UNIF 26

26th International Workshop on Unification

Proceedings

July 1, 2012
Manchester, United Kingdom

Santiago Escobar, Konstantin Korovin, Vladimir Rybakov (Eds.)

Preface

This volume contains the informal proceedings of the *26th International Workshop on Unification* (UNIF'2012), held on 1st July 2012 in Manchester, UK.

The International Workshop on Unification was initiated in 1987 as a yearly forum for researchers in unification theory and related fields to meet old and new colleagues, to present recent (even unfinished) work, and to discuss new ideas and trends. It is also a good opportunity for young researchers and researchers working in related areas to get an overview of the current state of the art in unification theory. The list of previous meetings can be found at the UNIF web page: http://www.pps.univ-paris-diderot.fr/~treinen/unif/.

Typically, the topics of interest include (but are not restricted to):

- Narrowing
- Matching
- Applications
- Constraint Solving
- Type reconstruction
- Typed Unification
- Foundations
- Unification in Special Theories
- Disunification
- Unification-Based approaches to Grammar
- General E-unification and Calculi
- Implementations
- Combination problem
- Higher-Order Unification

This year UNIF is a satellite event of the IJCAR 2012 conference which is part of the Alan Turing Year 2012, and collocated with The Alan Turing Centenary Conference. UNIF 2012 will be held on July 1st, at the, University of Manchester, School of Computer Science, UK.

There were seven original contributions to the workshop. The revised versions of these papers are included in this informal proceedings. Each contribution was reviewed by at least three Program Committee members. This volume also includes short abstracts of two invited speakers: Franz Baader from the Technische Universität Dresden, Germany and Emil Jeřábek from the Academy of Sciences of the Czech Republic. We would like to thank them for having accepted our invitation.

We would also like to thank all the members of the Program Committee and all the referees for their careful work in the review process. Finally, we express our gratitude to all members of the local organization of IJCAR 2012, whose work has made the workshop possible. We thank EasyChair for providing friendly environment for handling submissions and creating this proceedings.

June 2012 Santiago Escobar, Konstantin Korovin, Vladimir Rybakov

Organization

Program Chairs

– Santiago Escobar
Universitat Politècnica de València (Spain)
email: sescobar@dsic.upv.es
homepage: http://users.dsic.upv.es/~sescobar/

– Konstantin Korovin
The University of Manchester (UK)
email: korovin@cs.man.ac.uk
homepage: http://www.cs.man.ac.uk/~korovink

– Vladimir Rybakov
Manchester Metropolitan University (UK)
email: V.Rybakov@mmu.ac.uk
homepage: http://www2.docm.mmu.ac.uk/STAFF/V.Rybakov/

Program Committee

Franz Baader	Technische Universität Dresden, Germany
Christoph Benzmüller	Free University Berlin, Germany
Santiago Escobar	Universitat Politècnica de València, Spain
Maribel Fernández	King's College London, UK
Silvio Ghilardi	Università degli Studi di Milano
Rosalie Iemhoff	Utrecht University, The Netherlands
Konstantin Korovin	The University of Manchester, UK
Jordi Levy	IIIA - CSIC, Spain
Christopher Lynch	Clarkson University, USA
George Metcalfe	Universität Bern, Switzerland
Paliath Narendran	SUNY Albany, USA
Vladimir Rybakov	Manchester Metropolitan University, UK

Additional Reviewers

Siva Anantharaman	Barbara Morawska
Andrew Marshall	Stefan Borgwardt

Table of Contents

Unification and Related Problems in Modal and Description Logics 1
Franz Baader

Rules with Parameters in Modal Logic 2
Emil Jeřábek

Recent Advances in Unification for the EL Family 3
Franz Baader, Stefan Borgwardt and Barbara Morawska

Some Notes on Basic Syntactic Mutation 4
Christopher Bouchard, Kimberly Gero and Paliath Narendran

Amissibility and unification in subvarieties of pseudocomplemented lattices 10
Leonardo Manuel Cabrer

On Matching Concurrent Traces 14
Iliano Cervesato, Frank Pfenning, Jorge Luis Sacchini, Carsten Schuermann and Robert Simmons

Unification modulo a property of the El Gamal Encryption Scheme 20
Serdar Erbatur, Santiago Escobar and Paliath Narendran

Bounded Second-Order Unification Using Regular Terms 26
Tomer Libal

Experiments in Admissibility 31
George Metcalfe and Christoph Röthlisberger

Unification and Related Problems in Modal and Description Logics (Invited Talk)

Franz Baader
baader@tcs.inf.tu-dresden.de

Theoretical Computer Science, TU Dresden, Germany

Abstract. Unification modulo equational theories was originally introduced in automated deduction and term rewriting, but has recently also found applications in modal logics and description logics. In this talk, we review problems and results for unification in description logics, and relate them to equational unification and unification in modal logics. Related problems, like disunification, rigid E-unification, and admissibility of inference rules will also be considered.

References

1. Franz Baader and Silvio Ghilardi. Unification in modal and description logics. *Logic Journal of the IGPL*, 19(6):705–730, 2011.

Rules with Parameters in Modal Logic (Invited Talk)

Emil Jeřábek
jerabek@math.cas.cz

Institute of Mathematics of the Academy of Sciences, Czech Republic

Abstract. While admissible rules and unification are fairly well understood in transitive modal logics, rules with parameters have so far received less attention. We know from the work of Rybakov that admissibility of rules with parameters is decidable and complete sets of unifiers are computable for basic transitive modal logics. In this talk, we will discuss other aspects of rules with parameters in basic transitive logics, such as the computational complexity of admissibility and unification, and bases of admissible rules.

Recent Advances in Unification for the $\mathcal{EL}$ Family

Franz Baader, Stefan Borgwardt, and Barbara Morawska

Theoretical Computer Science, TU Dresden, Germany
{baader,stefborg,morawska}@tcs.inf.tu-dresden.de

Abstract

Unification in Description Logics (DLs) has been proposed as an inference service that can, for example, be used to detect redundancies in ontologies. For the DL $\mathcal{EL}$, which is used to define several large biomedical ontologies, unification is NP-complete. Several algorithms that solve unification in $\mathcal{EL}$ have previously been presented. In this paper, we summarize recent extensions of these algorithms that can deal with general concept inclusion axioms (GCIs), role hierarchies ($\mathcal{H}$), and transitive roles (R^+). For the algorithms to be complete, however, the ontology consisting of the GCIs and role axioms needs to satisfy a certain cycle restriction.

1 Introduction and Preliminaries

The description logic (DL) $\mathcal{EL}$ offers the constructors conjunction ($C \sqcap D$), existential restriction ($\exists r.C$), and the top concept ($\top$) to build concept descriptions, starting with a set of concept names $\mathsf{N_C}$ and role names $\mathsf{N_R}$. Although quite inexpressive compared to other DLs, $\mathcal{EL}$ is used to define biomedical ontologies, such as the large medical ontology SNOMED CT.[1] From the computational point of view, $\mathcal{EL}$ has the advantage over more expressive DLs that important inference problems, such as the subsumption problem, are polynomial, even in the presence of background knowledge formulated using so-called general concept inclusion axioms [12]. The $\mathcal{EL}$ family of description logics consists of several logics that extend $\mathcal{EL}$ by means of expressiveness that are useful for defining medical ontologies, but which do not increase the complexity of reasoning [7].

In all logics of the $\mathcal{EL}$ family, concept descriptions are interpreted by *interpretations* $\mathcal{I}$ as subsets of a *domain* $\Delta^{\mathcal{I}}$. Each concept name A is assigned a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name r a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Complex concept descriptions are then interpreted as follows: $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, and $(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$. For example, the concept description $\mathsf{Patient} \sqcap \exists \mathsf{finding}.(\mathsf{Injury} \sqcap \exists \mathsf{location}.\mathsf{Head})$ may be used to describe the set of all patients with a head injury.

The most expressive member of the $\mathcal{EL}$ family of description logics for which unification algorithms are available is $\mathcal{ELH}_{R^+}$. The concept descriptions of $\mathcal{ELH}_{R^+}$ are built in the same way as in $\mathcal{EL}$. The logics differ in the kind of axioms that are allowed in the background ontologies. A *general concept inclusion axiom (GCI)* is of the form $C \sqsubseteq D$ for two concept descriptions C, D and is *satisfied* by an interpretation $\mathcal{I}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A *role inclusion axiom* is of the form $r \circ r \sqsubseteq r$ (*transitivity axiom*) or $r_1 \sqsubseteq r_2$ (*role hierarchy axiom*) and is satisfied by $\mathcal{I}$ if $r^{\mathcal{I}} \circ r^{\mathcal{I}} \subseteq r^{\mathcal{I}}$ or $r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$, respectively. An $\mathcal{ELH}_{R^+}$*-ontology* $\mathcal{O}$ is a finite set of such axioms. Such an ontology is an $\mathcal{EL}$*-ontology* if it contains no role inclusions. An interpretation is a *model* of an ontology if it satisfies all its axioms. Ontologies are used to express background knowledge about an application domain. For example, the GCI

$$\exists \mathsf{finding}.\exists \mathsf{severity}.\mathsf{Severe} \sqsubseteq \exists \mathsf{status}.\mathsf{Emergency} \tag{1}$$

[1] see http://www.ihtsdo.org/snomed-ct/

expresses that every severe finding constitutes an emergency situation and the role inclusion axiom $\mathsf{partOf} \circ \mathsf{partOf} \sqsubseteq \mathsf{partOf}$ says that the role partOf should be interpreted as a transitive binary relation.

In the following, we consider an arbitrary $\mathcal{ELH}_{R^+}$-ontology $\mathcal{O}$. A concept description C is *subsumed* by a concept description D w.r.t. $\mathcal{O}$ (written $C \sqsubseteq_{\mathcal{O}} D$) if every model of $\mathcal{O}$ satisfies the GCI $C \sqsubseteq D$. We say that C is *equivalent* to D w.r.t. $\mathcal{O}$ (written $C \equiv_{\mathcal{O}} D$) if $C \sqsubseteq_{\mathcal{O}} D$ and $D \sqsubseteq_{\mathcal{O}} C$. If $\mathcal{O}$ is empty, we also write $C \sqsubseteq D$ and $C \equiv D$ instead of $C \sqsubseteq_{\mathcal{O}} D$ and $C \equiv_{\mathcal{O}} D$.

Unification

Unification in DLs has been proposed as a tool to detect redundancies in ontologies [11]. For example, assume that the following two concept descriptions were introduced independently into an ontology:

$$\exists \mathsf{finding}.(\mathsf{Head\_injury} \sqcap \exists \mathsf{severity}.\mathsf{Severe}) \tag{2}$$

$$\exists \mathsf{finding}.(\mathsf{Severe\_injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}) \tag{3}$$

The above descriptions are not formally equivalent, nevertheless they are meant to represent the same concept. They can be unified (i.e., made equivalent) by viewing $\mathsf{Head\_injury}$ and $\mathsf{Severe\_injury}$ as variables and substituting them respectively with $\mathsf{Injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}$ and $\mathsf{Injury} \sqcap \exists \mathsf{severity}.\mathsf{Severe}$.

Background knowledge can facilitate unification of concept descriptions. For example, assume that, instead of (3), the concept description

$$\exists \mathsf{finding}.(\mathsf{Severe\_injury} \sqcap \exists \mathsf{finding\_site}.\mathsf{Head}) \sqcap \exists \mathsf{status}.\mathsf{Emergency} \tag{4}$$

occurs in the ontology. The descriptions (2) and (4) are not unifiable. They can, however, be unified (with the same substitution as before) if the GCI (1) is in the background ontology.

To define unification more formally, we assume that the set $\mathsf{N_C}$ is partitioned into *concept variables* ($\mathsf{N_{var}}$) and *concept constants* ($\mathsf{N_{con}}$). A *substitution* σ maps every variable to a concept description and can be extended to concept descriptions in the usual way. A concept description is *ground* if it contains no variables and a substitution is *ground* if all concept descriptions in its range are ground. Similarly, an ontology is *ground* if it contains no variables. In the following, we assume that $\mathcal{O}$ is ground.

A *unification problem* w.r.t. $\mathcal{O}$ is a finite set $\Gamma = \{C_1 \equiv^? D_1, \ldots, C_n \equiv^? D_n\}$ of equations between concept descriptions. A substitution σ is a *unifier* of Γ w.r.t. $\mathcal{O}$ if σ *solves* all the equations in Γ w.r.t. $\mathcal{O}$, i.e. if $\sigma(C_1) \equiv_{\mathcal{O}} \sigma(D_1), \ldots, \sigma(C_n) \equiv_{\mathcal{O}} \sigma(D_n)$. We say that Γ is *unifiable* w.r.t. $\mathcal{O}$ if it has a unifier w.r.t. $\mathcal{O}$. We call Γ w.r.t. $\mathcal{O}$ an $\mathcal{EL}$- or $\mathcal{ELH}_{R^+}$-unification problem depending on whether $\mathcal{O}$ contains role inclusions.

Connection to E-Unification

We can equivalently express unification w.r.t. $\mathcal{ELH}_{R^+}$-ontologies as unification in the equational theory *SLmO* of semilattices with monotone operators, using additional identities to express GCIs and role inclusions [6, 15]. This unification-theoretic point of view sheds some light on our decision to restrict unification to the case of *ground* ontologies. In fact, if we lifted this restriction, then we would end up with an extension of rigid E-unification [14, 13] by a background theory. To the best of our knowledge, such variants of rigid E-unification have not been considered in the literature, and are probably quite hard to solve.

Cycle-Restricted Ontologies

Unfortunately, our unification algorithms are not complete for general ontologies. We call $\mathcal{O}$ *cycle-restricted* if $C \not\sqsubseteq_{\mathcal{O}} \exists w.C$ for every concept description C and every $w \in \mathsf{N}_\mathsf{R}^+$, where $\exists r_1 \dots r_n$ abbreviates $\exists r_1. \dots \exists r_n$. We can show that this condition needs to be checked only for the cases where C is a concept name or $\top$. This allows us to decide in polynomial time whether an $\mathcal{ELH}_{R^+}$-ontology is cycle-restricted [6].

The main reason why we need cycle-restrictedness of $\mathcal{O}$ is that it ensures that a substitution always induces a strict partial order on the variables: For a substitution γ and $X, Y \in \mathsf{N}_\mathsf{var}$, we define

$$X >_\gamma Y \quad \text{iff} \quad \gamma(X) \sqsubseteq_{\mathcal{O}} \exists w.\gamma(Y) \text{ for some } w \in \mathsf{N}_\mathsf{R}^+. \tag{5}$$

If $\mathcal{O}$ is cycle-restricted, this defines a strict partial order. This fact turns out to be an important prerequisite for the proofs of completeness of our algorithms.

2 Unification Algorithms

The basis of all $\mathcal{ELH}_{R^+}$-unification algorithms are lemmata that give recursive characterizations of the relation $\sqsubseteq_{\mathcal{O}}$. We have developed two approaches for proving these characterizations: one based on term rewriting [6] and another one based on a sequent calculus for subsumption [3, 5]. Previous algorithms for $\mathcal{EL}$-unification w.r.t. the empty ontology were based on an even simpler characterization of subsumption that only had to take into account the structure of the compared concept descriptions [8, 9, 10]. Each of the following algorithms is based on one of those earlier algorithms and generalizes it using one of the characterizations from [5] and [6].

Before we can describe the algorithms, we need some additional definitions. A *flat atom* is either a concept name or an existential restriction $\exists r.C'$, where C' is a concept name. We call a concept description *flat* if it is a conjunction of flat atoms. In the following, we restrict both the unification problem Γ and the TBox $\mathcal{T}$ to contain only flat concept descriptions. This restriction is without loss of generality [6].

The Brute-Force Algorithm

The main result underlying all the following $\mathcal{ELH}_{R^+}$-unification algorithms is that $\mathcal{ELH}_{R^+}$-unification is *local*, i.e. every solvable unification problem has a so-called *local unifier*. Let Γ be a flat unification problem and $\mathcal{O}$ be a flat, cycle-restricted $\mathcal{ELH}_{R^+}$-ontology. We will consider the set At_tr, which basically consists of all atoms occurring as subdescriptions in subsumptions in Γ or axioms in $\mathcal{O}$ and some additional flat atoms (see [6] for details). Furthermore, we define the set of *non-variable atoms* by $\mathsf{At}_\mathsf{nv} := \mathsf{At}_\mathsf{tr} \setminus \mathsf{N}_\mathsf{var}$. We call a function S that associates every variable $X \in \mathsf{N}_\mathsf{var}$ with a set $S_X \subseteq \mathsf{At}_\mathsf{nv}$ an *assignment*. For such an assignment S, we define $>_S$ as the transitive closure of $\{(X, Y) \in \mathsf{N}_\mathsf{var} \times \mathsf{N}_\mathsf{var} \mid Y \text{ occurs in an atom of } S_X\}$. We call the assignment S *acyclic* if $>_S$ is irreflexive (and thus a strict partial order). Any acyclic assignment S induces a unique substitution σ_S, which can be defined by induction along $>_S$:

- If X is a minimal element of N_var w.r.t. $>_S$, then we set $\sigma_S(X) := \bigsqcap_{D \in S_X} D$.
- Assume that $\sigma(Y)$ is already defined for all Y such that $X >_S Y$. Then we define $\sigma_S(X) := \bigsqcap_{D \in S_X} \sigma_S(D)$.

We call a substitution σ *local* if it is of this form, i.e., if there is an acyclic assignment S such that $\sigma = \sigma_S$.

In [3], we have shown that any unifiable $\mathcal{EL}$-unification problem has a local unifier. This also holds for $\mathcal{ELH}_{R^+}$-unification problems [5, 6]. Thus, one can test solvability of $\mathcal{ELH}_{R^+}$-unification problem in nondeterministic polynomial time by guessing an acyclic assignment S and then checking whether the induced substitution σ_S is a unifier, using the polynomial time algorithm for subsumption in $\mathcal{ELH}_{R^+}$ [7]. This is a direct extension of the guess-and-test algorithm for $\mathcal{EL}$ without background ontology from [8]. The following two algorithms try to generate acyclic assignments in a more goal-oriented way instead of blindly guessing arbitrary acyclic assignments.

The Rule-Based Algorithm

In [4] and [5], we extended the rule-based algorithm from [10] to deal with $\mathcal{EL}$- and $\mathcal{ELH}_{R^+}$-ontologies, respectively. The main idea underlying these algorithms is to guide the construction of an acyclic assignment by the equivalences of the unification problem Γ. The algorithm works by exhaustive application of certain rules to Γ. These rules can mark certain parts of Γ as solved, create new equivalences to be solved, and extend the current assignment, which is initially empty. Once Γ is completely solved, the current assignment yields a unifier of the original problem.

We show on a simple example how these rules work. Given the equivalence $\exists r.X \equiv^? \exists r.A$, where $X \in \mathsf{N_{var}}$ and $A \in \mathsf{N_{con}}$, we can employ a rule to create the new equivalence $X \equiv^? A$ and mark the old one as solved. Another rule can subsequently solve this smaller equivalence by adding A to S_X. This yields the substitution $X \mapsto A$, which solves the original identity. In contrast, the brute-force algorithm from above would have to guess any local assignment, yielding e.g. the substitution $X \mapsto A \sqcap \exists r.A$, only to realize later that this is not a unifier.

The length of every sequence of rule applications is bounded polynomially in the size of Γ. However, at each point, the algorithm has a nondeterministic choice which rule to apply. To restrict the amount of nondeterminism, several *eager* rules were introduced that are always applied first and leave no choice in their application. All of the rules are triggered by "unsolved parts" of the unification problem, and thus the constructed assignment contains only necessary non-variable atoms. In [4, 5], we added several *Mutation rules* to the original rules from [10] to take into account the axioms of an $\mathcal{ELH}_{R^+}$-ontology.

The Reduction to SAT

In this last approach, we reduce the unification problem to the propositional satisfiability problem [6], which has the advantage that we can employ highly optimized SAT solvers to solve unification problems. Basically, a satisfying propositional valuation yields a local unifier.

The propositional variables are of the form $[C \sqsubseteq D]$ for all atoms C, D of a unification problem Γ. Their intended meaning is as follows: if $[C \sqsubseteq D]$ is true, then the local substitution σ induced by the valuation satisfies $\sigma(C) \sqsubseteq \sigma(D)$. Using these propositional variables, a set of propositional clauses is constructed that (i) encodes Γ, (ii) expresses some relevant properties of subsumption in $\mathcal{ELH}_{R^+}$, and (iii) ensures that the generated assignment is acyclic. A satisfying propositional valuation of these clauses yields an acyclic assignment S, and thus a local substitution, in the following way: S_X contains all non-variable atoms D for which $[X \sqsubseteq D]$ is true. To account for $\mathcal{ELH}_{R^+}$-ontologies, the original reduction from [9] was modified in [6] using the mentioned characterization of subsumption. More precisely, the clauses encoding the properties of subsumption were extended to allow to take GCIs and role inclusions into account.

Consider $\mathcal{O} = \emptyset$ and the example $\exists r.X \equiv^? \exists r.A$ from before. This equivalence is encoded in the clauses $\rightarrow [\exists r.X \sqsubseteq \exists r.A]$ and $\rightarrow [\exists r.A \sqsubseteq \exists r.X]$. The clause $[\exists r.X \sqsubseteq \exists r.A] \rightarrow [X \sqsubseteq A]$ expresses that the subsumption $\exists r.X \sqsubseteq \exists r.A$ can only be solved by *decomposition*, i.e. stripping away the common top-level existential restriction. There are several clauses that prevent $[X \sqsubseteq \exists r.A]$ and $[X \sqsubseteq \exists r.X]$ from being true. Thus, this approach also yields only one unifier, namely $X \mapsto A$.

3 Minimal Unifiers

The brute-force algorithm and the SAT reduction yield *all* local unifiers in the sense that the successful runs of these nondeterministic procedures generate exactly the acyclic assignments that induce unifiers of the unification problem. In contrast, the rule-based algorithm only generates a subset of the local unifiers. However, it is still complete since it generates all *minimal* unifiers. To be more precise, we call a unifier *minimal* if it is minimal w.r.t. the order $\succeq$, where $\sigma \succeq \theta$ iff $\sigma(X) \sqsubseteq_{\mathcal{O}} \theta(X)$ holds for all $X \in \mathsf{N_{var}}$. In fact, locality of unification w.r.t. $\mathcal{O} = \emptyset$ was first shown in [8] by showing that every solvable unification problem has a minimal unifier and that every minimal unifier is local.

Generating exactly the minimal unifiers would be advantageous since there are considerably fewer minimal unifiers than local ones, and they are usually of smaller size. However, the rule-based algorithm also generates unifiers that are not minimal. If we assume a slight generalization of $\succeq$ to $\succeq_{\mathcal{X}}$, where $\succeq_{\mathcal{X}}$ compares the unifiers only w.r.t. a subset $\mathcal{X} \subseteq \mathsf{N_{var}}$, we are able to show [2] that there cannot be an NP-procedure that generates exactly the $\succeq_{\mathcal{X}}$-minimal unifiers in the sense that the successful runs of the procedure on a unification problem Γ generate exactly the acyclic assignments that yield $\succeq_{\mathcal{X}}$-minimal unifiers of Γ. It is still open whether this result also holds for the case of $\succeq = \succeq_{\mathsf{N_{var}}}$.

4 Future Work

The main objective for future work is to find a unification algorithm w.r.t. arbitrary, not necessarily cycle-restricted, $\mathcal{EL}\mathcal{H}_{R^+}$-ontologies. We have implemented the rule-based algorithm and the SAT reduction in our system UEL [1] for the case of acyclic terminologies. We have also modified the SAT reduction into a MaxSAT problem that yields only the minimal unifiers of a unification problem [1]. We plan on further optimizing our implementations and extending them to deal with cycle-restricted ontologies. The main difficulty is that the presence of $\mathcal{EL}\mathcal{H}_{R^+}$-ontologies other than acyclic terminologies increases the nondeterminism of our decision procedures considerably.

References

[1] Franz Baader, Stefan Borgwardt, Julian Alfredo Mendez, and Barbara Morawska. UEL: Unification solver for $\mathcal{EL}$. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Proc. of the 25th Int. Workshop on Description Logics (DL'12)*, volume 846 of *CEUR Workshop Proceedings*, pages 26–36, 2012.

[2] Franz Baader, Stefan Borgwardt, and Barbara Morawska. Computing minimal $\mathcal{EL}$-unifiers is hard. In Thomas Bolander, Torben Brauner, Silvio Ghilardi, and Lawrence Moss, editors, *Advances in Modal Logic 9 (AiML'12)*. College Publications, 2012.

[3] Franz Baader, Stefan Borgwardt, and Barbara Morawska. Extending unification in $\mathcal{EL}$ towards general TBoxes. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'12)*, pages 568–572. AAAI Press, 2012. Short paper.

[4] Franz Baader, Stefan Borgwardt, and Barbara Morawska. A goal-oriented algorithm for unification in $\mathcal{EL}$ w.r.t. cycle-restricted TBoxes. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Proc. of the 25th Int. Workshop on Description Logics (DL'12)*, volume 846 of *CEUR Workshop Proceedings*, pages 37–47, 2012.

[5] Franz Baader, Stefan Borgwardt, and Barbara Morawska. A goal-oriented algorithm for unification in $\mathcal{ELH}_R^+$ w.r.t. cycle-restricted ontologies. In Michael Thielscher and Dongmo Zhang, editors, *Proc. of the 25th Australasian Joint Conf. on Artificial Intelligence (AI'12)*, volume 7691 of *Lecture Notes in Artificial Intelligence*, pages 493–504. Springer-Verlag, 2012.

[6] Franz Baader, Stefan Borgwardt, and Barbara Morawska. SAT encoding of unification in $\mathcal{ELH}_{R^+}$ w.r.t. cycle-restricted ontologies. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Proc. of the 6th Int. Joint Conf. on Automated Reasoning (IJCAR'12)*, volume 7364 of *Lecture Notes in Artificial Intelligence*, pages 30–44. Springer-Verlag, 2012.

[7] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05)*, pages 364–369. Professional Book Center, 2005.

[8] Franz Baader and Barbara Morawska. Unification in the description logic $\mathcal{EL}$. In Ralf Treinen, editor, *Proc. of the 20th Int. Conf. on Rewriting Techniques and Applications (RTA'09)*, volume 5595 of *Lecture Notes in Computer Science*, pages 350–364. Springer-Verlag, 2009.

[9] Franz Baader and Barbara Morawska. SAT encoding of unification in $\mathcal{EL}$. In Christian G. Fermüller and Andrei Voronkov, editors, *Proc. of the 17th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'10)*, volume 6397 of *Lecture Notes in Computer Science*, pages 97–111. Springer-Verlag, 2010.

[10] Franz Baader and Barbara Morawska. Unification in the description logic $\mathcal{EL}$. *Logical Methods in Computer Science*, 6(3), 2010.

[11] Franz Baader and Paliath Narendran. Unification of concept terms in description logics. *Journal of Symbolic Computation*, 31(3):277–305, 2001.

[12] Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and - what else? In Ramon López de Mántaras and Lorenza Saitta, editors, *Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI'04)*, pages 298–302. IOS Press, 2004.

[13] Anatoli Degtyarev and Andrei Voronkov. The undecidability of simultaneous rigid E-unification. *Theoretical Computer Science*, 166(1–2):291–300, 1996.

[14] Jean Gallier, Paliath Narendran, David Plaisted, and Wayne Snyder. Rigid E-unification: NP-completeness and applications to equational matings. *Information and Computation*, 87(1–2):129–195, 1990.

[15] Viorica Sofronie-Stokkermansmans. Locality and subsumption testing in $\mathcal{EL}$ and some of its extensions. In Carlos Areces and Robert Goldblatt, editors, *Advances in Modal Logic 7 (AiML'08)*, pages 315–339. College Publications, 2008.

Some Notes on Basic Syntactic Mutation

Christopher Bouchard*, Kimberly A. Gero, and Paliath Narendran*

University at Albany–SUNY (USA)
{cbou,dran,kgero001}@cs.albany.edu

1 Introduction

Unification modulo an equational theory E (equational unification or E-unification) is an undecidable problem in general. Even in cases where it is decidable, it is often of high complexity. In their seminal paper "Basic Syntactic Mutation" Christopher Lynch and Barbara Morawska present syntactic criteria on equational axioms E that guarantee a polynomial time algorithm for the corresponding E-unification problem. As far as we know these are the only purely syntactic criteria that ensure a polynomial-time algorithm for unifiability. Our goal initially was to extend the Lynch-Morawska result *for convergent term rewriting systems* by relaxing their constraints, while still maintaining the polynomial time algorithm guarantee. However, we observed that their constraints were tight in the sense that relaxing any one of them would give rise to unification problems that are not in **P** (unless P = NP). We provide proofs that removing any of their constraints will lead to unification problems that are not in **P**.

We also investigate one of the computational issues raised by the Lynch-Morawska paper, namely, checking whether a convergent term rewriting system is *subterm-collapsing* — a term rewriting system is subterm-collapsing if and only if there is a term that is congruent to a proper subterm of itself. We show the undecidability of subterm collapse even for convergent term rewriting systems that satisfy the Lynch-Morawska conditions [7]. (For general convergent systems, this result was shown by Bürkert, Herold and Schmidt-Schauß [5].)

This is a preliminary report of our ongoing research. More details are given in [4].

2 Notation and Preliminaries

We assume the reader is familiar with the usual notions and concepts in term rewriting systems [2] and equational unification [3]. We consider rewrite systems over ranked signatures, usually denoted Σ, and a possibly infinite set of variables, usually denoted $\mathcal{X}$. The set of all terms over Σ and $\mathcal{X}$ is denoted as $T(\Sigma, \mathcal{X})$. An equation (Σ-identity) is an ordered pair of terms (s, t), usually written as $s \approx t$. Here s is the left-hand side and t is the right-hand side of the equation [2]. A rewrite rule is an equation $s \approx t$ where $Var(t) \subseteq Var(s)$, usually written as $s \to t$. A term rewriting system is a set of rewrite rules.

* Research supported in part by NSF grant CNS-0905286.

A set of equations E is *subterm-collapsing* if and only if there are terms t and u such that t is a proper subterm of u and $E \vdash t \approx u$ (or $t =_E u$) [5][1]. A term rewriting system is convergent if and only if it is confluent and terminating [2].

Given a set of equations E, the set of ground instances of E is denoted by $Gr(E)$. An equation e is *redundant in* E if and only if every ground instance e' of e is a consequence of equations in $Gr(E)$ which are smaller than e' modulo the ordering we use to show termination [7].

Following Hermann [6] the *forward-closure* of a convergent term rewriting system R is defined in terms of the following operation on rules in R: let $\rho_1 : l_1 \to r_1$ and $\rho_2 : l_2 \to r_2$ be two rules in R and let $p \in \mathcal{FP}os(r_1)$. Then

$$\rho_1 \rightsquigarrow_p \rho_2 = \sigma(l_1 \to r_1[r_2]_p)$$

where $\sigma = mgu(r_1|_p =^? l_2)$. Given rewrite systems R_1 and R_2 such that $R_2 \subseteq R_1$, we define $R_1 \rightsquigarrow R_2$ as the rules in

$$\{(l_1 \to r_1) \rightsquigarrow_p (l_2 \to r_2) \mid (l_1 \to r_1) \in R_1,\ (l_2 \to r_2) \in R_2 \text{ and } p \in \mathcal{FP}os(r_1)\}$$

which are not redundant in R_1.

We now define $FC_0(R) = R$ and $FC_{k+1}(R) = FC_k(R) \cup (FC_k(R) \rightsquigarrow R)$ for all $k \geq 0$. Finally,

$$FC(R) = \bigcup_{i=1}^{\infty} FC_i(R).$$

A set of rewrite rules R is forward-closed if and only if $FC(R) = R$.

Lynch and Morawska also define a *right-hand-side critical pair*. Since our focus is only on convergent term rewriting systems, this definition can be modified as

$$\frac{s \to t \qquad u \to v}{s\sigma \approx u\sigma}$$

where $\sigma = mgu(v, t)$ and $s\sigma \neq u\sigma$.

For an equational theory E, *RHS(E)* = { e | e is the conclusion of a Right-Hand-Side Critical Pair inference of two members of E } $\cup$ E [7].

A set of equations E is *quasi-deterministic* if and only if

1. No equation in E has a variable as its left-hand side or right-hand side,
2. No equation in E has the same root symbol on both the left-hand side and the right-hand side, and
3. No two equations of E have the same root symbols at their sides.

E is *deterministic* if and only if it is quasi-deterministic and non-subterm-collapsing.

[1] Non-subterm-collapsing theories are called *simple* theories in [5]

3 Lynch-Morawska Conditions

Given a confluent and terminating term rewriting system R, there are three conditions that must hold to maintain a polynomial time algorithm guarantee:

(i) R is non-subterm-collapsing,
(ii) R is forward-closed, and
(iii) $RHS(R)$ is quasi-deterministic.

In this section we will show that if any one of these conditions is relaxed there is no longer a polynomial time guarantee. Therefore the conditions given in the Lynch-Morawska paper are tight.

3.1 The case where R is subterm-collapsing

Consider the following single-rule ground term rewriting system R_1, where f is a binary function symbol and $c, 0, 1$ are constants.

$$f(0, f(0, f(1, c))) \to c$$

R_1 is forward-closed and $RHS(R_1)$ is quasi-deterministic. However, it is subterm-collapsing. The unification problem can be shown to be NP-hard by a reduction from monotone 1-in-3-SAT [2], which is known to be NP-complete.

For each clause $C_i = p_i \vee q_i \vee r_i$ we form the following equation, where Z_i is a variable that varies with each clause.

$$f(X_{p_i}, f(X_{q_i}, f(X_{r_i}, Z_i))) =^?_{R_1} Z_i$$

The unification problem S is the set of all these equations. Every unifier of $f(X_{p_i}, f(X_{q_i}, f(X_{r_i}, Z_i))) =^?_{R_1} Z_i$ replaces exactly one of X_{p_i}, X_{q_i}, X_{r_i} by 1 and the others by 0. In fact,

$$\sigma_1 = \{X_{p_i} \mapsto 0, X_{q_i} \mapsto 0, X_{r_i} \mapsto 1, Z_i \mapsto c\}$$
$$\sigma_2 = \{X_{p_i} \mapsto 1, X_{q_i} \mapsto 0, X_{r_i} \mapsto 0, Z_i \mapsto f(1, c)\}$$
$$\sigma_3 = \{X_{p_i} \mapsto 0, X_{q_i} \mapsto 1, X_{r_i} \mapsto 0, Z_i \mapsto f(0, f(1, c))\}$$

are the only normalized unifiers for the above equation. Clearly, then, S is unifiable iff C has a solution.

This (the ground term rewriting system R_1) solves an open problem posed by Lynch and Morawska [7] which asked if is there a system which is quasi-deterministic but subterm-collapsing, for which there is no polynomial time algorithm.

3.2 R is not forward-closed

$$R_2 = \{\, B(x, y) * B(u, v) \;\to\; B(x * u, y * v) \,\}$$

is deterministic (i.e quasi-deterministic and non-subterm-collapsing), but not forward-closed. There are no right-hand-side critical pairs, so $RHS(R_2) = R_2$. The unification problem is undecidable [1]. (Note that every forward-closed convergent system has a unification problem that is decidable in NP [7].)

3.3 $RHS(R)$ is not deterministic

The following system R_3 is non-subterm-collapsing and forward-closed. However $RHS(R_3)$ is not deterministic.

$$\begin{aligned} f(0,0,1) &\to c_1 & g(0,0,1) &\to c_1 \\ f(0,1,0) &\to c_2 & g(0,1,0) &\to c_2 \\ f(1,0,0) &\to c_3 & g(1,0,0) &\to c_3 \end{aligned}$$

where $c_1, c_2, c_3, 0$ and 1 are constants. Note that R_3 is deterministic, but $RHS(R_3)$ is not. Unifiability modulo R_3 can also be shown to be NP-hard by a reduction from monotone 1-in-3-SAT. We only present the key idea here.

For each proposional variable p we form a respective term variable V_p. For each clause C_i we form the following equation $\mathcal{EQ}_i$.

$$f(V_{p_i}, V_{q_i}, V_{r_i}) \ =^?_{R_2} \ g(V_{p_i}, V_{q_i}, V_{r_i})$$

It is not hard to see that the only unifiers of $\mathcal{EQ}_i$ are:

$$\begin{aligned} \sigma_1 &= \{V_{p_i} \mapsto 0, V_{q_i} \mapsto 0, V_{r_i} \mapsto 1\} \\ \sigma_2 &= \{V_{p_i} \mapsto 0, V_{q_i} \mapsto 1, V_{r_i} \mapsto 0\} \\ \sigma_3 &= \{V_{p_i} \mapsto 1, V_{q_i} \mapsto 0, V_{r_i} \mapsto 0\} \end{aligned}$$

4 Undecidability of Subterm-Collapse

For general convergent systems, this result was shown by Bürkert, Herold and Schmidt-Schauß [5]. We show that the property of subterm-collapsing is undecidable even when the convergent system R satisfies the other conditions of determinism, namely forward-closure and quasi-determinism of $RHS(R)$.

The proof is by reduction from the halting problem for reversible deterministic 2-counter Minsky machines, which is undecidable since reversible deterministic 2-counter Minsky machines are Turing-universal [8]. We use the same notation as Morita in [8].

Given such a machine M and configurations (q_0, k, p) and (q_L, k', p'), we construct a term rewriting system R_M. This system is forward-closed, and $RHS(R_M)$ is quasi-deterministic. We then show that R_M is subterm-collapsing if and only if the machine M, starting in configuration (q_0, k, p), will halt with (q_L, k', p') as its final configuration.

Our system is over the signature $\Sigma = \bigcup_{i=0}^{L}\{ q_i, f_i, f'_i \} \cup \{ 0, e, e', c, s, f, g, g' \}$ where 0, e, e', and $q_0, \ldots, q_L$ are constants, c has arity 4, every other symbol has arity 1. We encode a natural number n as a term $s^n(0)$. A state q_i will be encoded by a constant q_i. We can then encode a configuration (q_i, k, p) as a term $c(q_i, s^k(0), s^p(0), s^n(0))$, where n is the number of steps the machine has taken. We use the f and g symbols, with various subscripts and primes, to ensure termination and foward-closure of the resulting rewrite system.

To start, initialize R_M as

$$\begin{aligned} R_M \leftarrow \{\, & f(c(e, s^k(0), s^p(0), 0)) \to c(q_0, s^k(0), s^p(0), 0), \\ & f_L(c(q_L, s^{k'}(0), s^{p'}(0), z)) \to g(c(e', 0, 0, z)), \\ & g'(g(c(e', 0, 0, s(z)))) \to c(e', 0, 0, z), \\ & g'(g(c(e', 0, 0, 0))) \to e \,\} \end{aligned}$$

The first rule encodes initializing the machine. The second terminates the machine iff the configuration matches the given final configuration. The third checks that the step-counter encodes a valid natural number. The fourth rewrites a completely empty configuration to e, which could cause a subterm-collapse.

Next we encode the transition rules. For each quadruple in δ, extend R_M using one of the following transformations:

(a) $[\, q_i, 1, P, q_j \,]$: $R_M \leftarrow R_M \cup \{\, f_i(c(q_i, s(x), y, z)) \to c(q_j, s(x), y, s(z)) \,\}$
(b) $[\, q_i, 1, Z, q_j \,]$: $R_M \leftarrow R_M \cup \{\, f_i'(c(q_i, 0, y, z)) \to c(q_j, 0, y, s(z)) \,\}$
(c) $[\, q_i, 1, +, q_j \,]$: $R_M \leftarrow R_M \cup \{\, f_i(c(q_i, x, y, z)) \to c(q_j, s(x), y, s(z)) \,\}$
(d) $[\, q_i, 1, 0, q_j \,]$: $R_M \leftarrow R_M \cup \{\, f_i(c(q_i, x, y, z)) \to c(q_j, x, y, s(z)) \,\}$
(e) $[\, q_i, 1, -, q_j \,]$: $R_M \leftarrow R_M \cup \{\, f_i(c(q_i, s(x), y, z)) \to c(q_j, x, y, s(z)) \,\}$

For quadruples with 2 as their second argument, swap the second and third arguments of the c-terms.

Lemma 1 *Given a reversible deterministic 2-counter Minsky machine M, the system R_M has no RHS overlaps.*

Lemma 2 *The system R_M is forward-closed and quasi-deterministic.*

Lemma 3 *Given a reversible deterministic 2-counter Minsky machine M and configurations (q_0, k, p) and (q_L, k', p'), and rewrite system R_M constructed as above, R_M is subterm-collapsing if and only if the machine M, starting in configuration (q_0, k, p), will halt with (q_L, k', p') as its final configuration.*

Theorem 4. *It is undecidable, given a rewrite system R which is forward-closed and for which $RHS(R)$ is quasi-deterministic, whether $RHS(R)$ is subterm-collapsing.*

References

1. S. Anantharaman, S. Erbatur, C. Lynch, P. Narendran, M. Rusinowitch. "Unification modulo Synchronous Distributivity". Technical Report SUNYA-CS-12-01, Dept. of Computer Science, University at Albany—SUNY, 2012. Available at `www.cs.albany.edu/~ncstrl/treports/Data/README.html` (An abridged version to be presented at *IJCAR 2012.*)
2. F. Baader, T. Nipkow. *Term Rewriting and All That.* Cambridge Univ Press, 1999.
3. F. Baader, W. Snyder. "Unification Theory". *Handbook of Automated Reasoning*, pp. 440–526, Elsevier Science Publishers B.V., 2001.

4. C. Bouchard, K. Gero, P. Narendran. "Notes on Basic Syntactic Mutation". Technical Report SUNYA-CS-12-03, Dept. of Computer Science, University at Albany—SUNY. http://www.cs.albany.edu/ ncstrl/treports/Data/README.html.
5. H-J. Bürckert, A. Herold, M. Schmidt-Schauß. On Equational Theories, Unification, and (Un)Decidability. *Journal of Symbolic Computation* 8(1/2): 3-49 (1989).
6. M. Hermann. Chain properties of rule closures. *Formal Aspects of Computing* 2 207–225 (1990).
7. C. Lynch, B. Morawska. Basic Syntactic Mutation. In: *Proc. of CADE 2002* (A. Voronkov, ed.), LNCS 2392, pages 471–485.
8. K. Morita. Universality of a reversible two-counter machine. *Theoretical Computer Science* 168 303–320 (1996).

Admissibility and unification in subvarieties of pseudocomplemented lattices

Leonardo Manuel Cabrer

Institute of Mathematics
University of Oxford

Unification type and admissible rules have been studied for intermediate logics and their fragments by various authors [12], [9], [13], [15], [5] , [3] , [6], [2], [7], to mention some of them.

In this paper we initiate the study of admissible quasi-equations, admissible clauses and unification types of equational theories extending the equational theory of the $\{\neg, \bot\}$-fragment of propositional intuitionistic logic. This is the equational theory of the the variety of distributive pseudocomplemented lattices (p-lattices, for short). We calculate the unification type of all the subvarieties of p-lattice and we determine that only 3 of them are structurally complete. We also present basis for admissible quasi-equations and admissible clauses for some of the non structurally complete subvarieties of p-lattice. Finally, we determine the unification type of a particular unification problem in each subvariety of pseudocomplemented lattices, obtaining complete classifications for some particular subvarieties.

The main tools used in this paper are: First, the topological duality for bounded distributive lattices presented in [10], and its restriction to p-lattices developed in [11]; second, the characterization of subvarieties of p-lattices given in [8] and the description of finite projective p-lattices in these subvarieties given in [14]; and finally the algebraic approach to E-unification developed in [4].

An algebra $\mathbf{A} = (A, \wedge, \vee, \neg, 0, 1)$ is said to be a *distributive pseudocomplemented lattice* if $(A, \wedge, \vee, 0, 1)$ is a bounded distributive lattice and $\neg(a)$ is the maximal element of the set $\{b \in A \mid b \wedge a = 0\}$ for each $a \in A$. In what follows $\mathcal{PL}$ will denote the variety of p-lattices.

Let $\mathbf{B}_n = (B_n, \wedge, \vee, ^*, \bot, \top)$ denote the finite boolean algebra with n atoms and let $\mathbf{B}'_n$ be the pseudocomplemented lattice obtained by adding a fresh top 1 to $\mathbf{B}_n$ and defining the negation in $\mathbf{B}'_n$ as follows: $\neg(\bot) = 1$, $\neg(1) = \bot$ and $\neg(a) = a^*$ otherwise. We denote $\mathcal{B}_n = \mathbb{ISP}(\mathbf{B}'_n)$. In [8], Lee proved that for each $n = \{0, 1, \ldots\}$, $\mathcal{B}_n$ is a variety and that every non trivial proper subvariety of $\mathcal{PL}$ coincides with some $\mathcal{B}_n$. Observe that $\mathcal{B}_0$ and $\mathcal{B}_1$ are the varieties of Boolean algebras and Stone algebras respectively.

Unification:

In [4], Ghilardi proved that the variety $\mathcal{PL}$ has unification type 0. In this work we extend that result proving that every proper subvariety of $\mathcal{PL}$ different from the class of Boolean algebras has unification type 0. Precisely:

Theorem 1 *Let $\mathcal{V}$ be a non trivial subvariety of $\mathcal{PL}$, then the following hold:*

$$\text{Type}(\mathcal{V}) = \begin{cases} 1 & \text{if } \mathcal{V} = \mathcal{B}_0, \\ 0 & \text{otherwise.} \end{cases}$$

We also prove that there are no infinitary unification problems in any of the subvarieties of $\mathcal{PL}$, and we present an algorithm to determine the type of a unification problem in the varieties of Stone algebras and $\mathcal{B}_2$.

Admissibility:

In the case of equational theories, single-conclusion rules correspond to quasi-identities and multiple-conclusion rules correspond to clauses (implications between conjunctions of identities and disjunctions of identities). Admissibility then amounts to the validity of quasi-identities or clauses in the free algebra on countably infinitely many generators of the quasi-variety determined by the equational theory.

A quasi-variety $\mathcal{V}$ is called *structurally complete* (*universally complete*) if each admissible quasi-identity (clause) of $\mathcal{V}$ is satisfied by all members of $\mathcal{V}$.

It is well-known that the variety of Boolean algebras is structurally complete, and it can be easily seen that it is universally complete too. In [1], we have proven that the variety $\mathcal{B}_1$ of Stone algebras also is universally complete. We extend these results as follow:

Theorem 2 *Let $\mathcal{V}$ be a non trivial subvariety of $\mathcal{PL}$, then the following hold:*

(i) $\mathcal{V}$ is structurally complete iff $\mathcal{V} \in \{\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2\}$.

(ii) $\mathcal{V}$ is universally complete iff $\mathcal{V} \in \{\mathcal{B}_0, \mathcal{B}_1\}$.

The theorem above proves that all the subvarieties of p-lattices containing $\mathcal{B}_3$ ($\mathcal{B}_2$) have admissible quasi-equations (clauses) that are not valid. We started to determine axiomatizations for admissible quasi-equations and admissible clauses for subvarieties above $\mathcal{B}_2$. We summarize the results obtained so far in the following:

Theorem 3 *The admissible clauses of $\mathcal{B}_2$ are axiomatized by the following clause:*

$$x \wedge y \approx \bot, \quad \neg\neg(x \vee y) \approx x \vee y \quad \Rightarrow \quad x \approx \bot, \quad y \approx \bot.$$

The following quasi-equation axiomatizes the admissible quasi-equations of $\mathcal{B}_3$

$$x \wedge y \approx \bot, \quad \neg\neg(x \vee y) \wedge z \leq x \vee y \quad \Rightarrow \quad z \leq \neg x \vee \neg y.$$

References

[1] L.M. Cabrer and G. Metcalfe, Admissibility via natural dualities. Submitted (Research Report 2012-05, Mathematics Institute - University of Bern).

[2] P. Cintula and G. Metcalfe, Admissible rules in the implication-negation fragment of intuitionistic logic, *Annals of Pure and Applied Logic* **162**(2) (2010), 162171.

[3] W. Dzik, Splittings of lattices of theories and unification types, *Proceedings of the Workshop on General Algebra* **70**, Verlag Johannes Heyn (2006), 71-81.

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

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

[6] R. Iemhoff, Intermediate logics and Visser's rules, *Notre Dame Journal of Formal Logic* **46**(1) (2005), 65-81.

[7] R. Iemhoff and Paul Rozière, Unification in fragments of intermediate logics, Preprint.

[8] K.B. Lee, Equational classes of distributive pseudocomplemented lattices, *Canadian Jounal of Mathematics* **22** (1970), 881-891.

[9] P. Minari and A. Wroński, The property (HD) in intermediate logics: a partial solution of a problem of H. Ono, *Reports on Mathematical Logic* **22** (1988), 21-25.

[10] H.A. Priestley, Representation of distributive lattices by means of ordered Stone spaces, *Bulletin of the London Mathematical Society* **2** (1970), 186-190.

[11] H.A. Priestley, The construction of spaces dual to pseudocomplemented distributive lattices, *Quarterly Journal of Mathematics. Oxford Ser.* **26**(2) (1975), 215-228.

[12] T. Prucnal, On the structural completeness of some pure implicational propositional calculi, *Studia Logica* **32**(1) (1973), 45-50.

[13] P. Rozière, *Regles admissibles en calcul propositionnel intuitionniste*, PhD thesis, Universitè Paris VII (1992).

[14] A. Urquhart, Projective distributive p-algebras, *Bulletin of the Australian Mathematical Society* **24** (1981), 269-275.

[15] A. Wroński, Transparent Unification Problem, *Reports on Mathematical Logic* **29** (1995), 105-107.

On Matching Concurrent Traces*

Iliano Cervesato[1], Frank Pfenning[2], Jorge Luis Sacchini[1],
Carsten Schürmann[3], and Robert J. Simmons[2]

[1] Carnegie Mellon University – Doha, Qatar
iliano@cmu.edu, sacchini@qatar.cmu.edu
[2] Carnegie Mellon University – Pittsburgh, PA, USA
fp@cs.cmu.edu, rjsimmon@cs.cmu.edu
[3] IT University of Copenhagen – Copenhagen, Denmark
carsten@itu.dk

Abstract. Concurrent traces are sequences of computational steps where independent steps can be permuted and executed in any order. We study the problem of matching on concurrent traces. We outline a sound and complete algorithm for matching traces with one variable standing for an unknown subtrace.

1 State-Transition Concurrency

Computation in a concurrent system results from the interactions of computing units such as threads or agents. One popular approach to modeling concurrent computation views each such agent as being able to perform local transformations on a global state, possibly in parallel. This is the approach embraced by Petri nets [5]. This is also the approach underlying propositional multiset rewriting, which we will now describe in further detail.

A *multiset rewriting system* is determined by a set of *rules* of the form $\tilde{a} \to \tilde{b}$, where $\tilde{a}$ and $\tilde{b}$ are multisets over some support set S. We write "$\cdot$" for the empty multiset and "$\tilde{a}, \tilde{b}$" for the union of multisets $\tilde{a}$ and $\tilde{b}$. Multiset union is associative and commutative, and has the empty multiset as its unit. Therefore, the set of multisets over S is a commutative monoid with respect to "," and "$\cdot$". We give each rule $\tilde{a} \to \tilde{b}$ with a unique name, r, writing the association as $r : \tilde{a} \to \tilde{b}$. We write R for a set of labeled rules.

A *state* s is a set of pairs $x : a$ where $a \in S$ is an element of the support set S and x is a unique name. Abusing notation, we occasionally write such a state as $\tilde{x} : \tilde{a}$, where $\tilde{a}$ is the multiset of occurrences of elements of S in s and $\tilde{x}$ the corresponding names. We also use "," for set union in the context of states.

A rule $r : \tilde{a} \to \tilde{b}$ in a multiset rewriting system R is *enabled* in a state s if $s = s', (\tilde{x} : \tilde{a})$, i.e., if s contains its antecedent. In this circumstance, the

* Partially supported by the Qatar National Research Fund under grant NPRP 09-1107-1-168, the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program under Grant NGN-44, and the Danish Council for Strategic Research, Programme Commission on Strategic Growth Technologies under grant 10-092309.

application of r to s results in the state $s'' = s', (\tilde{y} : \tilde{b})$ where $\tilde{y}$ are fresh names: the portion of s matching $\tilde{a}$ has been replaced with new state elements for $\tilde{b}$. The triple $r(\tilde{x}; \tilde{y})$, generically denoted t, is a rule instance, or *transition*. We call $\tilde{x} : \tilde{a}$ the pre-condition of t and denote it as $\bullet t$ and $\tilde{y} : \tilde{b}$ its *post-condition*, denoted $t\bullet$. We formalize the state transformation embodied by the application of a rule by means of the multiset rewriting judgment $R \vdash s \xrightarrow{t} s''$. Then, application is defined simply as

$$R, (r : \tilde{a} \to \tilde{b}) \vdash \underbrace{s', \tilde{x} : \tilde{a}}_{s} \xrightarrow{r(\tilde{x};\tilde{y})} \underbrace{s', \tilde{y} : \tilde{b}}_{s''}$$

or more succinctly as $R \vdash s', \bullet t \xrightarrow{t} s', t\bullet$ for t an instance of a rule $r \in R$.

Multistep computation is obtained by iterating application. We write $R \vdash s \xRightarrow{\boldsymbol{t}} s'$ for the reflexive and transitive closure of our base judgment, where $\boldsymbol{t}$ records the rules that have been applied to go from s to s'. It is defined by the following grammar:

$$\boldsymbol{t} ::= \cdot \;\mid\; r(\tilde{x}; \tilde{y}) \;\mid\; \boldsymbol{t_1}; \boldsymbol{t_2}$$

We call $\boldsymbol{t}$ a *trace*. Here, "$\cdot$" represents the empty trace and "$\boldsymbol{t_1}; \boldsymbol{t_2}$" is the concatenation of $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$.

A concurrent trace $\boldsymbol{t}$ can be interpreted as a bipartite directed acyclic graph (BDAG) $G = (N_1, N_2, E)$ where N_1 is the set of transitions t in $\boldsymbol{t}$ and N_2 is the set of state elements $x : a$ mentioned in $\boldsymbol{t}$. There is an edge from $x : a$ to $t = r(\tilde{x}; \tilde{y})$ if and only if x occurs in $\tilde{x}$, and there is an edge from t to $y : b$ iff y occurs in $\tilde{y}$. These BDAGs are exactly what we get by graphically unfolding the computation of a place/transition Petri net.

2 Concurrent Traces

A trace is a precise record of all the steps that occurred in a concurrent computation. Indeed, given the initial state, it allows us to replay the computation exactly and determine the final state. If the latter is known, it can be replayed backward and reconstruct the initial state.

Traces have an interesting algebraic structure that is key to reasoning about concurrent computations. To expose it, it will be useful to define the natural extension of pre- and post-conditions:

$$\begin{cases} \bullet(\cdot) & = & \cdot \\ \bullet r(\tilde{x}; \tilde{y}) & = & \tilde{x} : \tilde{a} \\ \bullet(\boldsymbol{t_1}; \boldsymbol{t_2}) & = & \bullet \boldsymbol{t_1} \cup (\bullet \boldsymbol{t_2} \setminus \boldsymbol{t_1}\bullet) \end{cases} \qquad \begin{cases} (\cdot)\bullet & = & \cdot \\ r(\tilde{x}; \tilde{y})\bullet & = & \tilde{y} : \tilde{b} \\ (\boldsymbol{t_1}; \boldsymbol{t_2})\bullet & = & (\boldsymbol{t_1}\bullet \setminus \bullet \boldsymbol{t_2}) \cup \boldsymbol{t_2}\bullet \end{cases}$$

for $r : \tilde{a} \to \tilde{b}$. Two traces $\boldsymbol{t_1}$ and $\boldsymbol{t_2}$ are *independent*, written $\boldsymbol{t_1} \parallel \boldsymbol{t_2}$, if $\bullet \boldsymbol{t_1} \cap \boldsymbol{t_2}\bullet = \boldsymbol{t_1}\bullet \cap \bullet \boldsymbol{t_2} = \emptyset$. A name x is *internal* to a trace $\boldsymbol{t}$ if it does not occur in neither $\bullet \boldsymbol{t}$ nor $\boldsymbol{t}\bullet$.

Two traces $\boldsymbol{t}$ and $\boldsymbol{t'}$ are *equal*, written $\boldsymbol{t} = \boldsymbol{t'}$, if there are renamings ρ and ρ' of their internal variables such that $R \vdash s \stackrel{\rho\boldsymbol{t}}{\Longrightarrow} s'$ iff $R \vdash s \stackrel{\rho'\boldsymbol{t'}}{\Longrightarrow} s'$ for any s and s'. It is easy to prove that if $\boldsymbol{t} = \boldsymbol{t'}$, then $\bullet\boldsymbol{t} = \bullet\boldsymbol{t'}$ and $\boldsymbol{t}\bullet = \boldsymbol{t'}\bullet$.

Trace concatenation is associative with respect to trace equality, and the empty trace ("$\cdot$") is its left and right unit. Moreover, independent traces can be permuted, i.e., $\boldsymbol{t_1}; \boldsymbol{t_2} = \boldsymbol{t_2}; \boldsymbol{t_1}$ whenever $\boldsymbol{t_1} \parallel \boldsymbol{t_2}$.

Altogether, traces obey the following equational theory:

Associativity	$(\boldsymbol{t_1}; \boldsymbol{t_2}); \boldsymbol{t_3}$	$=$	$\boldsymbol{t_1}; (\boldsymbol{t_2}; \boldsymbol{t_3})$	
Left identity	$\cdot; \boldsymbol{t}$	$=$	$\boldsymbol{t}$	
Right identity	$\boldsymbol{t}; \cdot$	$=$	$\boldsymbol{t}$	
Independent commutativity	$\boldsymbol{t_1}; \boldsymbol{t_2}$	$=$	$\boldsymbol{t_2}; \boldsymbol{t_1}$	if $\boldsymbol{t_1} \parallel \boldsymbol{t_2}$

Two traces are equal whenever the corresponding BDAGs are isomorphic up to the name of internal state elements. Although the complexity of trace equality has not been determined as far as we know, the isomorphism problem for both bipartite graphs and directed acyclic graphs is known to be GI-complete, where GI is a complexity class between P and NP [6].

The equational theory we just gave is closely related to trace monoids, the laws that govern Mazurkiewicz trace theory [4]. Mazurkiewicz traces may contain several occurrences of what corresponds to our transitions, while our traces are limited to a single occurrence of each transition.

It should be observed that, although our definition of trace was based on finite concurrent computations, the same equational theory applies to traces of infinite computations, as produced by operating systems, streaming computations, and reactive systems.

3 Reasoning about Concurrent Traces

Reasoning about concurrent systems requires reasoning about concurrent computations, whether implicitly or explicitly. Typical reasoning tasks include 1) the analysis of a given computation, whether to gather information about it as done in profilers or to react to unexpected states as done by monitors; 2) reasoning about all possible computations from a given state, for example to verify correctness as in model checking, to ensure termination, and to detect deadlocks or starvation; and 3) verifying the soundness and often completeness of a program transformation, through some sort of (bi)simulation.

While many of these reasoning tasks can be performed by just looking at the successive states of the system, an analysis that explicitly works on the traces of the concurrent computation can be much more precise and economical, especially when the state is large or geographically distributed. To carry out such forms of trace-based reasoning, we must be able to work with traces that are not completely specified. This can be captured by extending our definition of trace with *trace variables*:

$$\boldsymbol{t} ::= X(\tilde{x}; \tilde{y}) \mid \cdot \mid r(\tilde{x}; \tilde{y}) \mid \boldsymbol{t_1}; \boldsymbol{t_2}$$

Equality and other notions introduced for traces carry over to traces containing variables. They also allow asking whether there are instances of said variables that make two traces equal, a typical unification problem. With the requirement that $\bullet \boldsymbol{t} = \bullet \boldsymbol{t'}$ and $\boldsymbol{t}\bullet = \boldsymbol{t'}\bullet$, we write

$$\boldsymbol{t} \stackrel{?}{=} \boldsymbol{t'}$$

for such a problem. A solution is a substitution $\theta = (X_1 \leftarrow \boldsymbol{t_1}, \ldots, X_n \leftarrow \boldsymbol{t_n})$ such that for each i, $\bullet X_i = \bullet t_i$ and $X_i\bullet = t_i\bullet$ and moreover $[\theta]\boldsymbol{t} = [\theta]\boldsymbol{t'}$ where substitution application is defined in the standard way.

Because the equational theory of traces is only partially commutative, standard results about ACU or string unification do not apply directly [1]. We have no proof that it is decidable. We know however that there can be multiple solutions to a trace unification problem since multiset unification can be encoded as a form of trace unification where all transitions are independent from each other.

4 Matching

While trace unification is necessary for reasoning tasks (2) and (3) above, matching is sufficient for task (1). In a matching problem $\boldsymbol{t} \stackrel{?}{=} \boldsymbol{t'}$, the trace $\boldsymbol{t'}$ does not contain trace variables. Matching is decidable: freeze the order of the transitions in $\boldsymbol{t}$ and consider all dependency-preserving permutations of $\boldsymbol{t'}$ in turn; each permutation gives rise to a string matching problem, which is solvable in linear time. Of course such a brute-force approach has exponential complexity.

In this section, we present an algorithm for the restriction of the matching problem with a single trace variable. This algorithm behaves better than the brute force approach just mentioned. These matching problems have therefore the form

$$\boldsymbol{t_1}; X(\tilde{x}; \tilde{y}); \boldsymbol{t'_1} \stackrel{?}{=} \boldsymbol{t_2}$$

where $\boldsymbol{t_1}$, $\boldsymbol{t'_1}$ and $\boldsymbol{t_2}$ do not contain trace variables.

Our algorithm will work as follows: strip common transitions from the beginning and the end of the two sides until only the trace variable is left on the left-hand side. While intuitively simple, care must be taken because independent transitions can be permuted and especially because the two sides may not use the same internal names. This means that as we strip initial and/or final transitions, we will need to apply a renaming to the rest of the trace.

Given sequences of names $\tilde{x}$ and $\tilde{x}'$ of the same length, we write $\rho = [\tilde{x}'/\tilde{x}]$ for the renaming that replaces each name x_i in $\tilde{x}$ with the corresponding name x'_i in $\tilde{x}'$. A renaming $\rho = [\tilde{x}'/\tilde{x}]$ is *legal* for a trace $\boldsymbol{t}$ if it is the identity for $\bullet\boldsymbol{t}$ and $\boldsymbol{t}\bullet$. Applying a renaming ρ to a trace $\boldsymbol{t}$, written $\rho\boldsymbol{t}$, preserves $\bullet\boldsymbol{t}$ and $\boldsymbol{t}\bullet$ if ρ is legal for $\boldsymbol{t}$.

Our matching algorithm is given next for a generic problem $\boldsymbol{t_1} \stackrel{?}{=} \boldsymbol{t_2}$. We fix the order of $\boldsymbol{t_1}$ but allow permutations in $\boldsymbol{t_2}$. We propagate the internal names in $\boldsymbol{t_2}$ to $\boldsymbol{t_1}$. Legality constraints ensure the invariant that $\bullet\boldsymbol{t_1} = \bullet\boldsymbol{t_2}$ and $\boldsymbol{t_1}\bullet = \boldsymbol{t_2}\bullet$.

1. $p(\tilde{x};\tilde{y});\boldsymbol{t_1} \stackrel{?}{=} p(\tilde{x};\tilde{y}');\boldsymbol{t_2}$:
   If $\tilde{y}'/\tilde{y}$ is legal for $p(\tilde{x};\tilde{y});\boldsymbol{t_1}$, then solve $[\tilde{y}'/\tilde{y}]\boldsymbol{t_1} \stackrel{?}{=} \boldsymbol{t_2}$, otherwise fail.
2. $\boldsymbol{t_1};p(\tilde{x};\tilde{y}) \stackrel{?}{=} \boldsymbol{t_2};p(\tilde{x}';\tilde{y})$:
   If $\tilde{x}'/\tilde{x}$ is legal for $\boldsymbol{t_1};p(\tilde{x};\tilde{y})$, then solve $[\tilde{x}'/\tilde{x}]\boldsymbol{t_1} \stackrel{?}{=} \boldsymbol{t_2}$, otherwise fail.
3. $X(\tilde{x};\tilde{y}) \stackrel{?}{=} \boldsymbol{t_2}$: Simply return the solution $X \leftarrow \boldsymbol{t_2}$.

This algorithm is sound and complete in the sense that computed solutions do yield equal traces, and that it computes all solutions. Proofs, although using a different presentation of traces, will appear in a forthcoming technical report.

Theorem 1 (Soundness). *Given a matching problem* $\boldsymbol{t_1};X(\tilde{x};\tilde{y});\boldsymbol{t_1'} \stackrel{?}{=} \boldsymbol{t_2}$*, if the matching algorithm reports a solution* $X \leftarrow \boldsymbol{t}$*, then there is a legal renaming* ρ *such that* $\rho\boldsymbol{t_1};\boldsymbol{t};\rho\boldsymbol{t_1'} = \boldsymbol{t_2}$.

Theorem 2 (Completeness). *Given a matching problem* $\boldsymbol{t_1};X(\tilde{x};\tilde{y});\boldsymbol{t_1'} \stackrel{?}{=} \boldsymbol{t_2}$*, if there is a trace* $\boldsymbol{t}$ *such that* $\boldsymbol{t_1};\boldsymbol{t};\boldsymbol{t_1'} = \boldsymbol{t_2}$*, then the matching algorithm will report the solution* $X \leftarrow \rho\boldsymbol{t}$ *for some renaming* ρ.

Because this algorithm embeds two trace equality subproblems, it is at least GI-complete. It may report multiple solutions.

5 Generalizations

In this paper, we have investigated the matching problem for the notion of concurrent traces emerging from multiset rewriting systems (or equivalently place/transition Petri nets [3]). In recent years, more expressive languages for describing concurrent computations have been successfully proposed. State elements can carry information to allow agents to store and exchange data. Some may be read repeatedly rather than consumed on access. In process algebras, synchronization changes the involved processes while our rules are immutable. Each of these extensions, and others, yields much more sophisticated notions of trace. The logical framework CLF [2] provides mechanisms rooted in linear type theory to support these forms of concurrency, and others. Being a type theory, its term language contains constructs that describe the associated traces.

We are extending the matching algorithm outlined here to a large sublanguage of CLF. Supporting reusable state elements, in particular, adds substantial complications. We have also started examining the general matching problem as well as unification for concurrent traces.

References

1. F. Baader and W. Snyder. *Handbook of Automated Reasoning*, chapter Unification Theory, pages 441–526. Elsevier, 2001.

2. I. Cervesato, F. Pfenning, D. Walker, and K. Watkins. A concurrent logical framework II: Examples and applications. Technical Report CMU-CS-02-102, Computer Science Department, Carnegie Mellon University, 2002.
3. I. Cervesato and A. Scedrov. Relating State-Based and Process-Based Concurrency through Linear Logic. *Information & Computation*, 207(10):1044–1077, 2009.
4. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.
5. Carl Adam Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. IFIP*, pages 386–390, Amsterdam, 1963. North Holland Publ. Comp.
6. V.N. Zemlyachenko, N.M. Korneenko, and R.I. Tyshkevich. Graph isomorphism problem. *Journal of Mathematical Sciences*, 29(4):1426–1481, 1985.

Unification modulo a property of the El Gamal Encryption Scheme

Serdar Erbatur[1], Santiago Escobar[2], and Paliath Narendran[1]

[1] University at Albany–SUNY (USA), {se,dran}@cs.albany.edu
[2] Universidad Politécnica de Valencia (Spain), sescobar@dsic.upv.es

1 Introduction

Equational Unification has recently been applied in the field of formal analysis of cryptographic protocols. Formal methods have been very useful in detecting nontrivial flaws in protocols and also to verify their correctness; see Meadows [7] for a survey of formal verification of cryptographic protocols. Terms in this approach are often assumed to be in the *free algebra,* i.e., the function symbols are not interpreted; in particular algebraic properties of function symbols are ignored. Thus two terms are equal only if they are syntactically equal; thus the analysis only requires standard unification. However, it is possible to extend this analysis by also considering algebraic properties of terms, to get a deeper analysis against attacks that can possibly exploit those properties [4, 5]. Therefore, E-unification algorithms provide tools to achieve this goal.

In this work, we consider an axiom which is observed in El Gamal encryption. We briefly explain how a message is encrypted within the El Gamal scheme. Let p be a prime, g a generator of Z_p and x the private key obtained from the range 1 to $p-2$. Define $h = g^x \, mod \, p$. The public key is the tuple (p, g, h). A message m is encrypted by first selecting a random integer r s.t. $gcd(r, p-1) = 1$ and then computing

$$a \equiv g^r \, mod \, p, \;\; b \equiv m * h^r \, mod \, p$$

The ciphertext of m is the pair (a, b). Let us define $B(m, r) = m * h^r$. When using this functionality, there is an important property of B that could be taken into account. This property is unfolded as follows:

$$B(m_1, r_1) * B(m_2, r_2) = m_1 * h^{r_1} * m_2 * h^{r_2}$$

and

$$B(m_1, r_1) * B(m_2, r_2) = m_1 * m_2 * h^{r_1 + r_2} \; (mod \, q)$$

where $h^{r_1+r_2}$ can be written (abstracted) as $r_1 \circledast r_2$. Therefore the equality of interest is

$$B(m_1, r_1) * B(m_2, r_2) = B(m_1 * m_2, \; r_1 \circledast r_2)$$

In the rest of the paper, we consider the theory with this equality, which we will denote by $\mathcal{E}$. In Section 2 we give an algorithm and prove decidability of

$\mathcal{E}$-unification. A novel feature of our approach is the use of types in detecting non-termination. We follow the standard notation in the literature, see [2] for more details.

2 Unification modulo $\mathcal{E}$

We present decidability of $\mathcal{E}$-unification by constructing an algorithm along with a proof of correctness. First, we define a set of inference rules based on standard forms and observe that those rules are complete and sound similarly to the case in [1]. This is not very different from our approach in earlier papers, but the novelty is that termination of the algorithm (i.e., inference rules) is obtained by introducing a type system for function symbols of $\mathcal{E}$. Note that $\mathcal{E}$ is not defined as a typed theory. However, using types as described later in this section allows us to identify a set of equivalence classes that does not grow. Then through a series of lemmas we show how to detect infinite splitting, which is the hardest type of failure to detect since new variables are continuously introduced into an $\mathcal{E}$-unification problem. Thus, the algorithm either transforms an $\mathcal{E}$-unification problem to dag-solved form or returns failure by finding (i) a function clash, (ii) (extended) cycle induced by relations among the variables, or (iii) variables which split indefinitely.

The function symbols B, $*$ and $\circledast$ are cancellative, i.e., if s_1, t_1, s_2, t_2 are ground terms in normal form, then $B(s_1, t_1) =_{\mathcal{E}} B(s_2, t_2)$ if and only if $s_1 =_{\mathcal{E}} s_2$ and $t_1 =_{\mathcal{E}} t_2$; similarly for the other symbols. This can be shown using the fact that $\mathcal{E}$ can be oriented either way to get a convergent rewrite system.

We now define several relations among variables:

- $U \succ_{r_*} V$ iff there is an equation $U = T * V$
- $U \succ_{l_*} V$ iff there is an equation $U = V * T$
- $U \succ_{r_\circledast} V$ iff there is an equation $U = T \circledast V$
- $U \succ_{l_\circledast} V$ iff there is an equation $U = V \circledast T$
- $U \succ_{r_B} V$ iff there is an equation $U = B(T, V)$
- $U \succ_{l_B} V$ iff there is an equation $U = B(V, T)$
- $U \succ_\circledast V$ iff $U \succ_{r_\circledast} V$ or $U \succ_{l_\circledast} V$
- $U \succ_* V$ iff $U \succ_{r_*} V$ or $U \succ_{l_*} V$
- $U \succ_B V$ iff $U \succ_{r_B} V$ or $U \succ_{l_B} V$

Let $\sim_{lp(*)}$ and $\sim_{lp(B)}$ be the reflexive, symmetric and transitive closures of $\succ_{l_*}$ and $\succ_{l_B}$ respectively. Also, let $\succ = \succ_\circledast \cup \succ_* \cup \succ_B$.

(a) *Variable Elimination*:

$$\frac{\{X =^? V\} \uplus \mathcal{EQ}}{\{X =^? V\} \cup [V/X](\mathcal{EQ})} \quad \text{if } X \text{ occurs in } \mathcal{EQ}$$

(b) *Cancellation on B*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V, Y),\ X =^? B(W, T)\}}{\mathcal{EQ} \cup \{X =^? B(V, Y),\ V =^? W,\ Y =^? T\}}$$

(c) *Cancellation on '$*$'*:

$$\frac{\mathcal{EQ} \uplus \{X =^? V * Y,\ X =^? W * T\}}{\mathcal{EQ} \cup \{X =^? V * Y,\ V =^? W,\ Y =^? T\}}$$

(d) *Cancellation on '$\circledast$'*:

$$\frac{\mathcal{EQ} \uplus \{X =^? V \circledast Y,\ X =^? W \circledast T\}}{\mathcal{EQ} \cup \{X =^? V \circledast Y,\ V =^? W,\ Y =^? T\}}$$

(e) *Splitting*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? W * Z\}}{\mathcal{EQ} \cup \{X =^? W * Z,\ V =^? V_0 * V_1,\ Y =^? Y_0 \circledast Y_1,\ W =^? B(V_0,Y_0),\ Z =^? B(V_1,Y_1)\ \}}$$

(f) *Failure Rule 1*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? W \circledast T\}}{FAIL}$$

(g) *Failure Rule 2*:

$$\frac{\mathcal{EQ} \uplus \{X =^? V * Y,\ X =^? W \circledast T\}}{FAIL}$$

(h) *Occur-Check*:

$$\frac{\mathcal{EQ}}{FAIL} \qquad \text{if } X \succ^+ X \text{ for some } X$$

A set of equations (i.e., a unification problem) is said to be in *dag-solved form* (or *d-solved form*) if and only if they can be arranged as a list

$$x_1 =^? t_1,\ \ldots,\ x_n =^? t_n$$

where (a) each left-hand side x_i is a distinct variable, and (b) $\forall\, 1 \leq i \leq j \leq n$: x_i does not occur in t_j [6].

The variable X in the splitting rule is called an *e-peak*. That is, an *e-peak* is a variable X such that $X \succ_{l_*} W$, $X \succ_{r_*} Z$, $X \succ_{l_B} V$ and $X \succ_{r_B} Y$ for some variables V, Y, W, Z. The rules (a) – (h) can be applied in any order but we propose the following strategy for efficiency in reaching the dag-solved form. Rules (a), (f), (g) and (h) have the highest priority, followed by (b), (c) and (d). Finally the rule (e) has the lowest priority.

Lemma 1. *Rules (a) – (h) are sound and complete for $\mathcal{E}$-unification.*

Proof. The result is obtained in a similar way to that of [1]. □

Apart from rules (f), (g) and (h), another failure case is infinite splitting. A necessary and sufficient condition for this, along with a failure rule, will be given later in this paper. Two example cases where rule (e) applies infinitely because of a variable shared between the first argument of B and an argument of $*$ are shown below. We underline those equations that give rise to a new e-peak in the conclusion of the inference rule.

(1) *Infinite Splitting Case 1*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? V * Z\}}{\mathcal{EQ} \cup \{X =^? V * Z,\ \underline{V =^? V_0 * V_1},\ Y =^? Y_0 \circledast Y_1,\ \underline{V =^? B(V_0,Y_0)},\ Z =^? B(V_1,Y_1)\ \}}$$

(2) *Infinite Splitting Case 2*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? W * V\}}{\mathcal{EQ} \cup \{X =^? W * V,\ \underline{V =^? V_0 * V_1},\ Y =^? Y_0 \circledast Y_1,\ W =^? B(V_0,Y_0),\ \underline{V =^? B(V_1,Y_1)}\ \}}$$

Note that if a variable is shared between the second argument of B and the first or second argument of $*$, this does not lead to infinite splitting:

(3) *Example*:

$$\frac{\mathcal{EQ} \uplus \{X =^? B(V,Y),\ X =^? W * Y\}}{\mathcal{EQ} \cup \{X =^? W * Y,\ V =^? V_0 * V_1,\ \underline{Y =^? Y_0 \circledast Y_1},\ W =^? B(V_0,Y_0),\ \underline{Y =^? B(V_1,Y_1)}\}}$$

By failure rule (f), case (3) results with a Function Clash. In contrast, note that cases (1) and (2) cause infinite splitting since they both give rise to *e-peaks* repeatedly. To explain this, we first assign the set of types $\{\alpha, \gamma\}$ to arguments of function symbols as follows:

$$B : \alpha \times \gamma \to \alpha,\ \ * : \alpha \times \alpha \to \alpha,\ \ \circledast : \gamma \times \gamma \to \gamma$$

This type mechanism is not strict: in fact one may consider $\{\alpha, \gamma\}$ as a set of attributes. Note that a term such as $B(X, X)$ would be problematic for strict typing but it is reasonable to assume that X has both α and γ as "attributes" in this case. This also explains how types are assigned to existing variables. As an example, the equation $U_1 =^? V * W$ and $U_2 =^? V \circledast W$ are typed properly in this discipline: U_1 with α, U_2 with γ and both V and W with α and γ, respectively.

We, in general, assign types to variables as follows. A variable V is assigned type α if and only if there exists a variable U such that one of $U \succ_* V, U \succ_{l_B} V$, $V \succ_* U$, or $V \succ_{l_B} U$ holds. Likewise, a variable V is assigned type γ if and only if there exists a variable U such that $U \succ_\circledast V$ or $U \succ_{r_B} V$ or $V \succ_\circledast U$ or $V \succ_{r_B} U$.

We can now observe that in (1) and (2) the new peaks have type α. However, the set $Var(\mathcal{S})$ for a problem $\mathcal{S}$ may get larger because of splitting. Also, if V is the representative of an equivalence class of variables with respect to a relation $R \in \{\sim_{lp(*)}, \sim_{lp(B)}\}$, i.e., $[V]_R$, then obviously all variables in $[V]_R$ have the same type as V.

As seen in rule (e), a variable T can split in two ways: either as $T = T_0 * T_1$ or as $T = T_0 \circledast T_1$. The splitting rule (e) may be applied further to new variables, and in general we may obtain a variable T_ω where $\omega \in \{0,1\}^*$ is a string of 0's and 1's. Therefore we adopt the general discipline for creating new variables as: $T_\omega = T_{\omega 0} * T_{\omega 1}$ **or** $T_\omega = T_{\omega 0} \circledast T_{\omega 1}$ [3]. Also, if $\omega = \lambda$, the empty string, then $T_\omega = T$, i.e., T is an original variable. For a variable V, define $\overline{V} = \{V_\omega \mid \omega \in \{0,1\}^*\}$. Note that $\overline{V}$ may be infinitely large.

Lemma 2. *Let V be of type α and $|\overline{V}|$ be infinite. Then any descendant V_ω, where $\omega \in \{0,1\}^*$ and $V \succ_*^+ V_\omega$, joins an originally existing $\sim_{lp(B)}$-equivalence class which has type α.*

Our main observations are (i) if a variable splits, then its descendants have the same type, see Lemma 2 and (ii) in case of infinite splitting the new e-peaks

[3] Using the type discipline given earlier, we assume that T is α-typed in the former case and γ-typed in the latter.

are always α-typed. We justify (ii) later in this section. Thus (i) and (ii) allow us to effectively leave γ-typed variables out.

Definition 1. *Let $\mathcal{V} = \{X_w \mid X$ is α-typed and $\omega \in \{0,1\}^*\}$ i.e., the set of variables which have type α in a given problem. In other words, $\mathcal{V}$ includes the original variables with type α and the new variables that α is assigned as type.*

Lemma 3. *Let $\mathcal{S}$ be an $\mathcal{E}$-unification problem and $X \in Var(\mathcal{S})$ such that X is an e-peak. Then X has type α, i.e., $X \in \mathcal{V}$.*

Let us consider grouping elements of $\mathcal{V}$ with respect to the equivalence relation $\sim_{lp(B)}$. That is, we write $\mathcal{V} = \biguplus[X]_{\sim_{lp(B)}}$ where $X \in \mathcal{V}$. Therefore, we have a set of $\sim_{lp(B)}$-equivalence classes such that the number of them remains the same even if splitting applies infinitely.

Lemma 4. *The number of $\sim_{lp(B)}$-equivalence classes in $\mathcal{V}$ does not increase.*

Let us define $\beta = \sim_{lp(B)} \circ \succ_* \circ \sim_{lp(B)}$. Then the following result gives us a way to detect infinite splitting.

Lemma 5. *If rule (e) applies infinitely, then there exists a β-cycle among $\sim_{lp(B)}$-equivalence classes in $\mathcal{V}$.*

We define an interpretation which gives a valid model for $\mathcal{E}$. If B is interpreted as projection to its first argument, i.e., left projection, then we get $m*n = m*n$ out of $\mathcal{E}$. This is useful since the unification problem is solvable only if its interpreted version is also solvable.

Lemma 6. *Let $\mathcal{P}$ be an $\mathcal{E}$-unification problem. If β is cyclic, then $\mathcal{P}$ is not solvable.*

Therefore we can define the following failure rule which deals with infinite splitting.

(i) *Infinite Splitting*:

$$\frac{\mathcal{EQ}}{FAIL} \qquad \text{if } \beta \text{ is cyclic in } \mathcal{V}$$

Lemma 7. *Unification modulo $\mathcal{E}$ is decidable using rules (a)-(i) above.*

3 Conclusion

We have shown decidability of unification modulo a theory with a single axiom which is a property of the El-Gamal public key cryptosystem.

In [3], we show decidability of unification for a theory with symbols B and $*$ through a similar outline of the results using the fact that the number of $\sim_{lp(B)}$-classes remain same. On the other hand, the number of $\sim_{lp(B)}$-classes here does not remain the same in general; only the number of $\sim_{lp(B)}$-classes in a special subclass, that we identified through a type system, is non-decreasing. Introducing types, which are not originally defined for the equational theory, proved useful to show termination of the algorithm. Furthermore, the version of $\mathcal{E}$ which has the same multiplication operator on the right, has an undecidable unification problem as shown recently in [1].

References

1. S. Anantharaman, S. Erbatur, C. Lynch, P. Narendran, M. Rusinowitch. "Unification modulo Synchronous Distributivity". Technical Report SUNYA-CS-12-01, Dept. of Computer Science, University at Albany—SUNY. Available at `www.cs.albany.edu/~ncstrl/treports/Data/README.html` (An abridged version to be presented at *IJCAR 2012.*)
2. F. Baader, W. Snyder. "Unification Theory". *Handbook of Automated Reasoning*, pp. 440–526, Elsevier Sc. Publishers B.V., 2001.
3. S. Erbatur, C. Lynch, P. Narendran. "Unification in Blind Signatures". Presented at FTP 2011. Available at `www.cs.albany.edu/~se/blindsig_ftp2011.pdf`
4. S. Escobar, C. Meadows, J. Meseguer. "Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties". In: *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures* (A. Aldini, G. Barthe, and R. Gorrieri, eds.) LNCS 5705, pages 1–50.
5. S. Escobar, C. Meadows, D. Kapur, C. Lynch, C. Meadows, J. Meseguer, P. Narendran, R. Sasse. "Protocol analysis in Maude-NPA using unification modulo homomorphic encryption". In: *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark* (P. Schneider-Kamp and M. Hanus, eds.), pages 65–76.
6. J.-P. Jouannaud, C. Kirchner. "Solving equations in abstract algebras: a rule-based survey of unification." *Computational Logic: Essays in Honor of Alan Robinson*, pp. 360–394, MIT Press, Boston (1991).
7. C. Meadows. "Formal Verification of Cryptographic Protocols: A Survey" *ASIACRYPT* , pp. 135–150 (1994).

Bounded Second-Order Unification Using Regular Terms

Tomer Líbal

Institute of Computer Languages (E185)
Vienna University of Technology
Favoritenstraße 9, 1040 Vienna, Austria
shaolin@logic.at

Abstract. We present an algorithm for the bounded unification of second-order problems. The algorithm extends G. P. Huet's pre-unification algorithm with rules for the generation and folding of regular terms. The concise form of the algorithm allows the reuse of the pre-unification correctness proof. Furthermore, the regular terms can be restricted in order to obtain termination over a restricted set of unifiers. Finally, the algorithm can be integrated into higher-order resolution calculi, such as constrained resolution (G. P. Huet), in order to obtain a complete calculus that enjoys eager unification.

1 Introduction

A complete pre-unification algorithm, which does not terminate, was given by G. P. Huet [3]. The problem was already shown to be undecidable for relatively simple second-order fragments [6]. The complexity of the problem is connected with the complexity of the generated terms, which can be further tracked down to two sources, the number of bounded variables in the terms and cycles in the unification problems. One approach to deal with this complexity using ramified types was given by J. Goubault-Larrecq [2]. Another approach, given by M. Schmidt-Schauß [10] [5], was to first fix a maximal number for the allowed occurrences of bounded variables and then search for minimal unifiers only. A known unification problem with a similar decidability proof is word unification [7], which belongs to the family of context unification problems. Unifiers of context unification problems are allowed to have terms with exactly one occurrence of each bounded variable in their range. It is still not known whether the unifiability of context problems is decidable.

When we are concerned not only with the unifiability problem, but with all most general unifiers, such an approach is not enough and we have to revert to the non-terminating pre-unification algorithm. This problem arises in some applications of unification, most notably in the resolution calculus, where both a search for all most general unifiers and a terminating unification algorithm are desired properties.

In this paper we present an algorithm that captures these two properties. First, we enumerate all pre-unifiers by the use of regular terms using the Kleene

star. Second, we can force the algorithm to terminate by fixing the number of iterations allowed over the Kleene star, thus obtaining, not only minimal unifiers, but in practice any possible pre-unifier.

Similar works are the finite representation of all unifiers for sub-classes of the word problem by using graphs and regular expressions [1] and the description of first-order cycles using finite automata [4].

2 Preliminaries

The unification problems dealt with in this paper are called bounded unification problems and differ from the ones presented in [8] by restricting the signature to unary second-order variables only. The restriction to unary second-order variables and even to second-order logic is done in order to simplify the presentation and does not restrict the results.

Let Σ be a signature of function symbols, denoted f, g, h and let ar be a function mapping Σ to natural numbers, which denotes its arity. Symbols with arity 0 are called constant symbols. Let V_1 be the set of first-order variables, denoted x, y, z and V_2 be the set of context variables, denoted X, Y, Z such that both are disjoint from each other and from the signature and let $V = V_1 \cup V_2$. We will denote a sequence of n terms (or variables, positions, etc.) using the notation $\overline{t_n}$. *Terms* are formed by the grammar: $t ::= x | f(t, .., t) | X(t)$. A term of the form $X(t)$ or x is called a *flex term* while $f(\overline{t_n})$ is called *rigid*. A variable not occurring as an argument to another variable in a term is called a *rigid variable occurrence*. Positions of sub-terms are defined as usual. *Contexts* extend the term grammar by $C[.] ::= [.] | f(t, .., C[.], .., t) | X(C[.])$ and denote terms containing exactly one occurrence of the hole ($[.]$), while *$\mathtt{n}$-contexts* extend the term grammar by $C[.] ::= [.] | f(C[.], .., C[.]) | X(C[.]) | t$ and denote terms containing at most $\mathtt{n}$ occurrences of the hole. A context different from $[.]$ is called non-trivial. The notation $C(t)$ means replacing all occurrences of $[.]$ in $C[.]$ with t. Given a function $\mathtt{bbound}$: $V_2 \to \mathbb{N}$, a *(ground) bounded substitution*, denoted σ, maps first-order variables to (ground) terms and context variables X to (ground) $\mathtt{n}$-contexts where $\mathtt{bbound}(X) = \mathtt{n}$. The definitions of *(ground) substitutions*, *unifiers*, *most general unifiers* and *pre-unifiers* are as in [12]. The *bounded unification problem (BUP)* is the search for bounded pre-unifiers. Given a BUP Γ, Var, Var_1 and Var_2 are the restrictions of V, V_1 and V_2 to the variables occurring in Γ. We further assume knowledge of Huet's pre-unification algorithm as presented in [12] as well as its correctness results and the definitions of *solved forms* and *pre-solved forms*.

3 Unification by Regular Terms

The algorithm presented in this section differs from the one in [8] in several aspects. First, it is based completely on the rules of Huet's algorithm, which makes the algorithm and its correctness proofs more concise. Second, it enumerates all pre-unifiers of a problem.

We first describe the concept of regular terms, which denote infinite sets of contexts. *Regular contexts* extend the context grammar by using the Kleene star $C_r[.] ::= C[.]|C(C_r[.])|C^*(C_r[.])$. *Regular variables* denote variables which can be mapped to members described by regular contexts only and their set is disjoint from the sets of first and context variables. We assume the existence of a mapping between regular variables and regular contexts and therefore denote regular variables by $[D]_X$ where D is a regular context.

Given a term $f(\overline{s_n})$ and a context variable X, the *partial bindings* $\mathit{PB}(f(\overline{s_n}), X)$ is the set $\{f(w_1, .., w_n)|w_i \in V_1 \cup V_2, \Sigma_{0<i\leq n}\mathtt{bbound}(w_i) = \mathtt{bbound}(X)\}$. For $\overline{x \in V_1}$ we always assume $\mathtt{bbound}(x) = 0$. A *cycle* c in a BUP Γ, is a sequence $\overline{t_n \doteq s_n}$ of equations of Γ such that for all $1 \leq i \leq n$, the head symbol of s_i is a context variable that occurs rigidly in $t_{i-1 \ mod \ n}$ and that there is $1 \leq j \leq n$, such that t_j is not a variable. A cycle is called a *standard cycle* if there is exactly one such j. The variables $\overline{X_n}$ are called *cyclic variables* of c and are also denoted by $X_i \in c$. The number $n = m - 1$, where m is the number of equations in the cycle, is called the *size of the cycle*. Another equivalent definition of the standard cycle size is the number of flex-flex equations in it whose heads are context variables. Let $t_j = C[X_{j+1 \ mod \ n}]$ in a standard cycle s for a non trivial context C, then C is called the *cycle context* and is denoted by $\mathtt{cc}(c)$. The set of all standard cycles in a unification problem Γ is denoted by $\mathtt{scy}(\Gamma)$.

The unification algorithm in Fig. 1 contains an additional mapping `ibound` from context variables to natural numbers, which denotes the maximal depth of terms which can be generated by applying the `(Imitate)` rule.

We define the *first-order bound* of a BUP Γ (denoted $\mathtt{fbound}(\Gamma)$) as $(k+1)*l$, where k is the number of variables and l is the maximum rigid depth of a term in Γ. Another key concept is the computation of regular contexts from standard cycles, which is done in two phases. In the first phase we replace, using the function `rec`, a context variable with a regular context variable while in the second phase, taking place in rules $(\mathtt{Rec}_0^1)$ and $(\mathtt{Rec}^*)$ through the function `holes+`, we insert some extra holes into the generated terms. The original hole will be referred to as the principal hole. Given a term T with multiple holes, the notation $T_p(t)$ means replacing only the principle hole with the term t. In the algorithm, we initialize $\mathtt{ibound}(Y) = \mathtt{fbound}(\Gamma)$ for all variables $Y \in Var_2$ and each call of `(Imitate)`, `(Project)` or `(Rec)` is followed by $(\mathtt{Bind}_X)$.

4 Correctness of the Algorithm

In this section we sketch the completeness proof for `BUA`with regard to the set of pre-unifiers generated by Huet's algorithm. The soundness of the algorithm trivially follows from the fact each non-trivial rule applies only substitutions. The proof is based on three things. First it is relatively easy to see that there can be at most m applications of the `(Project)` $(\mathtt{Bind}_X)$and `(Rec)`rules, where m is the total number of allowed bounded variable occurrences according to `bbound`. Second, the close relationship between first-order unification and `BUA` without the `(Project)`, $(\mathtt{Bind}_X)$ or `(Rec)` makes it possible to prove that if

$$\frac{\Gamma \cup \{t \doteq t\}}{\Gamma}\ \texttt{(Delete)} \qquad \frac{\Gamma \cup \{f(\overline{s_n}) \doteq f(\overline{t_n})\}}{\Gamma \cup \{s_1 \doteq t_1, .., s_n \doteq t_n\}}\ \texttt{(Decomp)}$$

$$\frac{\Gamma \cup \{x \doteq t\} \qquad x \notin Var(t)}{\Gamma\{x \mapsto t\}}\ \texttt{(Bind}_x\texttt{)} \qquad \frac{\Gamma \cup \{X([.]) \doteq t\} \qquad X \notin Var(t)}{\Gamma\{X \mapsto t\}}\ \texttt{(Bind}_X\texttt{)}$$

$$\frac{\Gamma \qquad X(s) \doteq t \in \Gamma, t \notin Var(\Gamma), u \in \texttt{PB}(X,t), \texttt{ibound}(X) > 0}{\Gamma \cup \{X([.]) \doteq u\}}\ \texttt{(Imitate)}^1$$

$$\frac{\Gamma \qquad X(s) \doteq t \in \Gamma, t \notin Var(\Gamma)}{\Gamma \cup \{X([.]) \doteq [.]\}}\ \texttt{(Project)}^2 \qquad \frac{\Gamma \qquad c \in \texttt{scy}(\Gamma), X \in c, C \in \texttt{rec}(c,X)}{\Gamma \cup \{X[.] \doteq C\}}\ \texttt{(Rec)}$$

$$\frac{\Gamma \qquad [L_0^*(L[.])]_X \doteq t \in \Gamma, L \in C_C}{\Gamma\{[L_0^*(L[.])]_X \mapsto \texttt{holes+}(L)\}}\ (\texttt{Rec}_0^1)^2 \qquad \frac{\Gamma \qquad [L^*(L_2[.])]_X \doteq t \in \Gamma, L_2 \in C_R}{\Gamma\{[L^*(L_2[.])]_X \mapsto [L_2[.]]_X\}}\ (\texttt{Rec}_0^2)$$

$$\frac{\Gamma \qquad [L^*(L_2[.])]_X \doteq t \in \Gamma}{\Gamma\{[L^*(L_2[.])]_X \mapsto (\texttt{holes+}(L))_p([L^*(L_2[.])]_X)\}}\ (\texttt{Rec}^*)$$

1. all fresh variables $w_i \in V_2$ introduced in u have $\texttt{ibound}(w_i) = \texttt{ibound}(X) - 1$.
2. for all variables $Y \in Var_2$, $\texttt{ibound}(Y) = \texttt{fbound}(\Gamma)$.

Fig. 1. `BUA` - Unification Rules

a context variable is not and cannot be a part of a cycle in the problem, then the maximal number of `(Imitate)` applications on it is bound by the first-order bound (`fbound`). The third and most involved claim is that given a standard cycle, then for any pre-unifier of the problem, there is a context variable in the cycle which is mapped by the unifier to a term generated by the functions `rec` and `holes+`. This can be proved by induction on the size of the standard cycles. The variables in cycles of size 1 must be mapped, in any pre-unifier, to an `n`-context of the form $\texttt{holes+}(C^*C')$ where C is the cycle context and C' is a prefix of C. For bigger cycles, either one variable is mapped to such a context or all variables are mapped to contexts with such a context as a prefix. By utilizing a technique called derailing [8] we can obtain a smaller cycle.

5 Termination and Minimal Unifiers

If we are interested in minimal unifiers only, then we can restrict the iterations over the Kleene star used in the regular contexts. The restriction is based on the exponent of periodicity, while the completeness proof is based on the periodicity lemma [9]. Termination of the algorithm in this case is shown by using the measure $\mu =< m_1, m_2, m_3, m_4, m_5, m_6 >$ where: m_1 is the sum of allowed bounded variables in the problem, $m_2 = size(Var_2)$, m_3 is the sum of remaining allowed iterations over the Kleene stars, $m_4 = \Sigma_{X \in Var_2} \texttt{ibound}(X)$, m_5 is the number of first-order variables and m_6 is the number of symbols other than $\doteq$ in the problem.

6 Conclusion

When comparing the algorithm given in this paper with those given in [8] and [10] we can say the following. First, the algorithm in Fig. 1 is a direct exten-

sion of Huet's algorithm and contains few rules only, in contrast to the other algorithms. Second, the correctness proofs are given with respect to Huet's algorithm, making them simpler. The most important difference is the enumeration of all pre-unifiers by the algorithm. The fact that termination can be obtained by restricting the $(\mathtt{Rec}^*)$ rule makes the algorithm a suitable replacement of Huet's algorithm in higher-order resolution. Minimal unifiers are obtained using the local exponent of periodicity, while the rest of the pre-unifiers are obtained by back-tracking and using non-local versions of the exponent. Another corollary of the algorithm is that bounded unification problems are finitary if the unification problem does not include any cycle. We hope that both these results will allow for practical resolution calculi for obtaining the refutations of concrete problems, especially mathematical problems. Investigations of real mathematical proofs containing one induction, especially if considering the arithmetical comprehension only [11], hint that for these classes we can restrict the resolution calculus to acyclic unification constraints and still preserve completeness.

References

1. Habib Abdulrab, Pavel Goralcik, and G. S. Makanin. Towards parametrizing word equations. *ITA*, 35(4):331–350, 2001.
2. Jean Goubault-Larrecq. Ramified higher-order unification. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, LICS '97, pages 410–, Washington, DC, USA, 1997. IEEE Computer Society.
3. Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975.
4. Philippe Le Chenadec. The finite automaton of an elementary cyclic set. Technical Report RR-0824, INRIA, April 1988.
5. Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011.
6. Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Inf. Comput.*, 159(1-2):125–150, 2000.
7. G. S. Makanin. On the decidability of the theory of free groups (in russian). In *FCT*, pages 279–284, 1985.
8. Manfred Schmidt-Schauß. Decidability of bounded second order unification. *Inf. Comput.*, 188:143–178, January 2004.
9. Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equation. In *RTA*, pages 61–75, 1998.
10. Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher-order unification. *J. Symb. Comput.*, 40(2):905–954, August 2005.
11. S.G. Simpson. *Subsystems of Second Order Arithmetic*. Perspectives in Logic. Cambridge University Press, 2010.
12. Wayne Snyder and Jean H. Gallier. Higher-order unification revisited: Complete sets of transformations. *J. Symb. Comput.*, 8(1/2):101–140, 1989.

Experiments in Admissibility

George Metcalfe and Christoph Röthlisberger

Mathematics Institute, University of Bern
Sidlerstrasse 5, Bern 3012, Switzerland

{george.metcalfe,roethlisberger}@math.unibe.ch

1 Introduction

The validity problem for finite algebras (and similarly, for finite-valued logics) is well-understood and "solved" in the sense that there exist general methods for obtaining proof systems for checking validity and standard optimization techniques for such systems (see, e.g., [10]). However, the corresponding problem of checking admissibility of quasiequations in finite algebras is not so well-understood. The problem is decidable (see [18]), but a naive approach leads to computationally unfeasible procedures even for small algebras. Our goal in this work (which extends [17]) is a general method for obtaining practical algorithms facilitating "experiments in admissibility" for arbitrary finite algebras.

Let us fix a (usually finite) non-trivial algebra $\mathbf{A}$ for a language $\mathcal{L}$ with formula algebra $\mathbf{Fm}_{\mathcal{L}}$, referring to [3] for undefined notions from universal algebra. An $\mathbf{A}$-*unifier* for a finite set of equations $\Sigma \subseteq \mathrm{Fm}_{\mathcal{L}}$ is a homomorphism $\sigma\colon \mathbf{Fm}_{\mathcal{L}} \to \mathbf{Fm}_{\mathcal{L}}$ satisfying $\models_{\mathbf{A}} \sigma(\varphi) \approx \sigma(\psi)$ for all $\varphi \approx \psi \in \Sigma$. A quasiequation $\Sigma \Rightarrow \varphi \approx \psi$ is $\mathbf{A}$-*admissible* if each $\mathbf{A}$-unifier of Σ is also an $\mathbf{A}$-unifier of $\varphi \approx \psi$. Note that Σ is $\mathbf{A}$-unifiable in $\mathbf{A}$ if and only if (henceforth, iff) $\Sigma \Rightarrow p \approx q$ is *not* $\mathbf{A}$-admissible where p, q are any distinct variables not occurring in Σ.

If $\mathbf{A}$ has n elements, then $\mathbf{A}$-admissibility amounts to validity in the finite free algebra $\mathbf{F}_{\mathbf{A}}(n)$ and is therefore decidable (see [18]).[1] However, even for small n, the size of $\mathbf{F}_{\mathbf{A}}(n)$ may be prohibitively large. This is particularly striking when compared with the fact that validity and admissibility in $\mathbf{A}$ may coincide ($\mathbf{A}$ is said to be *structurally complete*), or coincide at least for the cases where the premises Σ of a quasiequation $\Sigma \Rightarrow \varphi \approx \psi$ are $\mathbf{A}$-unifiable ($\mathbf{A}$ is then said to be *almost structurally complete*). In other cases, admissibility of a quasiequation corresponds to holding in other, often quite small, algebras (see, e.g., [14, 4, 16]). In this work we describe algorithms for discovering such facts, illustrating our methods with a selection of well-known finite algebras.

Admissibility (in tandem with unification) has been studied intensively in the context of intermediate and transitive modal logics in [18, 11, 6, 7, 13, 5], leading

[1] For a cardinal κ, $\mathbf{F}_{\mathcal{K}}(\kappa)$ denotes the free algebra of $\mathcal{K}$ with κ generators. Note that when considering admissibility, it can be helpful to view the elements of $\mathbf{F}_{\mathcal{K}}(\kappa)$ for $\kappa \leq \omega$ as equivalence classes $[\varphi]$ of formulas φ containing at most κ variables, defined with respect to the congruence relating φ and ψ whenever $\models_{\mathcal{K}} \varphi \approx \psi$.

also to proof systems for checking admissibility [8, 12, 1], and for certain many-valued logics in [18, 4, 14–16]. However, a general theory for the finite-valued case has so far been lacking. Our broader goal in this work is to automatically obtain admissible rules for finite algebras (and corresponding logics) that may then be used either to simplify and speed up derivations for checking validity, or to establish meta-level properties of the quasivarieties generated by these algebras.

2 Algorithms for Admissibility

Let us fix during this section a finite non-trivial algebra $\mathbf{A}$ for a language $\mathcal{L}$ and recall that checking whether a quasiequation $\Sigma \Rightarrow \varphi \approx \psi$ is $\mathbf{A}$-admissible is equivalent to checking the validity of $\Sigma \Rightarrow \varphi \approx \psi$ in the finite free algebra $\mathbf{F}_{\mathbf{A}}(|A|)$. The following observation, which provides the basis for our algorithms, shows that admissibility may also be equivalent to validity in certain (perhaps much smaller) subalgebras of $\mathbf{F}_{\mathbf{A}}(|A|)$.[2]

Proposition 1. *Given* $\mathbf{B} \in \mathbb{Q}(\mathbf{F}_{\mathbf{A}}(|A|))$ *and* $\mathbf{A} \in \mathbb{V}(\mathbf{B})$*:*

(a) $\mathbb{Q}(\mathbf{B}) = \mathbb{Q}(\mathbf{F}_{\mathbf{A}}(|A|))$.
(b) $\Sigma \Rightarrow \varphi \approx \psi$ *is admissible in* $\mathbf{A}$ *iff* $\Sigma \models_{\mathbf{B}} \varphi \approx \psi$.

In particular, admissibility amounts to validity in any subalgebra $\mathbf{B}$ of $\mathbf{F}_{\mathbf{A}}(|A|)$ of which $\mathbf{A}$ is a homomorphic image ($\mathbf{A} \in \mathbb{H}(\mathbf{B})$). This suggests a procedure:

(i) Find the smallest free algebra $\mathbf{F}_{\mathbf{A}}(m)$ ($m \leq |A|$) such that $\mathbf{A} \in \mathbb{H}(\mathbf{F}_{\mathbf{A}}(m))$.
(ii) Compute the set $\mathrm{Sub}(\mathbf{F}_{\mathbf{A}}(m))$ of subalgebras of $\mathbf{F}_{\mathbf{A}}(m)$.
(iii) Construct the set $\mathrm{Adm}(\mathbf{A})$ of all $\mathbf{B} \in \mathrm{Sub}(\mathbf{F}_{\mathbf{A}}(m))$ such that $\mathbf{A} \in \mathbb{H}(\mathbf{B})$.
(iv) Derive a proof system for checking satisfiability in a smallest $\mathbf{B} \in \mathrm{Adm}(\mathbf{A})$.

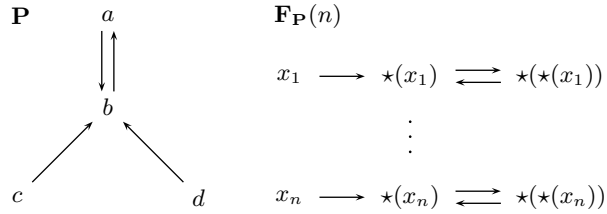
Steps (i)-(iii) have been implemented using macros implemented for the Algebra Workbench [20]. Step (iv) can be implemented directly making use of a system such as MUltlog/MUltseq [19, 9] or 3TAP [2]. Note, however, that computing all the subalgebras of a free algebra is rather inefficient. Hence, in our current implementation, subalgebras are generated and tested immediately for suitability, storing upper bounds for the best algebra, which leads to a significant reduction in the number of algebras to be tested.

It is reasonable to ask at this point whether the procedure computes the *smallest* algebra that generates $\mathbb{Q}(\mathbf{F}_{\mathbf{A}}(|A|))$. Unfortunately, this is not the case. Consider the algebra $\mathbf{P} = \langle \{a, b, c, d\}, \star \rangle$ where the unary operation $\star$ and the free algebras $\mathbf{F}_{\mathbf{P}}(n)$ are described by the following diagrams:

[2] A class of $\mathcal{L}$-algebras $\mathcal{K}$ is called a *variety* or *quasivariety* if it is axiomatized by a set of $\mathcal{L}$-equations or $\mathcal{L}$-quasiequations, respectively. The variety $\mathbb{V}(\mathcal{K})$ and quasivariety $\mathbb{Q}(\mathcal{K})$ *generated by* $\mathcal{K}$ are the smallest variety and quasivariety containing $\mathcal{K}$, respectively. Let $\mathbb{H}$, $\mathbb{I}$, $\mathbb{S}$, $\mathbb{P}$, and $\mathbb{P}_U$, be, respectively, the class operators of taking homomorphic images, isomorphic images, subalgebras, products, and ultraproducts. Then $\mathbb{V}(\mathcal{K}) = \mathbb{HSP}(\mathcal{K})$ and $\mathbb{Q}(\mathcal{K}) = \mathbb{ISPP}_U(\mathcal{K})$; moreover, for a finite algebra $\mathbf{A}$, the latter refines to $\mathbb{Q}(\mathbf{A}) = \mathbb{ISP}(\mathbf{A})$.

$\mathbf{A}$	$\lvert\mathbf{A}\rvert$	Quasivariety $\mathbb{Q}(\mathbf{A})$	Free algebra	|Output Algebra|
$\mathbf{Ł_3}$	3	algebras for Ł$_3$	$\lvert\mathbf{F_A}(1)\rvert = 12$	6
$\mathbf{Ł_3^{\to}}$	3	algebras for Ł$_3^{\to}$	$\lvert\mathbf{F_A}(2)\rvert = 40$	3
$\mathbf{B_1}$	3	Stone algebras	$\lvert\mathbf{F_A}(1)\rvert = 6$	3
$\mathbf{C_3}$	3	Kleene algebras	$\lvert\mathbf{F_A}(1)\rvert = 6$	4
$\mathbf{C_3^L}$	3	Kleene lattices	$\lvert\mathbf{F_A}(2)\rvert = 82$	4
$\mathbf{S_3^{\to\neg}}$	3	algebras for RM$^{\to\neg}$	$\lvert\mathbf{F_A}(2)\rvert = 264$	6
$\mathbf{S_3^{\to}}$	3	algebras for RM$^{\to}$	$\lvert\mathbf{F_A}(2)\rvert = 60$	3
$\mathbf{G_3}$	3	algebras for G$_3$	$\lvert\mathbf{F_A}(2)\rvert = 18$	3
$\mathbf{D_4^L}$	4	De Morgan lattices	$\lvert\mathbf{F_A}(2)\rvert = 166$	8
$\mathbf{D_4}$	4	De Morgan algebras	$\lvert\mathbf{F_A}(2)\rvert = 168$	10
$\mathbf{P}$	4	$\mathbb{Q}(\mathbf{P})$	$\lvert\mathbf{F_A}(2)\rvert = 6$	6
$\mathbf{B_2}$	5	$\mathbb{Q}(\mathbf{B_2})$	$\lvert\mathbf{F_A}(1)\rvert = 7$	5

Table 1. Experiments in admissibility



The smallest algebra obtained by the procedure is the free algebra $\mathbf{F_P}(2)$ on two generators which has six elements. However, $\mathbf{P}$ can also be embedded into $\mathbf{F_P}(1) \times \mathbf{F_P}(1)$ and hence $\mathbf{P}$ itself generates $\mathbb{Q}(\mathbf{F_P}(4))$. This problem can be avoided by considering the representation of a finite algebra $\mathbf{A}$ as a subdirect product of subdirectly irreducible algebras with respect to $\mathbb{Q}(\mathbf{A})$, and seeking embeddings of these algebras into $\mathbf{F_A}(|A|)$.

3 Experiments

In this section, we describe admissibility results obtained using our algorithm for some well-known (small) finite algebras, collating these findings in Table 1.

Note first that in some cases, $\mathbf{A}$-admissibility coincides with validity in $\mathbf{A}$; that is, $\mathbb{Q}(\mathbf{A}) = \mathbb{Q}(\mathbf{F_A}(|A|))$ and we say that $\mathbf{A}$ (or $\mathbb{Q}(\mathbf{A})$) is *structurally complete*. Consider for example the algebra (which provides algebraic semantics for the implicational fragment of the substructural logic RM) $\mathbf{S_3^{\to}} = \langle\{-1, 0, 1\}, \to\rangle$ with the binary operation $\to$ described by:

$\to$	-1	0	1
-1	1	1	1
0	-1	0	1
1	-1	-1	1

Then our procedure shows that the 60-element free algebra $\mathbf{F}_{\mathbf{S}_\mathbf{3}^{\rightarrow}}(2)$ (the smallest free algebra required) possesses a subalgebra isomorphic to $\mathbf{S}_\mathbf{3}^{\rightarrow}$, and hence that $\mathbf{S}_\mathbf{3}^{\rightarrow}$ is structurally complete. Similarly, known structural completeness results have been confirmed for

$\mathbf{Ł}_\mathbf{3}^{\rightarrow} = \langle\{0, \frac{1}{2}, 1\}, \rightarrow_{\text{Ł}}\rangle$	the 3-element Komori C-algebra
$\mathbf{B_1} = \langle\{0, \frac{1}{2}, 1\}, \min, \max, \neg_{\text{G}}\rangle$	the 3-element Stone algebra
$\mathbf{G_3} = \langle\{0, \frac{1}{2}, 1\}, \min, \max, \rightarrow_{\text{G}}\rangle$	the 3-element positive Gödel algebra

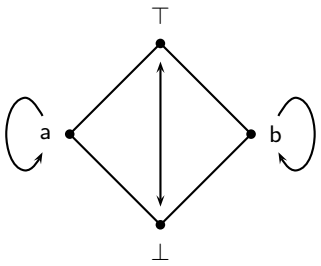
where $x \rightarrow_{\text{Ł}} y = \min(1, 1 - x + y)$, $x \rightarrow_{\text{G}} y$ is y if $x > y$, otherwise 1, and $\neg_{\text{G}} x = x \rightarrow_{\text{G}} 0$. A new structural completeness result has also been established for the pseudocomplemented distributive lattice $\mathbf{B_2}$ obtained by adding a top element to the 4-element Boolean algebra. Note, however, that the procedure timed out for the case of the 9-element algebra $\mathbf{B_3}$.

Observe now that in some cases, structural completeness fails but a quasiequation $\Sigma \Rightarrow \varphi \approx \psi$ is $\mathbf{A}$-admissible iff either $\Sigma \Rightarrow \varphi \approx \psi$ is valid in $\mathbf{A}$ or Σ is not $\mathbf{A}$-unifiable; that is, $\mathbf{A}$ (or $\mathbb{Q}(\mathbf{A})$) is said to be *almost structurally complete*. Let us note in passing that the following result provides a useful characterization of this property:

Proposition 2. *The following are equivalent for any finite algebra* $\mathbf{A}$ *and subalgebra* $\mathbf{B}$ *of* $\mathbf{F}_\mathbf{A}(1)$*:*

(1) $\mathbf{A}$ *is almost structurally complete.*
(2) $\mathbb{Q}(\mathbf{F}_\mathbf{A}(|A|)) = \mathbb{Q}(\mathbf{A} \times \mathbf{B})$.

Consider the 4-element algebra $\mathbf{D}_\mathbf{4}^\mathbf{L} = \langle\{\bot, a, b, \top\}, \wedge, \vee, \neg\rangle$ (which generates the variety of De Morgan lattices) consisting of a distributive lattice with an involutive negation defined as shown below:



Our procedure finds smallest suitable algebras isomorphic to $\mathbf{D}_4 \times \mathbf{2}$, where $\mathbf{2}$ is the 2-element Boolean lattice and a subalgebra of $\mathbf{F}_{\mathbf{D}_\mathbf{4}^\mathbf{L}}(1)$, so $\mathbf{D}_\mathbf{4}^\mathbf{L}$ is almost structurally complete. Other almost structurally complete algebras include

$\mathbf{Ł_3} = \langle\{0, \frac{1}{2}, 1\}, \rightarrow_{\text{Ł}}, \neg_{\text{Ł}}\rangle$	the 3-element Łukasiewicz algebra
$\mathbf{S}_\mathbf{3}^{\rightarrow\neg} = \langle\{-1, 0, 1\}, \rightarrow, \neg_{\text{S}}\rangle$	the 3-element algebra for the $\{\rightarrow, \neg\}$-fragment of RM

where $\neg_{\text{Ł}} = 1 - x$ and $\neg_{\text{S}} x = -x$.

On the other hand, consider the algebra $\mathbf{D_4} = \langle\{\bot, a, b, \top\}, \wedge, \vee, \neg, \bot, \top\rangle$ (which generates the variety of De Morgan algebras) defined as above for $\mathbf{D}_\mathbf{4}^\mathbf{L}$ but

with extra constants $\bot$ and $\top$. In this case, the smallest suitable algebra obtained by our procedure has 10 elements ($\mathbf{D}_4 \times \mathbf{2}$ with extra top and bottom elements, in fact). Finally, similar results have also been obtained for Kleene algebras and lattices (subvarieties of De Morgan algebras and lattices) generated by the 3-element chains $\mathbf{C_3} = \langle \{\top, a, \bot\}, \wedge, \vee, \neg, \bot, \top \rangle$ and $\mathbf{C_3^L} = \langle \{\top, a, \bot\}, \wedge, \vee, \neg \rangle$ where $\neg$ swaps $\bot$ and $\top$ and leaves a fixed. In both cases the smallest algebra found automatically by our procedure, is a 4-element chain.

References

1. S. Babenyshev, V. Rybakov, R. A. Schmidt, and D. Tishkovsky. A tableau method for checking rule admissibility in S4. In *Proceedings of UNIF 2009*, volume 262 of *ENTCS*, pages 17–32, 2010.
2. B. Beckert, R. Hähnle, P. Oel, and M. Sulzmann. The tableau-based theorem prover $_3$TAP, version 4.0. In *CADE '96*, volume 1104 of *LNCS*, pages 303–307. Springer, 1996.
3. S. Burris and H. P. Sankappanavar. *A Course in Universal Algebra*, volume 78 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1981.
4. P. Cintula and G. Metcalfe. Structural completeness in fuzzy logics. *Notre Dame Journal of Formal Logic*, 50(2):153–183, 2009.
5. P. Cintula and G. Metcalfe. Admissible rules in the implication-negation fragment of intuitionistic logic. *Annals of Pure and Applied Logic*, 162(10):162–171, 2010.
6. S. Ghilardi. Unification in intuitionistic logic. *Journal of Symbolic Logic*, 64(2):859–880, 1999.
7. S. Ghilardi. Best solving modal equations. *Annals of Pure and Applied Logic*, 102(3):184–198, 2000.
8. S. Ghilardi. A resolution/tableaux algorithm for projective approximations in IPC. *Logic Journal of the IGPL*, 10(3):227–241, 2002.
9. A. J. Gil and G. Salzer. Homepage of MUltseq. http://www.logic.at/multseq.
10. R. Hähnle. *Automated Deduction in Multiple-Valued Logics*. OUP, 1993.
11. R. Iemhoff. On the admissible rules of intuitionistic propositional logic. *Journal of Symbolic Logic*, 66(1):281–294, 2001.
12. R. Iemhoff and G. Metcalfe. Proof theory for admissible rules. *Annals of Pure and Applied Logic*, 159(1–2):171–186, 2009.
13. E. Jeřábek. Admissible rules of modal logics. *Journal of Logic and Computation*, 15:411–431, 2005.
14. E. Jeřábek. Admissible rules of Łukasiewicz logic. *Journal of Logic and Computation*, 20(2):425–447, 2010.
15. E. Jeřábek. Bases of admissible rules of Łukasiewicz logic. *Journal of Logic and Computation*, 20(6):1149–1163, 2010.
16. G. Metcalfe and C. Röthlisberger. Admissibility in De Morgan algebras. *Soft Computing*, to appear.
17. G. Metcalfe and C. Röthlisberger. Unifiability and admissibility in finite algebras. *Proceedings of CiE 2012*, to appear.
18. V. Rybakov. *Admissibility of Logical Inference Rules*, volume 136 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, Amsterdam, 1997.
19. G. Salzer. Homepage of MUltlog. http://www.logic.at/multlog.
20. M. Sprenger. Algebra Workbench. Homepage: http://www.algebraworkbench.net.

Author Index

B

Baader, Franz 1, 3
Borgwardt, Stefan 3
Bouchard, Christopher 4

C

Cabrer, Leonardo Manuel 10
Cervesato, Iliano 14

E

Erbatur, Serdar 20
Escobar, Santiago 20

G

Gero, Kimberly 4

J

Jeřábek, Emil 2

L

Libal, Tomer 26

M

Metcalfe, George 31
Morawska, Barbara 3

N

Narendran, Paliath 4, 20

P

Pfenning, Frank 14

R

Röthlisberger, Christoph 31

S

Sacchini, Jorge Luis 14
Schuermann, Carsten 14
Simmons, Robert 14