.

To define a class C of equational theories for which some procedure halts, we show that C only inludes theories that meet a specific syntactic criteria, or that C only include theories that are closed under a given inference rule.

3.1 Completion-based Decidability results

The simplest completion-based result is that the word problem is decidable for any theory which is closed under the Critical pair rule. In this case, every narrowing step reduces the size of the goal. The complexity of deciding the word problem depends on the ordering.

Shallow theories are a syntactic class of theories where no variable occurs at more than depth 1. In [8], there is the definition of a class of terms C, such that for inferences in a shallow theory, only terms in C will be created. The word problem is polynomial, and deciding the E-unification problem is NP-complete. A *linear standard theory* is a linear theory where a variable that occurs at greater than depth one can not occur on both sides of an equation. The results for shallow theories are extended to word problems for linear standard theories, and a more general class called standard theories, and for a more general class in [3]. But these methods do not give complexity results.

The *Paramodulation* inference rule is the same as the Critical Pair rule, with the restriction that $u[s] \not\preceq t$ removed. The paper [8] shows that E-unification is decidable in NP theories closed under Paramodulation. Each Narrowing step marks a term in the goal.

3.2 Goal-directed Decidability results

Similar results can be achieved with goal-directed theories, and sometimes better results have been achieved. For linear standard theories, deciding Varity 1 E-unification can be shown to be in P, by the same methods as above.[6]

E-unification for theories closed under paramodulation can be shown to be in NP by the same methods as above. However, if there is no term equivalent to a proper subset of itself, if no two equations in E have the same root symbol on their sides, if no equation has a variable on one side, and if no equation has the same root symbol on both sides, then it is possible to solve E-unification in P, and get a single most general unifier.[7].

There is also a class of theories called Finitely Closable Linear Equational Theories, which is closed under a particular inference rule. The theory of associativity and commutativity is a member of this class. E-unification can be solved in PSPACE for theories in this class. Each inference yields smaller subgoals.[5].

References

1. F. Baader and W. Snyder. Unification Theory. In *Handbook of Automated Reasoning*, ed. A. Robinson and A. Voronkov, 445-533, Elsevier and MIT Press, 2001.
2. C. Fermuller, A. Leitsch, T. Tammet and N. Zamov, *Resolution Methods for the Decision Problem.* Springer, 1993.
3. F. Jacquemard, Ch. Meyer, Ch. Weidenbach. Unification in Extensions of Shallow Equational Theories. In T. Nipkow, ed., *Rewriting Techniques and Applications, 9th International Conference, RTA-98*, LNCS, vol. 1379, Springer, 76-90, 1998.
4. C. Lynch and B. Morawska. Goal Directed E-Unification. In *RTA 12*, ed. A. Middeldorp, LNCS vol. 2051, 231-245, 2001.
5. C. Lynch and B. Morawska. Decidability and Complexity of Finitely Closable Linear Equational Theories. In R. Goré, A. Leitsch and T. Nipkow, eds., *Automated Reasoning. First International Joint Conference, IJCAR 2001*, Vol. 2083 of LNAI, 499-513, Springer, 2001.
6. C. Lynch and B. Morawska. Complexity of Linear Standard Theories. In R. Nieuwenhuis and A. Voronkov, eds., *LPAR 2001*, LNCS Vol. 2250, pp. 188–200, 2001.
7. C. Lynch and B. Morawska. Basic Syntactic Mutation. To appear in A. Voronkov, ed., *CADE 2002*.
8. R. Nieuwenhuis. Basic paramodulation and decidable theories. (Extended abstract), In *Proceedings 11th IEEE Symposium on Logic in Computer Science, LICS'96*, IEEE Computer Society Press, 473-482, 1996.

$ACID$-Unification, Rewrite Reachability and Set Constraints

Siva Anantharaman[1], Paliath Narendran[2*], and Michael Rusinowitch[3]

[1] LIFO - Orlans (France)
[2] University at Albany–SUNY (USA)
[3] LORIA - Nancy (France)

The unification problems for the theories AC (*Associativity-Commutativity*), ACI (*AC plus Idempotence*) and ACU (*AC with Unit element*) have been studied extensively in the past. On the other hand the theories $ACID$ (*ACI plus one Distributive symbol*) or $ACUID$ (*ACID plus a unit element U for addition, null for distribution*) serve as a basis for studying set constraints, but to our knowledge the unification problems over $ACID$ or $ACUID$ have not yet been looked into in detail.

Now, any ground term over an $AC(U)ID$-signature defines a homomorphism w.r.t. the AC-operator, because of the distributivity; therefore, a theory augmenting $AC(U)I$ with a finite family of homomorphisms (w.r.t. '+') is in a sense situated between $AC(U)I$ and $AC(U)ID$. If this family is *non-commuting*, then the theory has been labeled $ACUIh$, and the unification problem proved DEXPTIME-complete in [5]. Our first object of study will thus be the theory $ACUIH^C$, i.e. $ACUI$ plus a family of *commuting* homomorphisms. We show that $ACUIH^C$-unification is undecidable: The proof goes via reducing a reachability problem for commutative word rewriting (with constraints) to $ACUIH^C$-unification. We then show that the halting problem of Minsky counter machines can be reduced to this reachability problem for commutative word rewriting (with constraints).

It is a rather immediate consequence that the $ACID$-unification problem is undecidable when the symbol '$*$' is also assumed AC, in addition to being distributive w.r.t. '+'; this undecidability holds whether or not the ACI-symbol '+' admits a unit element U. If we assume that '$*$' is only associative besides being distributive over '+', then $ACID$-unification is still undecidable: this is established by reduction from the modified Post correspondence problem.

In the case where '$*$' does not satisfy other laws than distributivity, $ACID$-unification can be formulated as a problem of solvability in terms of *finite and non-empty* sets for a particular class of set constraints. We present an approach based on *labeled dag automata* (LDA) to show that $AC(U)ID$-unification is NEXPTIME-decidable. These LDA's are dag automata (as defined in [7]) enriched with labels, which are bit vectors expressing the constraints on the variables of the problem interpreted as set variables. On the other hand, by adapting the proof of the DEXPTIME-hardness of $ACUIh$-unification as given in [5], we

* Research supported in part by NSF grant CCR-9712396 and ONR grant N00014-01-1-0430.

deduce a DEXPTIME lower bound for both the $ACID$- and the $ACUID$- unification problems,

It might be of interest to note that for the particular class of set constraints expressing the $AC(U)ID$-unification problem, if the solutions can be arbitrary sets – i.e. to say possibly infinite, or empty – then satisfiability is again known to be NEXPTIME-decidable (cf. [2]) and DEXPTIME-hard (cf. [8]).

References

1. A. Aiken, E. Wimmers. *Solving Systems of Set Constraints.* In Proc. of the 7th IEEE Symposium on Logic in Computer Science, 1992 (LICS'92), pp. 329 – 340.
2. A. Aiken, D. Kozen,, M. Vardi, E. Wimmers. *The Complexity of Set Constraints.* In Proc. Conf. CSL'93, EACSL, September 1993, pp. 1–18.
3. S. Anantharaman, P. Narendran, M. Rusinowitch. *ACID-Unification, Rewrite Reachability and Set Constraints.* Research Report LIFO:2002-03, Université d'Orléans (Fr.) (`ftp://ftp-lifo.univ-orleans.fr/Users/siva/RR2002-03.ps`)
4. F. Baader. *Unification in Commutative Theories.* Journal of Symbolic Computation 8:479–497, 1989.
5. F. Baader, P. Narendran. *Unification of Concept Terms in Description Logics.* Journal of Symbolic Computation 31 (3): 277–305, 2001.
6. L. Bachmair, H. Ganzinger, U. Waldemann. *Set Constraints are the Monadic Class.* In Proc. of the 8th IEEE Symposium on Logic in Computer Science, 1993 (LICS'93), pp. 75 – 83.
7. W. Charatonik. *Automata on DAG representations of finite trees.* Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany.
8. W. Charatonik, A. Podelski. *Set Constraints with Intersection.* In Proc. of the 12th IEEE Symposium on Logic in Computer Science, Warsaw 1997 (LICS'97), pp. 362–372. (To appear in: *Information and Computation*).
9. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi. *Tree Automata Techniques and Applications.* http://www.grappa.univ-lille3.fr/tata/
10. R. Gilleron, S. Tison, M. Tommasi. *Solving Systems of Set Constraints using Tree Automata.* In Proc. STACS'93, Springer-Verlag, LNCS 665, 505–514.
11. M. Minsky. *Computation: Finite and Infinite Machines. Prentice-Hall International*, London, (1972).
12. J.-M. Talbot, Ph. Devienne, S. Tison. *Generalized Definite Set Constraints.* In CONSTRAINTS: an International Journal 5(1-2):161–202, 2000.

ACU-Unification with a Successor Symbol

Franz Baader[1] and Cesare Tinelli[2]

[1] TU Dresden, Germany, baader@inf.tu-dresden.de
[2] University of Iowa, USA, tinelli@cs.uiowa.edu

Abstract. Unification in the presence of an associative-commutative binary function symbol $+$ with a unit element 0 (ACU-unification) is well-investigated. This note is concerned with what happens if the signature is extended by a successor symbol s that satisfies the usual equation w.r.t. $+$, i.e., $x + \mathsf{s}(y) \equiv \mathsf{s}(x + y)$. We will show that unification in the theory obtained this way is basically the same as ACU-unification with an additional free constant symbol.

1 Introduction and preliminaries

Unification in the presence of an associative-commutative binary function symbol with a unit element (ACU-unification) has attracted considerable attention in unification theory (see [10, 5] among others, and look in [3] for more references) mainly because of its applications in term rewriting [8, 6] and automated deduction [9].

The solvability of elementary ACU-unification problems (where the terms contain only occurrences of the AC symbol, the unit symbol, and variables) is decidable in polynomial time [4], whereas the solvability of ACU-unification problems with (free) constants and of general ACU-unification problems (where the terms may contain additional free function symbols of arbitrary arity) are both NP-complete [7]. With respect to the unification type, ACU is unitary for elementary unification, and finitary for unification with constants and general unification. Elementary ACU-unification and ACU-unification with constants can be reduced to solving linear diophantine equations over the non-negative integers. General ACU-unification can be solved by applying general combination techniques [2].

If the associative-commutative function symbol is supposed to stand for the addition of non-negative integers $+$, then it is quite natural to have an additional unary function symbol s interpreted as the successor function. In addition to the identities satisfied by $+$ and the unit 0:

$$x + (y + z) \equiv (x + y) + z, \tag{1}$$

$$x + y \equiv y + x, \tag{2}$$

$$x + 0 \equiv x, \tag{3}$$

one then has the identity relating $+$ to s:

$$x + \mathsf{s}(y) \equiv \mathsf{s}(x + y). \tag{4}$$

The identities (1) and (2) axiomatize the theory AC, and (1)–(3) axiomatize ACU. We denote with ACU_{s} the theory axiomatized by the identities (1)–(4).

Throughout this paper, we will implicitly appeal to standard results about term rewriting systems modulo an equational theory. The reader is referred to [6] for more details about them. In the following, $\Sigma := \{+, \mathsf{s}, \mathsf{0}\} \cup \{\mathsf{c}_1, \mathsf{c}_2, \ldots\}$ denotes the signature consisting of the symbols $+$, s, and $\mathsf{0}$, together with a countably infinite set of (free) constants. It is well-known that orienting the identities (3) and (4) from left to right yields a canonical term rewriting system R modulo AC. Consequently, every Σ-term t has a computable R-normal form $t\!\downarrow$, and normal forms of ACU_{s}-equivalent terms are equivalent modulo AC. The normal form of any Σ-term t is of the form $t\!\downarrow = \mathsf{s}^m(r)$, where $m \geq 0$ and r is either $\mathsf{0}$ or a sum of variables and free constants.

In the following, we first show that ACU_{s}-unification is basically ACU-unification with an additional free constant c_0. For the convenience of readers not familiar with ACU-unification, we then sketch how an ACU_{s}-unification algorithm can be derived directly.

2 ACU_{s} as ACU with one additional constant

In [1], the notion of equivalence between equational theories over different signatures was introduced. Theories that are equivalent in this sense behave identically with respect to unification.

To define this notion of equivalence, we must first introduce the notion of signature transformation. Given two signatures Σ and Σ', a *signature transformation θ from Σ to Σ'* is a mapping from Σ-terms into Σ'-terms such that

1. $\theta(v) = v$ for every variable v,
2. $\theta(f(t_1, \ldots, t_n)) = (\theta(f(v_1, \ldots, v_n))\sigma$ for $\sigma := \{v_1 \mapsto \theta(t_1), \ldots, v_n \mapsto \theta(t_n)\}$.

It follows from this definition that a signature transformation is uniquely determined by the images of the flat terms $f(v_1, \ldots, v_n)$ where $f \in \Sigma$. For example, with $\Sigma = \{+, \mathsf{s}, \mathsf{0}\} \cup \{\mathsf{c}_1, \mathsf{c}_2, \ldots\}$, let $\Sigma' := \{+, \mathsf{0}\} \cup \{\mathsf{c}_0, \mathsf{c}_1, \mathsf{c}_2, \ldots\}$, where c_i is a constant ($i \geq 0$). We consider the signature transformation θ determined by the identity mapping on terms of the form $v_1 + v_2$ and on all constants in Σ, and by $\theta(s(v_1)) := v_1 + \mathsf{c}_0$. If we apply this transformation to the term $s(v_1 + s(\mathsf{c}_2))$, then we obtain $v_1 + \mathsf{c}_0 + (\mathsf{c}_2 + \mathsf{c}_0)$.

Let E and E' be equational theories over the signatures Σ and Σ', respectively. Then E and E' are *equivalent* iff there exists a signature transformation θ from Σ to Σ' and a signature transformation θ' from Σ' to Σ such that

1. $s =_E t$ implies $\theta(s) =_{E'} \theta(t)$ for all Σ-terms s, t, and
   $s' =_{E'} t'$ implies $\theta'(s') =_E \theta'(t')$ for all Σ'-terms s', t';
2. $\theta'(\theta(s)) =_E s$ for all Σ-terms s, and $\theta(\theta'(s')) =_{E'} s'$ for all Σ'-terms s'.

To show that ACU_{s} and ACU are equivalent (as equational theories over Σ and Σ', respectively), we introduce the signature transformation θ', which is determined by the identity mapping on terms of the form $v_1 + v_2$ and on constants different from c_0, and by $\theta'(\mathsf{c}_0) := \mathsf{s}(0)$. It is easy to see that the given θ and θ' satisfy the requirements of the above definition for $E = ACU_\mathsf{s}$ and $E' = ACU$.

Theorem 1. *ACU_s over Σ and ACU over Σ' are equivalent.*

This shows that ACU_s-unification is the same as ACU-unification with an additional free constant c_0. To unify t_1, t_2 modulo ACU_s, we compute $\theta(t_1)$ and $\theta(t_2)$, and unify these terms modulo ACU.

Since ACU-unification with constants is NP-complete, and the signature transformation θ can be computed in polynomial time, this shows that solvability of ACU_s-unification problems can be decided in NP. In contrast to ACU, however, elementary ACU_s-unification is already NP-hard. This is a consequence of the fact that ACU-unification with just one free constant is already NP-hard (see [3], page 67).

Corollary 1. *The solvability of elementary ACU_s-unification problems and that of ACU_s-unification problems with constants are NP-complete problems.*

Let Γ be an ACU_s-unification problem, and let $\theta(\Gamma)$ be the ACU-unification problem obtained by applying the signature transformation θ to all terms in Γ. Given a finite complete set $\{\sigma_1, \ldots, \sigma_\ell\}$ of ACU-unifiers of $\theta(\Gamma)$, one obtains a finite complete set of ACU_s-unifiers of Γ by applying θ' to every unifier σ_i, where $v\theta'(\sigma_i) := \theta'(v\sigma_i)$. In contrast to ACU, elementary ACU_s-unification is not unitary. This is due to the fact that ACU-unification with one free constant is not unitary, as shown by the unification problem $v_1 + v_2 =^? \mathsf{c}_0$, whose minimal complete set of unifiers consists of $\{v_1 \mapsto \mathsf{c}_0, v_2 \mapsto 0\}$ and $\{v_1 \mapsto 0, v_2 \mapsto \mathsf{c}_0\}$.

Corollary 2. *ACU_s is finitary (and not unitary) both for elementary unification and for unification with constants.*

3 A direct algorithm for ACU_s-unification

Assume that t_1, t_2 are two Σ-terms with variables from the set $\{v_1, \ldots, v_\ell\}$ (for some $\ell \geq 0$) and additional constant symbols $C := \{\mathsf{c}_1, \ldots, \mathsf{c}_{\ell'}\}$. It is easy to see that t_1, t_2 are ACU_s-unifiable iff there is an ACU_s-unifier σ of t_1, t_2 such that $v_j\sigma \in T(\{+, \mathsf{s}, 0\} \cup C, \emptyset)$ for all $j \in \{1, \ldots, \ell\}$.

Let σ be a substitution into ground $(\{+, \mathsf{s}, 0\} \cup C)$-terms. Without loss of generality we may assume that the terms t_1, t_2 as well as the terms $v_j\sigma$ are all in R-normal form, where R is the rewrite system introduced earlier. Under this assumption, it is easy to see that, modulo AC, t_i is of the form

$$t_i = \mathsf{s}^{m_i}(\sum_{j=1}^{\ell} a_{i,j}\, v_j + \sum_{k=1}^{\ell'} b_{i,k}\, \mathsf{c}_k),$$

where $a_{i,j}\, v_j$ and $b_{i,j}\, \mathsf{c}_j$ above abbreviate respectively the sum

$$\underbrace{v_j + \ldots + v_j}_{a_{i,j} \text{ times}} \quad \text{and} \quad \underbrace{\mathsf{c}_k + \ldots + \mathsf{c}_k}_{b_{i,k} \text{ times}}$$

for some $a_{i,j}, b_{i,k} \geq 0$.[1] Analogously, for every $j \in \{1, \ldots, \ell\}$ the term $v_j\sigma$ is of the form

$$v_j\sigma = \mathsf{s}^{p_j}(\sum_{k=1}^{\ell'} d_{j,k}\, \mathsf{c}_k).$$

Now consider the term $(t_i\sigma)\!\downarrow$ for $i = 1, 2$, which has the form $\mathsf{s}^{n_i}(r_i)$ where r_i is either a single constant from $C \cup \{\mathsf{0}\}$ or a sum of constants from C. It is not hard to show that σ is an ACU_{s}-unifier of t_1 and t_2 iff (1) $n_1 = n_2$ and (2) for each $k \in \{1, \ldots, \ell'\}$, the constant c_k occurs in r_1 the same number of times it occurs in r_2. In turn, these two conditions respectively hold iff

1. the tuple $(p_1, \ldots, p_\ell)$ solves the equation
$$a_{1,1}\, y_1 + \cdots + a_{1,\ell}\, y_\ell + m_1 = a_{2,1}\, y_1 + \cdots + a_{2,\ell}\, y_\ell + m_2 \qquad (5)$$
in the variables y_j over the non-negative integers, and
2. for each $k \in \{1, \ldots, \ell'\}$, the tuple $(d_{1,k}, \ldots, d_{\ell,k})$ solves the equation
$$a_{1,1}\, x_{1,k} + \cdots + a_{1,\ell}\, x_{\ell,k} + b_{1,k} = a_{2,1}\, x_{1,k} + \cdots + a_{2,\ell}\, x_{\ell,k} + b_{2,k} \qquad (6)$$
in the variables $x_{j,k}$ over the non-negative integers.

Since the solvability of linear equations over the non-negative integers is decidable (in NP), this shows directly that ACU_{s}-unification with constants is decidable (in NP).[2]

As with ACU-unification, the minimal solutions of the corresponding linear equations can be used to compute a complete set of unifiers of a given ACU_{s}-unification problem. We illustrate this by a simple example (see the papers on ACU-unification for details).

Assume we want to unify the terms $v_1 + \mathsf{s}(v_1) + \mathsf{c}_1$ and $\mathsf{s}(v_2) + \mathsf{s}(\mathsf{c}_2)$ modulo the theory ACU_{s}. The respective normal forms of the two terms to be unified are $t_1 := \mathsf{s}(v_1 + v_1 + \mathsf{c}_1)$ and $t_2 := \mathsf{s}^2(v_2 + \mathsf{c}_2)$. To decide solvability of this unification problem, we must solve three linear equations. One for s:

$$2y_1 + 1 = y_2 + 2 \qquad \text{(derived from (5))},$$

which has the tuple $t_{\mathsf{s}} := (1, 1)$ as (its only) minimal solution; one for the constant c_1:

$$2x_{1,1} + 1 = x_{2,1} \qquad \text{(derived from (6))},$$

which has the tuple $t_{\mathsf{c}_1} := (0, 1)$ as (its only) minimal solution; and one for the constant c_2:

$$2x_{1,2} = x_{2,2} + 1 \qquad \text{(derived from (6))},$$

[1] When every $a_{i,j}$ is zero, the summation $\sum_{j=1}^{\ell} a_{i,j}\, v_j$ stands for the constant $\mathsf{0}$.

[2] If we have more than one equation in the unification problem, then one obtains systems of linear equations.

which has the tuple $t_{\mathsf{c}_2} := (1, 1)$ as (its only) minimal solution. Since all these equations have solutions, the original ACU_{s}-unification problem is solvable.

If we are interested in computing a complete set of unifiers, then we must also consider a linear equation for a (prototypical) variable u in the image of the unifier:

$$2x_1 = x_2,$$

which has the tuple $t_u := (1, 2)$ as (its only) non-trivial minimal solution. These minimal solutions yield a complete set of unifiers consisting of a single unifier, namely: $\sigma := \{v_1 \mapsto \mathsf{s}(\mathsf{c}_2 + u),\ v_2 \mapsto \mathsf{s}(\mathsf{c}_1 + \mathsf{c}_2 + u + u)\}$.

In the example, there is only one unifier in the (minimal) complete set since the equations for the constants and for s had only one minimal solution each. In general, one obtains one unifier for each possible combination of minimal solutions. The fact that the equation for the prototypical variable has only one minimal solution in our example is responsible for the fact that the unifier σ introduces only one variable u. In general, one has one variable u_i for each minimal solution of the equation.

There is, of course, a very close connection between the algorithm sketched in this section and the usual ACU-unification algorithm applied to the transformed terms. The terms $t_1 = \mathsf{s}(v_1 + v_1 + \mathsf{c}_1)$ and $t_2 := \mathsf{s}^2(v_2 + \mathsf{c}_2)$ from our example are transformed into $\theta(t_1) = v_1 + v_1 + \mathsf{c}_1 + \mathsf{c}_0$ and $\theta(t_2) = v_2 + \mathsf{c}_2 + \mathsf{c}_0 + \mathsf{c}_0$. In the ACU-unification algorithm, the equation for the constant c_0 coincides with the equation for s considered above.

References

1. 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.
2. F. Baader and K. U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *J. Symbolic Computation*, 21:211–243, 1996.
3. F. Baader and J. H. Siekmann. Unification Theory. D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 41–125, 1994. Oxford University Press, Oxford, UK.
4. E. Domenjoud. *Outils pour la déduction automatique dans les théories associatives-commutatives.* Thèse de Doctorat, Université de Nancy I, France, 1991.
5. A. Herold and J. H. Siekmann. Unification in Abelian semigroups. *J. Automated Reasoning*, 3:247–283, 1987.
6. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Computing*, 15(4):1155–1194, 1986.
7. D. Kapur and P. Narendran. Complexity of unification problems with associative-commutative operators. *J. Automated Reasoning*, 9:261–288, 1992.
8. G. Peterson and M. E. Stickel. Complete sets of reductions for equational theories with complete unification algorithms. *J. of the ACM*, 28(2):233–264, 1981.
9. G. Plotkin. Building in equational theories. *Machine Intelligence*, 7:73–90, 1972.
10. M. E. Stickel. A unification algorithm for associative commutative functions. *J. of the ACM*, 28(3):423–434, 1981.

A constructive decision procedure for equalities modulo AC

Quang Huy Nguyen

LORIA & INRIA
BP 101, 54602 Villers-lès-Nancy Cedex, France
Quang-Huy.Nguyen@loria.fr

1 Introduction

We provide in this paper an optimised constructive decision procedure for AC equalities based on the *syntacticness* of AC theories. The original motivation for it comes from our work [5] to incorporate term rewriting into the Coq proof assistant [3] using ELAN [7]. The main idea is to perform term rewriting in ELAN and to only use Coq for checking purpose. When considering AC rewriting, proof checking requires an efficient method to prove AC equality in Coq using two axioms of associativity and commutativity or possibly a finite set of equalities derived from them.

Some notations need to be introduced before going into technical detail. Let s, t be two terms and $\mathcal{E}$ be a set of equations between terms, we write $s \rightarrow_{\mathcal{E}} t$ (or $s \rightarrow^{p}_{\mathcal{E}} t$) if there exist an equation $l = r$ in $\mathcal{E}$, a substitution σ and a position p such that $s|_p = l\sigma$ and $t|_p = r\sigma$. When $\mathcal{E}$ is clear from the context, we also write this relation $s \rightarrow^{p} t$. The symmetric closure of $\rightarrow_{\mathcal{E}}$ is denoted by $\leftrightarrow_{\mathcal{E}}$ and the transitive reflexive closure of $\leftrightarrow_{\mathcal{E}}$ is denoted by $\leftrightarrow^{*}_{\mathcal{E}}$ or by $=_{\mathcal{E}}$. By $s =^{?}_{\mathcal{E}} t$ we denote an equality constraint between s and t in $\mathcal{E}$.

Since this word problem is modular [8], we only consider here the elementary case where terms contain one AC symbol $+$ (*i.e.* terms in $\mathcal{T}(\{+\}, \mathcal{X})$). The number of occurrences of $+$ in a term t is called the *weight* of t and is denoted by $weight(t)$. In any equality $s =_{AC} t$, s and t have the same weight. This value is also called the weight of the equality and is denoted by $weight(s =_{AC} t)$. A tem t is said to be in *right associative normal form* (RACF) if $weight(t) = 0$ or t is of the form $t_1 + t_2$ where $weight(t_1) = 0$ and t_2 is in RACF.

The concept of syntactic theories was proposed by Kirchner [1] in order to compute the unification algorithms for a subclass of equational theories with a finite resolvent presentation. A set of axioms $\mathcal{E}$ is said to be *resolvent* if for any equality $t =_{\mathcal{E}} s$, there exists a proof which has at most one application of an axiom at root position (ϵ). An equational theory is *syntactic* if it admits a finite and resolvent presentation. In [2], Kirchner and Klay proved that AC theories are syntactic.

Theorem 1. *[2] If $+$ is an AC symbol we obtain the following finite and resolvent presentation of AC theories:*

$$\mathcal{E}_{AC} = \begin{cases} \textbf{ax\_syn} & w_1 + w_2 = w_1 + w_2 \\ \textbf{ax\_com} & w_1 + w_2 = w_2 + w_1 \\ \textbf{ax\_assoc} & (w_1 + w_2) + w_3 = w_1 + (w_2 + w_3) \\ \textbf{ax\_ac1} & w_1 + (w_2 + w_3) = (w_1 + w_3) + w_2 \\ \textbf{ax\_ac2} & (w_1 + w_2) + w_3 = (w_1 + w_3) + w_2 \\ \textbf{ax\_ac3} & w_1 + (w_2 + w_3) = w_3 + (w_1 + w_2) \\ \textbf{ax\_ac4} & (w_1 + w_3) + (w_2 + w_4) = (w_1 + w_4) + (w_2 + w_3) \end{cases}$$

Each equality in $\mathcal{E}_{AC}$ is an equational consequence of the two axioms of associativity and commutativity. In order to get an efficient decision procedure, we need to reduce as much as possible the search space. Our optimisation comes from the following observation. Any equality $s =_{AC} t$ can be proved by first computing using associativity only, the respective RACF s_A, t_A of s, t, then by proving $s_A =_{AC} t_A$. The proof of equality modulo associativity between a term and its RACF can be *computed* by a deterministic algorithm. Only the proof of $s_A =_{AC} t_A$ needs to be searched, but in a much more simpler set of equalities than $\mathcal{E}_{AC}$. In the next section, we present a *decision procedure* for AC equality of terms in RACF. This procedure is constructive, since its application to any equality modulo AC yields a trace which can be translated into a corresponding proof in Coq. Due to lack of space, most proofs are not given here but these missing proofs can be found in [5].

2 Optimised decision procedure

Let us first show that AC equality of terms in RACF can be proved using only a subset of $\mathcal{E}_{AC}$.

Lemma 1. *Let s, t be two terms in $\mathcal{T}(\{+\}, \mathcal{X})$. If both s and t are in RACF, then none of the four equalities* **ax_assoc**, **ax_ac1**, **ax_ac2** *and* **ax_ac4** *can be applied at root position in any proof of $s =_{AC} t$.*

Lemma 2. *Let $s = s_1 + s_2$ and $t = t_1 + t_2$ be two terms in $\mathcal{T}(\{+\}, \mathcal{X})$. If both s and t are in RACF, then* **ax_com** *can be applied at root position in any proof of $s =_{AC} t$ only if $weight(s_2) = 0$, while* **ax_ac3** *can be applied at root position in any proof of $s =_{AC} t$ only if $weight(s_2) > 0$.*

By Lemma 1 and Lemma 2, one can deduce the set of transformation rules $\mathcal{R}_{AC}$ in Figure 1 for proving AC equality of terms in RACF. These rules transform a *system of constraint* $\exists\overrightarrow{W}.\ E$ where $\overrightarrow{W}$ is a sequence of *existentially bound variables*, while $E = \bigwedge_{i=1}^{n} \alpha_i =^?_{AC} \beta_i$ is a conjunction of constraints. In order to avoid any ambiguity, all bound variable symbols are assumed *to be not in $\mathcal{X}$*. That is, all α_i and β_i are terms in $\mathcal{T}(\{+\}, \mathcal{X} \cup \{\overrightarrow{W}\})$ where $\{\overrightarrow{W}\}$ denotes the set of bound variables in $\overrightarrow{W}$. If both E and $\overrightarrow{W}$ are empty, the system $\exists\overrightarrow{W}.\ E$ is simply denoted by $\mathbb{T}$. The operator $\triangleleft$ adds a new bound variable to a sequence of bound variables.

syn	$\exists\overrightarrow{W}.\ E \wedge s_1 + s_2 =^?_{AC} t_1 + t_2 \mapsto\!\!\!\twoheadrightarrow \exists\overrightarrow{W}.\ E \wedge s_1 =^?_{AC} t_1 \wedge s_2 =^?_{AC} t_2$
com	$\exists\overrightarrow{W}.\ E \wedge s_1 + s_2 =^?_{AC} t_1 + t_2 \mapsto\!\!\!\twoheadrightarrow \exists\overrightarrow{W}.\ E \wedge s_1 =^?_{AC} t_2 \wedge s_2 =_{AC} t_1$ if $weight(s_2) = 0$
acR	$\exists\overrightarrow{W}.\ E \wedge s_1 + s_2 =^?_{AC} t_1 + t_2 \mapsto\!\!\!\twoheadrightarrow \exists\overrightarrow{W} \triangleleft w.\ E \wedge t_2 =^?_{AC} s_1 + w \wedge s_2 =^?_{AC} t_1 + w$ if $weight(s_2) > 0$
replR	$\exists\overrightarrow{W} \triangleleft w.\ E \wedge s =^?_{AC} w \mapsto\!\!\!\twoheadrightarrow \exists\overrightarrow{W}.\ E\{w \mapsto s\}$
replL	$\exists\overrightarrow{W} \triangleleft w.\ E \wedge w =^?_{AC} s \mapsto\!\!\!\twoheadrightarrow \exists\overrightarrow{W}.\ E\{w \mapsto s\}$
trivial	$\exists\overrightarrow{W}.\ E \wedge x =^?_{AC} x \mapsto\!\!\!\twoheadrightarrow \exists\overrightarrow{W}.\ E$

Fig. 1. Set of rules $\mathcal{R}_{AC}$ for AC equality of terms in RACF

Each of the three rules **syn**, **com**, **acR** transforms the rightmost constraint of the system into two new constraints. Moreover, **acR** generates a *new bound variable* which occurs in both two new constraints. The two rules **replR** and **replL** remove the rightmost constraint and instantiate a bound variable in the system by a term which may contain other bound variables. The rule **trivial** removes the trivial constraints in the form of syntactic equalities.

Example 1. The system $x_1 + (x_2 + x_3) =^?_{AC} x_2 + (x_3 + x_1)$ can be reduced to $\mathbb{T}$ as follows.

$$x_1 + (x_2 + x_3) =^?_{AC} x_2 + (x_3 + x_1) \mapsto\!\!\!\twoheadrightarrow_{\mathbf{acR}} \exists w_1.\ x_3 + x_1 =^?_{AC} x_1 + w_1 \wedge x_2 + x_3 =^?_{AC} x_2 + w_1 \mapsto\!\!\!\twoheadrightarrow_{\mathbf{syn}} \exists w_1.\ x_3 + x_1 =^?_{AC} x_1 + w_1 \wedge x_2 =^?_{AC} x_2 \wedge x_3 =^?_{AC} w_1$$
$$\mapsto\!\!\!\twoheadrightarrow_{\mathbf{replR}} x_3 + x_1 =^?_{AC} x_1 + x_3 \wedge x_2 =^?_{AC} x_2 \mapsto\!\!\!\twoheadrightarrow_{\mathbf{trivial}} x_3 + x_1 =^?_{AC} x_1 + x_3$$
$$\mapsto\!\!\!\twoheadrightarrow_{\mathbf{com}} x_3 =^?_{AC} x_3 \wedge x_1 =^?_{AC} x_1 \mapsto\!\!\!\twoheadrightarrow_{\mathbf{trivial}} x_3 =^?_{AC} x_3 \mapsto\!\!\!\twoheadrightarrow_{\mathbf{trivial}} \mathbb{T}$$

In the following, we prove that two terms s, t in RACF are equal modulo AC if and only if there exists a derivation by $\mathcal{R}_{AC}$ which reduces $s =^?_{AC} t$ to $\mathbb{T}$. For that purpose, two properties of a system of constraints have to be defined.

Definition 1. *A system of constraints* $S = \exists\overrightarrow{W}.\ \bigwedge_{i=1}^{n} \alpha_i =^?_{AC} \beta_i$ *is called*

1. *valid iff any bound variable occurs exactly twice in two different constraints;*
2. *satisfactory iff there exists a substitution* σ *from* $\overrightarrow{W}$ *to terms in* $\mathcal{T}(\{+\}, \mathcal{X})$ *such that* $\forall i \leq n:\ \alpha_i\sigma =_{AC} \beta_i\sigma$*;*

The previously defined set of rules $\mathcal{R}_{AC}$ has good properties *w.r.t.* valid and satisfactory systems of constraints, as stated in the two following propositions.

Proposition 1. *Let* s, t *be two terms in* $\mathcal{T}(\{+\}, \mathcal{X})$*. If a system* S *is derived from the system* $s =^?_{AC} t$ *by the rules in* $\mathcal{R}_{AC}$ *then* S *is valid.*

Proposition 2. *Let* S_1 *be a valid system and* S_2 *be a system such that* $S_1 \mapsto\!\!\!\twoheadrightarrow_{\mathcal{R}_{AC}} S_2$*. If* S_2 *is satisfactory then so is* S_1*.*

Theorem 2 (Correctness). *Let s and t be two terms in $\mathcal{T}(\{+\},\mathcal{X})$. If there exists a derivation by the rules in $\mathcal{R}_{AC}$ which reduces $s =^?_{AC} t$ to $\mathbb{T}$ then $s =_{AC} t$.*

Proof. $\mathbb{T}$ is a satisfactory system since it contains no constraint and no bound variable. By the two propositions 1 and 2, the system $s =^?_{AC} t$ must be satisfactory. Since this system contains no bound variable, we have $s =_{AC} t$.

Completeness is proved via the following lemma.

Lemma 3. *For any s, t, u in $\mathcal{T}(\{+\},\mathcal{X})$ such that*

1. *s is in RACF and $weight(t) = 0$;*
2. *$s =_{AC} t + u$;*

and for any non-empty E, there exists a derivation in $\mathcal{R}_{AC}$ which reduces $\exists\overrightarrow{W} \lhd w.\ E \wedge s =^?_{AC} t + w$ to $\exists\overrightarrow{W}.\ E\{w \mapsto v_A\}$ where $v_A =_{AC} u$ and v_A is in RACF.

Theorem 3 (Completeness). *Let s, t be two terms in $\mathcal{T}(\{+\},\mathcal{X})$. If $s =_{AC} t$ and s, t are in RACF then $s =^?_{AC} t$ can be reduced to $\mathbb{T}$ by $\mathcal{R}_{AC}$.*

In order to use $\mathcal{R}_{AC}$ as a decision procedure for equalities modulo AC, we still need to prove the termination of this set of rules. To this end, we define the size of a constraint $s =^?_{AC} t$ by $|s =^?_{AC} t| = size(s) + size(t)$ where

$$size(t) = \begin{cases} 1 & \text{if } t \in \mathcal{X} \\ size(t_1) + size(t_2) & \text{if } t = t_1 + t_2 \end{cases}$$

The termination of $\mathcal{R}_{AC}$ is not trivial. The difficulty is due to the interaction between the first three rules with two replacing rules. If the size of bound variables is defined as that of other symbols, then the first group of rules increases the number of constraints but decreases their size. On the contrary, the second group decreases the number of constraints but increases their size. Hence, we need a different size measurement for bound variables. The size of a bound variable is defined as *the size of the biggest term which may replace it* in the constraint. The size of w in the constraint $\alpha_k =^?_{AC} \beta_k$ of the system $\exists\overrightarrow{W}.\ \bigwedge_{i=1}^{n} \alpha_i =^?_{AC} \beta_i$ is $size(w) = max\{size(v)\}$ such that $v =^?_{AC} w$ or $w =^?_{AC} v$ can be derived from the system $\exists\overrightarrow{W}.\ \bigwedge_{i=1}^{k-1} \alpha_i =^?_{AC} \beta_i \bigwedge_{i=k+1}^{n} \alpha_i =^?_{AC} \beta_i$.

Lemma 4. *Let s, t be two terms in $\mathcal{T}(\{+\},\mathcal{X})$ and w be a bound variables. For any term v such that $v =^?_{AC} w$ or $w =^?_{AC} v$ can be derived from the system $\exists\overrightarrow{W} \lhd w.\ E \wedge s =^?_{AC} t + w$ while $v' =^?_{AC} w$ or $w =^?_{AC} v'$ cannot be derived from $\exists\overrightarrow{W}.\ E$, we have $size(v) < size(s)$.*

Theorem 4 (Termination). *If s and t are two terms in $\mathcal{T}(\{+\},\mathcal{X})$ then $\mathcal{R}_{AC}$ terminates on $s =^?_{AC} t$ for any control.*

Correctness, Completeness and Termination imply that non-deterministically applying the rules in $\mathcal{R}_{AC}$ yields a decision procedure for proving AC equality of terms in RACF. The next result gives the upper bound of the length of proofs and the cost of proof search.

Lemma 5. *Let s and t be two terms in $\mathcal{T}(\{+\}, \mathcal{X})$ which are in RACF and n be the weight of $s =_{AC} t$. The length of any derivation from $s =^?_{AC} t$ to $\mathbb{T}$ is bounded by $2^{n+1} - 1$, while the size of search tree is bounded by $2^{2^n - 1}$.*

3 Concluding remarks

In [5], we also give a system of rules for computing the proof of any equality modulo associativity between a term and its RACF and as a result get a decision procedure for AC equality of terms in $\mathcal{T}(\{+\}, \mathcal{X})$. The proof search procedure is implemented in ELAN which for each equality returns the corresponding derivation by $\mathcal{R}_{AC}$. This derivation is then translated into the calls of tactic which can be efficiently checked in Coq.

In [4], equality modulo AC between two ground terms can be decided by sorting their subterms using the lexicographic path ordering ($\succ_{lpo}$). If n is the weight of the equality then both the number of permutation steps and the number of necessary $\succ_{lpo}$ comparisons are bounded by $n * (n - 1)$. Nevertheless, permutation steps may occur inside terms and hence, require syntactic rewriting which makes proof checking in Coq more costly. With respect to complexity, our method seems to be more expensive. However, in this method, all transformation steps are performed at root position of equalities. This property is important since it allows the transformation steps to be translated into calls of Coq tactic. In other words, no syntactic rewriting is needed and proof checking in Coq is more efficient. In [6], Nipkow gives a system of rules for matching modulo AC including seven rules corresponding to $\mathcal{E}_{AC}$. The completeness is proved using a "geometric interpretation" but no proof of termination is given. In our method, thanks to the optimisation, the number of non-deterministic branches in each step of proof search is reduced from seven to two. This optimisation also allows to simplify the proofs of correctness, soundness and termination since less cases need to be considered.

References

1. Claude Kirchner. Computing unification algorithms. In *Proc. of LICS'86*, pages 206–216, 1986.
2. Claude Kirchner and F. Klay. Syntactic theories and unification. In *Proc. of LICS'90*, pages 270–277, 1990.
3. LogiCal/INRIA. The Coq homepage. `http://coq.inria.fr`.
4. U. Martin and T. Nipkow. Ordered rewriting and confluence. In M.E. Stickel, editor, *Proc. of CADE 10*, volume 449 of *LNCS*, pages 366–380, 1990.
5. Q-H. Nguyen, C. Kirchner, and H. Kirchner. External rewriting in constructive type theory based proof assistants. May 2002. Submitted. Available at `http://www.loria.fr/~nguyenqh/publication/eatr.ps.gz`.
6. T. Nipkow. Proof transformations for equational theories. In *Proc. of LICS'90*, pages 278–288, 1990.
7. PROTHEO/LORIA. The ELAN homepage. `http://elan.loria.fr`.
8. C. Ringeissen. *Combinaison de résolutions de contraintes.* PhD thesis, Université Henri Poincaré - Nancy I, December 1993.

Solution Count for Multiset Unification with Trailing Multiset Variables

Iliano Cervesato

ITT Industries, Inc. — AES Division

1 Motivations and Background

In this extended abstract, we consider a subproblem of AC1 unification [2, 4] applied to multisets [3], where the expressions being unified mention at most one multiset variable. This extends the unification problem of lists in Prolog-like languages with commutativity. We give an upper bound to the number of solutions and outline an algorithm for their generation.

This problem emerged in an attempt to automate the verification of the Group Diffie-Hellman cryptographic protocol [1], where an attacker may combine expressions of the form $g^{x_1 \cdots x_n X}$ to mount an attack against the protocol. Each x_i is a number in $\mathbb{Z}_p$ (p large prime) and X is a product of such numbers of unknown size. We symbolically model such expressions as the multiset of their exponents.

2 The Problem

A *multiset* over a set S is a function $M : S \longrightarrow \mathbb{N}$. The set S is called the *support* of M and will usually be assumed. In analogy with the usual extensional notation for finite sets, where we write $\{a_1, \ldots, a_n\}$ for the set consisting of the elements $a_1, \ldots, a_n$, we will denote a multiset with elements $a_1, \ldots, a_n$ (possibly with replications) as $\wr a_1, \ldots, a_n \int$. We write $|M|$ for the number of (possibly repeated) elements of M.

In this paper, we consider the problem of unifying *multiset expressions* of the form $\wr t_1, \ldots, t_n | R \int$ where t_i is either a constant a from the support set S or an *object variable* X that can be instantiated only to elements of S, and R is either the empty multiset $\wr\int$ or a *trailing multiset variable* whose values range over multisets over S. Multiset expressions where $R = \wr\int$ are called *simple*, and we will abbreviate them as $\wr t_1, \ldots, t_n \int$.

A *multiset equation* is then an expression of the form

$$\wr t_1, \ldots, t_n | R \int \stackrel{?}{=} \wr t'_1, \ldots, t'_{n'} | R' \int$$

where the two sides are multiset expressions. A *unifier* for this equation is a substitution θ that maps object variables to either constants or object variables, and multiset variables to multisets or multiset expressions such that

$$\wr t_1, \ldots, t_n | R \int[\theta] = \wr t'_1, \ldots, t'_{n'} | R' \int[\theta].$$

Given a multiset M, a *sequence* s_M on M is any total ordering of M. For any M with $m = |M|$ elements, there are at most $m!$ distinct sequences on M. This is an exact bound if M has no repeated elements. The number of distinct submultisets of M with n elements (with $0 \leq n \leq m$) is at most $\binom{m}{n} = \frac{m!}{n!\cdot(m-n)!}$. This bound is exact if all elements of M are distinct. This corresponds to the number of ways of picking n items out of a collection containing m distinct objects.

3 Solution Count

A multiset unification problem does not admit a unique unifier, in general, not even when limiting ourselves to most general unifiers. For example, the problem $\wr a, b \int \stackrel{?}{=} \wr X, Y \int$ has two solutions $(X \mapsto a, Y \mapsto b)$ and $(X \mapsto b, Y \mapsto a)$. There are however always finitely many most general unifiers. The purpose of this section is to prove this, and to compute exact worst-case bounds. We distinguish cases on the form of the two sides

3.1 Simple-Simple: $\wr t_1, \ldots, t_n \int \stackrel{?}{=} \wr t'_1, \ldots, t'_{n'} \int$

We consider first the situation in which both sides of the equation are simple terms (no multiset variables are present).

A convenient way to picture this situation, in particular when dealing with more general cases, is to rely on the graphical abstraction at right. We represent each of the t_i's an hollow bullet on the left and each of the t'_i's as a similar bullet on the right:

$$n\left\{\begin{array}{ccc}\circ & & \circ\\ \circ & & \circ\\ \vdots & & \vdots\\ \circ & & \circ\\ \circ & & \circ\end{array}\right\}n'$$

The unification problem is then reduced to drawing edges from the left half of the graph to its right-hand side so that exactly one edge enters/exits each node. The number of solutions depends on the values of n and n':

$\mathbf{n} \neq \mathbf{n'}$**:** 0 solutions. Clearly, if $n \neq n'$, at least one node will be left out.

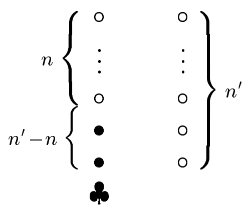
$\mathbf{n} = \mathbf{n'}$**:** $n!$ solutions. In this case, we can pair up the two sets of nodes in $n!$ ways (if we keep the left string fixed, each permutation of the right sequence is a potential solution).

3.2 Simple-General: $\wr t_1, \ldots, t_n | R \int \stackrel{?}{=} \wr t'_1, \ldots, t'_{n'} \int$ General-Simple: $\wr t_1, \ldots, t_n \int \stackrel{?}{=} \wr t'_1, \ldots, t'_{n'} | R' \int$

We now consider the case where one side of the equation is a simple multiset expression, while the other side is unrestricted. By symmetry, we can concentrate on the first possibility.

We extend our graphical reasoning by thinking about R as a magic hat from which we can extract new variables as the need arises. Indeed, if we know that R has at least one element, it is certainly the case that $R = \wr X | R^* \int$, where X and R^* are new variables.

This possibility somewhat relaxes the cardinality constraints of the *simple-simple* case. Since the two multisets must eventually have the same number of elements, we shall be able to pull $n' - n$ new variables out of R. Let us denote them with filled bullets in our graph, and write the residual multiset variable as $\clubsuit$ (clearly, it shall be instantiated with the empty multiset).



We can now count the solutions:

$\mathbf{n} > \mathbf{n}'$**:** 0 solutions. If there are originally more nodes on the left than on the right, there is no way we can obtain a one-to-one matching.

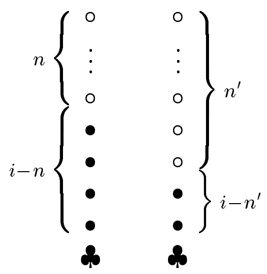
$\mathbf{n} = \mathbf{n}'$**:** $n!$ solutions. If the two side have originally the same number of hollow nodes, then we are essentially in the *simple-simple* situation.

$\mathbf{n} < \mathbf{n}'$**:** $\binom{n'}{n} \cdot n!$ solutions. If there are originally more nodes on the right than on the left, then we can create the missing filled bullets so that the two sides have the same number of nodes. Observe that the number of solutions is not $n'!$: while the precise way in which we relate hollow nodes counts, this does not apply to filled nodes. Indeed, if a and b correspond to hollow nodes, and X and Y to filled nodes, the mappings $(X \mapsto a, Y \mapsto b)$, and $(X \mapsto b, Y \mapsto a)$ are the same solution since X and Y are ultimately part of R.
Therefore, we shall pick $n' - n$ nodes out of n' from the left-hand side to be associated with the filled bullets (or equivalently pick n out n' nodes to be mapped to the hollow bullets). For each possibility, we shall consider all the permutations of the remaining n hollow bullets. Therefore, we obtain the reported total number of solutions. Observe that this situation implies the case where $n = n'$ as $\binom{n}{n} = 1$ for any n.

3.3 General-General: $\wr t_1, \ldots, t_n | R \int \stackrel{?}{=} \wr t'_1, \ldots, t'_{n'} | R' \int$

We will now consider the fully general case where both sides of the equation mention multiset variables. Observe that each t_i can be mapped to a t'_j but can also be related to a yet undefined member of R'. Dually, each t'_i can be associated with either a t_j or with some indistinct element of R. We will proceed by examining the cardinality of the multiset, M say, resulting from the instantiation of the two sides of the equation. For this purpose, we will again align hollow bullets for each t_i and t'_j, and we will extend each alignment with as many filled bullets as needed to achieve the hypothetical cardinality of M. Again, trailing multiset variables are rendered as $\clubsuit$.

First notice that $|M|$ must be at least $\max(n, n')$ for any solution to exist. On the other hand, values of $|M|$ greater than $n + n'$ do not contribute to the solution since this would imply that some of the filled bullets on the left are mapped to filled bullets on the right: they would be subsumed by solutions of cardinality $|M| - 1$ which push back these nodes into the respective $\clubsuit$'s.



Let $\max(n, n') \leq i \leq n + n'$ be the hypothesized cardinality of M. We count solutions as follows: first we shall pick $i - n$ out of n' hollow bullets from the right-hand side and map them to the filled bullets on the left; then we dually pick $i - n'$ out of n hollow bullets on the left-hand side and map them to the filled bullets on the right; finally, we associate the remaining $n + n' - i$ hollow bullets to each other in all possible ways.

The total number of most general solutions is then given by:

$$\sum_{i=\max(n,n')}^{n+n'} \binom{i-n'}{n} \cdot \binom{i-n}{n'} \cdot (n + n' - i)! \quad \text{solutions.}$$

Observe that this case always has a solution. Observe also that this situation subsumes the previous two as soon as we impose a strict bound on the cardinality of the solution multiset.

4 Algorithm

The previous solution count suggests an immediate algorithm for producing these solutions. The cost is clearly very high (exponential), and there are likely to be heuristics and data structures to lower this cost (at least when some constant elements are present).

4.1 Simple-Simple: $\lbag t_1, \ldots, t_n \rbag \stackrel{?}{=} \lbag t'_1, \ldots, t'_{n'} \rbag$

1. If $n \neq n'$, fail, otherwise proceed as follows.
2. Pick a permutation p_j of the indices $1..n$.
3. Construct the system of object equalities

$$S_j = \{t_1 \stackrel{?}{=} t'_{p_j(1)}, \quad \ldots \quad , t_n \stackrel{?}{=} t'_{p_j(n)}\}$$

4. If S_j is solvable add it to the solution set, otherwise discard it.
5. Repeat steps 2 to 4 until all $n!$ permutations of $1..n$ have been examined.

4.2 Simple-General: $\lbag t_1, \ldots, t_n | R \rbag \stackrel{?}{=} \lbag t'_1, \ldots, t'_{n'} \rbag$ General-Simple: $\lbag t_1, \ldots, t_n \rbag \stackrel{?}{=} \lbag t'_1, \ldots, t'_{n'} | R' \rbag$

We will concentrate on the Simple-General case: the General-Simple case is dual.

1. If $n > n'$, fail, otherwise proceed as follows.
2. Pick n elements out of $t'_1, \ldots, t'_{n'}$. Call these elements $\bar{t}'_1, \ldots, \bar{t}'_n$ and call the unselected elements $\hat{t}'_1, \ldots, \hat{t}'_{n'-n}$.
3. Pick a permutation p_j of the indices $1..n$.

4. Construct the system of object equalities
$$S_j = \{t_1 \stackrel{?}{=} \bar{t}'_{p_j(1)}, \quad \ldots \quad , t_n \stackrel{?}{=} \bar{t}'_{p_j(n)}\}$$
5. If S_j is solvable add it together with the multiset variable substitution
$$R \mapsto \wr \hat{t}'_1, \ldots, \hat{t}'_{n'-n} \wr$$
to the solution set, otherwise discard it.
6. Repeat steps 3 to 5 until all $n!$ permutations of $1..n$ have been examined.
7. Repeat steps 2 to 6 until all $\binom{n}{n'}$ ways of picking n out of n' elements of the right-hand side have been examined.

4.3 General-General: $\wr t_1, \ldots, t_n | R \wr \stackrel{?}{=} \wr t'_1, \ldots, t'_{n'} | R' \wr$

1. Set $i = \max(n, n')$.
2. Pick $n + n' - i$ elements out of $t'_1, \ldots, t'_{n'}$. Call these elements $\bar{t}'_1, \ldots, \bar{t}'_{n+n'-i}$ and call the unselected elements $\hat{t}'_1, \ldots, \hat{t}'_{i-n}$.
Similarly, pick $n + n' - i$ elements out of $t_1, \ldots, t_n$. Call these elements $\bar{t}_1, \ldots, \bar{t}_{n+n'-i}$ and call the unselected elements $\hat{t}_1, \ldots, \hat{t}_{i-n'}$.
3. Pick a permutation p_j of the indices $1..(n + n' - i)$.
4. Construct the system of object equalities
$$S^i_j = \{t_1 \stackrel{?}{=} \bar{t}'_{p_j(1)}, \quad \ldots \quad , t_{n+n'-i} \stackrel{?}{=} \bar{t}'_{p_j(n+n'-i)}\}$$
5. If S^i_j is solvable add it together with the multiset variable substitutions
$$R \mapsto \wr \hat{t}'_1, \ldots, \hat{t}'_{i-n} | R^* \wr \quad \text{and} \quad R' \mapsto \wr \hat{t}_1, \ldots, \hat{t}_{i-n'} | R^* \wr$$
to the solution set, otherwise discard it. R^* is a new multiset variable.
6. Repeat steps 3 to 5 until all $(n + n' - i)!$ permutations of $1..(n + n' - i)$ have been examined.
7. Repeat steps 2 to 6 until all $\binom{i-n}{n'} \cdot \binom{i-n'}{n}$ ways of picking $i - n$ out of n' elements on the right-hand side and $i - n'$ out of n elements on the left-hand side have been examined.
8. If $i \leq n + n'$, increment i by 1 and go back to step 1.

References

1. G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key management and friends. In *Proc. Conf. on Computer and Communication Security*. ACM Press, 1998.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
3. A. Dovier, A. Policriti, and G. Rossi. A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms. *Fundamenta Informaticae*, 36(2–3):201–235, 1998.
4. D. Kapur and P. Narendran. Double-exponential complexity of computing a complete set of AC-unifiers. In *Proc. Logic in Computer Science — LICS'92*, pages 11–21, Santa Cruz, CA, 1992.