TECHNISCHE
UNIVERSITÄT
DRESDEN

Technische Universität Dresden
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

The 38th International Workshop on Unification

UNIF 2024

Workshop Informal Proceedings

Santiago Escobar and Oliver Fernández Gil (Editors)

(July 2024)

LTCS-Report 24-04

Postal Address:
Lehrstuhl für Automatentheorie
Institut für Theoretische Informatik
TU Dresden
01062 Dresden

http://lat.inf.tu-dresden.de

Visiting Address:
Nöthnitzer Str. 46
Dresden

Preface

This volume contains the contributions presented at the 38th International Workshop on Unification (UNIF 2024). UNIF 2024 was a satellite event of the Conference on Automated Deduction (CADE), affiliated with the 12th International Joint Conference on Automated Reasoning (IJCAR 2024). It took place on July 2, 2024, in Nancy, France.

Unification is concerned with the problem of making two given terms equal, either syntactically or modulo an equational theory. It is a fundamental process used in various areas of computer science, including automated reasoning, term rewriting, logic programming, natural language processing, program analysis, knowledge representation, types, etc.

UNIF is a well-established event with more than three decades of history. It takes place annually and provides a 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 students, young researchers, and scientists working in related areas to get an overview of the current state of the art in unification theory. Information about previous editions can be found on the homepage of the UNIF international workshop: https://www.irif.fr/~treinen/unif.

The Program Committee of UNIF 2024 selected 9 contributions for presentation. Each submission was evaluated by at least three program committee members. In addition, the scientific program of the workshop included two invited talks given by Daniele Nantes-Sobrinho on *Frame Inference in Separation Logic via Associative-Commutative Matching*, and by George Metcalfe on *Independence in Logic and Algebra*.

We would like to thank all the members of the Program Committee for their detailed reviews and interesting discussions during the reviewing process. We are also grateful to the Conference Co-Chairs of IJCAR: Didier Galmiche, Stephan Merz and Christophe Ringeissen, the IJCAR Workshop Chair: Sophie Tourret, and the UNIF Steering Committee for all their support in the preparation of UNIF 2024. Finally, the work of the Program Committee was greatly helped by the EasyChair system, designed by Andrei Voronkov.

July 2024

Santiago Escobar
Oliver Fernández Gil

Organization

Workshop Chairs

Santiago Escobar	Universitat Politècnica de València
Oliver Fernández Gil	TU Dresden

Program Committee

Mauricio Ayala-Rincón	Universidade de Brasília
Franz Baader	TU Dresden
Philippe Balbiani	Institut de Recherche en Informatique de Toulouse
Stefan Borgwardt	TU Dresden
David M. Cerna	CAS ICS
Kimberly A. Cornell	University at Albany, SUNY
Serdar Erbatur	UT Dallas
Santiago Escobar	Universitat Politècnica de València
Oliver Fernández Gil	TU Dresden
Silvio Ghilardi	Dipartamento di Matematica, Universitá degli Studi di Milano
Temur Kutsia	RISC, Johannes Kepler University Linz
Jordi Levy	IIIA - CSIC
Christopher Lynch	Clarkson University
Andrew W. Marshall	University of Mary Washington
Barbara Morawska	University of Opole
Daniele Nantes-Sobrinho	Universidade de Brasília, Imperial College London
Paliath Narendran	University at Albany, SUNY
Christophe Ringeissen	INRIA
Julia Sapiña Sanchis	Universitat Politècnica de València
Manfred Schmidt-Schauss	Goethe-University Frankfurt am Main
Sam van Gool	IRIF

Table of Contents

Invited Talks

Accepted Contributions

Frame Inference in Separation Logic via Associative-Commutative Matching

Daniele Nantes-Sobrinho

Imperial College London
d.nantes-sobrinho@imperial.ac.uk

Abstract

Separation logic is a popular approach to concisely specifying the behaviour of programs that manipulate memory (the heap). Conciseness comes from the fact that separation logic allows the local analysis of programs, that is, it avoids the need to describe portions of the heap not altered by a command - the frame. However, the problem of identifying the frame is challenging, known as the frame inference problem, and several incomplete approaches were proposed. In this talk, I will present a polynomial algorithm to solve the frame inference problem based on transforming the problem to the well-known "distinct occurrences of AC-matching" (DO-ACM) problem. That is, finding a frame reduces to finding a matching in an undirected bipartite graph. Polynomiality is sensible to the choice of the expression language: our choice of language is standard and expressive. We illustrate our results with a simple memory model, but we believe that the results can be extended to more complex models. We show that the same approach can be used to infer the frame for incorrectness separation logic with just a small modification of the frame inference algorithm.

This is a joint work with Andreas Lööw and Philippa Gardner, Imperial College London.

Independence in Logic and Algebra

George Metcalfe

Mathematical Institute, University of Bern, Switzerland
george.metcalfe@unibe.ch

Abstract

This talk will explore a notion of independence for formulas considered by De Jongh and Chagrova for intuitionistic propositional logic in [1] that is closely related to a notion of independence for elements of an algebraic structure studied by Marczewski and others in the 1950s [2]. Terms $t_1, \ldots, t_n$ are said to be independent in a variety (equational class) V if any substitution mapping each variable x_i to t_i V-unifies only the equations in $x_1, \ldots, x_n$ that are already satisfied by V. In [1], it is shown that this property is decidable for Heyting algebras, using Pitts' proof of uniform interpolation for intuitionistic propositional logic [4], and a description is given of independent pairs of formulas. Following [3], this talk will consider the problems of deciding and describing independence for several other case studies from logic and algebra, including groups, semigroups, lattices, modal algebras, and MV-algebras, and explain how independence relates to the notions of coherence and admissibility.

References

[1] Dick De Jongh and L. A. Chagrova. The decidability of dependency in intuitionistic propositional logic. *J. Symb. Log.*, 60(2):498–504, 1995.

[2] Edward Marczewski. A general scheme of the notions of independence in mathematics. *Bull. Acad. Polon. Sci.*, 6:731–736, 1958.

[3] George Metcalfe and Naomi Tokuda. Deciding dependence in logic and algebra. In *Dick de Jongh on Intuitionistic and Provability Logics*, volume 28 of *Outstanding Contributions to Logic*. Springer, 2024.

[4] Andrew M. Pitts. On an interpretation of second order quantification in first order intuitionistic propositional logic. *J. Symb. Log.*, 57(1):33–52, 1992.

One is all you need: Second-order Unification without First-order Variables [*]

David M. Cerna[1] and Julian Parsert[23]

[1] Czech Academy of Sciences, Prague, Czechia
dcerna@cs.cas.cz
[2] University of Oxford, United Kingdom
[3] University of Innsbruck, Austria
julian.parsert@gmail.com

Abstract

We consider the fragment of Second-Order unification with the following properties: (i) only one second-order variable allowed, (ii) first-order variables do not occur. We show that Hilbert's 10^{th} problem is reducible to this fragment if the signature contains a binary function symbol and two constants. This generalizes known undecidability results. [1]

1 Introduction

In the 2014 addition of the unification workshop Levy [1] provided a comprehensive survey of decidability and undecidability results for second-order unification. While second-order unification without first-order variables was considered [2], 2 second-order variables were required to show undecidability. Furthermore, investigations proving undecidability of second-order unification with 1 second-order variable required first-order variables [2]. We generalize these result by showing one second-order variable is enough undecidability (no first-order variables). Proofs of all significant lemmas and theorems may be found in the *arxiv version* of the paper arxiv.org/abs/2404.10616.

2 Preliminaries

We consider a finite *signature* $\Sigma = \{f_1, \cdots, f_n, c_1, \cdots, c_m\}$ where $n, m \geq 1$, for $1 \leq i \leq n$, the arity of f_i is denoted $arity(f_i) \geq 1$, and for all $1 \leq j \leq m$, the arity of c_j is denoted $arity(c_j) = 0$ (*constants*). Furthermore, let $\Sigma^{\leq 1} \subseteq \Sigma$ be the set of *base symbols* defined as $\Sigma^{\leq 1} = \{c \mid c \in \Sigma \wedge arity(c) \leq 1\}$.

By $\mathcal{V}$ we denote a countably infinite set of *variables*. Furthermore, let $\mathcal{V}_i, \mathcal{V}_f \subset \mathcal{V}$ such that $\mathcal{V}_i \cap \mathcal{V}_f = \emptyset$. We refer to members of $\mathcal{V}_i$ as *individual variables*, denoted by x,y,z, $\cdots$ and members of $\mathcal{V}_f$ as *function variables*, denoted by F,G,H, $\cdots$. Members of $\mathcal{V}_f$ have an arity ≥ 1 which we denote by $arity(F)$ where $F \in \mathcal{V}_f$. By $\mathcal{V}_f^n$, where $n \geq 1$, we denote the set of all function variables with arity n. We will use h to denote a symbol in $\mathcal{V} \cup \Sigma$ when doing so would not cause confusion.

We refer to members of the term algebra $\mathcal{T}(\Sigma, \mathcal{V})$, as *terms*. By $\mathcal{V}_i(t)$ and $\mathcal{V}_f(t)$ ($\mathcal{V}_f^n(t)$ for $n \geq 1$) we denote the set of individual variables and function variables (with arity $= n$) occurring in t, respectively. We refer to a term t as n-second-order ground (n-SOG) if $\mathcal{V}_i(t) = \emptyset$,

[*] Funded by Czech Science Foundation Grant No. 22-06414L and Cost Action CA20111 EuroProofNet and the Austrian Science Fund (FWF) project AUTOSARD (36623).

[1] Full Results and proofs in Arxiv paper arxiv.org/abs/2404.10616.

$\mathcal{V}_f(t) \neq \emptyset$ with $\mathcal{V}_f(t) \subset \mathcal{V}_f^n$, first-order if $\mathcal{V}_f(t) = \emptyset$, and *ground* if t is first-order and $\mathcal{V}_i(t) = \emptyset$. The sets of n-SOG, first-order, and ground terms are denoted $\mathcal{T}_{SO}^n$, $\mathcal{T}_{FO}$, and $\mathcal{T}_G$, respectively. When possible, without causing confusion, we will abbreviate a sequence of terms $t_1, \cdots, t_n$ by $\overline{t_n}$ where $n \geq 0$.

The set of *positions* of a term t, denoted by $pos(t)$, is a set of strings of positive integers, defined as $pos(\mathsf{h}(t_1, \ldots, t_n)) = \{\epsilon\} \cup \bigcup_{i=1}^{n}\{i.p \mid p \in pos(t_i)\}$, $t_1, \ldots, t_n$ are terms, and ϵ denotes the empty string. For example, the term at position 1.1.2 of $g(f(x, a))$ is a. Given a term t and $p \in pos(t)$, then $t|_p$ denotes the subterm of t at position p. Given a term t and $p, q \in pos(t)$, we write $p \sqsubseteq q$ if $q = p.q'$ and $p \sqsubset q$ if $p \sqsubseteq q$ and $p \neq q$. The *set of subterms of a term t* is defined as $sub(t) = \{t|_p \mid p \in pos(t)\}$. The *head* of a term t is defined as $head(\mathsf{h}(t_1, \ldots, t_n)) = \mathsf{h}$, for $n \geq 0$. The number of occurrences of a term s in a term t is defined as $occ(s,t) = |\{p \mid s = t|_p \wedge p \in pos(t)\}|$. The number of occurrences of a symbol h in a term t is defined as $occ_\Sigma(\mathsf{h}, t) = |\{p \mid \mathsf{h} = head(t|_p) \wedge p \in pos(t)\}|$.

A *n-second-order ground (n-SOG) unification equation* has the form $u \overset{?}{=}_F v$ where u and v are n-SOG terms and $F \in \mathcal{V}_f^n$ such that $\mathcal{V}_f(u) = \{F\}$ and $\mathcal{V}_f(v) = \{F\}$. A *n-second-order ground unification problem* (n-SOGU problem) is a pair $(\mathcal{U}, F)$ where $\mathcal{U}$ is a set of n-SOG unification equations and $F \in \mathcal{V}_f^n$ such that for all $u \overset{?}{=}_G v \in \mathcal{U}$, $G = F$. Recall from the definition of n-SOG that $\mathcal{V}_i(u) = \mathcal{V}_i(v) = \emptyset$.

A *substitution* is set of bindings of the form $\{F_1 \mapsto \lambda\overline{y_{l_1}}.t_1, \cdots F_k \mapsto \lambda\overline{y_{l_k}}.t_k, x_1 \mapsto s_1, \cdots, x_w \mapsto s_w\}$ where $k, w \geq 0$, for all $1 \leq i \leq k$, t_i is first-order and $\mathcal{V}_i(t_i) \subseteq \{y_1, \cdots, y_{l_i}\}$, $arity(F_i) = l_i$, and for all $1 \leq i \leq w$, s_i is ground. Given a substitution σ, $dom_f(\sigma) = \{F \mid F \mapsto \lambda\overline{x_n}.t \in \sigma \wedge F \in \mathcal{V}_f^n\}$ and $dom_i(\sigma) = \{x \mid x \mapsto t \in \Sigma \wedge x \in \mathcal{V}_i\}$. We refer to a substitution σ as second-order when $dom_i(\sigma) = \emptyset$ and first-order when $dom_f(\sigma) = \emptyset$. We use postfix notation for substitution applications, writing $t\sigma$ instead of $\sigma(t)$. Substitutions are denoted by lowercase Greek letters. As usual, the application $t\sigma$ affects only the free variable occurrences of t whose free variable is found in $dom_i(\sigma)$ and $dom_f(\sigma)$. A substitution σ is a *unifier* of an n-SOGU problem $(\mathcal{U}, F)$, if $dom_f(\sigma) = \{F\}$, $dom_i(\sigma) = \emptyset$, and for all $u \overset{?}{=}_F v \in \mathcal{U}$, $u\sigma =_{\alpha\beta} v\sigma$.

We will use the following theorem due to Matiyasevich, Robinson, Davis, and Putnam, in later sections.

Theorem 2.1 (Hilberts 10^{th} problem or Matiyasevich–Robinson–Davis–Putnam theorem [3])**.** Given a polynomial $p(\overline{x})$ with integer coefficients, finding integer solutions to $p(\overline{x}) = 0$ is undecidable.

3 n-Multipliers and n-Counters

In this section, we define and discuss the n-multiplier and n-counter functions, which allow us to encode number-theoretic problems in second-order unification. These functions are motivated by the following simple observation about n-SOGU.

Lemma 3.1. Let $(\mathcal{U}, F)$ be a unifiable n-SOGU problem, and σ a unifier of $(\mathcal{U}, F)$. Then for all $c \in \Sigma^{\leq 1}$ and $u \overset{?}{=}_F v \in \mathcal{U}$, $occ_\Sigma(c, u\sigma) = occ_\Sigma(c, v\sigma)$.

Definition 3.1 (n-Mutiplier)**.** Let t be a n-SOG term such that $\mathcal{V}_f(t) \subseteq \{F\}$ and $F \in \mathcal{V}_f^n$ and $h_1, \cdots, h_n \geq 0$. Then we define $mul(F, \overline{h_n}, t)$ recursively as follows:

- if $t = b$ and $arity(b) = 0$, then $mul(F, \overline{h_n}, t) = 0$.
- if $t = f(t_1, \cdots, t_l)$, then $mul(F, \overline{h_n}, t) = \sum_{j=1}^{l} mul(F, \overline{h_n}, t_j)$

- if $t = F(\overline{t_n})$, then $\mathit{mul}(F, \overline{h_n}, t) = 1 + \sum_{i=1}^{n} h_i \cdot \mathit{mul}(F, \overline{h_n}, t_i)$

Furthermore, let $(\mathcal{U}, F)$ be an n-SOGU problem then, $\mathit{mul}_l(F, \overline{h_n}, \mathcal{U}) = \sum_{u \overset{?}{=}_F v \in \mathcal{U}} \mathit{mul}(F, \overline{h_n}, u)$ and $\mathit{mul}_r(F, \overline{h_n}, \mathcal{U}) = \sum_{u \overset{?}{=}_F v \in \mathcal{U}} \mathit{mul}(F, \overline{h_n}, v)$.

The *n-multiplier* captures the following property of a term: let t be a n-SOG term such that $\mathcal{V}_f(t) \subseteq \{F\}$, $f \in \Sigma$, and $\sigma = \{F \mapsto \lambda\overline{x_n}.s\}$ a substitution where $\mathit{occ}_\Sigma(f, s) \geq 0$, $\mathcal{V}_i(s) \subseteq \{\overline{x_n}\}$, and for all $1 \leq i \leq n$, $\mathit{occ}(x_i, s) = h_i$. Then $\mathit{occ}_\Sigma(f, t\sigma) \geq \mathit{occ}_\Sigma(f, s) \cdot \mathit{mul}(F, \overline{h_n}, t)$ where the $\overline{h_n}$ capture the duplication of the arguments to F. The following presents this idea using a concrete example.

Example 3.1. Consider the term $t = g(F(g(a, F(s(a)))), g(F(a), F(F(F(b)))))$. Then the n-multiplier of t is $\mathit{mul}(F, h, t) = \mathit{mul}(F, h, F(g(a, F(s(a))))) + \mathit{mul}(F, h, g(F(a), F(F(F(b))))) = (1 + h) + (1 + (1 + h \cdot (1 + h))) = 3 + 2 \cdot h + h^2$. Thus, when $h = 2$ we get $\mathit{mul}(F, h, t) = 11$. Observe $\mathit{occ}_\Sigma(g', t\{F \mapsto \lambda x.g'(x, x)\}) = 11$.

Next, we introduce the n-counter function. Informally, given an n-SOG term t such that $\mathcal{V}_f(t) \subseteq \{F\}$, a symbol $c \in \Sigma^{\leq 1}$, and a substitution σ with $\mathit{dom}_f(\sigma) = \{F\}$, the n-counter captures number of occurrences of c in $t\sigma$.

Definition 3.2 (n-Counter)**.** Let $c \in \Sigma^{\leq 1}$, t be a n-SOG term such that $\mathcal{V}_f(t) = \{F\}$ and $F \in \mathcal{V}_f^n$, and $h_1, \cdots, h_n \geq 0$. Then we define $\mathit{cnt}(F, \overline{h_n}, c, t)$ recursively as follows:

- if $t = b$, $\mathit{arity}(b) = 0$, and $b \neq c$, then $\mathit{cnt}(F, \overline{h_n}, c, t) = 0$.
- if $t = f(\overline{t_l})$ and $f \neq c$, then $\mathit{cnt}(F, \overline{h_n}, c, t) = \sum_{j=1}^{l} \mathit{cnt}(F, \overline{h_n}, c, t_j)$.
- if $t = c(t)$, then $\mathit{cnt}(F, \overline{h_n}, c, c(t)) = 1 + \mathit{cnt}(F, \overline{h_n}, c, t)$
- if $t = F(\overline{t_n})$, then $\mathit{cnt}(F, \overline{h_n}, c, t) = \sum_{i=1}^{n} h_i \cdot \mathit{cnt}(F, \overline{h_n}, c, t_i)$

Furthermore, let $(\mathcal{U}, F)$ be a n-SOGU problem them, $\mathit{cnt}_l(F, \overline{h_n}, c, \mathcal{U}) = \sum_{u \overset{?}{=}_F v \in \mathcal{U}} \mathit{cnt}(F, \overline{h_n}, c, u)$ and $\mathit{cnt}_r(F, \overline{h_n}, c, \mathcal{U}) = \sum_{u \overset{?}{=}_F v \in \mathcal{U}} \mathit{cnt}(F, \overline{h_n}, c, v)$.

The n-counter captures how many occurrences of a given constant or monadic function symbol will occur in a term $t\sigma$ where $\mathcal{V}_f(t) = \{F\}$, $\sigma = \{F \mapsto \lambda\overline{x_n}.s\}$, $\mathcal{V}_i(s) \subseteq \{\overline{x_n}\}$, and for all $1 \leq i \leq n$, $\mathit{occ}(x_i, s) = h_i$ A concrete instance is presented in Example 3.2.

Example 3.2. Consider the term $t = g(g(a, a), g(F(g(a, F(g(a, a)))), g(F(a), F(F(F(b))))))$. The counter of t is $\mathit{cnt}(F, h, a, t) = \mathit{cnt}(F, h, a, g(a, a)) + \mathit{cnt}(F, h, a, F(g(a, F(g(a, a))))) + \mathit{cnt}(F, h, a, g(F(a), F(F(F(b))))) = 2 + (h + 2 \cdot h^2) + h = 2 + 2 \cdot h + 2 \cdot h^2$. Thus, when $h = 2$ we get $\mathit{cnt}(F, h, a, t) = 14$. Observe $\mathit{occ}_\Sigma(a, t\{F \mapsto \lambda x.g(x, x)\}) = 14$.

The n-multiplier and n-counter functions differ in the following key aspects: the n-multiplier counts occurrences of a symbol occurring once in a given substitution with bound variable occurrences corresponding to $\overline{h_n}$, and the n-counter counts occurrences of a given symbol after applying the given substitution to a term.

Now we describe the relationship between the n-multiplier, n-counter, and the total occurrences of a given symbol.

Lemma 3.2. Let $c \in \Sigma^{\leq 1}$, t be a n-SOG term such that $\mathcal{V}_f(t) = \{F\}$, $h_1, \cdots, h_n \geq 0$, and $\sigma = \{F \mapsto \lambda\overline{x_n}.s\}$ a substitution such that $\mathcal{V}_i(s) \subseteq \{\overline{x_n}\}$ and for all $1 \leq i \leq n$ $\mathit{occ}(x_i, s) = h_i$. Then $\mathit{occ}(c, t\sigma) = \mathit{occ}(c, s) \cdot \mathit{mul}(F, \overline{h_n}, t) + \mathit{cnt}(F, \overline{h_n}, c, t)$.

This lemma captures an essential property of the n-multiplier and n-counter. This is again shown in the following example.

Example 3.3. Consider the term $t = g(g(a,a), g(F(g(a, F(g(a,a)))), g(F(a), F(F(F(b))))))$ and substitution $\{F \mapsto \lambda x.g(a, g(x,x))\}$. The n-counter of t at 2 is $cnt(F, 2, a, t) = 14$ and the n-multiplier of t at 2 is $mul(F, 2, t) = 11$. Observe $occ_\Sigma(a, t\{F \mapsto \lambda x.g(a, g(x,x))\}) = 25$ and $occ(a,s) \cdot mul(F, 2, t) + cnt(F, 2, a, t) = 25$.

Up until now we considered arbitrary terms and substitutions. We now apply these results to unification problems and their solutions. In particular, a corollary of Lemma 3.2 is that there is a direct relation between the n-multiplier and n-counter of a unifiable unification problem given a unifier of the problem. The following lemma describes this relation.

Lemma 3.3 (Unification Condition)**.** Let $(\mathcal{U}, F)$ be a unifiable n-SOGU problem such that $\mathcal{V}_f(\mathcal{U}) = \{F\}$, $h_1, \cdots, h_n \geq 0$, and $\sigma = \{F \mapsto \lambda\overline{x_n}.s\}$ a unifier of $(\mathcal{U}, F)$ such that $\mathcal{V}_i(s) = \{\overline{x_n}\}$ and for all $1 \leq i \leq n$, $occ(x,s) = h_i$. Then for all $c \in \Sigma^{\leq 1}$,

$$occ(c,s) \cdot (mul_l(F, \overline{h_n}, \mathcal{U}) - mul_r(F, \overline{h_n}, \mathcal{U})) = cnt_r(F, \overline{h_n}, c, \mathcal{U}) - cnt_l(F, \overline{h_n}, c, \mathcal{U}). \qquad (1)$$

The *unification condition* is at the heart of the undecidability proof presented in Section 4. Essentially, Equation 1 relates the left and right side of a unification equation giving a necessary condition for unification. The following example shows an instance of this property.

Example 3.4. Consider the 1-SOGU problem $F(g(a,a)) \overset{?}{=}_F g(F(a), F(a))$ and the unifier $\sigma = \{F \mapsto \lambda x.g(x,x)\}$. Observe $occ(a, g(x,x)) \cdot ((mul_l(F, 2, F(g(a,a))) - mul_r(F, 2, g(F(a), F(a)))) = 0 \cdot (1-2) = 0$ and $cnt_r(F, h, a, g(F(a), F(a))) - cnt_l(F, h, a, F(g(a,a))) = 4 - 4 = 0$.

4 Undecidability n-SOGU

We now use the ideas from the previous section to encode Diophantine equations in unification problems. As a result, we are able to transfer undecidability results Diophantine equations to satisfying the following unification condition for n-SOGU: for a given $c \in \Sigma^{\leq 1}$ and n-SOGU problem $(\mathcal{U}, F)$, does there exists $\overline{h_n} \geq 0$ such that $cnt_r(F, \overline{h_n}, c, \mathcal{U}) = cnt_l(F, \overline{h_n}, c, \mathcal{U})$. This unification condition is a necessary condition for unifiability.

For the remainder of this section, we consider a finite signature Σ such that $\{g, a, b\} \subseteq \Sigma$, $arity(g) = 2$, and $arity(a) = arity(b) = 0$. By $p(\overline{x_n})$ we denote a polynomial with integer coefficients over the variables $x_1, \cdots, x_n$ ranging over the natural numbers and by $mono(p(\overline{x_n}))$ we denote the set of monomials of $p(\overline{x_n})$. Given a polynomial $p(\overline{x_n})$ and $1 \leq i \leq n$, if for all $m \in mono(p(\overline{x_n}))$, there exists a monomial m' such that $m = x_i \cdot m'$ then we say $div(p(\overline{x_n}), x_i)$. Furthermore, $deg(p(\overline{x}_n)) = \max\{k \mid k \geq 0 \wedge m = x_i^k \cdot q(\overline{x_n}) \wedge 1 \leq i \leq n \wedge m \in mono(p(\overline{x_n}))\}$. Given a polynomial $p(\overline{x_n})$, a polynomial $p'(\overline{x_n})$ is a sub-polynomial of $p(\overline{x_n})$ if $mono(p'(\overline{x_n})) \subseteq mono(p(\overline{x_n}))$. Using the above definition we define distinct sub-polynomials based on divisibility by one of the input unknowns.

Definition 4.1 (monomial groupings)**.** Let $p(\overline{x_n}) = q(\overline{x_n}) + c$ be a polynomial where $c \in \mathbb{Z}$, $0 \leq j \leq n$, and $S_j = \{m \mid m \in mono(p(\overline{x_n})) \wedge \forall i(1 \leq i < j \Rightarrow \neg div(m, x_i))\}$. Then

- $p(\overline{x_n})_0 = c$,
- $p(\overline{x_n})_j = 0$ if there does not exists $m \in S_j$ such that $div(m, x_j)$,

- otherwise, $p(\overline{x_n})_j = p'(\overline{x_n})$, where $p'(\overline{x_n})$ is the sub-polynomial of $p(\overline{x_n})$ such that $mono(p'(\overline{x_n})) = \{m \mid m \in S_j \wedge div(m, x_j)\}$.

Furthermore, let $p(\overline{x_n})_j = x_j \cdot p'(\overline{x_n})$. Then $p(\overline{x_n})_j \downarrow = p'(\overline{x_n})$.

We now define a second-order term representation for arbitrary polynomials as follows:

Definition 4.2 (n-Converter)**.** Let $p(\overline{x_n})$ be a polynomial and $F \in \mathcal{V}_f^n$. Then we define the positive (negative) second-order term representation of $p(\overline{x_n})$, as $cvt^+(F, p(\overline{x_n}))(cvt^-(F, p(\overline{x_n})))$, where cvt^+ (cvt^-) is defined recursively as follows:

- if $p(\overline{x_n}) = p(\overline{x_n})_0 = 0$, then $cvt^+(F, p(\overline{x_n})) = cvt^-(F, p(\overline{x_n})) = b$
- if $p(\overline{x_n}) = p(\overline{x_n})_0 = c \geq 1$, then
  - $cvt^+(F, p(\overline{x_n})) = t$ where $occ_\Sigma(a, t) = |p(\overline{x_n})_0| + 1$ and t is ground.
  - $cvt^-(F, p(\overline{x_n})) = t$ where $occ_\Sigma(a, t) = 1$ and t is ground.
- if $p(\overline{x_n}) = p(\overline{x_n})_0 < 0$, then
  - $cvt^-(F, p(\overline{x_n})) = t$ where $occ_\Sigma(a, t) = |p(\overline{x_n})_0| + 1$ and t is ground.
  - $cvt^+(F, p(\overline{x_n})) = t$ where $occ_\Sigma(a, t) = 1$ and t is ground.
- if $p(\overline{x_n}) \neq p(\overline{x_n})_0$and $p(\overline{x_n})_0 = 0$, then for all $\star \in \{+, -\}$,

$$cvt^\star(F, p(\overline{x_n})) = F(cvt^\star(F, p(\overline{x_n})_1 \downarrow), \cdots, cvt^\star(F, p(\overline{x_n})_n \downarrow))$$

- if $p(\overline{x_n}) \neq p(\overline{x_n})_0$and $p(\overline{x_n})_0 \geq 1$, then
  - $cvt^+(F, p(\overline{x_n})) = g(t, F(cvt^+(F, p(\overline{x_n})_1 \downarrow), \cdots, cvt^+(F, p(\overline{x_n})_n \downarrow))$ where $occ_\Sigma(a, t) = p(\overline{x_n})_0$ and t is ground.
  - $cvt^-(F, p(\overline{x_n})) = F(cvt^-(F, p(\overline{x_n})_1 \downarrow), \cdots, cvt^-(F, p(\overline{x_n})_n \downarrow))$
- if $p(\overline{x_n}) \neq p(\overline{x_n})_0$, and $p(\overline{x_n})_0 < 0$, then
  - $cvt^-(F, p(\overline{x_n})) = g(t, F(cvt^-(F, p(\overline{x_n})_1 \downarrow), \cdots, cvt^-(F, p(\overline{x_n})_n \downarrow))$ where $occ_\Sigma(a, t) = p(\overline{x_n})_0$ and t is ground.
  - $cvt^+(F, p(\overline{x_n})) = F(cvt^+(F, p(\overline{x_n})_1 \downarrow), \cdots, cvt^+(F, p(\overline{x_n})_n \downarrow))$

Intuitively, the n-converter takes a polynomial in n unknowns separates it into $n+1$ variable disjoint subpolynomials. Each of these subpolynomials is assigned to one of the arguments of the second-order variable (except the subpolynomial representing an integer constant) and the n-converter is called recursively on these subpolynomials. The process stops when all the subpolynomials are integers. Example 4.1 illustrates the construction of a term from a polynomial. Example 4.2 & 4.3 construct the n-multiplier and n-counter of the resulting term, respectively.

Example 4.1. Consider the polynomial $p(x, y) = 3 \cdot x^3 + xy - 2 \cdot y^2 - 2$. The positive and negative terms representing this polynomial are as follows:

$$\begin{aligned} cvt^+(F, 3 \cdot x^3 + xy - 2 \cdot y^2 - 2) =& F(F(F(g(g(a, a), g(a, a)), b), g(a, a)), F(b, a)) \\ cvt^-(F, 3 \cdot x^3 + xy - 2 \cdot y^2 - 2) =& g(g(a, a), F(F(F(a, b), a), F(b, g(a, g(a, a)))) \end{aligned}$$

Observe that the n-converter will always produce a flex-rigid unification equation as long as the input polynomial is of the form $p(\overline{x_n}) = p'(\overline{x_n}) + c$ where $c \in \mathbb{Z}$. When $c = 0$, we get a flex-flex unification equation and there is always a solution.

Example 4.2. Consider the term from Example 4.1. The n-multiplier is as follows:
Thus, $mul(F, x, y, cvt^+(F, 3 \cdot x^3 + xy - 2 \cdot y^2 - 2)) = mul(F, x, y, cvt^-(F, 3 \cdot x^3 + xy - 2 \cdot y^2 - 2)) = 1 + x^2 + y$.

Example 4.3. Consider the term from Example 4.1. The n-counter is as follows:

$$\begin{aligned} cnt(F, x, y, a, cvt^+(F, 3 \cdot x^3 + xy - 2 \cdot y^2 - 2)) =& 4 \cdot x^3 + 2 \cdot xy + y^2 \\ cnt(F, x, y, a, cvt^-(F, 3 \cdot x^3 + xy - 2 \cdot y^2 - 2)) =& x^3 + xy + 3 \cdot y^2 + 2 \\ cnt(F, x, y, a, cvt^+(F, p(x, y))) - cnt(F, x, y, a, cvt^-(F, p(x, y))) =& 3x^3 + xy - 2 \cdot y^2 - 2 \end{aligned}$$

Using the operator defined in Definition 4.2, we can transform a polynomial with integer coefficients into a n-SOGU problem. The next definition describes the process:

Definition 4.3. Let $p(\overline{x_n})$ be a polynomial and $F \in \mathcal{V}_f^n$. Then $(\mathcal{U}, F)$ is the n-SOGU problem induced by $p(\overline{x_n})$ where $\mathcal{U} = \{cvt^-(F, p(\overline{x_n})) \overset{?}{=}_F cvt^+(F, p(\overline{x_n}))\}$.

The result of this translation is that the n-counter captures the structure of the polynomial and the n-multipliers cancel out.

Lemma 4.1. Let $n \geq 1$, $p(\overline{x_n})$ be a polynomial, and $(\mathcal{U}, F)$ an n-SOGU problem induced by $p(\overline{x_n})$ where $\mathcal{U} = \{cvt^-(F, p(\overline{x_n})) \overset{?}{=}_F cvt^+(F, p(\overline{x_n}))\}$. Then

$$p(\overline{x_n}) = cnt_r(F, \overline{x_n}, a, \mathcal{U}) - cnt_l(F, \overline{x_n}, a, \mathcal{U}) \quad \text{and} \quad 0 = mul_l(F, \overline{x_n}, \mathcal{U}) - mul_r(F, \overline{x_n}, \mathcal{U}).$$

A simply corollary of Lemma 4.1 concerns commutativity of unification equations:

Corollary 4.1. Let $n \geq 1$, $p(\overline{x_n})$ be a polynomial, and $(\{s \overset{?}{=} t\}, F)$ an n-SOGU problem induced by $p(\overline{x_n})$. Then $-p(\overline{x_n}) = cnt_r(F, \overline{x_n}, a, \{t \overset{?}{=} s\}) - cnt_l(F, \overline{x_n}, a, \{t \overset{?}{=} s\})$.

Both $p(\overline{x_n})$ and $-p(\overline{x_n})$ have the same roots and the induced unification problem cannot be further reduced without substituting into F, thus the induced unification problem uniquely captures the polynomial $p(\overline{x_n})$. We now prove that the unification condition as introduced in Lemma 3.3 is equivalent to finding the solutions to polynomial equations. The following shows how a solution to a polynomial can be obtained from the unification condition and vice versa.

Lemma 4.2. Let $p(\overline{x_n})$ be a polynomial and $(\mathcal{U}, F)$ the n-SOGU problem induced by $p(\overline{x_n})$ using the $c \in \Sigma^{\leq 1}$(Definition 4.2). Then there exists $h_1, \cdots, h_n \geq 0$ such that $cnt_l(F, \overline{h_n}, c, \mathcal{U}) = cnt_r(F, \overline{h_n}, c, \mathcal{U})$ (*unification condition*) if and only if $\{x_i \mapsto h_i \mid 1 \leq i \leq n \wedge h_i \in \mathbb{N}\}$ is a solution to $p(\overline{x_n}) = 0$.

Using Lemma 4.2, we now show that finding $h_1, \cdots, h_n \geq 0$ such that the *unification condition* holds is undecidable by reducing solving $p(\overline{x_n}) = 0$ for arbitrary polynomials over $\mathbb{N}$ (Theorem 2.1) to finding $h_1, \cdots, h_n \geq 0$ which satisfy the *unification condition.*

Lemma 4.3 (Equalizer Problem)**.** For a given n-SOGU problem, finding $h_1, \cdots, h_n \geq 0$ such that the *unification condition* (Lemma 3.3) holds is undecidable.

Theorem 4.1. There exists $n \geq 1$ such that n-SOGU is undecidable.

We prove Theorem 4.1 by assuming n-SOGU is decidable and using this assumption to show that the Equalizer Problem must be decidable, thus resulting in a contradiction.

In particular, we answer the question posed in Section 1 by proving that first-order variables occurrence does not impact the decidability of second-order unification.

References

[1] Jordi Levy. On the limits of second-order unification. In Temur Kutsia and Christophe Ringeissen, editors, *Proceedings of the 28th International Workshop on Unification, UNIF 2014, Vienna, Austria, July 13, 2014*, pages 5–14, 2014.

[2] Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Inf. Comput.*, 159(1-2):125–150, 2000.

[3] Yuri V. Matiyasevich. *Hilbert's tenth problem.* MIT Press, Cambridge, MA, USA, 1993.

Type Independent Unification of Higher-Order Patterns

Jean-Pierre Jouannaud

Université Paris-Saclay, Laboratoire de Méthodes Formelles, 91190 Gif-sur-Yvette, France.

1 Introduction

Unification is used to show confluence of a set of overlapping rewrite rules. We are interested here in higher-order rules whose left-hand sides are patterns, as introduced by Miller in the case of simple types [9], later considered for other typing disciplines [2], and finally adapted to untyped lambda calculi in [3]. Following up, we study here unification of patterns *independently of a given typing discipline*. To this end, we introduce the notion of *typed structure* by axiomatizing sets of typable terms without any syntactic notion of type, as subsets of untyped terms that satisfy some well crafted closure properties common to most type systems. We then show that typed structures enjoy most general unifiers for patterns, computable by the usual algorithm for unifying patterns. This axiomatization is somehow reminiscent of Girard's axiomatisation of typed structures on which strong normalization proofs of typed lambda calculi are based. We finally briefly discuss the associative-commutative case.

Typed rewriting structures, their unifiability properties, and their application to checking higher-order confluence are developed in [5], where the main result described here is proved.

2 Computations on higher-order terms

Given an *untyped* lambda calculus generated by a vocabulary made of three pairwise disjoint sets, a signature $\mathcal{F}$ of *function symbols*, a set $\mathcal{X}$ of *variables*, and a set $\mathcal{Z}$ of *meta-variables*, we are interested in $\lambda\mathcal{F}$, an untyped calculus whose reduction relation extends the β-rule of the underlying lambda calculus by a set of user-defined rewrite rules built over that vocabulary.

2.1 Terms

$\lambda\mathcal{F}$ is a mix of the pure lambda calculus and Klop's combinatory reduction systems [7]. Terms are those of the lambda calculus enriched with $\mathcal{F}$*-headed* terms of the form $f(\overline{u})$ with $f \in \mathcal{F}$, $\overline{u}$ denoting a list of terms separated by commas, and *meta-terms* of the form $Z[\overline{v}]$ with $Z \in \mathcal{Z}$. Only variables can be abstracted over. Elements of the vocabulary have arities, denoted by vertical bars as in $|f|$. Variables have arity zero, meta-variables have an arbitrary arity. The set of (open) terms, $\mathcal{T}_{\lambda\mathcal{F}}$, is defined by the following grammar rules:

$$\begin{aligned}&u, v := x \mid (u\ v) \mid \lambda x.v \mid f(\overline{u}) \mid Z[\overline{v}]\\ &\text{where } x \in \mathcal{X},\ f \in \mathcal{F},\ |\overline{u}| = |f|,\ Z \in \mathcal{Z} \text{ and } |\overline{v}| = |Z|\end{aligned}$$

We write a for $a(\,)$, X for $X[\,]$ and $f(x\ y)$ for $f((x\ y))$. We use the small letters $f, g, h, \ldots$ for function symbols, $x, y, z, \ldots$ for variables, and reserve capital letters $X, Y, Z, \ldots$ for meta-variables. When convenient, a small letter like x may denote any variable in $\mathcal{X} \cup \mathcal{Z}$. By function symbols we sometimes mean those in $\mathcal{F}$, as well as application and abstraction.

We use the notation $|\_|$ for various quantities besides symbols arities (length of lists, size of expressions, the cardinality of sets), and $[m..n]$ for the list of natural numbers from m to n.

Positions in higher-order terms, as in first-order terms, are words over the natural numbers, using Λ for the empty word, $\cdot$ for concatenation, $\leq_{\mathcal{P}}$ for the prefix order (*above*), $\geq_{\mathcal{P}}$ for its inverse (*below*), $<_{\mathcal{P}}$ and $>_{\mathcal{P}}$ for their strict parts, $p\#q$ for incomparable positions (*parallel*), and $p \geq_{\mathcal{P}} Q$ ($p \leq_{\mathcal{P}} Q$, resp.), Q a set of parallel positions, for $\exists q \in Q:\ p \geq_{\mathcal{P}} q$ ($p \leq_{\mathcal{P}} q$, resp.).

Given a term M, we use: $M(p)$ for its symbol at position p; $M|_p$ for the subterm of M at position p, a notion which is sometimes convenient and will be given a precise meaning later; $\mathcal{P}os(M)$, $\mathcal{FP}os(M)$, $\mathcal{VP}os(M)$, $\mathcal{MP}os(M)$ for the following respective sets of positions of M: all positions, the positions of function symbols, of free variables, and of meta-variables; $\mathcal{V}ar(M)$ for its sets of free variables; $\mathcal{MV}ar(M)$, $\mathcal{MV}ar^{l}(L)$ and $\mathcal{MV}ar^{nl}(M)$ for its sets of arbitrary, linear and non-linear meta-variables; A term M is *ground* if $\mathcal{V}ar(M) = \emptyset$, *closed* if $\mathcal{MV}ar(M) = \emptyset$, and *linear* if $\mathcal{MV}ar^{nl}(M) = \emptyset$. We use $\mathcal{T}$ for the set of closed terms.

2.2 Substitutions

A *substitution* is a map from variables and meta-variables to terms which extends to a *capture-avoiding* homomorphism on terms [7]. The result $t\sigma$ of substituting the term t by the substitution σ is called an *instance* of t. All substitutions considered here will have a finite domain, hence can be denoted in extension as in $\{x_1 \mapsto M_1, \dots x_n \mapsto M_n\}$ or $\{\overline{x} \mapsto \overline{M}\}$, where $\overline{x}$ is a list of variables or meta-variables. The substitution σ is *ground* (resp., *closed*) when so are all M_i's. The *domain* of σ is the set $\mathcal{D}om(\sigma) = \{x_i : \sigma(x_i) \neq x_i\}_i$ while $\mathcal{R}an(\sigma) = \bigcup_{x \in \mathcal{D}om(\sigma)} \mathcal{V}ar(\sigma(x)) \cup \mathcal{MV}ar(\sigma(x))$ is its *image*. A substitution σ can be *restricted to* or *deprived from* (meta-)variables in some set V, written $\sigma_{|V}$ and $\sigma_{\backslash V}$ respectively. We denote by $\mathcal{P}os(\sigma)$ the sequence $\{\mathcal{P}os(\sigma(x_i))\}_i$ of sets of positions of σ.

2.3 Splitting and sticking

Given a term u and a list $P = \{p_i\}_{i=1}^{i=n}$ of parallel positions in u, we define the term obtained by *splitting* u along P as $\underline{u}_P = u[Z_1(\overline{x_1})]_{p_1} \dots [Z_n(\overline{x_n})]_{p_n}$ (u is cut below P) and its associated substitution by $\overline{u}^P = \{Z_i \mapsto \lambda\overline{x_i}.u|_{p_i}\}_{i=1}^{i=n}$ (u is cut above P), where, for all $i \in [1, n]$, $\overline{x_i}$ is the list of all variables of $u|_{p_i}$ bound in u above p_i and Z_i is a fresh meta-variable of arity $|\overline{x_i}|$. The definition of substitution for meta-variables ensures that $\overline{u}^P \underline{u}_P = u$. Instantiating $\underline{u}_p$ by $\overline{u}^p$ amounts therefore to stick $u|_p$ in the *hole* of the *context* $u[\_]_p$, an operation that may capture free variables of $u|_p$: splitting gives a meaning for the operation of *sticking* a term inside another in terms of the familiar substitution operation. (Sticking is usually called *replacement* when no variable is captured.) We denote by $u[\_]_P$ a context with holes at a set $P = \{p_i\}$ of parallel positions in u, and by $u[\overline{v}]_P$ the term obtained by sticking each v_i at position p_i in u. The brackets used in contexts may sometimes collide with those used for meta-variables, requiring desambiguation by the user.

2.4 Reductions

Two different kinds of reductions coexist in $\lambda\mathcal{F}$, functional and higher-order reductions, both operating on closed terms. However, rewriting open terms will sometimes be needed, in which case rewriting is intended to rewrite all their closed instances at once.

2.5 Functional reductions

Functional reduction is the relation on terms generated by the rule $\beta_\alpha : (\lambda x.v\ w) \longrightarrow v\{x \mapsto w\}$. The usually omitted α-index stresses that renaming bound variables, called *α-conversion*, is built-in, that is, rewriting with β_α is *modulo* α-conversion (only those variables bound below the rewriting position need be renamed).

2.6 Higher-order reductions

Higher-order reductions result from rules whose left-hand sides are higher-order patterns in Miller's or Nipkow's sense [8], although they need not be typed here.

Definition 1 (Untyped pattern)**.** *A* pre-redex *of arity n in a term L is an unapplied meta-term $Z[\overline{x}]$ whose arguments $\overline{x}$ are n pairwise distinct variables. A* pre-pattern *is a ground term all of whose meta-variables occur in pre-redexes. An* untyped pattern*, or simply* pattern*, is a pre-pattern which is neither a pre-redex nor an abstraction.*

Note that erasing types from a Nipkow's pattern yields a pattern in our sense, since his pre-redexes being of base type, they cannot be applied. Observe that pre-redexes in pre-patterns can only occur at parallel positions.

We can now define higher-order rules and rewriting:

Definition 2 (Rule)**.** *A (higher-order)* rule *is a triple $i : L \to R$, whose (possibly omitted)* index *i is a name, left-hand side L is a pattern, and $\mathcal{MV}ar(R) \subseteq \mathcal{MV}ar(L)$.*

The use of capital letters for higher-order rules aims at pointing out that L, R are higher-order terms, that is, are built using the abstraction and application operators and meta-variables of arity at least one. In contrast, first-order terms have no abstractions, no applications, and no meta-variables of non-zero arity. We will use small letters for them, as in $l \to r$.

The β-reduction rule is a particular case of higher order rule written $(\lambda x.X[x]\ u) \to X[u]$.

Definition 3 (Higher-order rewriting)**.** *Given an open term u, a position $p \in \mathcal{P}os(u)$, and a rule $i : L \to R$, u rewrites with i at p, written $u \xrightarrow[i]{p} v$, iff $u|_p = L\gamma$ for some substitution γ, and $v = u[X[\overline{x}]]_p\{X \mapsto \lambda\overline{x}.R\gamma\} = u[R\gamma]_p$, where $\overline{x}$ is the list of variables of $u|_p$ which are bound above the position p in u. We write $u \xrightarrow[\mathcal{R}]{p} v$ for $\exists i \in \mathcal{R} : u \xrightarrow[i]{p} v$.*

*A $\lambda\mathcal{F}$-*rewrite system *is a pair $(\mathcal{F}, \mathcal{R})$ made of a user's signature $\mathcal{F}$ and a set $\mathcal{R}$ of higher-order rewrite rules on $\mathcal{F}$ containing beta, defining the rewrite relation of $\lambda\mathcal{F}$ as $\xrightarrow[\mathcal{R}]{}$.*

$\lambda\mathcal{F}$-rewrite systems are being used in a variety of proof assistants, notably in Agda, Isabelle, Dedukti, and Coq. As a higher-order rewriting format, $\lambda\mathcal{F}$ is a Combinatory Reduction System [10]. This is not surprising since all other known higher-order rewriting formats can be encoded as Combinatory Reduction Systems [11].

3 Typed rewriting structures

The role of typing is to characterize subsets of the set of higher-order closed terms that satisfy good properties for computing. Calling generically $\mathcal{TT}$ such subsets of closed terms, its elements are called *typed closed terms*. Computations are meant to operate on typed closed terms, but rewriting is based on open terms, that is terms with meta-variables.

We denote therefore by $\mathcal{TT}_{\lambda\mathcal{F}}$ the set of *typed open terms*, assuming $\mathcal{TT} \subseteq \mathcal{TT}_{\lambda\mathcal{F}}$. In order to dispense with explicit types, we say that a typed open substitution σ is *well-typed* for a typed open term u if $u\sigma$ is a typed open term, and write $\sigma \in \mathcal{TT}_{\lambda\mathcal{F}}(u)$. More generally, σ is *well-typed* for θ if σ is well-typed for all u such that $x \mapsto u \in \theta$. Splitting allows then to define whether the replacement of a subterm at a particular position by a typed term yields a typed term: we define $u \in \mathcal{TT}(w[\_]_p)$ iff $w[u]_p \in \mathcal{TT}_{\lambda\mathcal{F}}$, that is, iff $\{Z \mapsto \lambda\overline{z}.u\} \in \mathcal{TT}_{\lambda\mathcal{F}}(w[Z[\overline{z}]]_p)$. We omit mention of "open/closed" when it matters not or can be inferred from the context.

The *axioms* that a *typed structure* $\mathcal{TT}_{\lambda\mathcal{F}}$ such that $\mathcal{X} \cup \mathcal{Z} \subseteq \mathcal{TT}_{\lambda\mathcal{F}} \subseteq \mathcal{T}_{\lambda\mathcal{F}}$ should satisfy (and that $\mathcal{T}_{\lambda\mathcal{F}}$ itself satisfies) are the following:

H0 $\mathcal{TT}_{\lambda\mathcal{F}}$ is closed under α-conversion, and renaming of a free occurrence of a given variable (without capture) or meta-variable;

H1 abstraction: $u \in \mathcal{TT}_{\lambda\mathcal{F}}$ iff $\lambda z.u \in \mathcal{TT}_{\lambda\mathcal{F}}$;

H2 composition: $\sigma \in \mathcal{TT}_{\lambda\mathcal{F}}(u)$, $\tau \in \mathcal{TT}_{\lambda\mathcal{F}}(\sigma)$ and $\mathcal{D}om(\tau) \cap (\mathcal{V}ar(u) \cup \mathcal{MV}ar(u)) = \emptyset$ implies $\tau \in \mathcal{TT}_{\lambda\mathcal{F}}(u\sigma)$;

H3 splitting: $u\sigma \in \mathcal{TT}_{\lambda\mathcal{F}}$ implies $u \in \mathcal{TT}_{\lambda\mathcal{F}}$ and $\sigma \in \mathcal{TT}_{\lambda\mathcal{F}}(u)$;

H4 patterns: if L is a pattern, then $L \in \mathcal{TT}_{\lambda\mathcal{F}}$.

Interpreting membership to $\mathcal{TT}_{\lambda\mathcal{F}}$ by Curry-style typability for some typing context, sets of typed terms satisfy these assumptions for all usual type systems that have the unique type property.

Typed structures enjoy a few more important closure properties, notably subterm, monotonicity, stability, as well as unifiability:

Lemma 1 (Unifiability)**.** *Let u, v be two terms unifiable by a substitution σ such that $u\sigma = v\sigma$ is well-typed, that is, $u\sigma \in \mathcal{TT}_{\lambda\mathcal{F}}$. Then, $\forall w[\_]$ such that $(\mathcal{V}ar(w[\_]) \cup \mathcal{MV}ar(w[\_])) \cap \mathcal{D}om(\sigma) = \emptyset$, $u \in \mathcal{TT}_{\lambda\mathcal{F}}(w[\_])$ iff $v \in \mathcal{TT}_{\lambda\mathcal{F}}(w[\_])$.*

4 Unification in typed rewriting structures

We now investigate a major property of typed rewriting structures, the existence of most general unifiers for solvable *critical pair equations*, that is, equations $U = V$ such that one of U, V is a left-hand side of rule, and the other a subterm of a left-hand side of rule. In other words, if the equation $U = V$ is unifiable in the untyped world, then it is unifiable in a typed structure. Further, the most general unifier of the untyped structure happens to belong to any typed structure, hence must be most general in any typed structure.

Definition 4. *A* unification (equational) problem *is a conjunction of elementary equations. An* elementary equation *is either the constant $\perp$ or is of the form $u = v$ in which u is a pre-pattern and v is a pre-pattern.*

A set of transformation rules for higher-order unification of untyped patterns is described in [4] for linear patterns and meta-variables having a bounded arity, and its extension to non-linear ones is also sketched, following the standard path by adding a *Merge* rule. These unification rules are recalled in Figure 1. They are essentially those for simply typed patterns [9], see also [6]. As usual, the rules transform elementary equations into a conjunction thereof until some *solved form* is eventually obtained. They use the following definition:

Definition 5. *A free variable $x \in \mathcal{X}$ is* protected *in a pre-pattern u if all its occurrences in u belong to a pre-redex of u. We denote by $\mathcal{UV}ar(u)$ the set of unprotected variables of u.*

Dec-Fun	$f(\overline{u}) = f(\overline{v})$	$\longrightarrow$	$\bigwedge_{i=1}^{i=\lvert f\rvert} u_i = v_i$	if $f \in \mathcal{F} \cup \mathcal{Z} \cup \{@\}$
Dec-Abs	$\lambda x.u = \lambda y.v$	$\longrightarrow$	$u\{x \mapsto z\} = v\{y \mapsto z\}$	with z fresh
Merge	$X[\overline{x}] = u \wedge X[\overline{y}] = v$	$\longrightarrow$	$X[\overline{z}] = u\{\overline{x} \mapsto \overline{z}\} \wedge u\{\overline{x} \mapsto \overline{z}\} = v\{\overline{y} \mapsto \overline{z}\}$	if $\lvert u\rvert \leq \lvert v\rvert$
Swap	$u = Y[\overline{y}]$	$\longrightarrow$	$Y[\overline{y}] = u$	if u is not a pre-redex
Flip	$X[\overline{x}] = Y[\overline{y}]$	$\longrightarrow$	$Y[\overline{y}] = X[\overline{x}]$	if $\lvert X\rvert - \lvert\overline{x}\rvert > \lvert Y\rvert - \lvert\overline{y}\rvert$
Drop	$X[\overline{x}] = u[Y[\overline{y}]]_q$	$\longrightarrow$	$X[\overline{x}] = u[Z[\overline{z}]]_q \wedge Y[\overline{y}] = Z[\overline{z}]$	

where $\overline{z} = \overline{y} \cap (\overline{x} \cup \mathcal{BV}ar(u))$ and Z fresh s.t. $\lvert Z\rvert = \lvert Y\rvert - \lvert\overline{y}\rvert + \lvert\overline{z}\rvert$,
if $\overline{y} \not\subseteq \overline{x} \cup \mathcal{BV}ar(u), \lvert X\rvert = \lvert\overline{x}\rvert$ if $u(\Lambda) \in \mathcal{F} \cup \{@, \lambda\}, \mathcal{UV}ar(u) \subseteq \overline{x}$
and $\lvert Y\rvert - \lvert\overline{y}\rvert \geq \lvert X\rvert - \lvert\overline{x}\rvert$ if $q = \Lambda$,
where $\mathcal{UV}ar(u)$ denotes the set of variables of u whose one occurrence does not occur in an argument of a meta-variable.

Figure 1: Non-failure unification rules for equational problems

For an example, x is protected in $f(g(X[x]), X)$, but not in $f(g(X[x]), x)$ because of its second occurrence. Protected variables can be eliminated from a term by appropriately instantiating its meta-variables as done in the *Drop* rule.

The first rule of Figure 1 is the same as that for first-order unification. *Dec-Abs* is the particular case for abstractions. *Merge* eliminates all occurences of a non-linear meta-variable but one. *Swap*, *Flip* and *Drop* put the equations in a format appropriate for extracting the most general unifier.

The rules of Figure 1 suffice when a unification problem is known to be solvable, otherwise failure rules are also needed to detect non-unifiability. These rules are recalled in Figure 2. An important known observation exploited in rule *Fail-Protect* is that elementary unification problems for which a free variable occurs unprotected on one side, and does not occur at all on the other side, have no solution.

Theorem 1. *Assume σ is a well-typed unifier for some critical pair equation $U = V$. Then, $mgu(U{=}V) \in \mathcal{TT}_{\lambda\mathcal{F}}(U, V)$. It is obtained by applying the rules of Figure 1 until no more possible. Non-unifiability of an equational problem P_0 is obtained when the whole set of rules fails, that is, returns the constant $\bot$.*

Therefore, unifiability of typed patterns, and the expression of a most general unifier, does not depend upon a particular Curry-style type system for the lambda calculus, provided that the type system satisfies our axioms. This new result was already observed in particular cases.

Once soundeness of the rules is proved, the proof given in [5] is based on the preservation by the unification rules of Figure 1 of an appropriate invariant expressing that some substitution is a solution of the starting unification problem. In case no solution is known, the proof relies on termination of the whole set of unification rules, which must therefore end up with an application of *global-Failure* in case no solution exists to the starting unification problem.

Conflict	$f(\overline{u}) = g(\overline{v})$	$\longrightarrow$	$\bot$	if $f, g \in \mathcal{F} \cup \mathcal{X} \cup \{@, \lambda\}$ and $f \neq g$
Fail-Protect	$X[\overline{x}] = u$	$\longrightarrow$	$\bot$	if $\exists z {\in} \mathcal{UV}ar(u) \setminus \overline{x}$
Global-Failure	$P \wedge \bot$	$\longrightarrow$	$\bot$	

Figure 2: Failure unification rules

Let us illustrate some rules, using meta-variables according to our convention.

$$
\begin{array}{l}
f(\lambda z.X[z]) = f(\lambda z.z) \underset{Dec-Fun}{\longrightarrow} \lambda z.X[z] = \lambda z.z \underset{Dec-Abs}{\longrightarrow} X[z] = z \\
f(\lambda z.X) = f(\lambda z.z) \underset{Dec-Fun}{\longrightarrow} \lambda z.X = \lambda z.z \underset{Dec-Abs}{\longrightarrow} X = z \underset{\textit{Fail-Protect}}{\longrightarrow} \perp \\
f(\lambda y.f(U)) = f(X) \underset{Dec-Fun}{\longrightarrow} \lambda y.f(U) = X \underset{\textit{Swap}}{\longrightarrow} X = \lambda y.f(U) \underset{Meta-Abs}{\longrightarrow} X[y] = f(U) \\
f(Y) = f(\lambda y.f(U)) \underset{Dec-Fun}{\longrightarrow} Y = \lambda y.f(U) \underset{Meta-Abs}{\longrightarrow} Y[y] = f(U) \underset{\textit{Fail-Arity}}{\longrightarrow} \perp \\
T = \lambda y.Y[y] \underset{Meta-Abs}{\longrightarrow} T[y] = Y[y] \underset{\textit{Flip}}{\longrightarrow} Y[y] = T[y] \\
Y[z] = \lambda x.T[y,z] \underset{Meta-Abs}{\longrightarrow} Y[z,x] = T[y,z] \underset{\textit{Drop}}{\longrightarrow} Y[z,x] = Z[z] \wedge T[y,z] = Z[z]
\end{array}
$$

(where Z is a fresh variable of arity 1)

Since Associativity and Commutativity define a syntactic theory whose unification algorithm can be expressed by rewrite rules [1], we conjecture that AC-unification of higher-order patterns does not depend either upon a particular type system satisfying our axioms. Whether these result also extend to type systems having a principal type instead of a unique type is also open.

References

[1] Alexandre Boudet and Evelyne Contejean. ”syntactic” ac-unification. In Jean-Pierre Jouannaud, editor, *Constraints in Computational Logics, First International Conference, CCL'94, Munich, Germany, September 7-9, 1994*, volume 845 of *Lecture Notes in Computer Science*, pages 136–151. Springer, 1994.

[2] Gilles Dowek. Higher-order unification and matching. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1009–1062. Elsevier and MIT Press, 2001.

[3] Gilles Dowek, Gaspard Férey, Jean-Pierre Jouannaud, and Jiaxiang Liu. Confluence of left-linear higher-order rewrite theories by checking their nested critical pairs. *Mathematical Structures in Computer Science*, Special issue on Confluence:1–36, 2022.

[4] Gaspard Férey and Jean-Pierre Jouannaud. Confluence in UnTyped Higher-Order Theories by Means of Critical Pairs. draft hal-03126102, INRIA, march 2021.

[5] Jean-Pierre Jouannaud. Confluence in terminating rewriting computations. In Bertrand Meyer, editor, *The French School of Programming*. Springer Verlag, 2024.

[6] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT-Press, 1991.

[7] Jan Willem Klop. *Combinatory Reduction Systems*. Number 127 in Mathematical Centre Tracts. CWI, Amsterdam, The Netherlands, 1980. PhD Thesis.

[8] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.

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

[10] Terese. Term rewriting systems. In *Cambridge Tracts in Theoretical Computer Science, M. Bezem, J. W. Klop, and R. de Vrijer editors*. Cambridge University Press, 2003.

[11] Vincent van Oostrom and Femke van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. In Jan Heering, Karl Meinke, Bernhard Möller, and Tobias Nipkow, editors, *Higher-Order Algebra, Logic, and Term Rewriting, First International Workshop, HOA '93, Amsterdam, The Netherlands, September 23-24, 1993, Selected Papers*, volume 816 of *Lecture Notes in Computer Science*, pages 276–304. Springer, 1993.

Towards a Well-Founded $\preceq$ Relation for Permissive Nominal Terms

Alexander Baumgartner

Universidad de O'Higgins, Rancagua, Chile

Abstract

This work discusses the relation of more general terms ($\preceq$) in the permissive nominal language. First, we show that the permissive nominal language exhibits the same problem as the classical nominal language, namely, that the relation $\preceq$ is not well-founded. Second, we propose a modification in one of the original definitions that leads to a well-founded $\preceq$ relation. Third, we formulate an anti-unification algorithm that computes a unique least general generalization of two arbitrary input terms. The proposed modification yields a language that includes the original language from Dowek et al.. Since the original language is of generalization type zero, computed generalizations might be outside of it.

1 Introduction

This work discusses the relation that states that one term is more general than another one ($\preceq$), and the problem of finding a generalization of two input terms (called anti-unification problem), in the setting of permissive nominal terms [9]. Interesting generalizations are the least general ones (lgg). They represent parts of the input terms that they have in common. Finding an lgg has various real-world applications, e.g., in clone detection [5], analogy making [13], or parallel recursion scheme detection [1]. To be able to find an lgg, the relation $\preceq$ needs to be well-founded.

In the setting of nominal terms only atoms can be bounded. This yields some nice computational properties [14, 20] that are missing in λ-calculus where arbitrary variables may be bound [12, 15]. While the anti-unification problem has already been studied in various settings with binders [3, 6, 7, 8, 10, 17], including classical nominal terms [2, 4, 19], to the best of our knowledge, it has not yet been studied in the setting of permissive nominal terms.

It's a known fact that in the case of classical nominal terms $\preceq$ is not well-founded [2]. I.e., without restriction the anti-unification problem is of type zero (no lgg exists). One possible approach to overcome this issue is to introduce atom-variables [19]. Unfortunately, without any restrictions, that approach leads to intractable algorithms due to its intrinsic complexity.

First, we show that, like in the classical setting, $\preceq$ is not well-founded in the permissive case either. Second, we suggest a modification of the definition of permission sets which leads to a well-founded $\preceq$ relation. Permission sets define the atoms that are allowed to appear freely when instantiating a variable. The modification yields a term language that is a superset of the original one introduced by Dowek et al.. Third, we formulate an anti-unification algorithm.

2 Preliminaries

This work focuses on permissive nominal terms, introduced in [9]. We assume that the reader is familiar with them. In the following, main concepts and notions are being introduced.

Permissive Nominal Terms. Fix a countable infinite set of *atoms* $\mathbb{A} = \{a, b, c, \ldots\}$. Atoms are identified by their name, i.e., different names imply different elements from $\mathbb{A}$. In the permissive nominal setting, $\mathbb{A}$ is partitioned into two countably infinite sets $\mathbb{A}^<$ and $\mathbb{A}^>$. For instance, $\mathbb{A}^<$ might be identified with even numbers while $\mathbb{A}^>$ corresponds to the odd ones. A *signature* $\Sigma = \{f, g, \ldots\}$ is a set of *symbols* of certain arity, such that $\Sigma \cap \mathbb{A} = \emptyset$. If the arity of a symbol is clear from the context, we won't mention it. A *permutation* π is a bijection on $\mathbb{A}$ that is identity almost everywhere. Permutations are represented by finite sequences of swappings.[1] *Swappings* are written as $(a\ b)$ where $a, b \in \mathbb{A}$. *Id* denotes the identity permutation. The composition of two permutations π and π', denoted by $\pi \circ \pi'$, is equivalent to the concatenation of their representations as sequences of swappings.

The set $\mathcal{P}$ of all *permission sets* is defined as

$$\mathcal{P} = \{A \cup B \mid A \subseteq \mathbb{A}^<, B \subset \mathbb{A}^>, \text{ and } B \text{ finite}\}.$$

In contrast to [9], our permission sets are not necessarily infinite. This decision is discussed in section 3. S, T denote arbitrary permission sets. Since B is restricted to be finite in the definition of $\mathcal{P}$, the subtraction of a (finite number of) permission set(s) from $\mathbb{A}$ yields a countably infinite set. Therefore, permission sets are *coinfinite* w.r.t. $\mathbb{A}$, i.e., from $S \in \mathcal{P}$ follows that $\mathbb{A} \setminus S$ is an infinite set. Informally, this means that there are always "fresh" atoms available. It's an important property that guarantees that bound atoms can always be renamed.

For each permission set S, fix a countable infinite set of *variables* $\mathbb{X}^S = \{X^S, Y^S, Z^S, \ldots\}$ of sort S, disjoint from $\mathbb{A}$ and Σ. Moreover, $\mathbb{X}^S$ and $\mathbb{X}^T$ are disjoint for any permission set $T \neq S$.

Example 1. $X^{\{a,b\}}$ is of finite sort $\{a, b\}$, while $X^{\mathbb{A}^< \cup \{b\}}$ is of infinite sort $\mathbb{A}^< \cup \{b\}$. $X^\emptyset$ is of sort $\emptyset$. Sorts (i.e., permission sets) define the atoms that are allowed to appear freely in instantiations (see Example 2).

Permissive nominal terms are built by the grammar:

$$t, t_i ::= a \mid \lambda a.t \mid \pi \cdot X^S \mid f(t_1, \ldots, t_n) \qquad \pi ::= Id \mid (a\ b) \circ \pi$$

where a, b are atoms, f is a symbol of arbitrary but fixed arity n, $\lambda a.t$ denotes the *abstraction* of atom a in the permissive nominal term t, $\pi \cdot X^S$ is a *suspension* of the permutation π on the variable X^S, and $f(t_1, \ldots, t_n)$ is a *function application.* In the following, the word *term* refers to permissive nominal terms, if not specified otherwise.

Permutation application to a term is defined recursively and gets suspended in front of variables, as usual. We overload the notation, writing, e.g., $\pi \cdot t$ where π is a permutation and t a term. A permutation π may be applied to a permission set S by $\pi \cdot S = \{\pi(a) \mid a \in S\}$. The set of *free atoms* of some terms $t_1, \ldots, t_n$, denoted by $\mathtt{fa}(t_1, \ldots, t_n)$, is defined as $\mathtt{fa}(t_1, t_2, \ldots, t_n) = \mathtt{fa}(t_1) \cup \mathtt{fa}(t_2, \ldots, t_n)$, $\mathtt{fa}(a) = \{a\}$, $\mathtt{fa}(\lambda a.t) = \mathtt{fa}(t) \setminus \{a\}$, $\mathtt{fa}(\pi \cdot X^S) = \pi \cdot S$, and $\mathtt{fa}(f(t_1, \ldots, t_n)) = \mathtt{fa}(t_1, \ldots, t_n)$.

Two terms are α-*equivalent* if they are equal up to renaming of bound atoms (Figure 1). The predicate $=_\alpha$ is used to denote α-equivalence.

$$\frac{}{a =_\alpha a} \qquad \frac{\forall a \in S : \pi(a) =_\alpha \pi'(a)}{\pi \cdot X^S =_\alpha \pi' \cdot X^S}$$

$$\frac{s_1 =_\alpha t_1 \quad \cdots \quad s_n =_\alpha t_n}{f(s_1, \ldots s_n) =_\alpha f(t_1, \ldots, t_n)} \qquad \frac{s =_\alpha t}{\lambda a.s =_\alpha \lambda a.t} \qquad \frac{s =_\alpha (a\ b) \cdot t}{\lambda a.s =_\alpha \lambda b.t} \ a \notin \mathtt{fa}(t)$$

Figure 1: α-equivalence rules as defined in [9].

[1] Representations are not unique. In any case, an arbitrary representation can be chosen.

More General Relation $\preceq$. A *substitution* σ is a function that maps variables to terms such that for any arbitrary variable X^S holds that $\mathtt{fa}(\sigma(X^S)) \subseteq S$. Substitutions are identity almost everywhere. The identity substitution is denoted by *id* and arbitrary substitutions are denoted by σ, ρ. The composition of two substitutions and the action of a substitution on a term are defined as usual. We use the postfix notation like $t\sigma$, where t is a term and σ a substitution. The application of a substitution to a term is called *instantiation* and yields another term.

Given two terms t and s. We say that t is *more general* than s, denoted as $t \preceq s$, if there exists a substitution σ such that $t\sigma =_\alpha s$. The relation $t \prec s$ denotes that t is *strictly more general* than s, i.e., $t \preceq s$ but not $s \preceq t$. The notions of *less general* and *strictly less general* are defined analogously. Moreover, $t \simeq s$ means that t and s are *equi-general*, i.e., $t \preceq s$ and $s \preceq t$. A term r is a *generalization* of t and s if $r \preceq t$ and $r \preceq s$. It is a *least general generalization* (lgg) of t and s if there is no generalization r' of t and s such that $r \prec r'$.

Example 2. $\{X^\emptyset \mapsto \lambda a.f(a)\}$ is a substitution, while $\{X^\emptyset \mapsto f(a)\}$ is not. Therefore, $X^\emptyset \preceq \lambda a.f(a)$, while $X^\emptyset \not\preceq f(a)$. The instantiation $g(Y^S)\{Y^S \mapsto f(a)\} = g(f(a))$ is valid if $a \in S$.

3 Generalization Type

To discuss the generalization type of permissive nominal terms and our motivation of deviating from the original approach, we recall the original definition of permission sets [9]. Dowek et al. permission sets are defined as

$$\mathcal{P}_D = \{(\mathbb{A}^< \setminus A) \cup B \mid A \subset \mathbb{A}^<, B \subset \mathbb{A}^>, \text{ and } A, B \text{ are finite}\}.$$

Dowek et al. permission sets are infinite and coinfinite w.r.t. $\mathbb{A}$. On the other hand, our definition yields permission sets that are only coinfinite w.r.t. $\mathbb{A}$, but not necessarily infinite. Note that $\mathcal{P}$ also includes permission sets of the form $(\mathbb{A}^< \setminus A) \cup B$, where $A \subset \mathbb{A}^<, B \subset \mathbb{A}^>$ and A, B are finite. Therefore, $\mathcal{P}_D \subset \mathcal{P}$, i.e., our setting is more general.

Theorem 1. When using Dowek et al. permission sets, the relation $\preceq$ is not well-founded.

Proof. Since $\mathbb{A}^<$ and $\mathbb{A}^>$ are disjoint, any permission set from $\mathcal{P}_D$ corresponds to a set of the form $S \setminus A$ where S consists of all atoms from $\mathbb{A}^<$ and finitely many atoms from $\mathbb{A}^>$, and $A \subset \mathbb{A}^<$ is finite. By definition, any substitution σ must satisfy $\mathtt{fa}(\sigma(X^{S\setminus A})) \subseteq S \setminus A$ for any A. It follows that $X^{S\setminus\emptyset} \prec X^{S\setminus\{a_1\}} \prec X^{S\setminus\{a_1,a_2\}} \prec \dots$. Since A is finite, $\preceq$ is not well-founded. □

Corollary 1. Given two atoms a and b there is no lgg of a and b in the setting of $\mathcal{P}_D$.

This problem arises for any two arbitrary terms t and s that have a finite number of free atoms, i.e., where $\mathtt{fa}(t, s)$ is a finite set. For instance, the terms $f(a)$ and $g(b)$ do not have an lgg in the setting of Dowek et al.. Revising the proof of Theorem 1, it is easy to observe that the problem arises from the infinite nature of the Dowek et al. permission sets.

Also note that the same problem arises in the classical nominal setting. Baumgartner et al. [2] showed that, when considering an infinite supply of atoms, there are infinite chains of the form $\langle\emptyset, X\rangle \prec \langle\{a_1\#X\}, X\rangle \prec \langle\{a_1\#X, a_2\#X\}, X\rangle \prec \dots$. To overcome that issue, the supply of atoms may be restricted to a finite set $\mathcal{A}$ so that the statement $\langle\{a\#X \mid a \in \mathcal{A}\}, X\rangle$ becomes valid. For more details about the classical nominal setting we refer the reader to [2, 11, 18].

Our suggestion to allow finite sets in $\mathcal{P}$ was motivated by the goal of overcoming that issue.

Example 3. Considering finite permission sets, we get a variable $X^{\{a,b\}}$ as the lgg of two atoms a and b. I.e., only the atoms a and b are allowed to appear freely in instantiations of $X^{\{a,b\}}$. Note that $X^{\{a,b\}}$ is also an lgg of the two terms $f(a)$ and $g(b)$.

In our setting, there is still an infinite supply of atoms (e.g., to rename bound atoms), and, any valid term in the setting of Dowek et al. is also valid in our setting. Moreover, we can find an lgg of any two input terms without any restriction. Theorem 2 establishes that the generalization type becomes unitary in this setting.

Theorem 2. Given two arbitrary terms there exists a unique lgg up to $\simeq$.

The proof of Theorem 2 follows from section 4 and 5 where we construct an anti-unification algorithm that yields a unique output term of two arbitrary input terms and prove its properties.

4 Anti-Unification Algorithm NAU$_{\mathcal{P}}$

Given two terms t and s, an *anti-unification equation* is a triple $X^S : t \triangleq s$ where X^S is a variable of sort $\mathtt{fa}(t, s)$ that neither appears in t nor in s. X^S is called the *generalization variable*.

The anti-unification algorithm for permissive nominal terms, called NAU$_{\mathcal{P}}$, is formulated in terms of transformation rules that work on triples of the form E; Q; σ, where E and Q are sets of anti-unification equations, and σ is a substitution. Such triples are called the *states* of the algorithm. The transformation rules are given in Figure 2. A variable is called *fresh* if it didn't already appear in any of the former states of the transformation process. We use $\uplus$ to denote the disjoint union.

Atm: **Atom**

$$\{X^S : a \triangleq a\} \uplus E;\ Q;\ \sigma \Longrightarrow E;\ Q;\ \sigma\{X^S \mapsto a\}.$$

Dec: **Decomposition**

$$\{X^S : f(t_1, \dots, t_n) \triangleq f(s_1, \dots, s_n)\} \uplus E;\ Q;\ \sigma$$
$$\Longrightarrow \{Y_1^{S_1} : t_1 \triangleq s_1, \dots, Y_n^{S_n} : t_n \triangleq s_n\} \cup E;\ Q;\ \sigma\{X^S \mapsto f(Y_1^{S_1}, \dots, Y_n^{S_n})\},$$

where f is a symbol of arity $n \geq 0$, and $Y_i^{S_i}$ is a fresh variable of sort $S_i = \mathtt{fa}(t_i, s_i)$, for all $1 \leq i \leq n$.

Abs: **Abstraction**

$$\{X^S : \lambda a.t \triangleq \lambda b.s\} \uplus E;\ Q;\ \sigma \Longrightarrow \{Y^T : (c\ a) \cdot t \triangleq (c\ b) \cdot s\} \cup E;\ Q;\ \sigma\{X^S \mapsto \lambda c.Y^T\},$$

where $c \in \mathbb{A} \setminus S$ and Y^T is a fresh variable of sort $T = \mathtt{fa}((c\ a) \cdot t, (c\ b) \cdot s)$.[2]

Sol: **Solving**

$$\{X^S : t \triangleq s\} \uplus E;\ Q;\ \sigma \Longrightarrow E;\ Q \cup \{X^S : t \triangleq s\};\ \sigma,$$

if none of the previous rules is applicable.

Mer: **Merging**

$$E;\ \{X^S : t_1 \triangleq s_1, Y^T : t_2 \triangleq s_2\} \uplus Q;\ \sigma \Longrightarrow E;\ \{X^S : t_1 \triangleq s_1\} \cup Q;\ \sigma\rho,$$

where ρ is a substitution defined by $\{Y^T \mapsto \pi{\cdot}X^S\}$ and π is a permutation such that $\pi \cdot t_1 =_\alpha t_2$ and $\pi \cdot s_1 =_\alpha s_2$.[3]

Figure 2: Transformation rules of the anti-unification algorithm.

[2] Since $\mathbb{A} \setminus S$ isn't empty and $\mathtt{fa}(\lambda a.t, \lambda b.s) \subseteq S$, we can take $c \in \mathbb{A} \setminus S$ to rename the bound atoms (Figure 1).

[3] Since the sort of generalization variables is minimal w.r.t. the represented terms, ρ always exists if π exists.

Computing The Lgg. Given two terms t and s, $\mathrm{NAU}_{\mathcal{P}}$ works in the following manner:

1. Create the initial state $\{X^{\mathtt{fa}(t,s)} \colon t \triangleq s\};\ \emptyset;\ id$.
2. Apply the rules of Figure 2 exhaustively, that is $\{X^{\mathtt{fa}(t,s)} \colon t \triangleq s\};\ \emptyset;\ id \implies^* \emptyset;\ Q;\ \sigma$.
3. Apply the computed substitution σ to the generalization variable of the initial state $X^{\mathtt{fa}(t,s)}$, that is $X^{\mathtt{fa}(t,s)}\sigma$, to obtain the generalization of the input terms t and s.

We write $\mathrm{NAU}_{\mathcal{P}}(t,s)$ to denote the result of that process, i.e., it denotes the generalization.

Since, from the computed substitution only one mapping is needed to obtain the generalization, we will omit all the other mappings in the following derivation examples.

Example 4. Consider the input terms $f(a,b,a)$ and $f(b,a,c)$. The initial state is $\{X^{\{a,b,c\}} \colon f(a,b,a) \triangleq f(b,a,c)\};\ \emptyset;\ id$. Now we apply the rules of Figure 2 exhaustively.

$$\begin{array}{ll}
\{X^{\{a,b,c\}} \colon f(a,b,a) \triangleq f(b,a,c)\};\ \emptyset;\ id & \Longrightarrow_{\text{Dec}} \\
\{Y_1^{\{a,b\}} \colon a \triangleq b,\ Y_2^{\{a,b\}} \colon b \triangleq a,\ Y_3^{\{a,c\}} \colon a \triangleq c\};\ \emptyset;\ \{X^{\{a,b,c\}} \mapsto f(Y_1^{\{a,b\}}, Y_2^{\{a,b\}}, Y_3^{\{a,c\}})\} & \Longrightarrow^3_{\text{Sol}} \\
\emptyset;\ \{Y_1^{\{a,b\}} \colon a \triangleq b,\ Y_2^{\{a,b\}} \colon b \triangleq a,\ Y_3^{\{a,c\}} \colon a \triangleq c\};\ \{X^{\{a,b,c\}} \mapsto f(Y_1^{\{a,b\}}, Y_2^{\{a,b\}}, Y_3^{\{a,c\}})\} & \Longrightarrow_{\text{Mer}} \\
\emptyset;\ \{Y_1^{\{a,b\}} \colon a \triangleq b,\ Y_3^{\{a,c\}} \colon a \triangleq c\};\ \{X^{\{a,b,c\}} \mapsto f(Y_1^{\{a,b\}}, (a\ b) \cdot Y_1^{\{a,b\}}, Y_3^{\{a,c\}})\} &
\end{array}$$

The computed generalization is $f(Y_1^{\{a,b\}}, (a\ b) \cdot Y_1^{\{a,b\}}, Y_3^{\{a,c\}})$. Applying the substitution $\{Y_1^{\{a,b\}} \mapsto a, Y_3^{\{a,c\}} \mapsto a\}$ gives $f(a,b,a)$, and applying $\{Y_1^{\{a,b\}} \mapsto b, Y_3^{\{a,c\}} \mapsto c\}$ gives $f(b,a,c)$.

Example 5. Consider the input terms $f(\lambda b.b, a)$ and $f(\lambda a.X^S, X^S)$ where $S \subseteq \mathbb{A}^<$ and $a \in S$. The initial state is $\{Y^S \colon f(\lambda b.b, a) \triangleq f(\lambda a.X^S, X^S)\};\ \emptyset;\ id$.

$$\begin{array}{ll}
\{Y^S \colon f(\lambda b.b, a) \triangleq f(\lambda a.X^S, X^S)\};\ \emptyset;\ id & \Longrightarrow_{\text{Dec}} \\
\{Z_1^{S\setminus\{a\}} \colon \lambda b.b \triangleq \lambda a.X^S,\ Z_2^S \colon a \triangleq X^S\};\ \emptyset;\ \{Y^S \mapsto f(Z_1^{S\setminus\{a\}}, Z_2^S)\} & \Longrightarrow_{\text{Abs}} \\
\{Z_3^{(S\setminus\{a\})\cup\{c\}} \colon c \triangleq (a\ c) \cdot X^S,\ Z_2^S \colon a \triangleq X^S\};\ \emptyset;\ \{Y^S \mapsto f(\lambda c.Z_3^{(S\setminus\{a\})\cup\{c\}}, Z_2^S)\} & \Longrightarrow^2_{\text{Sol}} \\
\emptyset;\ \{Z_3^{(S\setminus\{a\})\cup\{c\}} \colon c \triangleq (a\ c) \cdot X^S,\ Z_2^S \colon a \triangleq X^S\};\ \{Y^S \mapsto f(\lambda c.Z_3^{(S\setminus\{a\})\cup\{c\}}, Z_2^S)\} & \Longrightarrow_{\text{Mer}} \\
\emptyset;\ \{Z_2^S \colon a \triangleq X^S\};\ \{Y^S \mapsto f(\lambda c.(a\ c) \cdot Z_2^S, Z_2^S)\} &
\end{array}$$

The computed result is $f(\lambda c.(a\ c) \cdot Z_2^S, Z_2^S)$. It is equi-general to $f(\lambda a.X^S, X^S)$, and, instantiating it by $\{Z_2^S \mapsto a\}$ results in $f(\lambda c.c, a)$ which is α-equivalent to $f(\lambda b.b, a)$.

Note that in the merge step of Example 5, we could have chosen to keep $Z_3^{(S\setminus\{a\})\cup\{c\}}$ instead of Z_2^S. There might be various possible rule applications to a certain state but the choice doesn't matter. The derivations are confluent and lead to equi-general results.

5 Properties of $\mathrm{NAU}_{\mathcal{P}}$

Lemma 1 (Termination)**.** $\mathrm{NAU}_{\mathcal{P}}$ generates $\mathcal{O}(n)$ states for any input of size n and terminates.

Proof. The size $\|t\|$ of a term t is defined as $\|a\| = 1$, $\|f(t_1, \ldots, t_n)\| = 1 + \sum_{i=1}^{n} \|t_i\|$, $\|\lambda a.s\| = 1 + \|s\|$, and $\|\pi \cdot X^S\| = 1$. The size of a set of anti-unification equations E is defined as $\|E\| = \sum_{X^S \colon t \triangleq s \in E} \|t\| + \|s\|$. Finally, the size of a state $E;\ Q;\ \sigma$ is defined by $2\|E\| + \|Q\|$.

Using that definition, we get that the initial state created by $\mathrm{NAU}_{\mathcal{P}}(t,s)$ is of size $2(\|t\| + \|s\|)$, i.e., linear by the size of the input terms t and s. Since every rule application strictly decreases the size of the state, $\mathrm{NAU}_{\mathcal{P}}$ generates at most $\mathcal{O}(n)$ states until no more rule is applicable, for any input of size n. □

Lemma 2 (Soundness)**.** Given terms t, s. Any term $\mathrm{NAU}_{\mathcal{P}}(t, s)$ is a generalization of t and s.

Proof. Given two input terms t and s, $\mathrm{NAU}_{\mathcal{P}}(t, s)$ creates an initial state E; Q; σ where $E = \{X^{\mathtt{fa}(t,s)} \colon t \triangleq s\}$, $Q = \emptyset$, and $\sigma = id$. It trivially holds that $X^{\mathtt{fa}(t,s)}\sigma$ is a generalization of t and s. By induction on the derivation process we will show that this is actually an invariant that is maintained by rule applications. More precisely, we show that, after any rule application E'; Q'; $\sigma' \Longrightarrow E''$; Q''; σ'', the term $X^{\mathtt{fa}(t,s)}\sigma''$ is a generalization of t and s, given that $X^{\mathtt{fa}(t,s)}\sigma'$ is a generalization of t and s. In order to prove that, two substitutions, that can be obtained from an arbitrary set of anti-unification equations F, are needed:

$$\rho_l^F := \{X^S \mapsto t \mid X^S : t \triangleq s \in F\} \qquad \rho_r^F := \{X^S \mapsto s \mid X^S : t \triangleq s \in F\}$$

For the initial state, it is trivial that $X^{\mathtt{fa}(t,s)}\sigma\rho_l^{E\cup Q} =_\alpha t$ and $X^{\mathtt{fa}(t,s)}\sigma\rho_r^{E\cup Q} =_\alpha s$. By case distinction on the rules of Figure 2 it can also easily be verified that $X^{\mathtt{fa}(t,s)}\sigma''\rho_l^{E''\cup Q''} =_\alpha t$ and $X^{\mathtt{fa}(t,s)}\sigma''\rho_r^{E''\cup Q''} =_\alpha s$ holds, given that $X^{\mathtt{fa}(t,s)}\sigma'\rho_l^{E'\cup Q'} =_\alpha t$ and $X^{\mathtt{fa}(t,s)}\sigma'\rho_r^{E'\cup Q'} =_\alpha s$. □

Lemma 3 (Completeness)**.** For any generalization r of some terms t, s holds $r \preceq \mathrm{NAU}_{\mathcal{P}}(t, s)$.

Proof. By structural induction on r we identify common parts of t and s. 4 possible cases are given by the term grammar. In the two cases where we encounter either an atom or a function application in r, it is easy to conclude that one of the rules Atm or Dec is applicable. Therefore, the same symbol will also appear in the substitution of the transformed state of $\mathrm{NAU}_{\mathcal{P}}$.

The third case treats abstractions in r. I.e., sub-terms like $\lambda a.t'$ and $\lambda b.s'$ in t, s. It corresponds to the case where Abs applies. Abs performs α-renaming and generalizes the abstraction.

The last case considers the appearance of a suspension $\pi \cdot X^S$ in r. It represents sub-terms t', s' of t and s, respectively. $\mathrm{NAU}_{\mathcal{P}}$ keeps the sort of generalization variables minimal, always. For t', s' it is $\mathtt{fa}(t', s')$. Therefore $\pi \cdot X^S$ is more general than the generalization variable used by $\mathrm{NAU}_{\mathcal{P}}$ to represent t' and s'. The rule Mer ensures that variables are shared, whenever possible. It follows that $\mathrm{NAU}_{\mathcal{P}}(t, s)$ is an lgg of t and s. □

6 Conclusion

The language of permissive nominal terms introduced by Dowek et al. exhibits the same problem, w.r.t. the generalization type, as classical nominal terms. In order to address that issue, we suggest a modification of the definition of permission sets. This leads to a term language that is a superset of the one introduced by Dowek et al.

Our work might be seen as a starting point for a broader revision of the (permissive) nominal setting. The modified definition probably leads to implications w.r.t. unification, computational complexity, and so on. It might be practical to restrict the permission sets so that they have a simple and finite representation. An interesting restriction could be to only consider finite permission sets and the original ones from Dowek et al., i.e., sets of the form $\{a_1, \ldots, a_n\}$ where $a_i \in \mathbb{A}$ and the ones in $\mathcal{P}_D$. Those are exactly the ones needed to get a well-founded $\preceq$ relation. Note that $\mathrm{NAU}_{\mathcal{P}}$ computes generalizations within that restricted setting if the input terms satisfy the restriction, e.g., if the input is from $\mathcal{P}_D$.

Due to the well-founded $\preceq$ relation, our setting might be closer to higher-order pattern [16] (HOP) than other nominal settings [14]. Note that HOP have a well-founded $\preceq$ relation too [3].

Questions like, implications of our setting w.r.t. the results from [9], computational complexity of $\mathrm{NAU}_{\mathcal{P}}$, connections to other languages (e.g., HOP), etc. remain open. They are possible directions of future work that continues to investigate on the proposed setting, with and without imposing restrictions on the permission sets.

References

[1] Adam D. Barwell, Christopher Brown, and Kevin Hammond. Finding parallel functional pearls: Automatic parallel recursion scheme detection in Haskell functions via anti-unification. *Future Generation Comp. Syst.*, 79:669–686, 2018.

[2] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal anti-unification. In Maribel Fernández, editor, *26th International Conference on Rewriting Techniques and Applications, RTA 2015, June 29 to July 1, Warsaw, Poland*, volume 36 of *LIPIcs*, pages 57–73. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[3] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reasoning*, 58(2):293–310, 2017.

[4] Alexander Baumgartner and Daniele Nantes-Sobrinho. A, C, and AC nominal anti-unification. In Temur Kutsia and Andrew M. Marshall, editors, *Proceedings of the 34th International Workshop on Unification, UNIF 2020, Linz, Austria, June 29, 2020*, pages 5:1–5:6, 2020.

[5] Petr Bulychev and Marius Minea. An evaluation of duplicate code detection using anti-unification. In *Proc. 3rd International Workshop on Software Clones*, 2009.

[6] David M. Cerna and Temur Kutsia. A generic framework for higher-order generalizations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 10:1–10:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[7] David M. Cerna and Temur Kutsia. Higher-order pattern generalization modulo equational theories. *Math. Struct. Comput. Sci.*, 30(6):627–663, 2020.

[8] David M. Cerna and Temur Kutsia. Anti-unification and generalization: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 6563–6573. ijcai.org, 2023.

[9] Gilles Dowek, Murdoch Gabbay, and Dominic Mulligan. Permissive nominal terms and their unification: An infinite, co-infinite approach to nominal techniques. *Logic Journal of the IGPL*, 18:769–822, 10 2010.

[10] Cao Feng and Stephen Muggleton. Towards inductive generalisation in higher order logic. In Derek Sleeman and Peter Edwards, editors, *Machine Learning Proceedings 1992*, pages 154–162. Morgan Kaufmann, San Francisco (CA), 1992.

[11] Maribel Fernández and Murdoch Gabbay. Nominal rewriting. *Inf. Comput.*, 205(6):917–965, 2007.

[12] Warren D Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[13] Ulf Krumnack, Angela Schwering, Helmar Gust, and Kai-Uwe Kühnberger. Restricted higher-order anti-unification for analogy making. In Mehmet A. Orgun and John Thornton, editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *LNCS*, pages 273–282. Springer, 2007.

[14] Jordi Levy and Mateu Villaret. Nominal unification from a higher-order perspective. In Andrei Voronkov, editor, *Rewriting Techniques and Applications*, pages 246–260, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[15] Claudio L Lucchesi. The undecidability of the unification problem for third order languages. *Report CSRR*, 2059:129–198, 1972.

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

[17] Frank Pfenning. Unification and anti-unification in the calculus of constructions. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 74–85. IEEE Computer Society, 1991.

[18] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.

[19] Manfred Schmidt-Schauß and Daniele Nantes-Sobrinho. Nominal anti-unification with atom-variables. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel*, volume 228 of *LIPIcs*, pages 7:1–7:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[20] Christian Urban, Andrew M Pitts, and Murdoch J Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1-3):473–497, 2004.

Combined Abstract Congruence Closure for Associative or Commutative Theories

Christophe Ringeissen and Laurent Vigneron

Université de Lorraine, CNRS, Inria, Loria, F-54000 Nancy, France.
FirstName.LastName@loria.fr

1 Introduction

We design rule-based satisfiability procedures modulo unions of axiomatized theories, targeting equational axioms such as Associativity or Commutativity. In the proposed approach, any function symbol can be uninterpreted, associative only, commutative only, but also associative and commutative. To tackle these unions of theories, we introduce a combined congruence closure procedure that can be viewed as a particular Nelson-Oppen combination method [8] using particular congruence closure procedures for the individual theories. The combined congruence procedure is based on the ping-ponging of entailed equalities between (shared) constants. Actually, the congruence closure procedures used for the individual theories allow us to deduce these equalities. In this context, we consider terminating congruence closure procedures, but also non-terminating ones. Hence, we have terminating congruence closure procedures for Commutativity and Associativity-Commutativity, while the one for Associativity is non-terminating. We show how all the congruence closure procedures, including the combined one, can be presented in a uniform and abstract way along the lines of [3, 5, 7].

Related Work. Congruence closure modulo Associativity-Commutativity has been successfully investigated in [3, 4]. It has been revisited more recently, showing how the method can be extended to take into account additional orientable axioms, for instance to handle the theory of Abelian Groups [5]. The case of flat permutation axioms, such as Commutativity, has been considered in [7]. The theory of Groups and all of its subtheories including Associativity is considered in [6], where the related congruence closure procedure is not necessarily terminating, contrarily to the one known for Associativity-Commutativity. In these papers, some particular unions of theories are studied, for instance to handle several symbols following the same equational axioms.

In our paper, we clearly focus on the combination of congruence closure procedures to cope with arbitrary unions of (signature-disjoint) theories. This combination of congruence closure procedures can be seen as a particular case of combination of deduction-complete satisfiability procedures, already investigated in [12]. In addition to Associativity-Commutativity, we believe that it is interesting to consider Associativity alone and Commutativity alone. On one hand, Associativity provides a significant case study of a non-terminating congruence closure procedure. On the other hand, Commutativity leads to a simple extension of the congruence closure procedure known for the theory of equality as done in [7].

Paper Outline. In this paper, after explaining the notations used, we describe our combination method based on two kinds of processes: the orchestrator whose role is to prepare and handle a combination of theories; a theory process whose role is to complete the set of rewrite rules for a specific theory. Then we discuss the completeness of the method.

2 Preliminaries

We assume the reader familiar with the notions of terms and term rewriting [1].

We consider n theories $\mathcal{E}_1, \dots, \mathcal{E}_n$ such that each theory $\mathcal{E}_i$ is given by a set of equalities over the signature Σ_i. The theories $\mathcal{E}_1, \dots, \mathcal{E}_n$ are assumed to be pairwise signature-disjoint, meaning that $\Sigma_i \cap \Sigma_j = \emptyset$ for any $i, j \in [1, n], i \neq j$. The union of theories $\mathcal{E}_1 \cup \dots \cup \mathcal{E}_n$ is denoted by $\mathcal{E}$ and the union of signatures $\Sigma_1 \cup \dots \cup \Sigma_n$ is denoted by Σ. We assume a set of ground equalities Γ and a set of ground disequalities Δ, where both Γ and Δ are expressed over the signature Σ.

The process described in this paper relies on a flattening of terms. For theory $\mathcal{E}_i$ including an operator, say $+$, such that $(x + y) + z \approx x + (y + z)$ occurs in $\mathcal{E}_i$, this flattening will be performed using $+$ as a variadic operator, eg. $a + (b + c)$ is flattened into $+(a, b, c)$.

The initial set of ground equalities Γ will be purified via flattening thanks to the introduction of new constants (K denotes the set of used new constants taken from an infinite countable set U disjoint from Σ), generating pure flat rewrite rules for each theory $\mathcal{E}_i$ (denoted by the set R_i); and further deductions between those rules may generate flat equalities in this theory (denoted by the set E_i).

The rewrite rules in R_i can have two shapes: D-rules denoted by $f(c_1, \dots, c_n) \to c$, where $f \in \Sigma_i$ and $c_1, \dots, c_n, c \in K$; E-rules denoted by $f(c_1, \dots, c_m) \to f(d_1, \dots, d_n)$, where $f \in \Sigma_i$ is a variadic operator and $c_1, \dots, c_m, d_1, \dots, d_n \in K$. For any rewrite rule $t \to s$, t has to be greater than s ($t \succ s$); the definition of an ordering may be difficult for deduction systems modulo equational theories; but in our case the ordering is very simple as we only have to consider D-rules and E-rules: for D-rules, it suffices to assume $\forall f \in \Sigma, \forall c \in K, f \succ c$; for E-rules, we have to compare lists of constants: if of the same length, this can be done with a lexicographic or a multiset extension of an arbitrary ordering comparing two constants of K (the choice is done for each theory), and if of different length, the longest is the biggest. For example, for an associative theory the lexicographic extension will be used, and for an associative-commutative theory the multiset extension will be used.

The equalities in E_i also have two shapes: D-equalities denoted by $f(c_1, \dots, c_n) \approx c$, where $f \in \Sigma_i$ and $c_1, \dots, c_n, c \in K$; E-equalities denoted by $f(c_1, \dots, c_m) \approx f(d_1, \dots, d_n)$, where $f \in \Sigma_i$ is a variadic operator and $c_1, \dots, c_m, d_1, \dots, d_n \in K$. An equality $c_1 \approx c_2$ between two constants of K is called a C-equality. E-rules and E-equalities will be generated only for variadic operators by the Superposition inference rule, because of the use of extended rewrite rules (see Section 3.2).

3 Combined Satisfiability Procedure

We describe in this section a procedure that aims at (semi-)deciding the satisfiability of any set of ground equalities Γ together with any set of ground disequalities Δ, modulo a combination of signature-disjoint equational theories $\mathcal{E}_i$. This procedure, called `CombCC`, is based on congruence closure and involves two kinds of processes: an orchestrator decomposing the problem to separate the different theories, and theory processes that complete rewrite rules, one process for each theory.

3.1 The Orchestrator

The role of the orchestrator is to purify and flatten the problem to be solved, to send each theory process the rewrite rules it has to handle, and to detect if any contradiction wrt. Δ is generated by one of the theory processes. For this purpose, several sets are handled in addition to Γ and Δ: the set of new constants K, and for each equational theory $\mathcal{E}_i$ a set of rewrite rules R_i and a set of equalities E_i. In the following inference rules, we will only indicate the involved sets.

The input problem is given by two sets: a set of ground equalities Γ built over Σ plus a set of ground disequalities Δ built over Σ.

The first task of the orchestrator is to transform the disequalities for hiding the theories involved. This is done with the following inference rule that replaces an arbitrary disequality by a disequality between two new constants together with the equalities associating each of these constants to the corresponding term:

Splitting: $$\frac{K, \Delta \cup \{t_1 \not\approx t_2\}, \Gamma}{K \cup \{c_1, c_2\}, \Delta \cup \{c_1 \not\approx c_2\}, \Gamma \cup \{t_1 \approx c_1, t_2 \approx c_2\}}$$

if $t_1, t_2 \notin K$, $c_1, c_2 \in U \setminus K$

Once all disequalities have been decomposed, the second task of the orchestrator is to purify the equalities of Γ, by generating rewrite rules that are purely in one theory. In this purpose, it applies the following inference rules:

Flattening: $$\frac{K, \Gamma[t], R_i}{K \cup \{c\}, \Gamma[c], R_i \cup \{t \to c\}}$$

if $t \to c$ is a D-rule, $c \in U \setminus K$, t occurs in some equality in Γ that is not a D-equality, and t is Σ_i-rooted.

Orientation: $$\frac{K \cup \{c\}, \Gamma \cup \{t \approx c\}, R_i}{K \cup \{c\}, \Gamma, R_i \cup \{t \to c\}}$$ if $t \approx c$ is a D-equality and t is Σ_i-rooted.

When all equations have been transformed ($\Gamma = \emptyset$), the orchestrator runs one process per equational theory $\mathcal{E}_i$, providing it two sets of information: the set of new constants K and the set of D-rules R_i defined over Σ_i and K.

Its final task is to manage equalities between new constants, when generated by a theory process in some set E_i; there are two possibilities: if the equality contradicts a disequality of Δ then the system can stop, otherwise a constant has to be replaced by the other in all sets.

Contradiction: $$\frac{K \cup \{c, d\}, \Delta \cup \{c \not\approx d\}, RE \cup (R_i, E_i \cup \{c \approx d\})}{\bot}$$

Compression: $$\frac{K \cup \{c, d\}, \Delta, RE \cup (R_i, E_i \cup \{c \approx d\})}{K \cup \{d\}, \Delta\langle c \mapsto d\rangle, RE\langle c \mapsto d\rangle \cup (R_i\langle c \mapsto d\rangle, E_i\langle c \mapsto d\rangle)}$$

if $c \succ d$; the notation $\langle c \mapsto d\rangle$ denotes the homomorphic extension of the mapping σ defined as $\sigma(c) = d$ and $\sigma(x) = x$ for $x \neq c$, and $S\langle c \mapsto d\rangle$ denotes the set of equalities/rules obtained by applying the mapping $\langle c \mapsto d\rangle$ to each term in set S.

The strategy of the orchestrator can therefore be described by: $\mathbf{Split}^* \cdot (\mathbf{Flat}^* \cdot \mathbf{Ori})^* \cdot (\mathbf{Cont} \cup \mathbf{Comp})^*$.

3.2 A Theory Process

A process run for an equational theory $\mathcal{E}_i$ will use inference rules to complete its term rewriting system R_i. Some inference rules are used for transforming the rewrite rules (Composition), for deducing new equalities added to a set E_i (Collapse, Superposition), and for handling those new equalities (Simplification, Orientation, Deletion). The set of new constants K is never modified, so not indicated, but it is useful in the process for checking if a constant is a new one or belongs to the theory.

Considering the theory $\mathcal{E}_i$, if two terms t_1 and t_2 are $\mathcal{E}_i$-equal in this theory, we write $t_1 \leftrightarrow^*_{\mathcal{E}_i} t_2$. By $(R_i, \mathcal{E}_i)$ we denote the rewriting system defined by $\{u' \to v \mid u \to v \in R_i \text{ and } u' \leftrightarrow^*_{\mathcal{E}_i} u\}$. For some theories, the inference system has to consider extended rewrite rules as we do not explicitly use the axioms of a theory: an extension is built wrt. a context defined from the theory axioms; a context is a term in which a non variable position (denoted by $\cdot$) is reserved for placing the term to extend; let us denote $Cont_{\mathcal{E}_i}$ the set of contexts for the theory $\mathcal{E}_i$; given a D-rule or a E-rule $u \to v$, its extended version by a context $Cont[\cdot] \in Cont_{\mathcal{E}_i}$ is written $Cont[u] \to Cont[v]$. The construction of contexts for generating extensions has been explained in [10, 9, 13].

In this paper, as we want to handle only shallow rewrite rules, we will consider only theories for which extended rewrite rules have a shallow form. For example, if an operator f is associative, from the axiom of this theory $f(f(x,y),z) \approx f(x,f(y,z))$, we can build three shallow contexts: $f(\cdot,x)$, $f(x,\cdot)$ and $f(x_1,\cdot,x_2)$. So, a rewrite rule $f(a,b) \to c$ has three extensions: $f(a,b,x) \to f(c,x)$, $f(x,a,b) \to f(x,c)$ and $f(x_1,a,b,x_2) \to f(x_1,c,x_2)$.

By $(R_i^e, \mathcal{E}_i)$ we denote the rewriting system extending $(R_i, \mathcal{E}_i)$ with all possible extended rewrite rules from R_i.

The inference rules used by a theory process are the following.

Simplification: $$\frac{R_i, E_i[t]}{R_i, E_i[s]}$$ where t occurs in some equality of E_i, and $t \to_{(R_i^e,\mathcal{E}_i)} s$.

Orientation: $$\frac{R_i, E_i \cup \{t \approx s\}}{R_i \cup \{t \to s\}, E_i}$$ if $t \succ s$ and $t \to s$ is a D-rule or a E-rule.

Deletion: $$\frac{R_i, E_i \cup \{t \approx s\}}{R_i, E_i}$$ if $t \leftrightarrow^*_{\mathcal{E}_i} s$.

Composition: $$\frac{R_i \cup \{t \to s, u \to v\}, E_i}{R_i \cup \{t \to s', u \to v\}, E_i}$$ if $s \to_{(\{u \to v\}^e,\mathcal{E}_i)} s'$.

Collapse: $$\frac{R_i \cup \{t \to s, u \to v\}, E_i}{R_i \cup \{u \to v\}, E_i \cup \{t' \approx s\}}$$ if $t \to_{(\{u \to v\}^e,\mathcal{E}_i)} t'$, and if $t \leftrightarrow^*_{\mathcal{E}_i} u$ then $s \succ v$.

Superposition: $$\frac{R_i \cup \{t_1 \to s_1, t_2 \to s_2\}, E_i}{R_i \cup \{t_1 \to s_1, t_2 \to s_2\}, E_i \cup \{Cont_1[s_1]\sigma \approx Cont_2[s_2]\sigma\}}$$

if the substitution σ is the ground substitution in a minimal complete set of $\mathcal{E}_i$-unifiers of $Cont_1[t_1]$ and $Cont_2[t_2]$, where $Cont_1[\cdot], Cont_2[\cdot] \in Cont_{\mathcal{E}_i}$ are selected to guarantee a useful ground new equality; the resulting equality will be written in flat form.

A strategy for combining all those inference rules is: $(\mathbf{Com}^* \cdot (\mathbf{Col} \cup \mathbf{Sup}) \cdot \mathbf{Sim}^* \cdot (\mathbf{Del} \cup \mathbf{Ori}))^*$ So this process starts with a set of rewrite rules R_i and, if terminating, it ends with R_i^∞ where there is no possible inference rule involving rewrite rules of R_i^∞; intermediate equalities are stored in E_i. If an equality between two constants of K is generated, it will be handled by the orchestrator.

For applying inference rules, this theory process has to use a $\mathcal{E}_i$-matching algorithm for applying rewriting steps with respect to $(R_i^e, \mathcal{E}_i)$. It also needs a simple $\mathcal{E}_i$-unification algorithm, simple because it will have to solve unification problems of the shape $Cont_1[t_1] =^? Cont_2[t_2]$, where t_1 and t_2 are ground; if there is a solution, it will be the unique most general unifier since the variables occurring in $Cont_i[\cdot]$ will be instantiated by subterms of the ground term t_{3-i}.

4 Completeness Results

The combined satisfiability procedure described in the previous section is defined for "some theories $\mathcal{E}_i$". But for guaranteeing its completeness, we consider only three kinds of theories (in addition to the empty theory of course).

- *Commutative* theories are represented by the set of axioms
$$\{f(x_1, \ldots, x_k) \approx f(x_{\sigma(1)}, \ldots, x_{\sigma(k)}) \mid \sigma \text{ is a permutation of } \{1, \ldots, k\}\}$$
With such theories, there is no extension of rewrite rules to be considered, so the Superposition inference rule cannot apply. For the ordering, the arguments of a commutative operator are compared with a multiset extension of the ordering between constants of K.

- *Associative* theories are represented by axioms $f(f(x_1, x_2), x_3) \approx f(x_1, f(x_2, x_3))$. They generate three possible extensions of rewrite rules, with the contexts $f(\cdot, x)$, $f(x, \cdot)$ and $f(x_1, \cdot, x_2)$. Those three contexts can be used for applying term rewriting steps with respect to $(R_i^e, \mathcal{E}_i)$. But for the Superposition inference rule between two rules $t_1 \rightarrow s_1$ and $t_2 \rightarrow s_2$, we only need to consider their extensions $f(t_1, x_1) \rightarrow f(s_1, x_1)$ and $f(x_2, t_2) \rightarrow f(x_2, s_2)$ because this is the only combination of contexts for which the unification problem $(f(t_1, x_1) =^? f(x_2, t_2))$ can generate a ground most general unifier (any use of another context would generate a redundant equation). For the ordering, the arguments of an associative operator are compared with a lexicographic extension of the ordering between constants of K.

- *Associative-Commutative* theories are represented by axioms $f(x_1, x_2) \approx f(x_2, x_1)$ and $f(f(x_1, x_2), x_3) \approx f(x_1, f(x_2, x_3))$. They generate only one possible extension of rewrite rules, with the context $f(\cdot, x)$, used for applying term rewriting steps by $(R_i^e, \mathcal{E}_i)$, and the Superposition inference rule. For the ordering, the arguments of an AC operator are compared with a multiset extension of the ordering between constants of K.

The CombCC procedure is refutationally complete, provided that deductions are fairly applied. Moreover, if the CombCC procedure terminates without finding a contradiction with disequalities of Δ, it generates a terminating confluent term rewriting system for the equational theory $\mathcal{E} \cup \Gamma$.

Theorem 1. *Let $\mathcal{E}$ be any disjoint union of empty, commutative, associative, and associative-commutative theories over the combined signature Σ which is assumed to include uninterpreted function symbols and constants. Consider Γ is any set of ground Σ-equalities and Δ is any set of ground Σ-disequalities. Given the input $\Gamma \cup \Delta$, the CombCC procedure halts on $\perp$ if $\Gamma \cup \Delta$ is $\mathcal{E}$-unsatisfiable. If the CombCC procedure halts on an output distinct from $\perp$, then $\Gamma \cup \Delta$ is $\mathcal{E}$-satisfiable, and the output provides a rewriting system R such that (1) R is terminating and confluent modulo $\mathcal{E}$ on $T(\Sigma \cup K)$, and (2) any two ground terms in $T(\Sigma)$ are $\mathcal{E} \cup R$-equal iff they are $\mathcal{E} \cup \Gamma$-equal. Moreover, the CombCC procedure is necessarily terminating if $\mathcal{E}$ does not involve associative theories.*

Example 1. *Our procedure may indeed not terminate (if no contradiction exists) with associative theories. For example, if f and g are associative, given the equalities $\{f(a, b) \approx c, f(a, c) \approx f(c, a), g(b, a) \approx c, g(a, c) \approx g(c, a)\}$, either the theory process of f, or the one of g, will generate* *an infinite number of rewrite rules, depending on the chosen ordering between constants a and c deciding of the orientation of the second and fourth equalities. If $c \succ a$, the infinitely generated rules can be schematized by $f(a, c^n, b) \to f(c, c^n)$. If $a \succ c$, they can be schematized by $g(b, c^n, a) \to f(c, c^n)$. We also have non-terminating examples with a single associative theory, but they may be less simple to explain.*

To prove the completeness of `CombCC`, we can rely on a Nelson-Oppen combination method [8] based on the ping-ponging of entailed equalities between (shared) constants. This combination method is applicable without loss of completeness because one can rely on a union of convex and stably infinite theories, using the same proof idea as the one initiated in [2]. The theories we focus on are convex, since equational theories are Horn theories, and Horn theories are known to be convex [11]. Actually, the convexity induces a particular way to decide the satisfiability of equalities plus a conjunction of disequalities: it allows us to consider each disequality separately. Assuming convexity, a satisfiable set of equalities Γ together with a set of disequalities Δ is satisfiable if and only if for any $s \not\approx t \in \Delta$, we have that $s \approx t$ is not entailed by Γ. In our context, arbitrary satisfiability problems are equi-satisfiable, via flattening, to satisfiability problems including only flat literals, meaning that all the disequalities in Δ are of the form $c \not\approx d$ where c and d are constants. Thus, we are looking for inference systems with the property of being deduction-complete [12], in order to *derive* each equality $c \approx d$ such that $\Gamma \Rightarrow c \approx d$ is valid in the underlying theory. This is exactly the purpose of a congruence closure procedure when it applies to an input set of flat equalities Γ. It generates all the equalities between constants that are logically entailed by Γ.

In this paper, we consider terminating congruence closure procedures, but also non-terminating ones. Compared to a classical use of the Nelson-Oppen combination method, we have to accommodate procedures that are not necessarily terminating, as exemplified by Associativity.

Let us shortly explain why `CombCC` is refutationally complete. According to the completeness of the Nelson-Oppen method, the satisfiability problem in any disjoint union of stably infinite theories is reducible to the satisfiability problems in the component theories, provided that all possible arrangements are guessed. Consequently, given any disjoint union of stably infinite theories, using refutationally complete procedures for the satisfiability problems in the component theories allows us to get a refutationally complete procedure for the satisfiability problem in the union. In our context, stably infinite theories are also convex and so the guessing of all possible arrangements can be replaced by a ping-ponging of entailed equalities between constants. Then, we use the property that all the entailed equalities between constants are eventually generated since our congruence closure procedures are deduction-complete.

5 Conclusion

We have implemented the combination of those three kinds of theories (plus the empty theory) by extending AbstractCC [3]. The result is a very efficient procedure, even if the initial set of ground (dis)equalities contains very big terms.

We have defined the orchestrator so that it does not need to handle specific algorithms of theories $\mathcal{E}_i$. It could be more efficient using other inference rules like Simplification and Deletion. But we did this on purpose for the clarity of the paper. We are considering several extensions of our procedure, to apply it to any theory having a deduction system preserving the groundness of generated rules/equalities. This applies to flat permutative theories, an extension of commutative theories. It also applies to extensions of associative or associative-commutative theories where axioms can be used as shallow collapsing rewrite rules.

References

[1] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[2] Franz Baader and Cesare Tinelli. A New Approach for Combining Decision Procedure for the Word Problem, and Its Connection to the Nelson-Oppen Combination Method. In William McCune, editor, *CADE-14, 14th Int. Conference on Automated Deduction, Townsville, North Queensland, Australia, July 1997, Proceedings*, volume 1249 of *LNCS*, pages 19–33. Springer, 1997.

[3] Leo Bachmair, Ashish Tiwari, and Laurent Vigneron. Abstract Congruence Closure. *J. Autom. Reason.*, 31(2):129–168, 2003.

[4] Deepak Kapur. A Modular Associative Commutative (AC) Congruence Closure Algorithm. In Naoki Kobayashi, editor, *6th Int. Conference on Formal Structures for Computation and Deduction, FSCD, Buenos Aires, Argentina, July 2021 (Virtual Conference)*, volume 195 of *LIPIcs*, pages 15:1–15:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[5] Deepak Kapur. Modularity and Combination of Associative Commutative Congruence Closure Algorithms enriched with Semantic Properties. *Log. Methods Comput. Sci.*, 19(1), 2023.

[6] Dohan Kim. Congruence closure modulo groups. *CoRR*, abs/2310.05014, 2023.

[7] Dohan Kim and Christopher Lynch. Congruence Closure Modulo Permutation Equations. In Temur Kutsia, editor, *Proceedings of the 9th Int. Symposium on Symbolic Computation in Software Science, SCSS 2021, Hagenberg, Austria, September 2021*, volume 342 of *EPTCS*, pages 86–98, 2021.

[8] Greg Nelson and Derek C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[9] Gerald E. Peterson and Mark E. Stickel. Complete Sets of Reductions for Some Equational Theories. *J. ACM*, 28(2):233–264, 1981.

[10] Gordon Plotkin. Building-in equational theories. *Machine intelligence*, 7:73–90, 1972.

[11] Cesare Tinelli. Cooperation of Background Reasoners in Theory Reasoning by Residue Sharing. *J. Autom. Reason.*, 30(1):1–31, 2003.

[12] Duc-Khanh Tran, Christophe Ringeissen, Silvio Ranise, and Hélène Kirchner. Combination of convex theories: Modularity, deduction completeness, and explanation. *J. Symb. Comput.*, 45(2):261–286, 2010.

[13] Laurent Vigneron. Positive Deduction modulo Regular Theories. In Hans Kleine Büning, editor, *Computer Science Logic, 9th Int. Workshop, CSL '95, Annual Conference of the EACSL, Paderborn, Germany, September 1995, Selected Papers*, volume 1092 of *LNCS*, pages 468–485. Springer, 1995.

Nominal Commutative Narrowing (Work in progress)[*]

Daniella Santaguida[1] and Daniele Nantes-Sobrinho[1,2]

[1] University of Brasília, Brasília, Brazil
daniella@mat.unb.br
[2] Imperial College London, London, U.K
d.nantes-sobrinho@imperial.ac.uk

Abstract

Narrowing equational problems is a well-known technique that adds to rewriting the required power to search for solutions of equational problems. Both rewriting and narrowing techniques are well-studied in first-order languages, but there is still a lot to investigate when dealing with languages with binders, such as nominal language. In a previous paper, by the second author, the nominal narrowing relation was introduced. In this abstract, we present a work in progress on the development of nominal commutative narrowing as a technique to solve nominal commutative unification problems. We have extended the definitions of nominal rewriting and nominal narrowing to take into account equational theories, and we are one step away from proving the Lifting Theorem relating nominal commutative narrowing and rewriting. The goal of this abstract is to present our ongoing research and its challenges as well as to obtain feedback from the community.

1 Introduction

The nominal framework [10] emerged as a promising approach to deal with languages with binders, such as lambda calculus, first-order logic, etc. In the nominal setting, equality coincides with the α-equivalence relation, and freshness constraints are part of the nominal reasoning, and not deemed to the meta-language. For example, we can express within the nominal language, as $a\#M$, the fact that if a name a occurs in a term M, it must be abstracted (in other words, a is fresh for M). To reason within this language, nominal unification [13] was developed.

Extensions of nominal unification with equational theories are being investigated. On the one hand, when an equational theory E can be presented by a convergent nominal rewrite system, nominal E-unification via nominal narrowing was already investigated [6]. On the other hand, when such presentation by convergent rewriting system does not exist, different extensions for dealing with rewriting modulo E were necessary. Initially, it was necessary to define the extensions of nominal equality with the theories associativity (A), commutativity (C) and associativity-commutativity (AC) [4]. Only recently, algorithms to solve nominal C-unification problems (and their formalisations) were proposed [1, 2, 7, 5].

In this abstract, we are interested in giving another step towards a more general development nominal E, treating the case in which E cannot be oriented as a convergent nominal rewrite system, thus extending the works [11, 14, 8]. We will report on our work in progress developing the nominal C-narrowing and C-rewriting relations, as well as present some of the extensions of the nominal language that were necessary to prove the nominal version of the Lifting Theorem (Corollary 3.5), that relates both narrowing and rewriting. The Lifting Theorem is fundamental

[*] FAPDF DE 00193.00001175/21-11 and CNPq Universal 409003/21-2

to prove that nominal narrowing provides a sound and complete procedure for nominal C-unification. The theorem is partially proven, we are one-step away from completing the proof due to some extensions that still need to be done in the nominal language and properties that need to be verified.

2 Nominal Rewriting and Narrowing modulo E

In this section, we introduce our novel definitions of *equational nominal rewriting systems* (ENRS) and *equational nominal narrowing*, sometimes abbreviated to nominal E-rewriting systems and nominal E-narrowing. While we assume the reader's familiarity with nominal techniques, we briefly recap some basic definitions. For more details, we refer to [9].

Background. Fix countable infinite pairwise disjoint sets of *atoms* $\mathbb{A} = \{a, b, c, \dots\}$ and *variables* $\mathcal{X} = \{X, Y, Z, \dots\}$. Let Σ be a finite set of function symbols disjoint from $\mathbb{A}$ and $\mathcal{X}$ such that for each $f \in \Sigma$, a unique non-negative integer n (arity of f) is assigned. A *permutation* π is a bijection on $\mathbb{A}$ with finite domain, i.e., the set $\mathtt{dom}(\pi) := \{a \in \mathbb{A} \mid \pi(a) \neq a\}$ is finite. Nominal terms are defined inductively by the grammar: $s, t, u ::= a \mid \pi \cdot X \mid [a]t \mid f(t_1, \dots, t_n)$, where a is an *atom*, $\pi \cdot X$ is a moderated variable, $[a]t$ is the *abstraction* of a in the term t, and $f(t_1, \dots, t_n)$ is a *function application* with $f \in \Sigma$ and $f : n$. A term is *ground* if it does not contain (moderated) variables. A substitution is a mapping from variables (from $\mathcal{X}$) to (nominal) terms. Note that atoms are objects that can be bound and moderated variables are objects that can be instantiated by a substitution.

There are two kinds of constraints: $s \approx_\alpha t$ is an equality constraint; and $a\#t$ is a freshness constraint which means that a cannot occur unabstracted in t. Primitive constraints have the form $a\#X$ and ∇, Δ denote finite sets of primitive constraints. Judgements have the form $\Delta \vdash s \approx_\alpha t$ and $\Delta \vdash a\#t$ and are derived using the rules in Figure 1. In Figure 1 we use the *difference set* of two permutations $ds(\pi, \pi') := \{n \mid \pi \cdot n \neq \pi' \cdot n\}$. So $ds(\pi, \pi')\#X$ represents the set of constraints $\{n\#X \mid n \in ds(\pi, \pi')\}$. For example, if $\pi = (a\ b)(c\ d)$ and $\pi' = (c\ b)$, then $ds(\pi, \pi') = \{a, b, c, d\}$, and $ds(\pi, \pi')\#X = \{a\#X, b\#X, c\#X, d\#X\}$.

A term in context $\Delta \vdash t$ expresses that the term t has the freshness constraints imposed by Δ. For example, $a\#X \vdash f(X, h(b))$ expresses that a cannot occur free in instances of X when instantiating the term $f(X, h(b))$. Nominal rewriting rules can be defined under freshness constraints, i.e., $\nabla \vdash l \to r$.

Nominal rewriting relation $\to_{\mathsf{R}}$ is as expected:

$$\Delta \vdash s \to_{\mathsf{R}} t \iff s = \mathbb{C}[s'] \wedge \Delta \vdash \nabla\theta \wedge s' \approx_\alpha \pi \cdot (l\theta) \wedge t \approx_\alpha \mathbb{C}[\pi \cdot (r\theta)],$$

for a substitution θ, a subterm s' of s, a position $\mathbb{C}$ and a nominal rule $\nabla \vdash l \to r$.

Nominal E-rewriting and E-narrowing. Recall that an *equational term rewriting system* (ETRS), denoted $\mathsf{R}\cup\mathsf{E}$, is a set consisting of a theory T containing a set of axioms that can be split into a set R of rules and a set E of identities. To define (ETRS) we need to define the extended relation $\approx_{\alpha,\mathsf{E}}$, which takes into account α-equivalence and equality modulo E.

Remark 2.1. To define $\approx_{\alpha,\mathsf{E}}$ we need to extend the rules of Figure 1 with the dedicated rules for the identities defining E. For example, in the case of the theory C, we need to add the following rule

$$\frac{\Delta \vdash s_0 \ \approx_{\alpha,\mathsf{C}} \ t_i \qquad \Delta \vdash s_1 \ \approx_{\alpha,\mathsf{C}} \ t_{1-i} \qquad i = 0, 1}{\Delta \vdash f^{\mathsf{C}}(s_0, s_1) \ \approx_{\alpha,\mathsf{C}} \ f^{\mathsf{C}}(t_0, t_1)} (\approx_{\alpha,\mathsf{C}} \text{C})$$

$$\frac{}{\Delta \vdash a\#b}(\#\text{atom}) \qquad \frac{(\pi^{-1}\cdot a\#X)\in\Delta}{\Delta\vdash a\#\pi\cdot X}(\#\text{var}) \qquad \frac{}{\Delta\vdash a\#[a]t}(\#a[a]) \qquad \frac{\Delta\vdash a\#t}{\Delta\vdash a\#[b]t}(\#a[b])$$

$$\frac{\Delta\vdash a\#t_1 \ \cdots \ \Delta\vdash a\#t_n}{\Delta\vdash a\#f(t_1,\cdots,t_n)}(\#\text{app}) \qquad \frac{}{\Delta\vdash a\approx_\alpha a}(\approx_\alpha \text{atom}) \qquad \frac{ds(\pi,\pi')\#X\in\Delta}{\Delta\vdash \pi\cdot X\approx_\alpha \pi'\cdot X}(\approx_\alpha \text{var})$$

$$\frac{\Delta\vdash s_1 \ \approx_\alpha \ t_1 \ \cdots \ \Delta\vdash s_n \ \approx_\alpha \ t_n}{\Delta\vdash f(s_1,\cdots,s_n) \ \approx_\alpha \ f(t_1,\cdots,t_n)}(\approx_\alpha \text{app}) \qquad \frac{\Delta\vdash s \ \approx_\alpha \ t}{\Delta\vdash [a]s \ \approx_\alpha \ [a]t}(\approx_\alpha [aa])$$

$$\frac{\Delta\vdash s \ \approx_\alpha \ (a\ b)\cdot t \qquad \Delta\vdash a\#t}{\Delta\vdash [a]s \ \approx_\alpha \ [b]t}(\approx_\alpha [ab])$$

Figure 1: Rules for # and $\approx_\alpha$

where f^{C} denotes that f^{C} is a commutative function symbol. Rule ($\approx_\alpha$ app) only applies when the function symbol f is not commutative. In addition, we need to modify the rules in Figure 1 to use $\approx_{\alpha,\mathsf{C}}$ instead of $\approx_\alpha$.

Extending the definition of ETRS to nominal terms with respect to a theory E, we obtain the following definition. Below, $[t]_{\approx_\mathsf{E}}$, denotes the equivalence class of the nominal term t modulo E, i.e., $[t]_{\approx_\mathsf{E}} = \{t' \mid t' \approx_{\alpha,\mathsf{E}} t\}$.

Definition 2.2 (Equational nominal rewrite system)**.** Let E be set of identities and R a set of nominal rewrite rules. A nominal term-in-context $\Delta \vdash s$, reduces with respect to R/E, when its equivalence class modulo E reduces via $\to_{\mathsf{R}/\mathsf{E}}$ as below.

$$\Delta \vdash ([s]_{\approx_\mathsf{E}} \to_{\mathsf{R}/\mathsf{E}} [t]_{\approx_\mathsf{E}}) \text{ iff there exist } s', t' \text{ such that } \Delta \vdash (s \approx_{\alpha,\mathsf{E}} s' \to_\mathsf{R} t' \approx_{\alpha,\mathsf{E}} t).$$

That said, we call R/E an *equational nominal rewrite system* (ENRS). In particular, R/C is a commutative nominal rewrite system.

Here we are dealing with α, E-congruence classes and they are in general infinite due to the availability of names for α-renaming. Although the pure $\approx_\alpha$ relation is decidable, when $\approx_\alpha$ is put together with an equational theory E which contains infinite congruence classes, the relation $\to_{\mathsf{R}/\mathsf{E}}$ may not be decidable (as in first-order). We will define the nominal relation $\to_{\mathsf{R},\mathsf{E}}$ that deals with nominal E-matching instead of inspecting the whole α, E-congruence class of a term.

Definition 2.3 (Nominal E-rewriting)**.** The *one-step* E*-rewrite relation* $\Delta \vdash s \to_{\mathsf{R},\mathsf{E}} t$ is the least relation such that for any $R = (\nabla \vdash l \to r) \in \mathsf{R}$, position $\mathbb{C}$, term s', permutation π, and substitution θ,

$$\frac{s \equiv \mathbb{C}[s'] \qquad \Delta \vdash \left(\nabla\theta,\ s' \approx_{\alpha,\mathsf{E}} \pi\cdot(l\theta),\ \mathbb{C}[\pi\cdot(r\theta)] \approx_{\alpha,\mathsf{E}} t\right)}{\Delta \vdash s \to_{\mathsf{R},\mathsf{E}} \ t}$$

The E-*rewrite relation* $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} t$ is the least relation that includes $\to_{\mathsf{R},\mathsf{E}}$ and satisfies: (i) for all Δ, s, s' we have $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} s'$ if $\Delta \vdash s \approx_{\alpha,\mathsf{E}} s'$; (ii) for all Δ, s, t, u we have that $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} t$ and $\Delta \vdash t \to^*_{\mathsf{R},\mathsf{E}} u$ implies $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} u$. If $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} t$ and $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} t'$, then we say that R is E-*confluent* when there exists a term u such that $\Delta \vdash t \to^*_{\mathsf{R},\mathsf{E}} u$ and $\Delta \vdash t' \to^*_{\mathsf{R},\mathsf{E}} u$. Also, R is said to be E-*terminating* if there is no infinite $\to_{\mathsf{R},\mathsf{E}}$ sequence. An ENRS R is called E-*convergent* if it is E-confluent and E-terminating. A term t is said to be in R, E-normal form (R/E-normal form) whenever one cannot apply another step of $\to_{\mathsf{R},\mathsf{E}}$ ($\to_{\mathsf{R}/\mathsf{E}}$).

In first-order languages, it is known that R,E-reducibility is decidable if the E-matching is decidable. Following Jouannaud et. al. [11], the existence of a finite and complete E-unification algorithm is a sufficient condition for that decidability. However solving nominal E-unification problems has the additional complication of dealing with renaming and freshness conditions, and these have a significant impact in obtaining finite and complete set of nominal E-unification algorithms.

Remark 2.4. Nominal C-unification is not finitary when one uses freshness constraints and substitutions for representing solutions [2], but the type of problems that generate an infinite set of C-unifiers are fixed-point equations $\pi\cdot X{}_?\overset{\mathsf{C}}{\approx}_?X$. For e.g., the nominal C-unification problem $(a\ b)\cdot X{}_?\overset{\mathsf{C}}{\approx}_?X$ has solutions $[X\mapsto a\oplus b], [X\mapsto (a\oplus b)\oplus(a\oplus b)], \ldots$. However, these problems do not appear in nominal C-matching [3]. Thus, the relation $\to_{\mathsf{R},\mathsf{C}}$ is decidable.

Now we define the nominal narrowing relation modulo E, extending previous works [6].

Definition 2.5 (Nominal E-narrowing)**.** The *one-step* E*-narrowing relation* $(\Delta\vdash s)\rightsquigarrow_{\mathsf{R},\mathsf{E}}(\Delta'\vdash t)$ is the least relation such that for any $R=(\nabla\vdash l\to r)\in\mathsf{R}$, position $\mathbb{C}$, term s', permutation π, and substitution θ,

$$\frac{s\equiv\mathbb{C}[s'] \qquad \Delta'\vdash\big(\nabla\theta,\ \Delta\theta,\ s'\theta\approx_{\alpha,\mathsf{E}}\pi\cdot(l\theta),\ (\mathbb{C}[\pi\cdot r])\theta\approx_{\alpha,\mathsf{E}}t\big)}{(\Delta\vdash s)\rightsquigarrow_{\mathsf{R},\mathsf{E}}(\Delta'\vdash t)}\,.$$

The *nominal* E*-narrowing relation* $(\Delta\vdash s)\rightsquigarrow^*_{\mathsf{R},\mathsf{E}}(\Delta'\vdash t)$ is the least relation that includes $\rightsquigarrow_{\mathsf{R},\mathsf{E}}$ and satisfies: (i) for all Δ,s,s' we have $(\Delta\vdash s)\rightsquigarrow^*_{\mathsf{R},\mathsf{E}}(\Delta\vdash s')$ if $\Delta\vdash s\approx_{\alpha,\mathsf{E}}s'$; (ii) for all $\Delta,\Delta',\Delta'',s,t,u$ we have that $(\Delta\vdash s)\rightsquigarrow^*_{\mathsf{R},\mathsf{E}}(\Delta'\vdash t)$ and $(\Delta'\vdash t)\rightsquigarrow^*_{\mathsf{R},\mathsf{E}}(\Delta''\vdash u)$ implies $(\Delta\vdash s)\rightsquigarrow^*_{\mathsf{R},\mathsf{E}}(\Delta''\vdash u)$.

The permutation π and substitution θ above are found by solving the E-unification problem $(\nabla\vdash l)\ {}_?\overset{\mathsf{E}}{\approx}_?\ (\Delta\vdash s')$. In this work, we will focus on the theory C, for which a nominal unification algorithm exists. From now on, our results are concentrated on $\to_{\mathsf{R}/\mathsf{C}},\to_{\mathsf{R},\mathsf{C}}$ and $\rightsquigarrow_{\mathsf{R},\mathsf{C}}$.

Since nominal C-narrowing is defined on nominal C-unification, which is not finitary when we use pairs (Δ',θ) of freshness contexts and substitutions for representing solutions, and following the Remark 2.4, we can conclude that our nominal C-narrowing trees are infinitely branching.

Example 1. Consider the signature $\Sigma=\{h:1,f:2,\oplus:2\}$, where $f,\oplus$ are commutative symbols. Let $R=\{\vdash h(Y)\to h(Y),\ \vdash f([a][b]\cdot Z,Z)\to f(h(Z),h(Z))\}$ be a set of rewrite[1] rules and $\mathsf{C}=\{\vdash f(X,Y)\approx f(Y,X),\ \vdash X\oplus Y\approx Y\oplus X\}$ be the axioms defining the theory.

Let $\vdash h(f([b][a]X,X))$ be a nominal term that we want to apply nominal narrowing modulo C. Observe that we can apply one step of narrowing, and then we obtain a branch that yields infinite branches due to the fixed-point equation (see Figure 2).

3 Nominal Lifting Theorem modulo C

Let $\mathsf{R}=\{\nabla_i\vdash l_i\to r_i\}$ be a C-convergent NRS. We want to establish correspondence between nominal C-narrowing and nominal C-rewriting. We will do that via an extension of the Nominal Lifting Theorem (cf. Theorem 12 [6]) for the extended relations $\rightsquigarrow_{\mathsf{R},\mathsf{C}}$ and $\to_{\mathsf{R},\mathsf{C}}$.

We start by defining a normalised substitution with respect to the relation $\to_{\mathsf{R},\mathsf{C}}$:

[1] $\vdash l\to r$ denotes $\emptyset\vdash l\to r$.

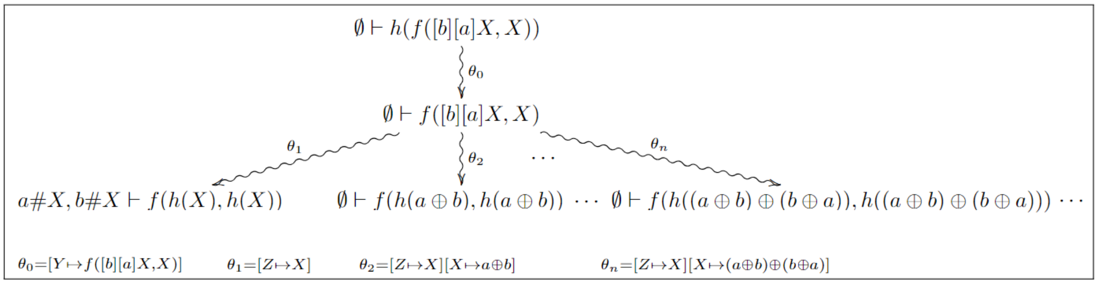

$\emptyset \vdash h(f([b][a]X, X))$
θ_0
$\emptyset \vdash f([b][a]X, X)$
θ_1
θ_2
θ_n
$\dots$
$a\#X, b\#X \vdash f(h(X), h(X))$
$\emptyset \vdash f(h(a \oplus b), h(a \oplus b))$ $\cdots$
$\emptyset \vdash f(h((a \oplus b) \oplus (b \oplus a)), h((a \oplus b) \oplus (b \oplus a)))$ $\cdots$
$\theta_0{=}[Y{\mapsto}f([b][a]X,X)]$
$\theta_1{=}[Z{\mapsto}X]$
$\theta_2{=}[Z{\mapsto}X][X{\mapsto}a\oplus b]$
$\theta_n{=}[Z{\mapsto}X][X{\mapsto}(a\oplus b)\oplus(b\oplus a)]$

Figure 2: Infinitely branching tree

$$\begin{tikzcd}
(\Delta_0 \vdash s_0) \arrow[r, squiggly, "\theta_0"] \arrow[d, dashed, "\rho_0"] & (\Delta_1 \vdash s_1) \arrow[r, squiggly, "*"] \arrow[d, dashed, "\rho_1"] & (\Delta_i \vdash s_i) \arrow[r, squiggly, "\theta_i"] \arrow[d, dashed, "\rho_i"] & (\Delta_{i+1} \vdash s_{i+1}) \arrow[r, squiggly, "*"] \arrow[d, dashed, "\rho_{i+1}"] & \dots \arrow[r, squiggly, "\theta_n"] & (\Delta_n \vdash s_n) \arrow[d, dashed, "\rho_n"] \\
\Delta \vdash s_0\rho_0 \arrow[r] & s_1\rho_1 \arrow[r, "*"] & s_i\rho_i \arrow[r] & s_{i+1}\rho_{i+1} \arrow[r, "*"] & \dots \arrow[r] & s_n\rho_n
\end{tikzcd}$$

Figure 3: Corresponding Narrowing to Rewriting Derivations

Definition 3.1 (Normalised substitution w.r.t $\to_{\mathsf{R,C}}$)**.** A substitution θ is *normalised in* Δ *with relation to* $\to_{\mathsf{R,C}}$ if $\Delta \vdash X\theta$ is a $\mathsf{R,C}$-normal form in R for every X. A substitution θ satisfies the freshness context Δ if there exists a freshness context ∇ such that $\nabla \vdash a\#X\theta$ for each $a\#X \in \Delta$. The minimal such ∇ is $\langle \Delta\theta \rangle_{nf}$, the latter which denotes the normal form of the set of freshness constraints $\Delta\theta$, after bottom-up simplification using the rules in Figure 1.

The next result (correctness) states that for each finite sequence of narrowing steps, there exists a finite sequence of rewriting steps.

Theorem 3.2. *($\rightsquigarrow^*_{\mathsf{R,C}}$ to $\to^*_{\mathsf{R,C}}$) Let $(\Delta_0 \vdash s_0) \rightsquigarrow^*_{\mathsf{R,C}} (\Delta_n \vdash s_n)$ be a nominal* C*-narrowing derivation. Let ρ be a substitution satisfying Δ_0, i.e., there exists Δ such that $\Delta \vdash \Delta_0\rho$. Then, there exists a rewriting derivation*

$$\Delta \vdash s_0\rho_0 \to^*_{\mathsf{R,C}} s_n\rho$$

such that $\Delta \vdash \Delta_i\rho_{i+1}$ and $\rho_i = \theta_i \dots \theta_{n-1}\rho$, for all $0 \le i < n$. In other words,

$$\Delta \vdash (s_0\theta)\rho \to^*_{\mathsf{R,C}} s_n\rho$$

where θ is the composition of the successive $\mathsf{R,C}$*-narrowing substitutions.*

Proof. By induction on the length of the narrowing derivation and illustrated in Figure 3. □

The proof of the converse (completeness) is more challenging. In the first-order case, the approach to prove that each finite sequence of $\to_{\mathsf{R,C}}$ steps corresponds to a finite sequence of $\rightsquigarrow_{\mathsf{R,C}}$ steps relies on an additional property: E-coherence. This property can be extended to the nominal framework as follows:

Definition 3.3 (Nominal E-Coherence)**.** The relation $\Delta \vdash \_ \to_{\mathsf{R,E}} \_$ is called E-*coherent* iff for all t_1, t_2, t_3, t_4 such that $\Delta \vdash t_1 \approx_{\alpha,\mathsf{E}} t_2$ and $\Delta \vdash t_1 \to_{\mathsf{R,E}} t_3 \to^*_{\mathsf{R/E}} t_4$, there exist t_5, t_6, t_7 such that $\Delta \vdash t_4 \to^*_{\mathsf{R/E}} t_5$, $t_2 \to_{\mathsf{R,E}} t_6 \to^*_{\mathsf{R/E}} t_7$ and $\Delta \vdash t_5 \approx_{\mathsf{E}} t_7$, for some Δ.

$$\begin{tikzcd}
\Delta \vdash t_1 \arrow[r, "\mathsf{R,E}"'] \arrow[d, squiggly, "\approx_{\alpha,\mathsf{E}}"] & t_3 \arrow[r, "*", "\mathsf{R/E}"'] & t_4 \arrow[r, dashed, "*", "\mathsf{R/E}"'] & t_5 \arrow[d, squiggly, "\approx_{\alpha,\mathsf{E}}"] \\
\Delta \vdash t_2 \arrow[r, dashed, "\mathsf{R,E}"'] & t_6 \arrow[rr, dashed, "*", "\mathsf{R/E}"'] & & t_7
\end{tikzcd}$$

The diagram above illustrates nominal E-coherence: the dashed lines represent existentially quantified reductions.

Jouannaud et. al. [11, 12] proved, in the first-order case, that $\rightarrow_{\mathsf{R/E}}$ and $\rightarrow_{\mathsf{R,E}}$ coincide, if $\rightarrow_{\mathsf{R,E}}$ is E-coherent. We conjecture that the same result holds for the nominal framework, as long as E is a first-order theory (it does not contain bindings and/or freshness constraints).

Conjecture 3.1. *Let* E *be a first-order theory and* R *be a nominal rewrite system that is* E*-confluent and* E*-terminating. Then the* R,E*- and* R/E*-normal forms of any term* t *are* E*-equal iff* $\rightarrow_{\mathsf{R,E}}$ *is* E*-coherent.*

Proof. As in the first order case, we conjecture that the proof will follow by case analysis on the normal forms $t{\downarrow_{\mathsf{R,E}}}$ and $t{\downarrow_{\mathsf{R/E}}}$ of a term t. In addition, it will use the fact that $\rightarrow_{\mathsf{R,E}}\subseteq\rightarrow_{\mathsf{R/E}}$. □

Conjecture 3.1 is used in the proof of Theorem 3.4, which we call a *naive completeness*: the exact conditions on the freshness contexts have to be further investigated and Conjecture 3.1 has to be proven.

Theorem 3.4 (Naive version of Proposition 3 in [11])**.** *Let* $\mathsf{R}\cup\mathsf{C}$ *be an ENRS such that* R *is* C*-confluent and* C*-terminating and* $\rightarrow_{\mathsf{R,C}}$ *is* C*-coherent. Let* V_0 *be a finite set of variables containing* $V = V(\Delta_0, s_0)$*. Then, for any* R,C*-derivation*

$$\Delta \vdash t_0 = s_0\rho_0 \rightarrow^*_{\mathsf{R,C}} t_0{\downarrow}$$

to any of its R,C*-normal forms, say* $t_0{\downarrow}$*, where* $\mathit{dom}(\rho_0) \subseteq V(s_0) \subseteq V_0$ *and* ρ_0 *is a* R,C*-normalised substitution that satisfies* Δ_0 *with* Δ*, there exist a* R,C*-narrowing derivation*

$$(\Delta_0 \vdash s_0) \rightsquigarrow^*_{\mathsf{R,C}} (\Delta_n \vdash s_n),$$

for each i*,* $0 \le i < n$*, with the composition of substitutions* θ*, and a* R,C*-normalised substitution* ρ_n *such that* $\Delta \vdash s_n\rho_n \approx_{\alpha,\mathsf{C}} t_0{\downarrow}$ *and* $\Delta \vdash \rho_0|_V \approx_{\alpha,\mathsf{C}} \theta\rho_n|_V$*.*

Proof. By induction on the length of the rewriting derivation. □

Thus, we obtain our main result:

Corollary 3.5 (C-Lifting Theorem)**.** *Nominal lifting modulo* C *is a consequence of Theorem 3.2 and Theorem 3.4.*

The C-Lifting Theorem is fundamental to prove that nominal narrowing provides a sound and complete procedure for nominal C-unification.

4 Conclusion and Future Work

In this work, we proposed definitions for nominal R,C-rewriting and R,C-narrowing and proved some properties relating them. So far, we have partially proved the nominal version of the Lifting Theorem taking into account commutativity. The next step is to complete its proof. Since nominal C-unification based on freshness constraints only is not finitary, our nominal C-narrowing tree is not finite. As future work, we plan to investigate alternative approaches to nominal C-unification for which the representation of solutions is finite, such as the approach using fixed-point constraints.

References

[1] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, and Daniele Nantes-Sobrinho. Nominal c-unification. In Fabio Fioravanti and John P. Gallagher, editors, *Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017, Namur, Belgium, October 10-12, 2017, Revised Selected Papers*, volume 10855 of *Lecture Notes in Computer Science*, pages 235–251. Springer, 2017.

[2] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, and Daniele Nantes-Sobrinho. On solving nominal fixpoint equations. In Clare Dixon and Marcelo Finger, editors, *Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings*, volume 10483 of *Lecture Notes in Computer Science*, pages 209–226. Springer, 2017.

[3] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, and Daniele Nantes-Sobrinho. A formalisation of nominal C-matching through unification with protected variables. In Beniamino Accattoli and Carlos Olarte, editors, *Proc. of the 13th Workshop on Logical and Semantic Frameworks with Applications, LSFA 2018, Fortaleza, Brazil, September 26-28, 2018*, volume 344 of *ENTCS*, pages 47–65. Elsevier, 2018.

[4] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, Daniele Nantes-Sobrinho, and Ana Cristina Rocha Oliveira. A formalisation of nominal α-equivalence with A, C, and AC function symbols. *Theor. Comput. Sci.*, 781:3–23, 2019.

[5] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, Gabriel Ferreira Silva, and Daniele Nantes-Sobrinho. Formalising nominal c-unification generalised with protected variables. *Math. Struct. Comput. Sci.*, 31(3):286–311, 2021.

[6] Mauricio Ayala-Rincón, Maribel Fernández, and Daniele Nantes-Sobrinho. Nominal narrowing. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[7] Mauricio Ayala-Rincón, Maribel Fernández, and Daniele Nantes-Sobrinho. Fixed-point constraints for nominal equational unification. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[8] Santiago Escobar, José Meseguer, and Ralf Sasse. Variant narrowing and equational unification. In Grigore Rosu, editor, *Proceedings of the Seventh International Workshop on Rewriting Logic and its Applications, WRLA 2008, Budapest, Hungary, March 29-30, 2008*, volume 238 of *Electronic Notes in Theoretical Computer Science*, pages 103–119. Elsevier, 2008.

[9] Maribel Fernández and Murdoch Gabbay. Nominal rewriting. *Inf. Comput.*, 205(6):917–965, 2007.

[10] Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects Comput.*, 13(3-5):341–363, 2002.

[11] Jean-Pierre Jouannaud, Claude Kirchner, and Hélène Kirchner. Incremental construction of unification algorithms in equational theories. In Josep Díaz, editor, *Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings*, volume 154 of *Lecture Notes in Computer Science*, pages 361–373. Springer, 1983.

[12] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986.

[13] Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.

[14] Emanuele Viola. E-unifiability via narrowing. In Antonio Restivo, Simona Ronchi Della Rocca, and Luca Roversi, editors, *Theoretical Computer Science, 7th Italian Conference, ICTCS 2001, Torino, Italy, October 4-6, 2001, Proceedings*, volume 2202 of *Lecture Notes in Computer Science*, pages 426–438. Springer, 2001.

On Testing Convexity of 2DNF

Josué A. Ruiz[1], Amir Masoumzadeh[1], Paliath Narendran[1], and Padmavathi Iyer[2]

[1] University at Albany–SUNY (USA),
e-mail: {jaruiz, amasoumzadeh, pnarendran}@albany.edu
[2] Drury University (USA)
e-mail: riyer@drury.edu

Abstract

Testing the convexity of boolean formulae has applications in problems such as the convertibility of access control policies. In this paper, we report on our ongoing work on developing a polynomial algorithm for testing the convexity of the set of minterms of a disjunctive normal form formula where every term has exactly two literals in it (2DNF).

1 Introduction and Preliminaries

Our goal in this paper is to develop an algorithm for testing the convexity of the set of minterms of a boolean formula in disjunctive normal form where every term has exactly two literals in it (2DNF). The motivation for this problem comes from the convertibility problem for rule-based access control policies [4, 5].

Let $X = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. Expressions, terms (products or conjuncts), and minterms are defined as usual [3]. Given a set of Boolean variables X, we denote the set of all possible minterms as $\mathcal{M}_X$. For a Boolean expression Ψ, let $\mu(\Psi)$ be the set of its minterms. Note that each minterm of an expression can also be viewed as a (representation of a) satisfying assignment for that expression.

We define a partial order $\leq$ on bit strings (of the same length) as follows: first define the order on single bits as $0 \leq 0,\ 0 \leq 1$ and $1 \leq 1$. This is extended to bit strings X and Y as

- $X \leq Y$ iff $X[j] \leq Y[j]$ for all j and
- $X < Y$ iff $X \leq Y$ and $X \neq Y$.

Definition 1.1. *A set of minterms M is* convex *if and only if for every pair $m_1 \leq m_2 \in M$:*

$$\{ \ m \mid m_1 \leq m \leq m_2 \} \ \subseteq \ M.$$

Definition 1.2. *Let M be a set of minterms. Its* upward closure *$M^{\uparrow}$ is defined as*

$$M^{\uparrow} = \{ \ u \mid \exists m \in M : m \leq u \}.$$

Definition 1.3. *A set of minterms M is* upward closed *if and only if $M = M^{\uparrow}$.*

Definition 1.4. *For a term t, the product of its positive literals is denoted by $\Pi(t)$ and the product of its negative literals is denoted by $\mathcal{N}(t)$.*

If there are no positive literals in a term t, then $\Pi(t) = 1$. Similarly, if there are no negative literals in a term t, then $\mathcal{N}(t) = 1$.

Definition 1.5. *Given two product terms t_1 and t_2, their* separator*, denoted as* $\mathrm{sep}(t_1, t_2)$*, is $\Pi(t_2)\mathcal{N}(t_1)$, i.e., the conjunction of the positive literals of t_2 and the negative literals of t_1.*

For instance, if $t_1 = x_1\overline{x_2}$ and $t_2 = x_3\overline{x_4}$ then $sep(t_1, t_2) = \overline{x_2}x_3$.

In an earlier paper [5], we showed that:

Lemma 1.1. *The set of minterms of a boolean expression Φ in DNF is convex if and only if every separator is an implicant of Φ.*

Another characterization of convexity is as follows.

Lemma 1.2. *The set of minterms of a boolean expression Φ in DNF is convex if and only if there exist* positive *DNF expressions Ψ_1 and Ψ_2 such that*

$$\Phi \equiv \Psi_1 \wedge \neg\Psi_2$$

Note that a boolean expression in DNF is said to be in *positive* DNF form if and only if no negated literals appear in it.

Our goal in this paper is to design an efficient algorithm for checking the convexity of an expression in DNF. By Lemma 1.2 this problem can be formulated as a matching problem as follows:

Instance: A formula Φ in DNF.

Question: Are there *positive* DNF formulae $\mathcal{A}$ and $\mathcal{B}$ such that $\Phi \equiv \mathcal{A} \wedge \neg\mathcal{B}$?

We consider in this paper a restricted version of this problem where every product term in the DNF formula *has exactly two literals.* There are three main cases:

(a) There exists an all-positive term and an all-negative term.

(b) Every term is *mixed*, i.e., every term has a negated literal and an unnegated literal. Thus we have neither all-positive terms nor all-negative terms. We refer to such formulae as “all-mixed 2DNF.”

(c) There is an all-positive term, but no all-negative term. (The dual case where there is an all-negative term but no all-positive terms is similar.)

Case (a) is the most straightforward. The set of minterms $\mu(t)$ of an all-positive term t contains the highest minterm. Similarly if t is all-negative, then $\mu(t)$ contains the lowest minterm.

Hence, the set of minterms of such a formula is convex if and only if the formula is valid [4]. The validity of such formulae can be checked in linear time [1].

In Section 2, we discuss Case (b), i.e., where every product term is of the form $x_i\overline{x_j}$. We derive a graph-theoretic characterization of the implication graph of the *negation* of such formulae (which clearly will be in CNF). This leads to a linear algorithm for testing convexity. We also briefly discuss a quadratic-time algorithm for Case (c) in Section 3, and conclude the paper in Section 4.

2 Linear-Time Algorithm For All-Mixed Case

Let Φ be an all-mixed 2DNF formula and let $\neg\Phi$ be its complement in CNF. Let $IG(\neg\Phi)$ be the implication graph of $\neg\Phi$ [1]:

- For each variable x_i, we add two nodes named x_i and $\overline{x}_i$.
- For each clause $u \vee v$ of $\neg\Phi$, we add edges $\overline{u} \rightarrow v$ and $\overline{v} \rightarrow u$.

Note that every clause in $\neg\Phi$ is of the form $\overline{x_i}\ \vee\ x_j$. Hence we only keep the part of the graph with nodes with positive literals since there are no edges between nodes with positive literals and nodes with negative literals.

Lemma 2.1. *Let Φ be an all-mixed 2DNF formula and let x_1 and x_2 be two distinct variables. Then $x_1\overline{x_2}$ is an implicant of Φ if and only if there is a path from x_1 to x_2 in $IG(\neg\Phi)$.*

Lemma 2.2. *Let Φ be an all-mixed 2DNF formula. Then Φ is convex if and only if the following holds for all distinct variables x_1, x_2, x_3, x_4:*

> *if $x_1\overline{x_2}$ and $x_3\overline{x_4}$ are terms in Φ then there are paths in $IG(\neg\Phi)$ from x_1 to x_4 and from x_3 to x_2.*

Proof. Follows from Lemmas 1.1 and 2.1, since $x_1\overline{x_4}$ and $\overline{x_2}x_3$ are separators. □

Since the implication graph may be cyclic, we will also need to consider the *component graph* $CIG(\neg\Phi)$ of the implication graph. This is obtained by coalescing all nodes in a strongly-connected component (SCC) into one node. This component graph can be constructed in time linear in the size of the original digraph [2].

Lemma 2.3. *Let Φ be an all-mixed DNF formula. Then Φ is convex* only if *the following holds for all distinct variables x_1, x_2, x_3, x_4:*

> *If $x_1\overline{x_2}\ \vee\ x_2\overline{x_3}\ \vee\ x_3\overline{x_4}$ is a subexpression of Φ then x_2 and x_3 belong to the same strongly-connected component in $CIG(\neg\Phi)$.*

Proof. There must be a path in $IG(\neg\Phi)$ from x_3 to x_2 by Lemma 2.2. □

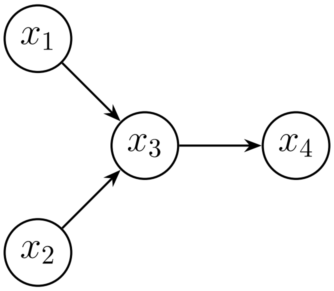
Corollary 2.3.1. *Let Φ be an all-mixed DNF formula. If Φ is convex, then $CIG(\neg\Phi)$ cannot contain a path of 3 edges.*

We now have to consider two separate cases:

1. All paths in $CIG(\neg\Phi)$ are of length 1.
2. There is a path of length 2 in $CIG(\neg\Phi)$.

(There is also the case where $CIG(\neg\Phi)$ has exactly one node, but that case is taken care of by the definition of radial dags given below.)

We call a dag G *radial* if there is a unique node v such that there is an edge from every source node to v, there is an edge to every sink node from v, and every node other than v is either a source node or a sink node. In other words, the set of nodes V can be partitioned into 3 disjoint subsets (V_1, $\{v\}$, V_2) such that V_1 is the set of source nodes, V_2 is the set of sink nodes, every node in V_1 is connected to v and v is connected to every node in V_2. (See Figure 1: Nodes x_1 and x_2 are source nodes, x_4 is the only sink node and x_3 is the "middle node.")



A *bipartite dag* is a dag where the set of nodes V can be partitioned into two disjoint subsets V_1 and V_2, such that every edge is from a node in V_1 to a node in V_2. In other words, every node in V_1 is a source node, and every node in V_2 is a sink node. A *complete bipartite dag* is a dag where the set of nodes V can be partitioned into two disjoint subsets V_1 and V_2, such that every node in V_1 has an edge to every node in V_2.

Lemma 2.4. *Let Φ be an all-mixed DNF formula such that $CIG(\neg\Phi)$ has no edges at all. Then Φ is convex if and only if $CIG(\neg\Phi)$ has only one node.*

Lemma 2.5. *Let Φ be an all-mixed DNF formula such that all paths in $CIG(\neg\Phi)$ are of length 1. Then Φ is convex if and only if $CIG(\neg\Phi)$ a complete bipartite dag.*

Lemma 2.6. *Let Φ be an all-mixed DNF formula such that there is a path of length 2 in $CIG(\neg\Phi)$. Then Φ is convex if and only if $CIG(\neg\Phi)$ is a radial dag.*

Theorem 1. *An all-mixed DNF formula Φ is convex if and only if $CIG(\neg\Phi)$ is either a complete bipartite dag or a radial dag.*

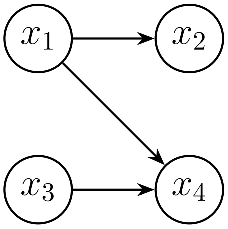
Proof. Follows from Corollary 2.3.1 and lemmas 2.5 and 2.6. □

The above theorem provides a linear-time algorithm for the all-mixed case.

Example 2.1. *Consider the DNF expression* $\Phi = x_1\overline{x_2} \ \vee \ \overline{x_1}x_2$*. CIG*$(\neg\Phi)$ *has exactly one node* $\{1, 2\}$*. Thus this expression is convex.*

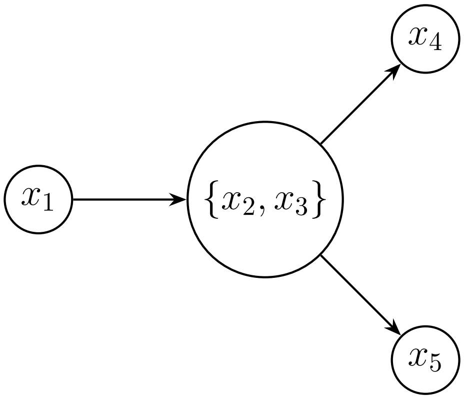
Example 2.2. *Consider the DNF expression* $\Phi = x_1\overline{x_2} \ \vee \ \overline{x_1}x_2 \ \vee \ x_3\overline{x_4} \ \vee \ \overline{x_3}x_4$*. CIG*$(\neg\Phi)$ *has two nodes but no edges. This formula is not convex because* 1100 *is not a minterm of* Φ*, whereas* 1110 *and* 1000 *are.*

Example 2.3. *Consider the DNF expression* $\Phi = x_1\overline{x_2} \ \vee \ x_1\overline{x_4} \ \vee \ x_3\overline{x_4}$*. Then the CIG*$(\neg\Phi)$ *is:*



Since the graph is not a complete bipartite dag*, according to Lemma 2.5,* Φ *is not convex. Note that* $\overline{x_2}x_3$*, which is a separator, is not an implicant.*

Example 2.4. *Consider the DNF expression* $\Phi = x_1\overline{x_2} \vee x_2\overline{x_3} \vee \overline{x_2}x_3 \vee x_2\overline{x_4} \vee x_3\overline{x_5}$*. Then CIG*$(\neg\Phi)$ *is as follows:*



Note that this dag is radial*. The given expression is convex.*

3 Quadratic-Time Algorithm For Case with An All-Positive Term

In this case it can be shown that the formula is convex if and only if it is upward-closed.

Lemma 3.1. *A formula Φ in DNF is upward-closed if and only if it is convex and contains an all-positive term.*

Lemma 3.2. *A formula Φ in DNF is upward-closed if and only if for every term t in Φ, $\Pi(t)$ is an implicant of Φ.*

Proof. This follows from Lemma 1.1: if Φ contains an all-positive term, then from every mixed term t we can get $\Pi(t)$ as a separator. □

A quadratic algorithm can be derived fairly easily since all we need to do is to check for every mixed term $u\overline{v}$ whether u is an implicant of the formula.

4 Conclusion and Future Work

In this paper, we investigated the problem of testing the convexity of DNF formula where every term has exactly two literals, splitting the problem into three main cases. We showed that the problem is easiest when both an all-positive and an all-negative term exist, since in that case convexity reduces to validity. We provided a linear-time algorithm to solve the second case, where every term is mixed, based on a characterization of the implication graph of the negation of the formula. We also showed that the third case, where the formula contains either an all-positive term or an all-negative term (but not both), can be solved in polynomial time. As part of future work we plan to derive another graph-theoretic characterization of the last case.

References

[1] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Inf. Process. Letters*, 8(3):121–123, 1979.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[3] Yves Crama and Peter L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2011.

[4] Amirreza Masoumzadeh, Paliath Narendran, and Padmavathi Iyer. Towards a Theory for Semantics and Expressiveness Analysis of Rule-Based Access Control Models. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, SACMAT '21, pages 33–43. Association for Computing Machinery, 2021.

[5] Josué A. Ruiz, Amirreza Masoumzadeh, Paliath Narendran, and Padmavathi Iyer. Converting Rule-Based Access Control Policies: From Complemented Conditions to Deny Rules. In *Proceedings of the 29th ACM Symposium on Access Control Models and Technologies (to appear)*, 2024. Preprint Available at https://www.cs.albany.edu/~amir/papers/ruiz2024sacmat.pdf.

Undecidability of Static Equivalence in Leaf Permutative Theories

Serdar Erbatur[1], Andrew M. Marshall[2], Paliath Narendran[3], and Christophe Ringeissen[4]

[1] University of Texas at Dallas, Richardson, TX, USA
[2] University of Mary Washington, Fredericksburg, VA, USA
[3] University at Albany, SUNY, Albany, NY, USA
[4] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

1 Introduction

We are interested in knowledge problems that occur in the analysis of security protocols, namely deduction, distinguishability, and static equivalence. In this context, the capabilities of an intruder are specified by an equational theory, possibly expressed by a term rewrite system. For example, a number of decision procedures for these knowledge problems have been developed for the particular case of subterm convergent rewrite systems [1]. Non-orientable axioms are also useful in practice and the prominent case of Associativity-Commutativity (*AC* for short) has been successfully investigated and a number of decision procedures have been developed, for example see [1,4]. The *AC* equational theory is not the only example of non-orientable axioms that deserve to be investigated. The simpler case of shallow permutative axioms, like Commutativity, and decision procedures for the above knowledge problems have been developed in [9]. In this paper, we consider permutative theories in general. For the deduction problem, decidability for the class of permutative theories can be easily proven. The main contribution of the paper is to prove the undecidability of the static equivalence problem in the more restricted class of leaf permutative theories which is a subclass of permutative theories. To retrieve decidability of static equivalence, one can restrict to some particular variable-permuting theories [8].

2 Preliminaries

We use the standard notation of equational unification [6] and term rewriting systems [5].

Notions of Knowledge. The applied pi calculus and frames are used to model attacker knowledge [2]. In this model, the set of messages or terms which the attacker knows, and which could have been obtained from observing one or more protocol sessions, are the set of terms in $Ran(\sigma)$ of the frame $\phi = \nu\tilde{n}.\sigma$, where σ is a substitution ranging over ground terms. We also need to model cryptographic concepts such as nonces, keys, and publicly known values. We do this by using names, which are essentially free constants. Here also, we need to track the names which the attacker knows, such as public values, and the names which the attacker does not know a priori, such as freshly generated nonces. $\tilde{n}$ consists of a finite set of restricted names, these names represent freshly generated names which remain secret from the attacker. The set of names occurring in a term t is denoted by $fn(t)$. For any frame $\phi = \nu\tilde{n}.\sigma$, let $fn(\phi)$ be the set of names $fn(\sigma)\backslash\tilde{n}$ where $fn(\sigma) = \bigcup_{t\in Ran(\sigma)} fn(t)$; and for any term t, let $t\phi$ denote by a slight abuse of notation the term $t\sigma$. We say that a term t *satisfies the name restriction (of ϕ)* if $fn(t) \cap \tilde{n} = \emptyset$.

Let us now define the knowledge problems we consider in this paper.

Definition 1 (Deduction)**.** *Let $\phi = \nu\tilde{n}.\sigma$ be a frame, and t a ground term. We say that t is* deduced from ϕ modulo E*, denoted by $\phi \vdash_E t$, if there exists a term ζ such that $\zeta\sigma =_E t$ and $fn(\zeta) \cap \tilde{n} = \emptyset$. The term ζ is called a* recipe *of t in ϕ modulo E.*

Definition 2 (Static Equivalence)**.** *Two terms s and t are* equal *in a frame $\phi = \nu\tilde{n}.\sigma$ modulo an equational theory E, denoted $(s =_E t)\phi$, if $s\sigma =_E t\sigma$, and $\tilde{n} \cap (fn(s) \cup fn(t)) = \emptyset$. The set of all equalities $s = t$ such that $(s =_E t)\phi$ is denoted by $Eq(\phi)$. Given a set of equalities Eq, the fact that $(s =_E t)\phi$ for all $s = t \in Eq$ is denoted by $\phi \models Eq$. Two frames $\phi = \nu\tilde{n}.\sigma$ and $\psi = \nu\tilde{n}.\tau$ are* statically equivalent modulo E*, denoted as $\phi \approx_E \psi$, if $Dom(\sigma) = Dom(\tau)$, $\phi \models Eq(\psi)$ and $\psi \models Eq(\phi)$.*

Alternatively, one can consider the negation of this problem, finding a term pair that shows two frames are not static equivalent. That is, it distinguishes the two frames.

Definition 3 (Frame Distinguishability)**.** *Given frames $\phi = \nu\tilde{n}.\sigma$ and $\psi = \nu\tilde{n}.\tau$, we say that ϕ is distinguishable from ψ in theory E, denoted $\phi \not\approx_E \psi$, if there exists two terms, t and s (with $\tilde{n} \cap (fn(s) \cup fn(t)) = \emptyset$), such that $t\sigma =_E s\sigma$ and $t\tau \neq_E s\tau$.*

Classes of Permutative Theories. We also need to define the (sub)classes of permutative theories we consider in this paper. This is important not only for properly defining the results proven here but also because there are some previous definitions of leaf permutative theories which don't match the definition given here. For example, the definition of leaf permutative given in this paper differs from the one given in [7] (See Definition 6 below). In [7](Definition 6) the permutation is restricted to just the variables and is only applicable to linear terms. Thus, the definition in [7] would perhaps be better named as a linear variable permutative, while the one given here is just a permutation if the leaf-nodes of the term, see Definition 5.

Definition 4 (Permutative Theory)**.** *An equational theory E is* permutative *if for each axiom $l = r$ in E, l and r contain the same symbols with the same number of occurrences.*

One can easily check that $A = \{f(x, f(y, z)) = f(f(x, y), z)\}$ (Associativity), $C = \{f(x, y) = f(y, x)\}$ (Commutativity) and $AC = \{f(x, f(y, z)) = f(f(x, y), z),\ f(x, y) = f(y, x)\}$ (Associativity-Commutativity) are permutative.

Two important subclasses of permutative theories are given by considering the cases where the permutations only occur on leafs or on variables.

Definition 5 (Leaf Permutative Theory)**.** *An equational theory E is* Leaf permutative *if for each axiom $l = r$ in E, r is a leaf permutation of l, i.e., $r = l\sigma$, where σ is a permutation of the leaf nodes of l (variables and constants of l).*

For example, C is leaf permutative, but A is not.

Definition 6 (Variable-Permuting Theory)**.** *An axiom $l = r$ is said to be* variable-permuting *[12] if all the following conditions are satisfied:*

1. *the set of occurrences of l is identical to the set of occurrences of r,*
2. *for any non-variable occurrence p of l, $l(p) = r(p)$,*
3. *for any $x \in Var(l) \cup Var(r)$, the number of occurrences of x in l is identical to the number of occurrences of x in r.*

Both deduction and distinguishability are known to be decidable in subterm convergent rewrite systems [1]. It has already been shown in [8] that deduction is decidable in permutative theories.

Theorem 1 ([8])**.** *Deduction is decidable in any permutative theory.*

This is due to the fact that you can put a bound on the number of terms you need to consider since permutative theories are non-size reducing. However, when considering distinguishability, the problem becomes more difficult as we show in the next section.

3 Distinguishability is Undecidable for Leaf Permutative Theories

In this section we prove that the frame distinguishability problem, and thus static equivalence, is undecidable for leaf permutative theories. The results proceeds as follows, we first state an undecidable result for Linear Bounded Automata (LBA) which we will use in the reduction. LBA are Turing Machines with a tape that is bounded by the size of the input string (plus two tape end-caps) [10, 11]. Next, we show how to create a leaf permutative TRS from an LBA. Finally, we use the TRS and the frame distinguishability problem to solve the undecidable problem for LBA.

Lemma 1. *Given a deterministic LBA, M, with input alphabet Σ, it's undecidable if there exists a string, $w \in \Sigma^*$, such that M accepts w.*

Proof. Easy reduction from the LBA empty language problem proved undecidable in [3]. There it is shown that it is undecidable if for an arbitrary deterministic LBA M, $L(M) = \emptyset$. □

Lemma 2. *Given a deterministic LBA, M, one can construct a leaf permutative TRS R such that if M accepts a string w then there exists a term t which encodes the initial configuration of M on input w and a term s that encodes the final accepting configuration of M on w such that $t{\downarrow_R} = s$.*

Proof. Here we modify the encoding from [14] to obtain a conversion from an LBA to a leaf permutative TRS.

Let $M = (Q, \Sigma, \Gamma, q_0, q_a, q_r, \delta)$. Assume $\Sigma = \{a_1, a_2, \ldots, a_n\}$ and $\Gamma = \{<, >, \sqcup\} \cup \Sigma$, where $<, >$ are the left and right end caps respectively, and $\sqcup$ is the blank symbol.

We now need to construct the terms that will represent the tape of the LBA. We introduce three new non-constant function symbols, f, g, h and three new constants, a, b, and P. We use each as follows:

- h has arity $|Q| + 1$ and is used to represent the state of the LBA. To represent state q_i, a constant b is placed at the i^{th} position with the remaining positions containing a constants. For example, if $|Q| = 2$ then q_0 is represented as $h(b, a, a)$. The final configuration, with constant b in the final position, is used to represent a non-state or "dummy state", which the use of is described below. We denote this dummy state as q_d. **We use the notation $h(q_i)$ to abbreviate the encoding of the state q_i using h.**

- g has arity $|\Gamma|$ and is used to encode the alphabet characters. We place a constant b at position i in g with constants a placed at all other positions to encode a_i. Positions $n + 1, \ldots, n + 3$ are used to encode $\{<, >, \sqcup\}$ in the same way. **We use the notation $h(a_i)$ to abbreviate the encoding of the character 'a_i' using h.**

- f is used to form terms which consist of an encoding of a state, an encoding of a single character, and an f-rooted subterm or the constant P. The subterm is used to encode the rest of the string. The constant P is used to stop the encoding. For example, the dummy state and the character a_i can be encoded as the term $f(h(q_d), g(a_i), P)$.

We now form terms representing the state of the LBA and its tape as follows:

- For any string $w \in \Gamma^+$ we represent w as a layered f-term, one layer per alphabet character. The last layer is ended using the constant P. The dummy state is used by default for each of the state positions in the f-terms. **We use the notation $f(\overline{w})$ to abbreviate the encoding of the string w using f-terms**.

We now need to construct the leaf permutative TRS. Let's consider the moves of the transition function, δ, and construct from them a TRS R:

- For each right move, $\delta(q_i, a_i) = (q_j, a_j, R)$, we create a rule for each possible character $a_k \in \Gamma$ below, of the form:

$$f(h(q_i), g(a_i), f(h(q_d), g(a_k), x)) \to f(h(q_d), g(a_j), f(h(q_j), g(a_k), x))$$

- For each left move, $\delta(q_i, a_i) = (q_j, a_j, L)$, we create a rule for each possible character $a_k \in \Gamma$ below, of the form:

$$f(h(q_d), g(a_k), f(h(q_i), g(a_i), x)) \to f(h(q_j), g(a_k), f(h(q_d), g(a_j), x))$$

Finally, we need to describe the initial configurations for the LBA. The LBA will start in the configuration $q_o\langle w\rangle$ for input w. We encode this as an f-term, where all the states are initially $h(q_d)$ except the first (leftmost) h. That is, we encode using the term $t = f(h(q_0), g(<), f(\overline{w>}))$.

Notice that each of the rules in R are leaf permutative. In addition, the LBA accepts the string w iff $f(h(q_0), g(<), f(\overline{w>})) \to_R^* s$ such that s is a term that includes just one encoded $h(q_a)$. □

Lemma 3. *Let M be a deterministic LBA. Then the leaf permutative TRS, R, constructed from M is locally confluent. Furthermore, if M is both deterministic and terminating then R is convergent.*

Next, it's easy to show that from an LBA M_1 one we can construct the following LBA.

Lemma 4. *Let M_1 be a deterministic LBA that before accepting and halting it replaces the tape (except the end caps) with blank symbols and stops (enters the accept or reject state) with the tape head over the left end-cap. One can construct an LBA M_2 from M_1 such that $L(M_2) = \emptyset$ and every transition M_1 and M_2 are the same except the accepting transitions from M_1 are now rejecting in M_2.*

Notice that for each LBA M_1 and M_2 there is a corresponding leaf permutative TRS, R_1 and R_2 respectively. Next we can easily combine these two TRSs into a single TRS.

Lemma 5. *Given deterministic LBAs M_1 and M_2 and their leaf permutative TRSs R_1 and R_2 respectively. Let q_{0_i} be the initial state for M_i. One can construct a leaf permutative TRS $R_{1,2}$ such that for input string w and term $t_i = f(h(q_{0_i}), g(<), f(\overline{w>}))$, that $t_i \to_{R_{1,2}}^* s_i$ iff $t_i \to_{R_i}^* s_i$, $i \in \{1, 2\}$.*

Proof. One just needs to ensure that $\{Q_1 \setminus \{q_a, q_r\}\} \cap \{Q_2 \setminus \{q_a, q_r\}\} = \emptyset$. Then, the rules of R_1 and R_2 are disjoint. Starting a configuration with a start state in Q_i will ensure each of the following configurations, except the final with shared states q_a or q_r, are disjoint. □

Theorem 2. *Frame distinguishability is undecidable in general when E is a leaf permutative theory.*

Proof. We can proceed by reduction using Lemma 1. Assume we are given a deterministic LBA M_1. Without loss of generality assume that before halting and entering q_a or q_r that M_1 erases its tape and halts with the head over the left end-cap. Furthermore, assume that as an initial step M_1 scans the tape from left to right end-cap and then back to the left. Since these steps could be added to any LBA without changing its language, they don't represent a restriction. We proceed as follows.

1. From M_1 construct the always rejecting LBA M_2 as in Lemma 4. We can assume w.l.g., that $\{Q_1 \setminus \{q_a, q_r\}\} \cap \{Q_2 \setminus \{q_a, q_r\}\} = \emptyset$. This can be done by simply creating a marked version of $Q_1 \setminus \{q_a, q_r\}$ and using that for the set of states for M_2, i.e., Q_2. Let q_{0_1} be the start state for M_1 and q_{0_2} for M_2.

2. From M_1 and M_2 construct R_1 and R_2 respectively as in Lemma 2.

3. Construct $R_{1,2}$ from R_1 and R_2 as in Lemma 5.

4. Construct two frames, such that $\tilde{n} = \emptyset$ for each frame:

   a) $\phi_1 = \nu\tilde{n}.\sigma_1 = \{x \mapsto q_{0_1}, y \mapsto q_a\}$ b) $\phi_2 = \nu\tilde{n}.\sigma_2 = \{x \mapsto q_{0_2}, y \mapsto q_a\}$

   Assume that we have an algorithm for the frame distinguishability problem and it returns a term pair (t, s). Then, $t\sigma_1 \rightarrow^*_{R_{1,2}} s\sigma_1$ and $t\sigma_2 \not\rightarrow^*_{R_{1,2}} s\sigma_2$. Notice, it can't be that $s = t$ nor can t be ground, otherwise $t\sigma_2 \rightarrow^*_{R_{1,2}} s\sigma_2$. Thus, $t\sigma_1 \rightarrow^+_{R_1} s\sigma_1$. We now need to show the following:

   (a) $t\sigma_i$ is a well formed term encoding an initial configuration of an LBA M_i for some initial input string w.

   - Notice that the rules of $R_{1,2}$ can only be applied to well-formed subterms of t. Since M_1 (and thus M_2) first scan the tape from left to right the rewrite derivation on $t\sigma_1$ would stop when any malformed subterm was reached and before the final configuration. However, since the steps of M_2, except the step entering the final configuration, are the same as those for M_1, M_2 would stop at the same point in the derivation from $t\sigma_2$. That is, $t\sigma_2 \rightarrow^*_{R_{1,2}} s\sigma_2$.
   - We can assume the initial rewrite step in the derivations $t\sigma_i \rightarrow^*_{R_{1,2}} s\sigma_i$, occur at ϵ since any part of the term above the subterm $f(h(x), g(<), f(\overline{w >}))$ will not change and thus can be discarded.

   (b) $s\sigma_1$ is a final configuration of an LBA M_1 and represents an accepting configuration.

   - $s\sigma_1$ must be a final *accepting* configuration, otherwise $t\sigma_2 \rightarrow^*_{R_{1,2}} s\sigma_2$. Recall that all the transitions of M_2, except the final ones entering a final configuration, are the same. Thus, starting from a well-formed initial configuration, the only way to have $t\sigma_1 \rightarrow^*_{R_{1,2}} s\sigma_1$ and $t\sigma_2 \not\rightarrow^*_{R_{1,2}} s\sigma_2$ is for $s\sigma_1$ to be a final accepting configuration.

Therefore, a frame distinguishablity algorithm would allow us to decide if for an arbitrary LBA M_1 if there exist some string, w, accepted by M_1, violating Lemma 1. □

If the frame distinguishability problem is undecidable then we also get undecidability of the static equivalence problem.

Corollary 1. *Static equivalence modulo E is undecidable in general when E is a leaf permutative theory.*

4 Conclusion

In the context of security protocols, it is crucial to identify classes of theories with decidable knowledge problems. When a theory is given by a subterm convergent term rewrite system, these knowledge problems are known to be decidable thanks to the seminal work initiated in [1]. Furthermore, decidability of the knowledge problems for shallow permutative theories has been shown in [9]. Then, it is possible to stay on the decidability side when considering a subterm rewrite system which is convergent modulo a shallow permutative theory [9]. In [8] we constructed a restricted subclass of variable-permuting theories, called *SVP* theories, where static equivalence is decidable. These theories restrict the roots of the left and right-hand sides of the rules such that decidability is achieved.

We plan to continue the study of the knowledge problems in equational theories given by rewrite systems modulo permutative axioms. On the one hand, it is possible to consider extensions of subterm rewrite systems, such as contracting rewrite systems [8,13]. On the other hand, we can now envision to go beyond shallow permutative axioms. Due to the undecidability result reported here, it is clear now that we cannot consider any arbitrary set of permutative axioms.

References

[1] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.*, 367(1-2):2–32, 2006.

[2] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 104–115. ACM, 2001.

[3] Zümrüt Akçam, Kimberly A. Cornell, Daniel S. Hono II, Paliath Narendran, and Andrew Pulver. On problems dual to unification: The string-rewriting case, 2023. Available at `https://doi.org/10.48550/arXiv.2103.00386`.

[4] Mauricio Ayala-Rincón, Maribel Fernández, and Daniele Nantes-Sobrinho. Intruder deduction problem for locally stable theories with normal forms and inverses. *Theoretical Computer Science*, 672:64–100, 2017.

[5] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998.

[6] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

[7] Thierry Boy De La Tour, Mnacho Echenim, and Paliath Narendran. Unification and matching modulo leaf-permutative equational presentations. In *International Joint Conference on Automated Reasoning*, pages 332–347. Springer, 2008.

[8] Carter Bunch, Saraid Dwyer Satterfield, Serdar Erbatur, Andrew M. Marshall, and Christophe Ringeissen. Knowledge problems in protocol analysis: Extending the notion of subterm convergent, 2024. Available at https://doi.org/10.48550/arXiv.2401.17226.

[9] Serdar Erbatur, Andrew M. Marshall, and Christophe Ringeissen. Computing knowledge in equational extensions of subterm convergent theories. *Math. Struct. Comput. Sci.*, 30(6):683–709, 2020.

[10] John E Hopcroft and Jeffrey D Ullman. Some results on tape-bounded Turing machines. *Journal of the ACM*, 16(1):168–177, 1969.

[11] Sige-Yuki Kuroda. Classes of languages and linear-bounded automata. *Information and control*, 7(2):207–223, 1964.

[12] Paliath Narendran and Friedrich Otto. Single versus simultaneous equational unification and equational unification for variable-permuting theories. *J. Autom. Reason.*, 19(1):87–115, 1997.

[13] Saraid Dwyer Satterfield, Serdar Erbatur, Andrew M. Marshall, and Christophe Ringeissen. Knowledge problems in security protocols: Going beyond subterm convergent theories. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 30:1–30:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[14] Manfred Schmidt-Schauß. Unification in permutative equational theories is undecidable. *J. Symb. Comput.*, 8(4):415–421, 1989.

On Anti-Unification over Absorption, Associative, and Commutative Theories

Andrés Felipe González Barragán[1], David M. Cerna[2], Mauricio Ayala-Rincón[1], and Temur Kutsia[3]

[1] University of Brasilia, Brasilia D.F., Brazil
andres.felipe@aluno.unb.br, ayala@unb.br
[2] Czech Academy of Sciences, Prague, Czech Republic
dcerna@cs.cas.cz
[3] RISC, Johannes Kepler University, Linz, Austria
kutsia@risc.jku.at

Abstract

Algebras as monoids, semi-groups, and Abelian semi-groups, including absorption operators with their relative absorption constants, are also equipped with commutative (C) and associative (A) properties such as the product operator with the constant zero: $x * 0 \approx 0 * x \approx 0$. We present a sound algorithm and some examples of the anti-unification problem for absorption ($\mathfrak{a}$) theories, including A or C operators.

1 Introduction

Anti-unification (AU) or generalization is a crucial method of reasoning. The problem of AU consists of finding commonalities between two expressions. algorithms aiming to solve this problem find a set of terms that minimally express all possible similarities between input expressions. The problem was introduced by Plotkin and Reynolds, addressing the (syntactic) first-order languages [7, 8]. AU has been studied in several equational theories, such as theories with associative (A) and commutative (C) operators [1], unital [5], and absorption ($\mathfrak{a}$) theories [3]. Moreover, one of the relatively unexplored areas is the investigation of combinations between these theories, as highlighted in related works in [2] and [4]. This abstract discusses the combinations of absorption theories with associative or commutative operators. For a survey on anti-unification, see [6]. In a recent paper [3], the authors presented a sound and complete algorithm that solves anti-unification modulo absorption theories, theories with operators that satisfy the axioms $\{f(\varepsilon_f, x) \approx \varepsilon_f, f(x, \varepsilon_f) \approx \varepsilon_f\}$. This work aims to present recent advancements, introducing two distinct extensions of the anti-unification problem modulo absorption. We consider absorption symbols together with associative and commutative symbols, treated separately in the same set of axioms. The inclusion of this kind of symbols raises new generalizations that were not considered before either in $\mathfrak{a}$- or C- or A-theories, then we assemble the existing algorithms in [3, 1] and introduce new rules to handle these generalizations. It is important to highlight that here in this new approach, the role of $\star$ in the expansions of the absorption constants within commutative or associative properties could lead us to new generalizations that need to be captured for the algorithm. This algorithm is terminating, sound, and capable of capturing generalizations for this kind of combination.

1.1 Preliminaries

Let $\mathcal{V}$ be a countable set of variables and $\mathcal{F}$ a set of function symbols with a fixed arity. Additionally, we assume $\mathcal{F}$ to contain a special constant $\star$, referred to as the *wild card*. The set of terms derived from $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F},\mathcal{V})$, whose members are constructed using the grammar $t ::= x \mid f(t_1,\ldots,t_n)$, where $x \in \mathcal{V}$ and $f \in \mathcal{F}$ with arity $n \geq 0$. When $n = 0$, f is called a *constant*. The set of variables occurring in t is denoted by $var(t)$. The *size* of a term is defined inductively as: $size(x) = 1$, and $size(f(t_1,\ldots,t_n)) = 1 + \sum_{i=1}^{n} size(t_i)$. Let σ be a substitution, $dom(\sigma)$, and $rvar(\sigma)$ denote the domain and the set of variables occurring in terms of the range of σ, respectively. The *head* of a term t is defined as $head(x) = x$ and $head(f(t_1,\ldots,t_n)) = f$, for $n \geq 0$.

The focus of this work is anti-unification modulo equational theories E that may include commutative symbols, for short C-symbols, with axioms for commutativity, $\{f(x,y) = f(y,x)\}$, associative symbols (A-symbols), with axioms for Associativity, $\{f(f(x,y),z) = f(x,f(y,z))\}$, and absorption symbols, for short $\mathfrak{a}$-symbols, with absorption axioms, $\{f(x,\varepsilon_f) \approx \varepsilon_f,\ f(\varepsilon_f,x) \approx \varepsilon_f\}$. Symbols f and ε_f are called *related* $\mathfrak{a}$*-symbols*. An $(E)(E')$-theory includes E-symbols and E'-symbols and an EE'-theory includes symbols holding E- and E'-axioms simultaneously.

Definition 1 (E-generalization, $\preceq_E$). *The generalization relation of the theory induced by E holds for terms $r, s \in \mathcal{T}(\mathcal{F},\mathcal{V})$, written $r \preceq_E s$, if there exists a substitution σ such that $r\sigma \approx_E s$. An E-generalization r of s and t is a term r such that $r \preceq_E s$ and $r \preceq_E t$.*

Example 1. *Consider $\mathfrak{a}C = \{f(\varepsilon_f,x) \approx \varepsilon_f, f(x,y) \approx f(y,x)\}$. Then $f(x,a)$ is an $\mathfrak{a}C$-generalization of ε_f and $f(a,a)$: $f(x,a)\{x \mapsto \varepsilon_f\} \approx_{\mathfrak{a}C} \varepsilon_f$ and $f(x,a)\{x \mapsto a\} \approx_{\mathfrak{a}C} f(a,a)$.*

Definition 2 (Minimal complete set of E-generalizations). *The* minimal complete set of E-generalizations *of the terms s and t, denoted as $mcsg_E(s,t)$, is a subset of $\mathcal{G}_E(s,t)$, the set of all E-generalizations of s and t, satisfying: (i) for each $r \in \mathcal{G}_E(s,t)$ there exists $r' \in mcsg_E(s,t)$ such that $r \preceq_E r'$; (ii) if $r, r' \in mcsg_E(s,t)$ and $r \preceq_E r'$, then $r = r'$.*

Example 2. *Continuing Example 1, notice that $mcsg_{\mathfrak{a}C}(\varepsilon_f, f(a,a)) = \{f(x,a), f(x,x)\}$, and $mcsg_{\mathfrak{a}}(\varepsilon_f, f(a,a)) = \{f(x,a), f(a,x), f(x,x)\}$.*

Definition 3 (Anti-unification type). *The anti-unification type of an equational theory E is said to be* unitary *if $mcsg_E(s,t)$ is a singleton for all terms s and t; it is* finitary *if it is not unitary but $mcsg_E(s,t)$ is always finite; it is* infinitary *if it is neither unitary nor finitary but $mcsg_E(s,t)$ always exists; otherwise, it is said to be* nullary.

Syntactic AU is *unitary* [7, 8], AU over (A) and (C) theories is *finitary* [1], AU over ($\mathfrak{a}$) theories is *infinitary* [4], and AU with a disjoint combination of unital equations is *nullary* [5].

An *anti-unification triple* (AUT), $s \triangleq_x t$, consists of a *label* $x \in \mathcal{V}$, and two terms s and t. Given a set A of AUTs, $labels(A) = \{x \mid s \triangleq_x t \in A\}$ and $size(A) = \sum_{s \triangleq_x t \in A} \big(size(s) + size(t)\big)$. A set of AUTs is *valid* if its labels are pairwise disjoint. A *wild* AUT is of the form either $\star \triangleq_x s$ or $s \triangleq_x \star$. A non-wild AUT $s \triangleq_x t$ is *solved* over an absorption theory $\mathfrak{a}$ if $head(s)$ and $head(t)$ are different and they are not related $\mathfrak{a}$-symbols.

The label x in an AUT $s \triangleq_x t$, as a variable, is a most general generalization of the terms s and t, and it is used to associate the generalizations of s and t. The wild card plays an important role when anti-unification problems are decomposed, and related $\mathfrak{a}$-symbols appear in the head of AUTs; they will represent any possible term expanding and $\mathfrak{a}$-constant symbol needs to be expanded ($\varepsilon_f \approx_{\mathfrak{a}} f(\varepsilon_f,\star)$ or $\varepsilon_f \approx_{\mathfrak{a}} f(\star,\varepsilon_f)$), see [3].

2 Anti-Unification in Absorption Theories with Commutative or Associative Properties

Several algebras having absorption property like semi-groups, Abelian semi-groups, and monoids may include the associative or/and commutative property. Interesting examples of these algebras are the integers with multiplication with zero as absorption constant; the integers with the greatest common divisor gcd with one as the absorption constant; the $n \times n$-matrices over reals with the product and the zero matrix; the powerset of a given set with the intersection $\cap$ with $\varnothing$ as absorption constant; Boolean algebras with two binary operations, where each operation is associative, commutative, and has zero element. This section shows how generalizations for $\mathfrak{a}$ theories with A, C symbols differ from generalizations of pure $\mathfrak{a}$ theories presented in [3].

Example 3. *The set $mcsg_{\mathfrak{a}}(\varepsilon_f, f(a,a)) = \{f(x,a), f(a,x), f(x,x)\}$, which is different from the set $mcsg_{\mathfrak{a}C}(\varepsilon_f, f(a,a))$ (see Example 2), is computed by the algorithm in [3]. Also, for the more elaborated example, $mcsg_{\mathfrak{a}C}(\varepsilon_f, f(f(a,a), f(a,a)))$ does not include $\mathfrak{a}$ minimal generalizations as $f(f(a,a), f(u,a))$ and $f(f(a,u), f(a,u))$ in $mcsg_{\mathfrak{a}}(\varepsilon_f, f(f(a,a), f(a,a)))$.*

An algorithm to compute generalizations in ($\mathfrak{a}$)($\mathfrak{a}$C)(C)-theories should include rules to treat C symbols as in [1], and also adaptations of the expansion and merge rules introduced in [3] for $\mathfrak{a}$ theories, to deal with $\mathfrak{a}$C symbols.

Example 4. *The set $mcsg_{\mathfrak{a}A}(g(\varepsilon_f, a), g(f(f(a,a), f(a,a)), f(a, f(a,a))))$ includes the $\mathfrak{a}$A-generalization $g(f(x,y), y)$, where g is a syntactic symbol and f is an $\mathfrak{a}$A-symbol. Notice that this is not an $\mathfrak{a}$-generalization.*

In the case of ($\mathfrak{a}$)($\mathfrak{a}$A)(A)-theories, standard flattened notation is used, and for an A-symbol, f, the flattened term $f(t)$ equals t. An algorithm to compute the generalizations requires designing specialized rules, adapted from [3], to deal with $\mathfrak{a}$A symbols.

3 Algorithm for Absorption with Commutative or Associative Theories

Tables 1, 2, and 3 present inference rules for theories with $\mathfrak{a}$-, $\mathfrak{a}$C-, $\mathfrak{a}$A-, A-, and C-symbols. The algorithm AUnif consists of applying these rules exhaustively, returning a set of objects from which generalizations of the input AUTs may be derived. The inference rules work on *configurations*, defined below.

Definition 4 (Configuration)**.** *A configuration is a quadruple of the form $\langle A; S; D; \theta\rangle$, where, A is a valid set of AUTs (*active set*); S is a valid set of solved AUTs (*store*); D is a valid set of* wild *AUTs (*delayed set*); θ is a* substitution *such that $rvar(\theta) = labels(A) \cup labels(S) \cup labels(D)$ (*anti-unifier*); and with the property that $labels(A), labels(S), labels(D)$, and $dom(\theta)$ are pairwise disjoint.*

All terms occurring in a configuration are in their $\mathfrak{a}$-normal forms. For $\mathfrak{a}$A- and A-symbols, all terms in a configuration are considered in the flattened form.

Table 1 contains rules: Decompose ($\xrightarrow{Dec}$), Solve ($\xrightarrow{Sol}$), Expansions for Left Absorption, ($\xrightarrow{ExpL1}$ and $\xrightarrow{ExpL2}$), Expansions for Right Absorption ($\xrightarrow{ExpR1}$ and $\xrightarrow{ExpR2}$), and Expansion Absorption in Both sides ($\xrightarrow{ExpB1}$ and $\xrightarrow{ExpB2}$), representing the common rules. Table 2 has the extra rules

Commutative ($\overset{Com}{\Longrightarrow}$), and Table 3 shows the additional rules Associativity Left and Right ($\overset{AL}{\Longrightarrow}$) and ($\overset{AR}{\Longrightarrow}$), and Absorption-Associative Left and Right 1,2 ($\overset{\mathfrak{a}\mathrm{AL1}}{\Longrightarrow}$), ($\overset{\mathfrak{a}\mathrm{AL2}}{\Longrightarrow}$), ($\overset{\mathfrak{a}\mathrm{AR1}}{\Longrightarrow}$), and ($\overset{\mathfrak{a}\mathrm{AR2}}{\Longrightarrow}$).

By $\mathcal{C} \Longrightarrow^* \mathcal{C}'$ we denote a finite sequence of inference rule applications starting at $\mathcal{C}$ and ending with $\mathcal{C}'$. In both cases we say $\mathcal{C}'$ is *derived* from $\mathcal{C}$. An initial configuration is a configuration of the form $\langle A; \varnothing; \varnothing; \iota\rangle$, where $\iota = \{\mathsf{f}_A(x) \mapsto x \mid x \in \mathit{labels}(A)\}$ with $\mathsf{f}_A : \mathcal{V} \to (\mathcal{V} \setminus \mathit{labels}(A))$ being a bijection over variables. A configuration $\mathcal{C}$ is referred to as *final* if no inference rule applies to $\mathcal{C}$. We denote the set of final configurations finitely derived from an initial configuration $\mathcal{C}$ by AUnif($\mathcal{C}$).

Example 5. *Notice that* AUnif *computes the generalization $f(x,a)$ for the problem in Example 3 using the rules (ExpL1) and (Sol); and the generalization $g(f(x,z),y)$ for the problem in Example 4 using the rules (Dec),($\mathfrak{a}AL1$) for $k = 1$, and (Sol).*

Table 1: Inference rules common to all theories.

$$(\overset{\mathit{Dec}}{\Longrightarrow}) \quad \frac{\langle\{f(s_1,\ldots,s_n) \triangleq_x f(t_1,\ldots,t_n)\} \uplus A; S; D; \theta\rangle}{\langle\{s_1 \triangleq_{y_1} t_1, \ldots, s_n \triangleq_{y_n} t_n\} \cup A; S; D; \theta\{x \mapsto f(y_1,\ldots,y_n)\}\rangle}$$

where f is any symbol, $n \geq 0$, and $y_1,\ldots,y_n$ are fresh variables.

$$(\overset{\mathit{Sol}}{\Longrightarrow}) \quad \frac{\langle\{s \triangleq_x t\} \uplus A; S; D; \theta\rangle}{\langle A; \{s \triangleq_x t\} \cup S; D; \theta\rangle}$$

where $\mathit{head}(s) \neq \mathit{head}(t)$ and they are not related $\mathfrak{a}$-symbols.

$$(\overset{\mathit{Mer}}{\Longrightarrow}) \quad \frac{\langle\varnothing; \{s_1 \triangleq_x t_1, s_2 \triangleq_y t_2\} \cup S; T; \theta\rangle}{\langle\varnothing; \{s_2 \triangleq_y t_2\} \cup S; T; \theta\{x \mapsto y\}\rangle}$$

where $s_1 \approx_E s_2$, $t_1 \approx_E t_2$, $x \neq y$, and E is an equational theory.

In the following rules, f is an $\mathfrak{a}$-, $\mathfrak{a}$C-, or $\mathfrak{a}$A-symbol, and y_1, y_2 are fresh variables:

$$(\overset{\mathit{ExpL1}}{\Longrightarrow}) \quad \frac{\langle\{\varepsilon_f \triangleq_x f(t_1,t_2)\} \uplus A; S; D; \theta\rangle}{\langle\{\varepsilon_f \triangleq_{y_1} t_1\} \cup A; S; \{\star \triangleq_{y_2} t_2\} \cup D; \theta\{x \mapsto f(y_1,y_2)\}\rangle}$$

$$(\overset{\mathit{ExpL2}}{\Longrightarrow}) \quad \frac{\langle\{\varepsilon_f \triangleq_x f(t_1,t_2)\} \uplus A; S; D; \theta\rangle}{\langle\{\varepsilon_f \triangleq_{y_2} t_2\} \cup A; S; \{\star \triangleq_{y_1} t_1\} \cup D; \theta\{x \mapsto f(y_1,y_2)\}\rangle}$$

$$(\overset{\mathit{ExpR1}}{\Longrightarrow}) \quad \frac{\langle\{f(s_1,s_2) \triangleq_x \varepsilon_f\} \uplus A; S; D; \theta\rangle}{\langle\{s_1 \triangleq_{y_1} \varepsilon_f\} \cup A; S; \{s_2 \triangleq_{y_2} \star\} \cup D; \theta\{x \mapsto f(y_1,y_2)\}\rangle}$$

$$(\overset{\mathit{ExpR2}}{\Longrightarrow}) \quad \frac{\langle\{f(s_1,s_2) \triangleq_x \varepsilon_f\} \uplus A; S; D; \theta\rangle}{\langle\{s_2 \triangleq_{y_2} \varepsilon_f\} \cup A; S; \{s_1 \triangleq_{y_1} \star\} \cup D; \theta\{x \mapsto f(y_1,y_2)\}\rangle}$$

$$(\overset{\mathit{ExpB1}}{\Longrightarrow}) \quad \frac{\langle\{\varepsilon_f \triangleq_x \varepsilon_f\} \uplus A; S; D; \theta\rangle}{\langle A; S; \{\varepsilon_f \triangleq_{y_1} \star,\ \star \triangleq_{y_2} \varepsilon_f\} \cup D; \theta\{x \mapsto f(y_1,y_2)\}\rangle}$$

$$(\overset{\mathit{ExpB2}}{\Longrightarrow}) \quad \frac{\langle\{\varepsilon_f \triangleq_x \varepsilon_f\} \uplus A; S; D; \theta\rangle}{\langle A; S; \{\star \triangleq_{y_1} \varepsilon_f,\ \varepsilon_f \triangleq_{y_2} \star\} \cup D; \theta\{x \mapsto f(y_1,y_2))\}\rangle}$$

Lemma 1 (Configuration Preservation). *Let $\mathcal{C}$ be a configuration and $\mathcal{C} \Longrightarrow^* \mathcal{C}'$. Then $\mathcal{C}'$ is a configuration.*

Proof. According to the rules in Tables 1,2 and 3 we can have the following two cases:

Table 2: Inference rule for 𝔞C- and C-symbols.

$$(\xrightarrow{Com}) \quad \frac{\langle \{f(s_1, s_2) \triangleq_x f(t_1, t_2)\} \dot{\cup} A; S; T; \theta\rangle}{\langle \{s_1 \triangleq_{y_1} t_2, s_2 \triangleq_{y_2} t_1\} \cup A; S; T; \theta\{x \mapsto f(y_1, y_2)\}\rangle}$$

for f an 𝔞C- or C-symbol and y_1, y_2 fresh variables.

Table 3: Inference rule for 𝔞A and A symbols.

In the next two rules, g is either an 𝔞A-symbol or an A-symbol:

$$(\xrightarrow{AL}) \quad \frac{\langle \{g(s_1, \dots, s_n) \triangleq_x g(t_1, \dots, t_m)\} \dot{\cup} A; S; T; \theta\rangle}{\langle \{s_1 \triangleq_{y_1} g(t_1, \dots, t_k), g(s_2, \dots, s_n) \triangleq_{y_2} g(t_{k+1}, \dots, t_m)\} \cup A; S; T; \theta\{x \mapsto g(y_1, y_2)\}\rangle}$$

for $1 \le k \le m - 1$ and y_1, y_2 are fresh variables.

$$(\xrightarrow{AR}) \quad \frac{\langle \{\{\{g(s_1, \dots, s_n) \triangleq_x g(t_1, \dots, t_m)\} \dot{\cup} A; S; T; \theta\rangle}{\langle \{g(s_1, \dots, s_k) \triangleq_{y_1} t_1, g(s_{k+1}, \dots, s_n) \triangleq_{y_2} g(t_2, \dots, t_m)\} \cup A; S; T; \theta\{x \mapsto g(y_1, y_2)\}\rangle}$$

for $1 \le k \le n - 1$ and y_1, y_2 are fresh variables.

Next five rules apply to 𝔞A-symbols, and $1 \le k \le n - 1$, and y_1, y_2 are fresh variables:

$$(\xrightarrow{\mathfrak{a}\mathbf{AL1}}) \quad \frac{\langle \{\varepsilon_f \triangleq_x f(t_1, \dots, t_n)\} \dot{\cup} A; S; T; \theta\rangle}{\langle \{\varepsilon_f \triangleq_{y_1} f(t_1, \dots, t_k)\} \cup A; S; \{\varepsilon_f \triangleq_{y_2} f(t_{k+1}, \dots, t_n)\} \cup T; \theta\{x \mapsto f(y_1, y_2)\}\rangle}$$

$$(\xrightarrow{\mathfrak{a}\mathbf{AL2}}) \quad \frac{\langle \{\varepsilon_f \triangleq_x f(t_1, \dots, t_n)\} \dot{\cup} A; S; T; \theta\rangle}{\langle \{\varepsilon_f \triangleq_{y_2} f(t_{k+1}, \dots, t_n)\} \cup A; S; \{\varepsilon_f \triangleq_{y_1} f(t_1, \dots, t_k)\} \cup T; \theta\{x \mapsto f(y_1, y_2)\}\rangle}$$

$$(\xrightarrow{\mathfrak{a}\mathbf{AR1}}) \quad \frac{\langle \{f(s_1, \dots, s_n) \triangleq_x \varepsilon_f\} \dot{\cup} A; S; T; \theta\rangle}{\langle \{f(t_1, \dots, t_k) \triangleq_{y_1} \varepsilon_f\} \cup A; S; \{f(t_{k+1}, \dots, t_n) \triangleq_{y_2} \varepsilon_f\} \cup T; \theta\{x \mapsto f(y_1, y_2)\}\rangle}$$

$$(\xrightarrow{\mathfrak{a}\mathbf{AR2}}) \quad \frac{\langle \{f(s_1, \dots, s_n) \triangleq_x \varepsilon_f\} \dot{\cup} A; S; T; \theta\rangle}{\langle \{f(t_{k+1}, \dots, t_n) \triangleq_{y_2} \varepsilon_f\} \cup A; S; \{f(t_1, \dots, t_k) \triangleq_{y_1} \varepsilon_f\} \cup T; \theta\{x \mapsto f(y_1, y_2)\}\rangle}$$

- A rule removes an AUT $s \triangleq_x t$ from the active set of $\mathcal{C}$. Then either $s \triangleq_x t$ occurs in the store of $\mathcal{C}'$, or the anti-unifier component of $\mathcal{C}'$ is the composition of the anti-unifier component of $\mathcal{C}$ with $\{x \mapsto r\}$, where $var(r)$ are fresh variables labeling newly added AUTs in the active and delayed sets of $\mathcal{C}'$.

- A rule removes an AUT $s \triangleq_x t$ from the store of $\mathcal{C}$. Then the store of $\mathcal{C}'$ is a subset of the store of $\mathcal{C}$ and the anti-unifier component of $\mathcal{C}'$ is the composition of the anti-unifier component of $\mathcal{C}$ with $\{x \mapsto y\}$, where y is a label of an AUT in the store of $\mathcal{C}$ such that $x \neq y$.

In both cases, the properties of a configuration are preserved. □

Theorem 1 (Termination)**.** *Let $\mathcal{C}$ be a configuration. Then* AUnif$(\mathcal{C})$ *is finitely computable.*

Proof. The termination of AUnif is proved using a lexicographical measure over configurations. The measure for $\mathcal{C} = \langle A; S; T; \theta\rangle$ is given by $(size(A), size(S))$. All rules except (Mer) decrease the first component, and (Mer) maintains the first but decreases the second component. □

Termination (Theorem 1) guarantees that always is possible to obtain a final configuration. Configurations of the form $\langle \varnothing; S; D; \theta\rangle$ where S has no duplicated AUTs, except for the label, are final configurations since no rule can be applied. Since all rules, except for (Mer), decrease the size of the active set, it becomes empty, and the rule (Mer) will eliminate all such possible duplication in the store. Configuration preservation (Lemma 1) and termination allow proving the soundness of AUnif for the combination of the $\mathfrak{a}$-, $\mathfrak{a}$C-, $\mathfrak{a}$A-, C-, and A-theories.

Theorem 2 (Soundness). *Let* $\langle\varnothing; S_n; D_n; \theta_n\rangle \in \mathrm{AUnif}(\langle A_0; S_0; D_0; \theta_0\rangle)$*, and* E *be any combination of the theories* $\mathfrak{a}$*,* $\mathfrak{a}C$*,* $\mathfrak{a}A$*,* C*, and* A*. Then, for all* $s \triangleq_x t \in A_0 \cup S_0$*,* $x\theta_n \in \mathcal{G}_E(s,t)$*.*

Proof. The proof is by induction on the length of derivations, analyzing each rule application. □

4 Work in progress

Work in progress addresses adaptation of AUnif to allow combinations in which (AC)- and ($\mathfrak{a}$AC)-symbols are also allowed. Of course, it also aims to prove completeness. For theories with C-, $\mathfrak{a}$-, and $\mathfrak{a}$C-symbols, currently under study, the proof requires induction on the occurrence of variables in the possible generalizations interrelated with structural analysis of the AUTs under the action of AUnif. The analysis is much more elaborate than the applied on the proof of completeness for $\mathfrak{a}$-theories in [3]. Additionally, succeeding in the completeness proof will imply that the anti-unification problem is infinitary.

Acknowledgments. The research is supported by the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES), under the PrInt program. It is partially supported by the Austrian Science Fund (FWF), under project P 35530, the Georgian Rustaveli National Science Foundation Project FR-21-16725, and by the Czech Science Foundation Grant No. 22-06414L, and the Cost Action CA20111 EuroProofNet.

References

[1] María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014.

[2] María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. Order-sorted equational generalization algorithm revisited. *Ann. Math. Artif. Intell.*, 90(5):499–522, 2022.

[3] Mauricio Ayala-Rincón, David M. Cerna, Andrés Felipe González Barragán, and Temur Kutsia. Equational anti-unification over absorption theories. In *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, Proceedings*, volume 14740 of *Lecture Notes in Computer Science*. Springer, 2024.

[4] David M. Cerna. Anti-unification and the theory of semirings. *Theor. Comput. Sci.*, 848:133–139, 2020.

[5] David M. Cerna and Temur Kutsia. Unital anti-unification: Type and algorithms. In *5th Int. Conference on Formal Structures for Computation and Deduction, FSCD*, volume 167 of *LIPIcs*, pages 26:1–26:20, 2020.

[6] David M. Cerna and Temur Kutsia. Anti-unification and generalization: A survey. In *Proceedings of the 32nd Int. Joint Conference on Artificial Intelligence, IJCAI*, pages 6563–6573, 2023.

[7] Gordon D. Plotkin. A note on inductive generalization. *Machine Intell.*, 5(1):153–163, 1970.

[8] John C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. *Machine Intell.*, 5(1):135–151, 1970.

Computing Generalizers over Intersection and Union Type Theories

Gabriela Ferreira[1], David M. Cerna[2], Mauricio Ayala-Rincón[1], and Temur Kutsia[3]

[1] University of Brasilia (UnB), Brasília D.F., Brazil
222114451@aluno.unb.br, ayala@unb.br

[2] Czech Academy of Sciences Institute of Computer Science (CAS ICS), Prague, Czech Republic
dcerna@cs.cas.cz

[3] Research Institute for Symbolic Computation,
Johannes Kepler University Linz, Austria
kutsia@risc.jku.at

Abstract

We discuss how to extend anti-unification to generalize higher-order terms with different types by extending the type system used within an existing framework with union and intersection types. We provide examples illustrating desirable properties of the corresponding least general generalizers.

Keywords: Anti-Unification, higher-order terms, intersection and union types.

1 Introduction

Anti-unification was introduced and studied by Plotkin [18] and Reynolds [19] in the 1970s. It requires identifying similarities between two symbolic expressions and retaining them as a new symbolic expression, called a generalizer of the given ones. At the same time, the differences between the given expressions are also reflected in their generalization in the form of new variables. This new symbolic expression is referred to as *least general* (or most particular) when it maximally captures the structure of the input expressions and abstracts the differences by new variables uniformly. For instance, a least general generalizer (lgg) of two first-order terms $f(a, g(a))$ and $f(b, g(b))$ is $f(x, g(x))$. In the first-order syntactic case, the least general generalizer (lgg) is unique. But there are theories and problems for which there exist more than one (even infinitely many) lggs, or lggs do not exist at all.

In recent years, research on anti-unification has intensified, mainly due to its various applications. Questions about anti-unification have been studied in different syntactic and semantic frameworks; for example, Baader [4] studied it for a class called commutative theories, Alpuente *et al.* [2] considered anti-unification over equational theories with associative (A), and commutative (C) operators, Cerna and Kutsia studied theories with idempotent operators [8] and most recently, Ayala-Rincón *et al.* studied theories with absorbing operators [3]. Concerning higher-order anti-unification for simply-typed λ-terms, Cerna and Kutsia [11] proposed a generic framework of algorithms producing top-maximal (i.e., retaining maximal common top part of the given terms) and shallow (i.e., forbidding nested generalization variables) generalization variants, while Cerna and Buran [9] proved nullarity of anti-unification in this calculus for the unrestricted case. Some of those equational and higher-order anti-unification algorithms have been implemented and are accessible online [1, 6]. The recent survey [12] gives more detailed information about equational and higher-order anti-unification.

As an example of one of the variants included in the framework presented in [11], namely, the *common subterms variant* or, shortly, CS-variant, consider terms $s = \lambda x.f(g(x), g(g(g(x))))$ and $t = \lambda x.f(g(x), h(h(g(x))))$. They have unique top-maximal shallow CS-lgg $r = \lambda x.f(g(x), X(g(x)))$, which retains not only the topmost maximal common structure of s and t, but also keeps the common subterm $g(x)$ that appears under distinct symbols in them. This common subterm appears in r under the generalization variable X. An example of a non-shallow top-maximal lgg of s and t is $\lambda x.f(g(x), X(X(g(x))))$, where generalization variables appear nested.

Both mentioned papers [9, 11], as well as some other ones (e.g., [7, 13, 16]) assume that in generalization problems, the input terms have the same type. Relaxing this restriction would widen the practical application area of anti-unification techniques. This abstract presents work in progress towards lifting this restriction in the context of intersection and union types. We employ these types to capture the semantics of generalizing terms of different types and illustrate our approach through examples.

2 Preliminaries

Types are constructed from a set of *base types* π using the grammar $\tau ::= \pi \mid \tau \rightarrow \tau \mid \tau \wedge \tau \mid \tau \vee \tau$, where $\wedge$ stands for type *intersection* and $\vee$ for type *union*. We use Greek letters τ, σ and ρ to denote types.

λ-*terms* (typically s, t, r) are built using the grammar $t ::= x \mid c \mid \lambda x.t \mid t\ t$, where x is a variable and c is a constant. Notions as α-conversion, β-reduction, η-long, and β-normal forms are defined as usual (e.g., [14]). Unless otherwise stated, we only consider λ-terms in β-normal η-long form and use *term* and λ-term synonymously. A complete system for typing terms is presented in [5] (see also [17]).

The *subtype relation* is formalized as the set of valid consequences derived using the inference rules presented below (S-ref, S-tran, and S-arrow); these rules derive statements of the form $\tau_1 \leq \tau_2$, read as "τ_1 is a subtype of τ_2" or "τ_2 is a supertype of τ_1".

$$\text{(S-ref)}\ \ \sigma \leq \sigma \qquad \text{(S-tran)}\ \frac{\sigma_1 \leq \sigma_2 \qquad \sigma_2 \leq \sigma_3}{\sigma_1 \leq \sigma_3} \qquad \text{(S-arrow)}\ \frac{\sigma \leq \sigma' \qquad \tau' \leq \tau}{\sigma' \rightarrow \tau' \leq \sigma \rightarrow \tau}$$

The subtyping relation with respect to a given type system $\mathcal{T}$ has the following property: If the judgment $\Gamma \vdash_{\mathcal{T}} t : \sigma$ holds and $\sigma \leq \tau$, then $\Gamma \vdash_{\mathcal{T}} t : \tau$ holds:

$$\text{(T-Sub)}\ \frac{\Gamma \vdash t : \sigma \qquad \sigma \leq \tau}{\Gamma \vdash t : \tau}$$

Properties of subtyping over the intersection and union types relevant to our work are presented below. Note, $\sigma \sim \tau$ denotes that both $\sigma \leq \tau$ and $\tau \leq \sigma$ hold.

$$1.\ \sigma \wedge \sigma \sim \sigma \sim \sigma \vee \sigma, \qquad 2.\ \sigma_i \leq \sigma_1 \vee \sigma_2,\ i = 1, 2, \qquad 3.\ \sigma_1 \wedge \sigma_2 \leq \sigma_i,\ i = 1, 2.$$

This abstract assumes the following:

A1. $\sigma \vee \tau$ is the least upper bound of σ and τ w.r.t. the subtyping relation,

A2. $\sigma \wedge \tau$ is the greatest lower bound of σ and τ w.r.t. the subtyping relation.

A *substitution* (typically θ) is a finite set of pairs $\{X_1 \mapsto t_1, \cdots, X_n \mapsto t_n\}$, where $X_i \neq X_j$ if $i \neq j$. Postfix notation, e.g., $t\theta$, denotes substitution application.

Let the relation $\preceq$ ($\prec$ is the strict part) be a preorder over terms defined as follows: $r \preceq t$ if a substitution θ exists such that $r\theta = t$. A term r is a *generalizer* of two terms s and t if $r \preceq s$ and $r \preceq t$. A term r is a *least general generalizer* (lgg) of s and t if there is no term r' such that r' is a generalizer of s and t and $r \prec r'$. The *anti-unification problem* for s and t, denoted as $s \triangleq t$, is defined as

Given: terms $t : \tau_1$ and $s : \tau_2$ in η-long β-normal form.

Find: an lgg $r : \tau$ of s and t such that $\tau_1 \vee \tau_2 = \tau$.

The intuition behind the lgg is that it should express the common structure of typed input expressions as much as possible while "minimizing the subtyping distance" between its type and the input types. Generalizers represent the divergences in the input term structures by variables. Thus, generalizers can be instantiated into the input terms of the problem. As for the generalizer type, it should be the least possible common supertype of the original ones. From the generalization definition, it follows that there should exist substitutions θ_1 and θ_2 such that $r\theta_1 = s : \tau$ and $r\theta_2 = t : \tau$, where τ is a supertype of both τ_1 and τ_2. There can be many supertypes of τ_1 and τ_2. However, τ is selected as the most specific supertype of τ_1 and τ_2, i.e., τ must be the least upper bound of the types of s and t, which, by assumption 2, is exactly $\tau_1 \vee \tau_2$.

3 Extending Generalization to Types

Generalizing applications. Consider an anti-unification problem $f(a) \triangleq g(b)$ with $f : \sigma \to \tau$, $g : \sigma' \to \tau'$, $a : \rho_a \leq \sigma$ and $b : \rho_b \leq \sigma'$. It is straightforward that the term XY with an adequate type is its solution. The main problem is how to systematically build the adequate generalizer and its type. Such a mechanism is still a work in progress, not addressed in this abstract. Instead, the focus is on the desired properties of such generalizations regarding works on HO-generalization as [10].

Looking at XY above, we see that X must be of function type, i.e., $X : \gamma_1 \to \gamma_2$ because it applies to Y. Furthermore, X is a generalizer of f and g. Thus, its type must be a supertype of both types of f and g: $\sigma \to \tau \leq \gamma_1 \to \gamma_2$ and $\sigma' \to \tau' \leq \gamma_1 \to \gamma_2$. To satisfy both relations, it is necessary that $\gamma_1 \leq \sigma$ and $\gamma_1 \leq \sigma'$, and that $\tau \leq \gamma_2$ and $\tau' \leq \gamma_2$. Choose $\gamma_1 = \sigma \wedge \sigma'$ and $\gamma_2 = \tau \vee \tau'$. Then, the subtyping statements $\sigma \to \tau \leq \gamma_1 \to \gamma_2$ and $\sigma' \to \tau' \leq \gamma_1 \to \gamma_2$ follow from a derivation by the (S-arrow) rule:

$$\frac{\sigma \wedge \sigma' \;\leq\; \sigma \qquad \tau \;\leq\; \tau \vee \tau'}{\sigma \to \tau \leq (\sigma \wedge \sigma') \to (\tau \vee \tau')} \qquad\qquad \frac{\sigma \wedge \sigma' \;\leq\; \sigma' \qquad \tau' \;\leq\; \tau \vee \tau'}{\sigma' \to \tau' \leq (\sigma \wedge \sigma') \to (\tau \vee \tau')}$$

Next, since Y generalizes a and b, the type of Y, say ρ_y, must be a supertype of both ρ_a and ρ_b. Also, $\rho_y \leq \gamma_1$ since XY should be well-typed. It implies that

$$\rho_a \vee \rho_b \leq \rho_y \leq \sigma \wedge \sigma'. \tag{1}$$

Suppose that condition (1) holds, and select the lower bounds of the supertypes obtained above, i.e.,

$$\begin{aligned} &X : \gamma_1 \to \gamma_2 = (\sigma \wedge \sigma') \to (\tau \vee \tau') \\ &Y : \rho_y = \rho_a \vee \rho_b. \end{aligned}$$

Then, it follows that we have $f(a) : \tau$, $g(b) : \tau'$ and their generalizer $XY : \tau \vee \tau'$.

The generalizer XY of $f(a)$ and $g(b)$ is not a shallow term. It was chosen to facilitate readers's comprehension since it is an application (as the input terms). If $a \neq b$, then the unique top-maximal shallow generalizer of $f(a)$ and $g(b)$ is just X.

Cerna and Kutsia introduced a so-called common-subterm variant for HO-generalization (CS-variant, see [11]). It is one of the special cases of top-maximal shallow generalization. In this variant, every free generalization variable occurring in an lgg of two terms s and t and generalizing their subterms s' and t', should apply to maximal common subterms that appear in s' and t'. For example, $X(h(a))$ is a CS-generalizer of $f(h(a), a) \triangleq g(b, h(a))$ while $X(Y, Z)$ is not (although it is their generalizer).

Now consider $f(a) \triangleq g(a)$ with $f : \sigma \to \tau$, $g : \sigma' \to \tau'$, $a : \rho_1 \leq \sigma$ and $a : \rho_2 \leq \sigma'$. The CS-generalizer of this problem is $X(a)$ with the types $X : \gamma_1 \to \gamma_2 = (\sigma \wedge \sigma') \to (\tau \vee \tau)'$ and $a : \rho_1 \vee \rho_2$, subject of an additional constraint $\rho_1 \vee \rho_2 \leq \sigma \wedge \sigma'$.

Generalizing abstractions. Consider an anti-unification problem where both input terms are different identity functions: $\lambda x.x \triangleq \lambda y.y$ where $x : \sigma$ and $y : \tau$. It is straightforward that $\lambda z.z$ with the appropriated type will be the generalizer of the input terms.

To see what should be the generalization type, first notice that $r = \lambda z.z$ must have a function type, i.e., $r : \gamma_1 \to \gamma_2$. Also, this type must be a supertype of both input types: $\sigma \to \sigma \leq \gamma_1 \to \gamma_2$ and $\tau \to \tau \leq \gamma_1 \to \gamma_2$. To satisfy both relations, it is necessary that $\gamma_1 \leq \sigma$, $\gamma_1 \leq \tau$, $\sigma \leq \gamma_2$ and $\sigma \leq \gamma_2$. By a reasoning analogous to the application case above, taking into account that $z : \sigma \wedge \tau$ implies $z : \sigma \vee \tau$, we get that

$$r = \lambda z.z : (\sigma \wedge \tau) \to (\sigma \vee \tau). \tag{2}$$

Observe in (2) that the typing condition for the generalizer decreased the domain and increased the range of those input terms. Otherwise, it would not satisfy the requirement that the type of the generalizer must be a supertype of the types of the input terms. Obviously, $(\sigma \wedge \tau) \to (\sigma \wedge \tau)$ is a supertype neither of $\sigma \to \sigma$ nor of $\tau \to \tau$. The same is true for $(\sigma \vee \tau) \to (\sigma \vee \tau)$. This is a consequence of the contravariance in the rule (S-arrow).

Example 1. Now, consider the identity function defined in different sets: $\texttt{ID}_{\mathbb{N}}(n) = n$ and $\texttt{ID}_{\mathbb{Z}}(z) = z$. They are expressed in λ-calculus by $\lambda(x : \mathbb{N}).(x : \mathbb{N})$ and $\lambda(y : \mathbb{Z}).(y : \mathbb{Z})$, then the generalizer of those functions will be $\lambda(k : \mathbb{N} \wedge \mathbb{Z}).(k : \mathbb{N} \vee \mathbb{N})$. Therefore, the generalizer is $\texttt{ID}_{\texttt{gen}}(k) = k$ with domain $\mathbb{N} \cap \mathbb{Z}$ and range $\mathbb{N} \cup \mathbb{Z}$ which means that the computed generalizer is a non-surjective identity function from $\mathbb{N}$ to $\mathbb{Z}$.

This abstract does not discuss the inhabitation of intersection types. It is clear that by interpreting types as sets, some intersection types may get uninhabited. In these special cases, some generalizations will not have semantic meaning, suggesting that generalizations of the input terms do not exist. (A similar phenomenon was observed in the calculus of constructions [15] where there is no semantic interpretation of the generalization of abstract kinds, Set, Prop, Type, etc.) For instance, consider two identity functions: one defined on the set of rational numbers and the other one on the set of irrational numbers, respectively: $\texttt{ID}_{\mathbb{Q}}(q) = q$ and $\texttt{ID}_{\mathbb{I}}(i) = i$. The generalizer should be $\texttt{ID}_{\texttt{gen}}(g) = g$ with domain $\mathbb{Q} \cap \mathbb{I}$ and range $\mathbb{Q} \cup \mathbb{I}$; however, this domain set is empty. Consequently, the generalizer does not exist.

Now, consider a generalization problem $\lambda x.f(x, a) \triangleq \lambda y.g(b, y)$ with $x : \sigma_x \leq \sigma$, $f : \sigma \to \tau$, $y : \sigma_y \leq \sigma'$ and $g : \sigma' \to \tau'$, and the requirement that a solution of this problem should retain the common top-maximal structure of the given terms. Since both terms are abstractions, the desired generalizer should be an abstraction, too. Also, since the scopes of the input terms have different heads f and g, the scope of the generalizer must be a free variable. Therefore, $\lambda z.X(z)$ is the lgg that can be transformed to the original expressions via the substitutions $\theta_1 = \{X \mapsto \lambda u.f(u,a)\}$ and $\theta_2 = \{X \mapsto \lambda u.g(b,u)\}$:

$$\lambda z.X(z)\theta_1 = \lambda z.(\lambda u.f(u,a))(z) =_\beta \lambda z.f(z,a) =_\alpha \lambda x.f(x,a)$$
$$\lambda z.X(z)\theta_2 = \lambda z.(\lambda u.f(b,u))(z) =_\beta \lambda z.g(b,z) =_\alpha \lambda y.g(b,y).$$

What about the type of this generalizer? Let $\lambda z.X(z) : \gamma_1 \to \gamma_2$. Again, to infer the conditions for this to be an adequate type, we have $\sigma_x \to \tau \leq \gamma_1 \to \gamma_2$ and $\sigma_y \to \tau' \leq \gamma_1 \to \gamma_2$, which imply

$$\lambda z.X(z) : (\sigma_x \wedge \sigma_y) \to (\tau \vee \tau').$$

Then $z : \sigma_x \wedge \sigma_y$ and $X(z) : \tau \vee \tau'$. Since X applies z and has range $\tau \vee \tau'$, it follows that $X : (\sigma_x \wedge \sigma_y) \to (\tau \vee \tau')$.

Example 2. With this example, we illustrate how generalization with intersection and union types can be used to synthesize a generic function from two concrete instances. As the given concrete ones, consider two functions, congruences modulo 3 and 5, defined respectively as

$$\mathtt{MOD}_3(n : \mathbb{N}) : \{0, \ldots, 2\} = \text{if } n < 3 \text{ then } n \text{ else } \mathtt{MOD}_3(n-3) \tag{3}$$
$$\mathtt{MOD}_5(n : \mathbb{N}) : \{0, \ldots, 4\} = \text{if } n < 5 \text{ then } n \text{ else } \mathtt{MOD}_5(n-5), \tag{4}$$

where the explicit types indicate that $\mathtt{MOD}_3 : \mathbb{N} \to \{0, \ldots, 2\}$ and $\mathtt{MOD}_5 : \mathbb{N} \to \{0, \ldots, 4\}$.

We aim to synthesize a generic function for congruence modulo, from which proper instantiations are used to obtain these concrete ones. This we do in two steps, where only the first one concerns generalizer computation:

Step 1. Anti-unify (3) and (4). It will give

$$X(n) = \text{if } n < k \text{ then } n \text{ else } X(n-k) \tag{5}$$

where X and k are generalization variables of types respectively $X : (\mathbb{N} \wedge \mathbb{N}) \to (\{0, \ldots, 2\} \vee \{0, \ldots, 4\}) = \mathbb{N} \to \{0, \ldots, 4\}$ and $k : \mathbb{N}$. The original expressions (3) and (4) can be obtained by the substitutions, respectively:

$$\{X \mapsto \lambda x.\mathtt{MOD}_3(x),\ k \mapsto 3\}$$
$$\{X \mapsto \lambda x.\mathtt{MOD}_5(x),\ k \mapsto 5\}.$$

Step 2. Notice that although (5) is a (least general) generalizer of two function definitions (3) and (4), it is not a function definition itself because of those free generalization variables. Now, we turn it into such a definition by introducing a new function name `GENMOD` (meaning generic `MOD`) instead of X. It has both n and k as its arguments:

$$\mathtt{GENMOD}(n,k) = \text{if } n < k \text{ then } n \text{ else } \mathtt{GENMOD}(n-k,k) \tag{6}$$

With some further processing that is beyond the scope of this abstract, one can connect the type to `GENMOD` to k as, e.g., $\mathbb{N} \to \mathbb{N} \to \{0, \ldots, k-1\}$.

4 Final Remarks

This abstract presented preliminary investigations about higher-order generalization with intersection and union types, aiming at extending existing HO-generalization problems (e.g., [7, 10, 11]) to this setting. We illustrated some desirable properties of such generalizers. An anti-unification algorithm to construct generalizers and an algorithm to compute their (minimal) types are subjects of ongoing work. Future research should cover anti-unification with abstractions and terms of different structures.

Acknowledgments. The Brazilian Higher Education Council (CAPES) supported the Brazilian-Austrian cooperation through the PrInt program finance code 001. Funded by the Austrian Science Fund (FWF), under project P 35530; Czech Science Foundation, grant No. 22-06414L; Cost Action CA20111 EuroProofNet; Brazilian Research Council (CNPq), grants Universal 409003/21-2, RG 313290/21-0; and the Brazilian Federal District Research Foundation FAPDF, grant DE 00193-00001175/2021-11.

References

[1] María Alpuente, Demis Ballis, Angel Cuenca-Ortega, Santiago Escobar, and José Meseguer. Acuos^2: A high-performance system for modular ACU generalization with subtyping and inheritance. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 171–181. Springer, 2019.

[2] María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Inf. Comput.*, 235:98–136, 2014.

[3] Mauricio Ayala-Rincón, David M. Cerna, Andrés Felipe Gonzáez Barragán, and Temur Kutsia. Equational anti-unification over absorption theories. In *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, Proceedings*, volume 14740 of *Lecture Notes in Computer Science*. Springer, 2024.

[4] Franz Baader. Unification, weak unification, upper bound, lower bound, and generalization problems. In *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*, volume 488 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 1991.

[5] Franco Barbanera and Mariangiola Dezani-Ciancaglini. Intersection and union types. In Takayasu Ito and Albert R. Meyer, editors, *Theoretical Aspects of Computer Software, International Conference TACS '91, Sendai, Japan, September 24-27, 1991, Proceedings*, volume 526 of *Lecture Notes in Computer Science*, pages 651–674. Springer, 1991.

[6] Alexander Baumgartner and Temur Kutsia. A library of anti-unification algorithms. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2014.

[7] Alexander Baumgartner, Temur Kutsia, Jordi Levy, and Mateu Villaret. Higher-order pattern anti-unification in linear time. *J. Autom. Reason.*, 58(2):293–310, 2017.

[8] David Cerna and Temur Kutsia. Idempotent anti-unification. *ACM Trans. Comput. Log.*, 21(2):10:1–10:32, 2020.

[9] David M. Cerna and Michal Buran. One or nothing: Anti-unification over the simply-typed lambda calculus. *ACM Trans. Comput. Logic*, 2024. Accepted.

[10] David M. Cerna and Temur Kutsia. Higher-order equational pattern anti-unification. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduc-*

tion, FSCD 2018, July 9-12, 2018, Oxford, UK, volume 108 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[11] David M. Cerna and Temur Kutsia. A generic framework for higher-order generalizations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 10:1–10:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[12] David M. Cerna and Temur Kutsia. Anti-unification and generalization: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 6563–6573. ijcai.org, 2023.

[13] Cao Feng and Stephen H. Muggleton. Towards inductive generalization in higher order logic. In Derek H. Sleeman and Peter Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1-3, 1992*, pages 154–162. Morgan Kaufmann, 1992.

[14] J Roger Hindley. *Basic simple type theory.* Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.

[15] Frank Pfenning. Unification and anti-unification in the calculus of constructions. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 74–85. IEEE Computer Society, 1991.

[16] Brigitte Pientka. Higher-order term indexing using substitution trees. *ACM Trans. Comput. Log.*, 11(1):6:1–6:40, 2009.

[17] Benjamin C. Pierce. *Types and programming languages.* MIT Press, 2002.

[18] Gordon D. Plotkin. A note on inductive generalization. *Machine Intelligence 5*, 5:153–163, 1970.

[19] John C. Reynolds. Transformational system and the algebraic structure of atomic formulas. *Machine Intelligence 5*, 5:135–151, 1970.