Informal Proceedings of the
35th International Workshop on Unification (UNIF 2021)
18th of July 2021, Buenos Aires, Argentina

Preface

This volume contains the papers presented at the 35th International Workshop on Unification (UNIF 2021). UNIF 2021 was affiliated with the 6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021) and held on July 18, 2021 in Buenos Aires, Argentina (online due to the COVID-19 pandemic).

Unification is concerned with the problem of identifying given (first- or higher-order) terms, either syntactically or modulo a theory. It is a fundamental technique that is employed in various areas of Computer Science and Mathematics. In particular, unification algorithms are key components in completion of term rewriting systems, resolution-based theorem proving, and logic programming. But unification is, for example, also investigated in the context of natural language processing, program analysis, types, modal logics, and in knowledge representation. UNIF is a well-established yearly event. Its purpose is to bring together researchers interested in unification theory and its applications, as well as closely related topics, such as matching (i.e., one-sided unification), anti-unification (i.e., the dual problem to unification), disunification (i.e., solving equations and inequations) and the admissibility problem (which generalizes unification in modal logics). The UNIF workshop series provides a forum for presenting recent (even unfinished) work, and discuss new ideas and trends in this and related fields.

The Program Committee of UNIF 2021 selected 9 contributions for presentation. Each submission was evaluated by at least three reviewers. We would like to thank all members of the Program Committee and the Organizers of FSCD for their support in the preparation of the UNIF 2021 workshop. The EasyChair system, designed by Andrei Voronkov, was a great help for organizing the reviewing process.

July 2021

Franz Baader
Alexander Baumgartner

Workshop Organization

Chairs

Franz Baader	TU Dresden
Alexander Baumgartner	Universidad de O'Higgins

Program Committee

Mauricio Ayala-Rincón	Universidade de Brasilía
Franz Baader	TU Dresden
Philippe Balbiani	Institut de recherche en informatique de Toulouse
Alexander Baumgartner	Universidad de O'Higgins
David Cerna	RISC, Johannes Kepler University Linz
Besik Dundua	Institute of Applied Mathematics, Tbilisi State University
Serdar Erbatur	Ludwig Maximilian University of Munich
Santiago Escobar	Universitat Politècnica de València
Silvio Ghilardi	Dipartimento di Matematica, Università degli Studi di Milano
Temur Kutsia	RISC, Johannes Kepler University Linz
Jordi Levy	IIIA - CSIC
Christopher Lynch	Clarkson University
Andrew M. Marshall	University of Mary Washington
Barbara Morawska	Ahmedabad University
Daniele Nantes-Sobrinho	Universidade de Brasilía
Paliath Narendran	University at Albany, SUNY
Veena Ravishankar	University of Mary Washington
Christophe Ringeissen	INRIA
David Sabel	Ludwig-Maximilians-University Munic
Manfred Schmidt-Schauß	Goethe-University Frankfurt am Main
Yu Zhang	University of Albany, SUNY

Table of Contents

About the unification type of modal logic K5 and its extensions
Majid Alizadeh, Mohammad Ardeshir, Philippe Balbiani and Mojtaba Mojtahedi

Restricted Unification in the Description Logic FL0
Franz Baader, Oliver Fernández Gil and Maryam Rostamigiv

Nominal Disunification via Fixed-Point Constraints
Leonardo M. Batista, Maribel Fernández, Daniele Nantes-Sobrinho and Deivid Vale

Nominal Anti-Unification with Atom-Variables
Manfred Schmidt-Schauß

Checking Symbolic Security of Cryptographic Modes of Operation
Hai Lin and Christopher Lynch

Formal Analysis of Symbolic Authenticity
Hai Lin and Christopher Lynch

What, Again? Automatic Deductive Synthesis of the Unification Algorithm
Richard Waldinger

When First-order Unification Calls itself
David M. Cerna

A Polynomial-time Algorithm for the Common Left Multiplier Problem for Forward-closed String Rewriting Systems
Wei Du, Paliath Narendran and Bharvee Acharya

About the unification type of modal logic $\mathbf{K}5$ and its extensions*

Majid Alizadeh[1], Mohammad Ardeshir[2], Philippe Balbiani[3], and Mojtaba Mojtahedi[1]

[1] School of Mathematics, Statistics and Computer Science, College of Science, University of Tehran, Tehran, Iran
[2] Department of Mathematical Sciences, Sharif University of Technology, Tehran, Iran
[3] Toulouse Institute of Computer Science Research, CNRS — Toulouse University, Toulouse, France

1 Introduction

We prove that all extensions of $\mathbf{K}45$ have projective unification and $\mathbf{K}5$ and some of its extensions are of unification type 1. The breakdown of the paper is as follows. Firstly, we prove in Proposition 9 that if $\mathbf{L}$ contains $\mathbf{K}45$ then every formula has extension property in $\mathbf{L}$. Secondly, generalizing some results obtained in [7], we prove in Proposition 13 that if $\mathbf{L}$ contains $\mathbf{K}5$ then every $\mathbf{L}$-unifiable formula is $\mathbf{L}$-filtering. Thirdly, imitating arguments used in [19, 20], we prove in Proposition 20 that if $\mathbf{L}$ contains $\mathbf{K}5$ then every formula having extension property in $\mathbf{L}$ is $\mathbf{L}$-projective. Fourthly, we prove in Proposition 21 that if $\mathbf{L}$ contains $\mathbf{K}5$ and $\mathbf{L}$ is global[1] then for all substitutions σ, every formula $\mathbf{L}$-unified by σ is implied by an $\mathbf{L}$-projective formula based on the variables of the given formula and having σ as one of its $\mathbf{L}$-unifiers. The proofs of some of our results can be found in [1].

2 Syntax and semantics

Let $\mathbf{VAR}$ be a countably infinite set of *variables* (with typical members denoted x, y, etc). The set $\mathbf{FOR}$ of all *formulas* (with typical members denoted φ, ψ, etc) is inductively defined by

- $\varphi := x \mid \bot \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \Box\varphi$.

We adopt the standard rules for omission of parentheses. The Boolean connectives $\top$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ and the modal connective $\Diamond$ are defined as usual. For all $\varphi{\in}\mathbf{FOR}$, let $\mathtt{var}(\varphi)$ be the set of all variables occurring in φ. For all finite $X{\subseteq}\mathbf{VAR}$, let $\mathbf{FOR}_X$ be the set of all $\varphi{\in}\mathbf{FOR}$ such that $\mathtt{var}(\varphi){\subseteq}X$.

A *substitution* is a triple (X,Y,σ) where $X,Y{\subseteq}\mathbf{VAR}$ are finite and $\sigma: \mathbf{FOR}_X \longrightarrow \mathbf{FOR}_Y$ is a homomorphism. The sets X and Y are respectively its *domain* and its *codomain*. Let $\mathbf{SUB}$ be the set of all substitutions. We say that $(X,Y,\sigma){\in}\mathbf{SUB}$ is *variable-free* if $Y{=}\emptyset$. It is possible to compose two substitutions if the codomain of the first is equal to the domain of the second. The *composition* of $(X,Y,\sigma),(Y,Z,\tau){\in}\mathbf{SUB}$ (in symbols $(X,Y,\sigma)\circ(Y,Z,\tau)$) is the substitution (X,Z,υ) such that for all $x{\in}X$, $\upsilon(x){=}\tau(\sigma(x))$. When its domain and its codomain can be guessed from the context, the substitution (X,Y,σ) will be simply written σ[2]. For all finite $X,Y{\subseteq}\mathbf{VAR}$, let $\mathbf{SUB}_{X,Y}$ be the set of all $\sigma{\in}\mathbf{SUB}$ such that the domain of σ is X and the codomain of σ is Y.

We say that $\mathbf{L}{\subseteq}\mathbf{FOR}$ is a *modal logic* if the following conditions hold[3]: $\mathbf{L}$ contains all tautologies, $\mathbf{L}$ contains the formula $\Box(x \rightarrow y) \rightarrow (\Box x \rightarrow \Box y)$, $\mathbf{L}$ is closed for *modus ponens* (for all $\varphi,\psi{\in}\mathbf{FOR}$, if

*A long version of this paper is currently submitted to the *Logic Journal of the IGPL*.

[1] Globality is defined in Section 2.

[2] However, when we write that two substitutions are equal, this will imply in any case that their domains are equal and their codomains are equal.

[3] The modal logics considered in this paper are exactly the *normal* modal logics considered in standard textbooks such as [13, 14, 28].

$\varphi \to \psi \in \mathbf{L}$ and $\varphi \in \mathbf{L}$ then $\psi \in \mathbf{L}$), $\mathbf{L}$ is closed for *generalization* (for all $\varphi \in \mathbf{FOR}$, if $\varphi \in \mathbf{L}$ then $\Box\varphi \in \mathbf{L}$), $\mathbf{L}$ is closed for *uniform substitution* (for all $\varphi \in \mathbf{FOR}$, if $\varphi \in \mathbf{L}$ then for all substitutions (X, Y, σ), if $\mathtt{var}(\varphi) \subseteq X$ then $\sigma(\varphi) \in \mathbf{L}$). For all modal logics $\mathbf{L}$ and for all $\varphi \in \mathbf{FOR}$, we write $\mathbf{L} \oplus \varphi$ for the least modal logic containing $\mathbf{L}$ and φ. The following modal logics — and their extensions — are considered in this paper: $\mathbf{K} \oplus \Diamond x \to \Box\Diamond x$ (denoted $\mathbf{K}5$), $\mathbf{K}5 \oplus \Box x \to \Box\Box x$ (denoted $\mathbf{K}45$), $\mathbf{K}5 \oplus \Box x \to x$ (denoted $\mathbf{S}5$), $\mathbf{K}$ denoting the least modal logic[4]. We say that a modal logic $\mathbf{L}$ is *consistent* if $\mathbf{L} \neq \mathbf{FOR}$. From now on in this paper, **let $\mathbf{L}$ be a consistent modal logic.** Let $\equiv_{\mathbf{L}}$ be the equivalence relation on $\mathbf{FOR}$ defined for all $\varphi, \psi \in \mathbf{FOR}$, by $\varphi \equiv_{\mathbf{L}} \psi$ if and only if $\varphi \leftrightarrow \psi \in \mathbf{L}$. We shall say that $\mathbf{L}$ is *locally tabular* if for all finite $X \subseteq \mathbf{VAR}$, $\equiv_{\mathbf{L}}$ possesses finitely many equivalence classes on $\mathbf{FOR}_X$.

Proposition 1. *If $\mathbf{L}$ contains $\mathbf{K}5$ then $\mathbf{L}$ is locally tabular.*

We say that $\varphi \in \mathbf{FOR}$ is $\mathbf{L}$-*derivable* from $\Gamma \subseteq \mathbf{FOR}$ (in symbols $\Gamma \vdash_{\mathbf{L}} \varphi$) if there exists $n \geq 1$ and there exists $\varphi_1, \ldots, \varphi_n \in \mathbf{FOR}$ such that $\varphi_n = \varphi$ and for all $k \in \{1, \ldots, n\}$, at least one of the following 4 conditions holds: *(i)* $\varphi_k \in \mathbf{L}$, *(ii)* $\varphi_k \in \Gamma$, *(iii)* there exists $i, j \in \{1, \ldots, n\}$ such that $i, j < k$ and $\varphi_i = \varphi_j \to \varphi_k$, *(iv)* there exists $i \in \{1, \ldots, n\}$ such that $i < k$ and $\varphi_k = \Box\varphi_i$. Substitutions being completely defined by the restrictions to their domains, it is possible to compare two substitutions by means of these restrictions if their domains are equal. Let $\simeq_{\mathbf{L}}$ be the equivalence relation on $\mathbf{SUB}$ defined for all $(X, Y, \sigma), (X, Z, \tau) \in \mathbf{SUB}$, by $(X, Y, \sigma) \simeq_{\mathbf{L}} (X, Z, \tau)$ if and only if for all $x \in X$, $\sigma(x) \leftrightarrow \tau(x) \in \mathbf{L}$[5]. Let $\preccurlyeq_{\mathbf{L}}$ be the quasi-order on $\mathbf{SUB}$ defined for all $(X, Y, \sigma), (X, Z, \tau) \in \mathbf{SUB}$, by $(X, Y, \sigma) \preccurlyeq_{\mathbf{L}} (X, Z, \tau)$ if and only if there exists $(Z, T, \upsilon) \in \mathbf{SUB}$ such that for all $x \in X$, $\sigma(x) \leftrightarrow \upsilon(\tau(x)) \in \mathbf{L}$[6].

Proposition 2. *If $\mathbf{L}$ is locally tabular then for all finite $X, Y \subseteq \mathbf{VAR}$, $\simeq_{\mathbf{L}}$ possesses finitely many equivalence classes on $\mathbf{SUB}_{X,Y}$.*

A *frame* is a couple (W, R) where W is a non-empty set and R is a binary relation on W[7]. In a frame (W, R), for all $s \in W$, let $R(s) = \{t \in W : sRt\}$ and for all $U \subseteq W$, let $R(U) = \{t \in W$: there exists $s \in U$ such that $sRt\}$. We say that a frame (W, R) is generated from $s \in W$ if for all $t \in W$, there exists $n \geq 0$ and there exists $u_0, \ldots, u_n \in W$ such that $u_0 = s$, $u_n = t$ and for all $i \in \{1, \ldots, n\}$, $u_{i-1} R u_i$. A *valuation* on a frame (W, R) is a function assigning to each variable a subset of W. Given a frame (W, R) and a valuation V on (W, R), the *satisfiability* of $\varphi \in \mathbf{FOR}$ at $s \in W$ (in symbols $(W, R), V, s \models \varphi$) is inductively defined as follows:

- $(W, R), V, s \models x$ if and only if $s \in V(x)$,
- $(W, R), V, s \not\models \bot$,
- $(W, R), V, s \models \neg\varphi$ if and only if $(W, R), V, s \not\models \varphi$,
- $(W, R), V, s \models \varphi \vee \psi$ if and only if either $(W, R), V, s \models \varphi$, or $(W, R), V, s \models \psi$,
- $(W, R), V, s \models \Box\varphi$ if and only if for all $t \in W$, if sRt then $(W, R), V, t \models \varphi$.

We say that a formula φ is *valid* in a frame (W, R) (in symbols $(W, R) \models \varphi$) if for all valuations V on (W, R) and for all $s \in W$, $(W, R), V, s \models \varphi$. We say that $\mathbf{L}$ is *valid* in a frame (W, R) (in symbols $(W, R) \models \mathbf{L}$) if for all $\varphi \in \mathbf{L}$, $(W, R) \models \varphi$. For all frames (W, R), for all substitutions (X, Y, σ) and for

[4]Obviously, $\mathbf{K}45$ contains $\mathbf{K}5$. Moreover, as is well-known [23, Chapter 3], $\mathbf{S}5$ contains $\mathbf{K}45$.

[5]Obviously, for all $(X, Y, \sigma), (X, Z, \tau) \in \mathbf{SUB}$, if $(X, Y, \sigma) \simeq_{\mathbf{L}} (X, Z, \tau)$ then for all $\varphi \in \mathbf{FOR}_X$, $\sigma(\varphi) \leftrightarrow \tau(\varphi) \in \mathbf{L}$.

[6]Obviously, for all $(X, Y, \sigma), (X, Z, \tau) \in \mathbf{SUB}$, if $(X, Y, \sigma) \preccurlyeq_{\mathbf{L}} (X, Z, \tau)$ then there exists $(Z, T, \upsilon) \in \mathbf{SUB}$ such that for all $\varphi \in \mathbf{FOR}_X$, $\sigma(\varphi) \leftrightarrow \upsilon(\tau(\varphi)) \in \mathbf{L}$. Moreover, for all $(X, Y, \sigma), (X, Z, \tau) \in \mathbf{SUB}$, if $(X, Y, \sigma) \simeq_{\mathbf{L}} (X, Z, \tau)$ then $(X, Y, \sigma) \preccurlyeq_{\mathbf{L}} (X, Z, \tau)$.

[7]We assume the reader is at home with the relational semantics of modal logics. For more on this, see [13, 14, 28].

all valuations V on (W,R), let V^σ be the valuation on (W,R) such that for all $x\in\mathbf{VAR}$, if $x\in X$ then $V^\sigma(x)=\{s\in W:(W,R),V,s\models\sigma(x)\}$ else $V^\sigma(x)=V(x)$[8].

Proposition 3. *Let (W,R) be a frame, (X,Y,σ) be a substitution and V be a valuation on (W,R). For all $\varphi\in\mathbf{FOR}_X$ and for all $s\in W$, $(W,R),V^\sigma,s\models\varphi$ if and only if $(W,R),V,s\models\sigma(\varphi)$.*

Proposition 4. *Let (W,R) be a frame such that $(W,R)\models\mathbf{L}$. If $\mathbf{L}$ contains $\mathbf{K}5$ then for all $s\in W$, if (W,R) is generated from s then exactly one of the following 3 conditions holds: (i) $W=\{s\}$ and $R=\emptyset$, (ii) $R=W\times W$, (iii) there exists $A,B\subseteq W$ such that $A\neq\emptyset$, $A\subseteq B$, $s\notin B$, $W=\{s\}\cup B$ and $R=(\{s\}\times A)\cup(B\times B)$. If $\mathbf{L}$ contains $\mathbf{K}45$ then for all $s\in W$, if (W,R) is generated from s then exactly one of the following 3 conditions holds: (iv) $W=\{s\}$ and $R=\emptyset$, (v) $R=W\times W$, (vi) there exists $A\subseteq W$ such that $A\neq\emptyset$, $s\notin A$, $W=\{s\}\cup A$ and $R=(\{s\}\times A)\cup(A\times A)$.*

Let $\mathcal{S}$ be a frame (W,R) such that $\mathtt{Card}(W)=1$ and $R=\emptyset$. For all $m\geq1$, let $\mathcal{T}_m$ be a frame (W,R) such that $\mathtt{Card}(W)=m$ and $R=W\times W$. For all $m\geq1$ and for all $n\geq0$, let $\mathcal{U}_{(m,n)}$ be a frame (W,R) such that there exists $s\in W$ and there exists $A,B\subseteq W$ such that $A\neq\emptyset$, $A\subseteq B$, $s\notin B$, $W=\{s\}\cup B$, $R=(\{s\}\times A)\cup(B\times B)$, $\mathtt{Card}(A)=m$ and $\mathtt{Card}(B)=m+n$.

Proposition 5. *If $\mathbf{L}$ contains $\mathbf{K}5$ then exactly one of the following conditions holds:* (i) *for all $m\geq1$, $\mathcal{T}_m\models\mathbf{L}$ and $\mathcal{S}\models\mathbf{L}$,* (ii) *for all $m\geq1$, $\mathcal{T}_m\models\mathbf{L}$ and $\mathcal{S}\not\models\mathbf{L}$,* (iii) *there exists $m\geq1$ such that $\mathcal{T}_m\models\mathbf{L}$, there exists $n\geq1$ such that $\mathcal{T}_n\not\models\mathbf{L}$ and $\mathcal{S}\models\mathbf{L}$,* (iv) *there exists $m\geq1$ such that $\mathcal{T}_m\models\mathbf{L}$, there exists $n\geq1$ such that $\mathcal{T}_n\not\models\mathbf{L}$ and $\mathcal{S}\not\models\mathbf{L}$,* (v) *for all $m\geq1$, $\mathcal{T}_m\not\models\mathbf{L}$.*

We say that $\mathbf{L}$ is *global* if for all $m,m'\geq1$ and for all $n'\geq0$, if $m=m'+n'$ and $\mathcal{T}_m\models\mathbf{L}$ then $\mathcal{U}_{(m',n')}\models\mathbf{L}$. For all positive integers l, let $\varphi_l=\bigwedge\{\Diamond\Diamond x_k:\ 0\leq k\leq l\}\to\bigvee\{\Diamond\Diamond(x_i\wedge x_j):\ 0\leq i<j\leq l\}$.

Proposition 6. *If either $\mathbf{L}=\mathbf{K}5$, or $\mathbf{L}=\mathbf{K}5\oplus\Diamond\top$, or $\mathbf{L}=\mathbf{K}5\oplus\varphi_l$ for some positive integer l, or $\mathbf{L}=\mathbf{K}5\oplus\varphi_l\oplus\Diamond\top$ for some positive integer l, or $\mathbf{L}=\mathbf{K}5\oplus\Box\bot$ then $\mathbf{L}$ is global.*

Proposition 7. *If $\mathbf{L}$ contains $\mathbf{K}5$ and $\mathbf{L}$ is global then either $\mathbf{L}=\mathbf{K}5$, or $\mathbf{L}=\mathbf{K}5\oplus\Diamond\top$, or $\mathbf{L}=\mathbf{K}5\oplus\varphi_l$ for some positive integer l, or $\mathbf{L}=\mathbf{K}5\oplus\varphi_l\oplus\Diamond\top$ for some positive integer l, or $\mathbf{L}=\mathbf{K}5\oplus\Box\bot$.*

Proposition 8. *If $\mathbf{L}$ contains $\mathbf{K}5$ then for all $\varphi\in\mathbf{FOR}$, if $\varphi\notin\mathbf{L}$ then there exists a finite frame (W,R), there exists a valuation V on (W,R) and there exists $s\in W$ such that $(W,R)\models\mathbf{L}$, (W,R) is generated from s and $(W,R),V,s\not\models\varphi$.*

For all finite frames (W,R), for all valuations V on (W,R), for all $s\in W$ and for all finite $X\subseteq\mathbf{VAR}$, we say that a valuation V' on (W,R) is a *variant* of V with respect to s and X if for all $x\in X$, $V'(x)\setminus\{s\}=V(x)\setminus\{s\}$. We say that $\varphi\in\mathbf{FOR}$ has *extension property* in $\mathbf{L}$ if for all finite frames (W,R), for all valuations V on (W,R) and for all $s\in W$, if $(W,R)\models\mathbf{L}$ and (W,R) is generated from s then there exists a variant V' of V with respect to s and $\mathtt{var}(\varphi)$ such that $(W,R),V',s\models\Diamond\Box\varphi\to\varphi$. The following result is essential for the proof of Proposition 23.

Proposition 9. *If $\mathbf{L}$ contains $\mathbf{K}45$ then for all $\varphi\in\mathbf{FOR}$, φ has extension property in $\mathbf{L}$.*

For all finite $X\subseteq\mathbf{VAR}$, for all finite frames (W,R), for all valuations V on (W,R) and for all $s\in W$, let $\mathtt{for}_X((W,R),s,V)=\{\chi\in\mathbf{FOR}_X:\ (W,R),V,s\models\chi\}$. Obviously, $\mathtt{for}_X((W,R),s,V)$ is an infinite subset of $\mathbf{FOR}_X$. Nevertheless, when $\mathbf{L}$ is locally tabular, we will treat $\mathtt{for}_X((W,R),s,V)$ as if it is a finite subset of $\mathbf{FOR}_X$. In that case, $\mathtt{for}_X((W,R),s,V)$ will also denote the conjunction of all formulas in this finite subset.

[8] Such definition is standard [4, 16, 19, 20].

3 Unification

An $\mathbf{L}$-*unifier* of $\varphi\in\mathbf{FOR}$ is a substitution $(\mathtt{var}(\varphi), X, \sigma)$ such that $\sigma(\varphi)\in\mathbf{L}$. We write $\Sigma_{\mathbf{L}}(\varphi)$ to mean the set of all $\mathbf{L}$-unifiers of $\varphi\in\mathbf{FOR}$. We say that $\varphi\in\mathbf{FOR}$ is $\mathbf{L}$-*unifiable* if $\Sigma_{\mathbf{L}}(\varphi)\neq\emptyset$. Since $\mathbf{L}$ is closed for uniform substitution, for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, $\Sigma_{\mathbf{L}}(\varphi)$ contains variable-free substitutions. We say that an $\mathbf{L}$-unifier σ of $\varphi\in\mathbf{FOR}$ is a *most general* $\mathbf{L}$-*unifier* of φ if for all $\mathbf{L}$-unifiers τ of φ, $\tau\preccurlyeq_{\mathbf{L}}\sigma$. We say that a set Σ of $\mathbf{L}$-unifiers of an $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$ is *complete* if for all $\mathbf{L}$-unifiers σ of φ, there exists $\tau\in\Sigma$ such that $\sigma\preccurlyeq_{\mathbf{L}}\tau$[9]. We say that a complete set Σ of $\mathbf{L}$-unifiers of an $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$ is a *basis* for φ if for all $\sigma,\tau\in\Sigma$, if $\sigma\preccurlyeq_{\mathbf{L}}\tau$ then $\sigma=\tau$[10].

Proposition 10. *For all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$ and for all bases Σ,Δ for φ, Σ and Δ have the same cardinality.*

As a consequence of Proposition 10, an important question is the following: when $\varphi\in\mathbf{FOR}$ is $\mathbf{L}$-unifiable, is there a basis for φ? When the answer is "yes", how large is this basis? For all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, we say that φ is *of type* 1 if there exists a basis for φ with cardinality 1, φ is *of type* ω if there exists a basis for φ with finite cardinality ≥ 2, φ is *of type* ∞ if there exists a basis for φ with infinite cardinality, φ is *of type* 0 if there exists no basis for φ[11]. We say that $\mathbf{L}$ is *of type* 1 if every $\mathbf{L}$-unifiable formula is of type 1, $\mathbf{L}$ is *of type* ω if every $\mathbf{L}$-unifiable formula is either of type 1, or of type ω and there exists an $\mathbf{L}$-unifiable formula of type ω, $\mathbf{L}$ is *of type* ∞ if every $\mathbf{L}$-unifiable formula is either of type 1, or of type ω, or of type ∞ and there exists an $\mathbf{L}$-unifiable formula of type ∞, $\mathbf{L}$ is *of type* 0 if there exists an $\mathbf{L}$-unifiable formula of type 0[12]. For all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, we say that φ is $\mathbf{L}$-*filtering* if for all $\mathbf{L}$-unifiers σ,τ of φ, there exists an $\mathbf{L}$-unifier υ of φ such that $\sigma\preccurlyeq_{\mathbf{L}}\upsilon$ and $\tau\preccurlyeq_{\mathbf{L}}\upsilon$.

Proposition 11. *Let $\varphi\in\mathbf{FOR}$ be $\mathbf{L}$-unifiable. If φ is $\mathbf{L}$-filtering then φ is either of type 1, or of type 0.*

We say that $\mathbf{L}$ has *filtering unification* if for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, φ is $\mathbf{L}$-filtering.

Proposition 12. *If $\mathbf{L}$ has filtering unification then $\mathbf{L}$ is either of type 1, or of type 0.*

The following result is essential for the proof of Proposition 25.

Proposition 13. *If $\mathbf{L}$ contains $\mathbf{K}5$ then for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, φ is $\mathbf{L}$-filtering.*

For all $\varphi\in\mathbf{FOR}$, a substitution $(\mathtt{var}(\varphi),\mathtt{var}(\varphi),\sigma)$ is $\mathbf{L}$-*projective* for φ if for all $x\in\mathtt{var}(\varphi)$, $\varphi\vdash_{\mathbf{L}} x\leftrightarrow\sigma(x)$.

Proposition 14. *Let $\varphi\in\mathbf{FOR}$. Let (W,R) be a finite frame, V be a valuation on (W,R) and $s\in W$ be such that $(W,R)\models\mathbf{L}$, (W,R) is generated from s and $(W,R),V,s\models\Diamond\Box\varphi$. If $\mathbf{L}$ contains $\mathbf{K}5$ then for all $\mathbf{L}$-projective substitutions υ for φ, V^{υ} is a variant of V with respect to s and $\mathtt{var}(\varphi)$.*

Proposition 15. *Let $\varphi\in\mathbf{FOR}$ and σ be an $\mathbf{L}$-projective substitution for φ. For all $\psi\in\mathbf{FOR}_{\mathtt{var}(\varphi)}$, $\varphi\vdash_{\mathbf{L}}\psi\leftrightarrow\sigma(\psi)$.*

Proposition 16. *Let $\varphi\in\mathbf{FOR}$ and σ be an $\mathbf{L}$-projective substitution for φ. For all $\mathbf{L}$-projective substitutions τ for φ, $\sigma\circ\tau$ is $\mathbf{L}$-projective for φ.*

[9] Obviously, for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, $\Sigma_{\mathbf{L}}(\varphi)$ is a complete set of $\mathbf{L}$-unifiers of φ.

[10] Obviously, for all complete sets Σ of $\mathbf{L}$-unifiers of an $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, Σ is a basis for φ if and only if Σ is a minimal complete set of $\mathbf{L}$-unifiers of φ, i.e. for all $\Delta\subseteq\Sigma$, if Δ is a complete set of $\mathbf{L}$-unifiers of φ then $\Delta=\Sigma$.

[11] Obviously, the types 1, ω, ∞ and 0 constitute a set of jointly exhaustive and pairwise distinct situations for each $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$.

[12] That is to say, the types 1, ω, ∞ and 0 being ordered by $1<\omega<\infty<0$, the unification type of $\mathbf{L}$ is the greatest one among the types of its unifiable formulas.

Proposition 17. *Let $\varphi\in\mathbf{FOR}$ and σ be an $\mathbf{L}$-projective substitution for φ. For all $\mathbf{L}$-unifiers τ of φ, $\tau\preccurlyeq_{\mathbf{L}}\sigma$.*

For all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, we say that φ is $\mathbf{L}$-*projective* if there exists an $\mathbf{L}$-projective $\mathbf{L}$-unifier of φ.

Proposition 18. *Let $\varphi\in\mathbf{FOR}$ be $\mathbf{L}$-unifiable. If φ is $\mathbf{L}$-projective then φ is of type 1.*

We say that $\mathbf{L}$ has *projective unification* if for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, φ is $\mathbf{L}$-projective.

Proposition 19. *If $\mathbf{L}$ has projective unification then $\mathbf{L}$ is of type 1.*

The following result is essential for the proof of Proposition 23.

Proposition 20. *If $\mathbf{L}$ contains $\mathbf{K}5$ then for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, φ is $\mathbf{L}$-projective if and only if φ has extension property in $\mathbf{L}$.*

The following result is essential for the proof of Proposition 25.

Proposition 21. *If $\mathbf{L}$ contains $\mathbf{K}5$ and $\mathbf{L}$ is global then for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$ and for all $\mathbf{L}$-unifiers σ of φ, there exists $\psi\in\mathbf{FOR}_{\mathtt{var}(\varphi)}$ such that $\sigma(\psi)\in\mathbf{L}$, $\psi\rightarrow\varphi\in\mathbf{K}$, ψ is $\mathbf{L}$-projective.*

4 Extensions of $\mathbf{K}5$

Firstly, let us consider the extensions of $\mathbf{K}45$.

Proposition 22. *If $\mathbf{L}$ contains $\mathbf{K}45$ then for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, φ is $\mathbf{L}$-projective.*

Proposition 23. *If $\mathbf{L}$ contains $\mathbf{K}45$ then $\mathbf{L}$ has projective unification.*

Secondly, let us consider the extensions of $\mathbf{K}5$.

Proposition 24. *If $\mathbf{L}$ contains $\mathbf{K}5$ and $\mathbf{L}$ is global then for all $\mathbf{L}$-unifiable $\varphi\in\mathbf{FOR}$, φ is of type 1.*

Proposition 25. *If $\mathbf{L}$ contains $\mathbf{K}5$ and $\mathbf{L}$ is global then $\mathbf{L}$ is of type 1.*

Notice that the line of reasoning leading to Propositions 23 and 25 rules out neither the possibility that all extensions of $\mathbf{K}5$ have projective unification, nor the possibility that some nonglobal extension of $\mathbf{K}5$ is either of type ω, or of type ∞, or of type 0[13].

5 Conclusion

A property similar to the extension property has been used by Ghilardi who has proved both in Intuitionistic Logic [19] and in transitive modal logics like $\mathbf{K}4$ and $\mathbf{S}4$ [20] that it is equivalent to the projectivity of formulas. This property has also been considered in [12] where formulas verifying it are called *extendible formulas*. As a matter of fact, Bezhanishvili and de Jongh have provided a complete characterization in Intuitionistic Logic of the set of all extendible formulas with at most 2 variables. However, the question remains unsettled whether a complete characterization in Intuitionistic Logic of the set of all extendible formulas with at least 3 variables can be given. Within the context of extensions of $\mathbf{K}5$, we believe that it is probably easier to give a complete characterization of the set of all formulas verifying the extension property.

[13] No modal logic is known to be of type ∞ [16, Chapter 5].

Funding

The preparation of this paper has been supported by *French Minister of Europe and Foreign Affairs, French Minister of Higher Education, Research and Innovation, Center for International Scientific Studies and Collaboration* of *Iranian Minister of Research, Science and Technology* and *Embassy of France in Tehran* (Programme Gundishapur, project *40903VF*).

Acknowledgement

Special acknowledgement is heartily granted to Çiğdem Gencer (Istanbul Aydın University, Istanbul, Turkey), Maryam Rostamigiv (Toulouse Institute of Computer Science Research, Toulouse, France) and Tinko Tinchev (Sofia University St. Kliment Ohridski, Sofia, Bulgaria) for many stimulating discussions about modal logics and the unification problem.

References

[1] ALIZADEH, M., M. ARDESHIR, P. BALBIANI, and M. MOJTAHEDI, 'About the unification type of modal logic $\mathbf{K}5$ and its extensions', HAL-CNRS (2021) hal-03252589.

[2] BAADER, F., and S. GHILARDI, 'Unification in modal and description logics', *Logic Journal of the IGPL* **19** (2011) 705–730.

[3] BAADER, F., and W. SNYDER, 'Unification theory', In: *Handbook of Automated Reasoning,* Elsevier (2001) 439–526.

[4] BABENYSHEV, S., and V. RYBAKOV, 'Unification in linear temporal logic $\mathbf{LTL}$', *Annals of Pure and Applied Logic* **162** (2011) 991–1000.

[5] BALBIANI, P., 'Remarks about the unification type of several non-symmetric non-transitive modal logics', *Logic Journal of the IGPL* **27** (2019) 639–658.

[6] BALBIANI, P., and Ç. GENCER, '$\mathbf{KD}$ is nullary', *Journal of Applied Non-Classical Logics* **27** (2017) 196–205.

[7] BALBIANI, P., and Ç. GENCER, 'Unification in epistemic logics', *Journal of Applied Non-Classical Logics* **27** (2017) 91–105.

[8] BALBIANI, P., and Ç. GENCER, 'About the unification type of modal logics between $\mathbf{KB}$ and $\mathbf{KTB}$', *Studia Logica* **108** (2020) 941–966.

[9] BALBIANI, P., Ç. GENCER, M. ROSTAMIGIV, and T. TINCHEV, 'About the unification type of $\mathbf{K} + \Box\Box\bot$', (submitted for publication).

[10] BALBIANI, P., and T. TINCHEV, 'Unification in modal logic $\mathbf{Alt}_1$', In: *Advances in Modal Logic,* College Publications (2016) 117–134.

[11] BALBIANI, P., and T. TINCHEV, 'Elementary unification in modal logic $\mathbf{KD}45$', *Journal of Applied Logics* **5** (2018) 301–317.

[12] BEZHANISHVILI, N., and D. DE JONGH, 'Extendible formulas in two variables in intuitionistic logic', *Studia Logica* **100** (2012) 61–89.

[13] BLACKBURN, P., M. DE RIJKE, and Y. VENEMA, *Modal Logic,* Cambridge University Press (2001).

[14] CHAGROV, A., and M. ZAKHARYASCHEV, *Modal Logic,* Oxford University Press (1997).

[15] VAN DITMARSCH, H., W. VAN DER HOEK, and B. KOOI, *Dynamic Epistemic Logic,* Springer (2008).

[16] DZIK, W., *Unification Types in Logic,* Wydawnicto Uniwersytetu Slaskiego (2007).

[17] DZIK, W., and P. WOJTYLAK, 'Projective unification in modal logic', *Logic Journal of the IGPL* **20** (2012) 121–153.

[18] FAGIN, R., J. HALPERN, Y. MOSES, and M. VARDI, *Reasoning About Knowledge,* MIT Press (1995).

[19] GHILARDI, S., 'Unification in intuitionistic logic', *Journal of Symbolic Logic* **64** (1999) 859–880.

[20] GHILARDI, S., 'Best solving modal equations', *Annals of Pure and Applied Logic* **102** (2000) 183–198.

[21] GHILARDI, S., and L. SACCHETTI, 'Filtering unification and most general unifiers in modal logic', *Journal of Symbolic Logic* **69** (2004) 879–906.

[22] HALPERN, J., and L. RÊGO, 'Characterizing the $\mathbf{NP}$-$\mathbf{PSPACE}$ gap in the satisfiability problem for modal logic', *Journal of Logic and Computation* **17** (2007) 795–806.

[23] HUGHES, G., and M. CRESSWELL, *An Introduction to Modal Logic,* Methuen (1968).

[24] IEMHOFF, R., 'A syntactic approach to unification in transitive reflexive modal logics', *Notre Dame Journal of Formal Logic* **57** (2016) 233–247.

[25] JEŘÁBEK, E., 'Logics with directed unification', In: *Algebra and Coalgebra meet Proof Theory,* Utrecht, Netherlands (2013).

[26] JEŘÁBEK, E., 'Blending margins: the modal logic $\mathbf{K}$ has nullary unification type', *Journal of Logic and Computation* **25** (2015) 1231–1240.

[27] KOST, S., 'Projective unification in transitive modal logics', *Logic Journal of the IGPL* **26** (2018) 548–566.

[28] KRACHT, M., *Tools and Techniques in Modal Logic,* Elsevier (1999).

[29] NAGLE, M., 'The decidability of normal $\mathbf{K}5$ logics', *Journal of Symbolic Logic* **46** (1981) 319–328.

[30] NAGLE, M., and S. THOMASON, 'The extensions of the modal logic $\mathbf{K}5$', *Journal of Symbolic Logic* **50** (1985) 102–109.

[31] ROSTAMIGIV, M., 'About the Type of Modal Logics for the Unification Problem', *Doctoral Thesis,* Toulouse University (2020).

[32] RYBAKOV, V., 'A criterion for admissibility of rules in the model system $S4$ and the intuitionistic logic', *Algebra and Logic* **23** (1984) 369–384.

[33] RYBAKOV, V., *Admissibility of Logical Inference Rules,* Elsevier (1997).

Restricted Unification in the Description Logic $\mathcal{FL}_0$

Franz Baader[1], Oliver Fernández Gil[1], and Maryam Rostamigiv[2]

[1] Theoretical Computer Science, TU Dresden, Dresden, Germany
franz.baader@tu-dresden.de, oliver.fernandez@tu-dresden.de
[2] Département d'Informatique, Paul Sabatier University, Toulouse, France
Maryam.Rostamigiv@irit.fr

Abstract

Unification in the Description Logic (DL) $\mathcal{FL}_0$ is known to be ExpTime-complete, and of unification type zero. We investigate in this paper whether a lower complexity of the unification problem can be achieved by either syntactically restricting the role depth of concepts or semantically restricting the length of role paths in interpretations. We will show that the answer to this question depends on whether the number formulating such a restriction is encoded in unary or binary. We will also show that the unification type of $\mathcal{FL}_0$ improves from type zero to unitary (finitary) for unification without (with) constants.

1 Introduction

Unification of concept patterns has been proposed as an inference service in Description Logics that can, for example, be used to detect redundancies in ontologies. For the DL $\mathcal{FL}_0$, which has the concept constructors conjunction ($\sqcap$), value restriction ($\forall r.C$), and top concept ($\top$), unification was investigated in detail in [4]. It was shown there that unification in $\mathcal{FL}_0$ corresponds to unification modulo the equational theory *ACUIh* since (modulo equivalence) conjunction is associative (A), commutative (C), idempotent (I) and has top as a unit (U), and value restrictions behave like homomorphisms for conjunction and top (h). For this equational theory, it had already been shown in [1] that it has unification type zero, which means that a solvable unification problem need not have a minimal complete set of unifiers, and thus in particular not a finite one. From the DL point of view, the decision problem is, however, more interesting than the unification type. Since *ACUIh* is a commutative/monoidal theory [1, 12], solvability of *ACUIh* unification problems (and thus of unification problems in $\mathcal{FL}_0$) can be reduced to solvability of systems of linear equations in a certain semiring, which for the case of *ACUIh* consists of finite languages over a finite alphabet, with union as semiring addition and concatenation as semiring multiplication [4]. By a reduction to the emptiness problem for certain tree automata, it was then shown in [4] that solvability of the linear equations corresponding to an $\mathcal{FL}_0$ unification problem can be decided in exponential time. In addition, by a reduction from the intersection emptiness problem for deterministic root-to-frontier tree automata [14], ExpTime-hardness of this problem was proved in [4].

In the present paper, we investigate two kinds of restrictions on unification in $\mathcal{FL}_0$. On the one hand, we *syntactically restrict the role depth* (i.e., the maximal nesting of value restrictions) in the concepts obtained by applying a unifier to be below a certain bound k. This restriction was motivated by a similar restriction used in research on least common subsumers (lcs) [13], where imposing a bound on the role depth guarantees existence of the lcs also in the presence of a (possibly cyclic) terminology. Also note that such a restriction was used in [9] for the theory *ACh*, for which unification is known to be undecidable [11]. It is shown in [9] that the problem becomes decidable if a bound on the maximal nesting of applications of homomorphisms is imposed. On the other hand, we consider a *semantic restriction* where, when defining the semantics of concepts, only interpretations for which the length of role paths is bounded by a

given number k are considered. A similar restriction (for $k = 1$) was employed in [7] to improve the unification type from type zero for the modal logic **K** [8] to unitary or finitary for $\mathbf{K}+\Box\Box\bot$.

In the present paper we show that both the syntactic and the semantic restriction ensures that the unification type of $\mathcal{FL}_0$ (and equivalently, of the theory *ACUIh*) improves from type zero to unitary for unification without constants and finitary for unification with constants. Regarding the decision problem, we can show that the complexity depends on whether the bound k is assumed to be encoded in unary or binary.[1] For binary encoding of k, the complexity stays ExpTime, whereas for unary coding it drops from ExpTime to PSpace. This is again the case both for the syntactic and the semantic restriction. Detailed proofs of these result can be found in [3]. A longer version of this abstract is accepted by the conference FroCoS 2021 [2].

2 Unification in $\mathcal{FL}_0$

Starting with mutually disjoint countably infinite sets N_C and N_R of concept and role names, respectively, the set of $\mathcal{FL}_0$ concepts is inductively defined as follows:

- $\top$ (top concept) and every concept name $A \in N_C$ is an $\mathcal{FL}_0$ concept,
- if C, D are $\mathcal{FL}_0$ concepts and $r \in N_R$ is a role name, then $C \sqcap D$ (conjunction) and $\forall r.C$ (value restriction) are $\mathcal{FL}_0$ concepts.

The *semantics* of $\mathcal{FL}_0$ concepts is defined using first-order interpretations $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consisting of a non-empty domain $\Delta^\mathcal{I}$ and an interpretation function $\cdot^\mathcal{I}$ that assigns a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ to each concept name A, and a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ to each role name r. This function is extended to $\mathcal{FL}_0$ concepts as follows:

$$\top^\mathcal{I} = \Delta^\mathcal{I},\ \ (C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I},\ \ (\forall r.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \forall y \in \Delta^\mathcal{I}\colon (x, y) \in r^\mathcal{I} \Rightarrow y \in C^\mathcal{I}\}.$$

Given two $\mathcal{FL}_0$ concepts C and D, we say that C is subsumed by D (written $C \sqsubseteq D$) if $C^\mathcal{I} \subseteq D^\mathcal{I}$ holds for all interpretations $\mathcal{I}$, and that C is equivalent to D (written $C \equiv D$) if $C \sqsubseteq D$ and $D \sqsubseteq C$. It is well known that subsumption (and thus also equivalence) of $\mathcal{FL}_0$ concepts can be decided in polynomial time [10].

In unification, we consider concepts that may contain variables, which can be replaced by concepts. More formally, we introduce a countably infinite set N_V of concept variables, which is disjoint with N_C and N_R. An $\mathcal{FL}_0$ concept pattern is an $\mathcal{FL}_0$ concept that is constructed using $N_C \cup N_V$ as concept names. The semantics of such concept patterns is defined as for concepts, i.e., concept variables are treated like concept names when defining the semantics. This way, the notions of subsumption and equivalence (both in the restricted and in the unrestricted setting) transfer from concepts to concept patterns in the obvious way.

A substitution σ is a mapping from N_X into the set of all $\mathcal{FL}_0$ concept patterns such that $dom(\sigma) := \{X \in N_V \mid \sigma(X) \neq X\}$ is finite. This mapping is extended to concept patterns as follows:

- $\sigma(A) := A$ for all $A \in N_C \cup \{\top\}$,
- $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$ and $\sigma(\forall R.C) := \forall R.\sigma(C)$.

An $\mathcal{FL}_0$ unification problem is an equation of the form $C \stackrel{?}{\equiv} D$ where C, D are $\mathcal{FL}_0$ concept patterns. A unifier of this equation is a substitution σ such that $\sigma(C) \equiv \sigma(D)$.

[1] For unary coding, the size of the input k is the k, whereas for binary coding it is $\log k$.

It was shown in [4] that the question of whether a given $\mathcal{FL}_0$ unification problem has a unifier or not can be reduced to solving linear language equations, i.e., equations of the form

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

where $S_0, \ldots, S_n, T_0, \ldots, T_n$ are finite languages of words over the alphabet Δ of all role names and $X_1, \ldots, X_n$ are variables for finite languages over Δ. A solution of the equation (1) is an assignment θ of finite languages $\theta(X_i) \subseteq \Delta^*$ to the variables X_i (for $i = 1, \ldots, n$) such that

$$S_0 \cup S_1{\cdot}\theta(X_1) \cup \cdots \cup S_n{\cdot}\theta(X_n) = T_0 \cup T_1{\cdot}\theta(X_1) \cup \cdots \cup T_n{\cdot}\theta(X_n), \tag{2}$$

where $\cup$ is interpreted as union and $\cdot$ as concatenation. Note that a word $w = r_1 \ldots r_\ell$ in such a solution corresponds to a conjunct $\forall r_1. \cdots \forall r_\ell.A$ in the unified concept $\sigma(C) \equiv \sigma(D)$.

It was shown in [4] that checking solvability of linear language equations can be reduced to testing emptiness of tree automata. More precisely, the tree automata employed in [4] work on finite node-labelled trees, going from the root to the leaves. Such automata are called root-to-frontier tree automata (RFAs) in [4]. Basically, given a linear language equation, one can construct an RFA whose size is exponential in the size of the language equation, and which accepts some tree iff the language equation has a solution. Since the emptiness problem for RFAs is polynomial, this yields an ExpTime upper bound for solvability of linear language equations. The matching ExpTime lower bound was proved in [4] by a reduction from the intersection emptiness problem for deterministic RFAs (DRFAs).

3 Syntactically Restricted Unification in $\mathcal{FL}_0$

The role depth of an $\mathcal{FL}_0$ concept is the maximal nesting of value restrictions in this concept. To be more precise, we define the role depth $rd(C)$ of an $\mathcal{FL}_0$ concept C by induction:

- $rd(\top) = rd(A) = 0$ for all $A \in N_C$,
- $rd(C \sqcap D) = \max(rd(C), rd(D))$ and $rd(\forall r.C) = 1 + rd(C)$.

For an integer $k \geq 1$ and $\mathcal{FL}_0$ concepts C and D (equal $\top$ or not containing any occurrences of $\top$), we define subsumption and equivalence restricted to concepts of role depth $\leq k$ as follows:

- $C \sqsubseteq^k_{syn} D$ if $C \sqsubseteq D$ and $\max(rd(C), rd(D)) \leq k$,
- $C \equiv^k_{syn} D$ if $C \sqsubseteq^k_{syn} D$ and $D \sqsubseteq^k_{syn} C$.

The effect of this definition is that subsumption and equivalence can only hold for concepts that satisfy the restriction of the role depth by k. For concepts satisfying this syntactic restriction, the relations $\sqsubseteq^k_{syn}$ and $\equiv^k_{syn}$ coincide with the classical subsumption and equivalence relation on $\mathcal{FL}_0$ concepts.

For an integer $k \geq 1$, a *syntactically k-restricted unification problem* is an equation of the form $C \overset{?}{\equiv}{}^k_{syn} D$, where C, D are $\mathcal{FL}_0$ concept patterns (equal $\top$ or not containing any occurrences of $\top$). A unifier of this equation is a substitution σ such that $\sigma(C) \equiv^k_{syn} \sigma(D)$.

Due to the definition of $\equiv^k_{syn}$ and the correspondence between words in a solution of (1) and nested value restrictions in the unified concept, the question of whether a given syntactically k-restricted unification problem has a unifier or not can be reduced to checking whether language equations of the form (1) have solutions θ such that

$$S_0 \cup S_1{\cdot}\theta(X_1) \cup \cdots \cup S_n{\cdot}\theta(X_n) \;=\; T_0 \cup T_1{\cdot}\theta(X_1) \cup \cdots \cup T_n{\cdot}\theta(X_n) \;\subseteq\; \Delta^{\leq k}, \tag{3}$$

where $\Delta^{\leq k}$ denotes the set of words over Δ of length at most k.

We can test for the existence of such a solution by modifying the automata construction in [4] for testing existence of a solution in the unrestricted case. The depth of the trees accepted by the constructed automaton $\mathcal{A}$ corresponds to the length of the longest word in the solved equation. Thus, to find a solution satisfying (3), we must check whether $\mathcal{A}$ accepts a tree of depth $\leq k$. We can achieve this restriction by adding a counter to the states of the automaton that is decremented whenever we go from a node in the tree to a successor node. As soon as the counter reaches 0, no more transitions are possible. This idea can be used to show the complexity upper bounds in the following theorem.

Theorem 1. *Given an integer $k \geq 1$ and $\mathcal{FL}_0$ concepts C, D as input, the problem of deciding whether the syntactically k-restricted unification problem $C \overset{?}{\equiv}{}^{k}_{syn} D$ has a unifier or not is ExpTime-complete if the number k is assumed to be encoded in binary, and PSpace-complete if k is assumed to be encoded in unary.*

Regarding the PSpace upper bound, one cannot construct the exponentially large modified automaton $\mathcal{A}^k_{syn}$ before testing it for emptiness, but rather constructs the relevant parts of $\mathcal{A}^k_{syn}$ on-the-fly while doing the emptiness test. The ExpTime lower bound is an easy consequence of the fact that the automaton $\mathcal{A}$ accepts a tree iff it accepts one of depth linear in the size of $\mathcal{A}$ (which is exponential in the size of the input equation). Showing the PSpace lower bound is more challenging. For this, we had to prove the following k-restricted variant of Seidl's ExpTime hardness result [14] for the intersection emptiness problem for deterministic RFAs (DRFAs). The *k-restricted intersection emptiness problem for DRFAs* asks whether a given finite collection of DRFAs accepts a common tree of depth at most k.

Proposition 2. *The k-restricted intersection emptiness problem for DRFAs is PSpace-complete if the number k is represented in unary.*

4 Semantically Restricted Unification in $\mathcal{FL}_0$

For an integer $n \geq 1$ and a given interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, a role path of length n is a sequence $d_0, r_1, d_1, \ldots, d_{n-1}, r_n, d_n$, where $d_0, \ldots, d_n$ are elements of $\Delta^{\mathcal{I}}$, $r_1, \ldots, r_n$ are role names, and $(d_{i-1}, d_i) \in r_i^{\mathcal{I}}$ holds for all $i = 1, \ldots, n$. The interpretation $\mathcal{I}$ is called k-restricted if it does not admit any role paths of length $> k$.

For an integer $k \geq 1$ and $\mathcal{FL}_0$ concepts C and D, we define subsumption and equivalence restricted to interpretations with role paths of length $\leq k$ as follows:

- $C \sqsubseteq^k_{sem} D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all k-restricted interpretations $\mathcal{I}$,
- $C \equiv^k_{sem} D$ if $C \sqsubseteq^k_{sem} D$ and $D \sqsubseteq^k_{sem} C$.

The effect of this notion of equivalence is that all concepts occurring at a role depth $> k$ can be replaced by $\top$, and thus can be removed.

For an integer $k \geq 1$, a *semantically k-restricted unification problem* is an equation of the form $C \overset{?}{\equiv}{}^{k}_{sem} D$, where C, D are $\mathcal{FL}_0$ concept patterns. A unifier of this equation is a substitution σ such that $\sigma(C) \equiv^k_{sem} \sigma(D)$.

Whereas in the syntactically restricted case a sequence of value restrictions of depth $> k$ (a word of length $> k$) destroys the property of being a unifier (solution), in the semantically restricted case one can simply ignore such sequences (words). Thus, one can reduce the question

of whether a given semantically k-restricted unification problem has a unifier or not to checking whether, for language equations of the form (1), there is an assignment θ such that

$$(S_0 \cup S_1{\cdot}\theta(X_1) \cup \cdots \cup S_n{\cdot}\theta(X_n)) \cap \Delta^{\leq k} \quad = \quad (T_0 \cup T_1{\cdot}\theta(X_1) \cup \cdots \cup T_n{\cdot}\theta(X_n)) \cap \Delta^{\leq k}. \quad (4)$$

Note that, in general, such an assignment need not be a solution of (1), but clearly any solution θ of (1) also satisfies (4).

To check for the existence of a solution of (1) satisfying (4), we again add a counter to the states of the automaton $\mathcal{A}$, but now accept as soon as the counter goes below the value 0. This yields the complexity upper bounds in the following theorem. The PSpace lower bound for the unary case can be shown by a reduction from syntactically k-restricted unification. If k is encoded in binary, then this reduction is no longer polynomial. It is an open problem whether, for the case of binary coding, the ExpTime upper bound stated below is tight.

Theorem 3. *Given an integer $k \geq 1$ and $\mathcal{FL}_0$ concepts C, D as input, the problem of deciding whether the semantically k-restricted unification problem $C \overset{?}{\equiv}{}^{k}_{sem} D$ has a unifier or not is in ExpTime if the number k is assumed to be encoded in binary, and PSpace-complete if k is assumed to be encoded in unary.*

5 The Unification Type

Until now, we were mainly interested in the complexity of deciding solvability of unification problems. Now, we want to investigate the question of whether all unifiers of a given unification problem can be represented as instances of a finite set of unifiers, where the instance relation between unifiers is defined in the usual way [6, 3].

A set of unifiers M of an $\mathcal{FL}_0$ unification problem $C \overset{?}{\equiv} D$ is *complete* if any unifier of $C \overset{?}{\equiv} D$ is an instance of some element of M. This set is *minimal* if no two distinct elements of M are comparable w.r.t. the instance relation. The unification problem $C \overset{?}{\equiv} D$ has *type zero* if it does not have a minimal complete set of unifiers. Note that this implies that $C \overset{?}{\equiv} D$ does not have a finite complete set of unifiers since such a set could be made minimal by removing unifiers that are instances of others [6]. Saying that $\mathcal{FL}_0$ has *unification type zero* means that there is an $\mathcal{FL}_0$ unification problem that has type zero.

The fact that $\mathcal{FL}_0$ has unification type zero follows from a result in [1], which states that the equational theory $ACUIh$ has unification type zero. In fact, it was shown in [4] that equivalence $\equiv$ of $\mathcal{FL}_0$ concepts can be axiomatized by the equational theory

$$\begin{aligned} ACUIh \quad := \quad & \{\ (x \wedge y) \wedge z = x \wedge (y \wedge z),\ x \wedge y = y \wedge x,\ x \wedge x = x,\ x \wedge 1 = x\ \} \\ & \cup \{\ h_r(x \wedge y) = h_r(x) \wedge h_r(y),\ h_r(1) = 1 \mid r \in N_R\ \}, \end{aligned}$$

where $\wedge$, h_r, and 1 in the terms respectively correspond to $\sqcap$, $\forall r.$, and $\top$ in the concepts. These identities say that $\wedge$ is associative (A), commutative (C), and idempotent (I) with unit 1 (U), and that the unary function symbols h_r behave like homomorphisms (h) for $\wedge$ and 1.

For our restricted settings, the unification type of $\mathcal{FL}_0$ is *finitary* for unification with constants and *unitary* for unification without constants. The former statement means that any solvable (semantically or syntactically) k-restricted unification problem has a finite complete set of unifiers. The latter statement means that any solvable (semantically or syntactically) k-restricted unification problem not containing concept constants from N_C has a complete set of unifiers of cardinality 1.

For the *semantically restricted case*, it is easy to see that the equivalence $\equiv^k_{sem}$ can be axiomatized by adding identities to *ACUIh* that say that nesting of homomorphisms of depth $> k$ produces the unit. Given a word $u = r_1 r_2 \ldots r_n \in N_R^*$, we denote a term of the form $h_{r_1}(h_{r_2}(\cdots h_{r_n}(t)\cdots))$ as $h_u(t)$. It is now easy to see that $\equiv^k_{sem}$ is axiomatized by

$$ACUIh^k := ACUIh \cup \{h_u(x) = 1 \mid u \in N_R^* \text{ with } |u| = k+1\}.$$

Both *ACUIh* and $ACUIh^k$ are so-called commutative/monoidal theory [1, 12, 5], for which unification can be reduced to solving linear equations over a corresponding semiring. For *ACUIh* this semiring consists of finite languages over the alphabet Δ of all role names with union as addition and concatenation as multiplication [4]. As shown in [3], the semiring corresponding to $ACUIh^k$ consists of the subsets of $\Delta^{\leq k}$, with union as addition and the following multiplication: $L_1 \cdot_k L_2 = (L_1 \cdot L_2) \cap \Delta^{\leq k}$.

We can now apply general results for commutative/monoidal theories [1, 12, 5] to determine the unification type of $\mathcal{FL}_0$ in the semantically restricted case. These results imply that the unification type of a commutative/monoidal theory is unitary (finitary) for unification without (with) constants if the corresponding semiring is finite. Since the semiring corresponding to $ACUIh^k$ consists of the subsets of the finite set $\Delta^{\leq k}$, it is clearly finite, which yields the following theorem.

Theorem 4. *Unification in $ACUIh^k$, and thus also semantically k-restricted unification in $\mathcal{FL}_0$, is unitary for unification without constants and finitary for unification with constants.*

For the *syntactically restricted case*, the results for commutative/monoidal theories do not apply directly, but we can show the same results as for the semantically restricted case, using the ideas underlying the proofs in [1, 12].

Theorem 5. *Syntactically k-restricted unification in $\mathcal{FL}_0$ is unitary for unification without constants and finitary for unification with constants.*

The first step towards showing these results is to restrict the number of variables that need to occur in elements of a complete set of unifiers. To be more precise, let c denote the (finite) cardinality of $\Delta^{\leq k}$. Then we can show that, if a syntactically k-restricted unifier of $C \overset{?}{\equiv}{}^k_{syn} D$ introduces more than $2^{c \cdot n}$ variables, it is an instance of a syntactically k-restricted unifier that introduces at least one variable less. Thus, there always is a complete set of unifiers that contains only unifiers introducing at most $2^{c \cdot n}$ different variables. Once we have restricted the unifiers in the complete set to ones using only finitely many variables, it is easy to show that, up to equivalence, there can be only finitely many unifiers in this set. This proves that syntactically k-restricted unification in $\mathcal{FL}_0$ is finitary for unification with constants.

For unification without constants, one can combine the unifiers in a finite complete set $\{\sigma_1, \ldots, \sigma_\kappa\}$ of syntactically k-restricted unifiers of $C \overset{?}{\equiv}{}^k_{syn} D$ into a single substitution σ by setting

$$\sigma(X) = \sigma_1(X) \sqcap \ldots \sqcap \sigma_\kappa(X) \quad \text{for all variables } X \text{ occurring in } C, D.$$

If we assume (without loss of generality) that the variables occurring in the ranges of the unifiers σ_i are disjoint and that no concept constant occurs in the range, then it is easy to show that σ is also a syntactically k-restricted unifier of $C \overset{?}{\equiv}{}^k_{syn} D$, and has the substitutions $\sigma_1, \ldots, \sigma_\kappa$ as instances. Consequently, the singleton set $\{\sigma\}$ is a complete set of unifiers of $C \overset{?}{\equiv}{}^k_{syn} D$.

References

[1] F. Baader. Unification in commutative theories. *J. Symbolic Computation*, 8(5):479–497, 1989.

[2] F. Baader, O. Fernández Gil, and M. Rostamigiv. Restricted unification in the description logic $\mathcal{FL}_0$. In B. Konev and G. Reger, editors, *Proc. of the 13th International Symposium on Frontiers of Combining Systems (FroCoS 2021)*, Lecture Notes in Computer Science. Springer, 2021. To appear.

[3] F. Baader, O. Fernández Gil, and M. Rostamigiv. Restricted unification in the DL $\mathcal{FL}_0$ (extended version). LTCS-Report 21-02, Chair of Automata Theory, Institute of Theoretical Computer Science, Technische Universität Dresden, Dresden, Germany, 2021. https://lat.inf.tu-dresden.de/research/reports/2021/BaGiRo21.pdf.

[4] F. Baader and P. Narendran. Unification of concept terms in description logics. *J. Symbolic Computation*, 31(3):277–305, 2001.

[5] F. Baader and W. Nutt. Combination problems for commutative/monoidal theories: How algebra can help in equational reasoning. *J. Applicable Algebra in Engineering, Communication and Computing*, 7(4):309–337, 1996.

[6] F. Baader and W. Snyder. Unification theory. In J. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 447–533. Elsevier Science Publishers, 2001.

[7] P. Balbiani, C. Gencer, M. Rostamigiv, and T. Tinchev. About the unification type of $\mathbf{K}+\Box\Box\bot$. In *Proc. of the 34th International Workshop on Unification (UNIF 2020)*, pages 4:1–4:6. RISC-Linz, 2020.

[8] E. Jerabek. Blending margins: The modal logic K has nullary unification type. *J. Logic and Computation*, 25(5):1231–1240, 2015.

[9] A. Kumar Eeralla and C. Lynch. Bounded ACh unification. *Math. Struct. Comput. Sci.*, 30(6):664–682, 2020.

[10] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[11] P. Narendran. Solving linear equations over polynomial semirings. In *Proc. of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 466–472. IEEE Computer Society, 1996.

[12] W. Nutt. Unification in monoidal theories. In M. E. Stickel, editor, *Proc. of the 10th Int. Conf. on Automated Deduction (CADE 1990)*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 618–632, Kaiserslautern, Germany, 1990. Springer.

[13] R. Peñaloza and A. Turhan. A practical approach for computing generalization inferences in $\mathcal{EL}$. In G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, editors, *Proc. of the 8th Extended Semantic Web Conference (ESWC 2011)*, volume 6643 of *Lecture Notes in Computer Science*, pages 410–423. Springer, 2011.

[14] H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52:57–60, 1994.

Nominal Disunification via Fixed-Point Constraints (Work in Progress)

Leonardo M. Batista[2,*], Maribel Fernández[1], Daniele Nantes-Sobrinho[2], and Deivid Vale[3]

[1] King's College London, London, UK
maribel.fernandez@kcl.ac.uk
[2] Universidade de Brasília, Brasília, Brazil
L.M.Batista@mat.unb.br, dnantes@unb.br
[3] Radboud University Nijmegen, Nijmegen, The Netherlands
deividvale@cs.ru.nl

Abstract

This is a work in progress on solving equations and disequations between nominal terms, i.e, we are interested in the *nominal disunification problem*. In the standard nominal syntax, α-equality ($s \approx_\alpha t$) between two nominal terms s and t is defined using a freshness predicate ($a\#t$) meaning "atom a is fresh for t". Recently, an alternative syntax was proposed using fixed-point constraints instead of freshness constraints. Using fixed-point constraints, nominal commutative (C) unification is finitary whilst it is not finitary if freshness constraints are used to represent solutions. With the (future) goal of investigating nominal disunification problems modulo equational theories, whose solvability via freshness constraints may be problematic, we exploit this fixed-point approach to solve nominal disunification problems: we provide an algorithm to compute finite and complete sets of solutions for nominal disunification problems consisting of equations, disequations and fixed-point constraints. This is a first step towards solving nominal C-disunification problems.

1 Introduction

This paper is about solving equations ($s \approx_\alpha^? t$) and disequations ($s \not\approx_\alpha^? t$) between nominal terms, that is, it concerns the *nominal disunification problem* [3], which has the form $\langle s_i \approx_\alpha^? t_i \ (1 \leq i \leq n), u_j \not\approx_\alpha^? v_j \ (1 \leq j \leq m)\rangle$.

Nominal techniques are useful for the treatment of languages involving binders [6]. In this approach, bindings are implemented through the abstraction of atoms, and atom permutations are used to implement renamings. For this, freshness constraints (which have the form $a\#t$) and α-equivalence constraints (which have the form $s \approx_\alpha t$) are considered. Intuitively, $a\#t$ means that the atom a cannot occur free in the term t. This concept was formalised in [7] using the quantifier *new* (И) which, in nominal logic, quantifies over new names. Such formalisation is expressed by the following sentence: $a\#x \Leftrightarrow (\text{И}a')(a\ a') \cdot x = x$, that is, a is fresh in x if, and only if, for any new atom a', the permutation $(a\ a')$ fixes x. For example, consider the formula $\phi = \forall[a]P$. In this case, a is an abstracted name, therefore $a\#\phi$, which is equivalent to saying that the renaming of a by a new name a' still preserves ϕ, that is, $(\text{И}a')(a\ a') \cdot \phi \approx_\alpha \phi$.

This observation lead to a new axiomatisation of α-equivalence of nominal terms using fixed-point constraints instead of freshness constraints [2]. Fixed-point constraints have the form $\pi \cdot t \curlywedge_\alpha t$ (read "the permutation π fixes the term t"). For nominal unification problems,

*Author partially funded by Capes and CNPq.

there is a direct correspondence between solutions expressed using freshness constraints and solutions expressed using fixed-point constraints, but in the presence of commutative theories the method via fixed point stands out.

Let $\approx_{\alpha,\mathtt{C}}$ denote α-equivalence modulo commutativity. Using freshness constraints, the equation $(a\ b)\cdot X \approx^?_{\alpha,\mathtt{C}} X$ has a unifier $\langle a, b\#X, Id\rangle$ [9], but this is not the only solution. Indeed there are infinite solutions $\{X \mapsto a+b\}, \{X \mapsto (a+b)+(a+b)\}, \{X \mapsto f(a+b)\}, \ldots$ While nominal unification is finitary [9], when equational theories are involved, this property is lost if solutions are represented using freshness constraints, as shown in [1]. Note that $(a\ b)\cdot(a+b) = b+a \approx_{\alpha,\mathtt{C}} a+b$, so the permutation $(a\ b)$ fixes the term $a+b$, although the atoms a and b occur free in $a+b$. With the fixed-point approach, the nominal unification algorithm (modulo commutativity) computes a finite complete set of unifiers [2]. Fixed-point equations appeared for the first time in [8] in the context of nominal unification with recursive let, but the authors used the standard notion of freshness to develop their work.

Recently, in [3], a method was proposed to decide the nominal disunification problem, which is an extension of the first-order disunification problem defined in [4], where the α-equivalence relation is built using permutations and freshness. The idea is simple: one checks if the set of equations associated to the disequations, i.e., $\langle u_j \approx^?_\alpha v_j (1 \le j \le m)\rangle$ is satisfiable, if yes (with solution set S), remove the solutions of $\langle s_i \approx^?_\alpha t_i (1 \le i \le n)\rangle$ that are instances of S. Therefore, the proposed nominal disunification algorithm relies on the existence of a finite representation of solutions for nominal unification problems. For this reason, the nominal disunification algorithm proposed in [3], which represents solutions using freshness constraints, cannot be used to solve nominal C-disunification problems.

In this work we define the *nominal disunification problem via fixed-point constraints* (Definition 3.1) and we extend to this approach several concepts necessary for the study of its decidability: a new notion of *solution* for this problem (Definition 3.4) which depends on the concept of *pair with exceptions* (Definition 3.2) as well as its consistency (Definition 3.3). We prove consistency results (Corollary 3.1) and present the algorithm (Algorithm 2) to obtain a complete set of solutions (Theorem 3.1). This is a first step towards the development of extensions of the nominal disunification problem that involve equational theories.

2 Preliminaries

We assume familiarity with nominal techniques and briefly recall basic notions for a fixed-point approach to nominal syntax. For a detailed treatment, the reader is referred to [2].

Nominal Terms. We fix countable infinite pairwise-disjoint sets of *atoms* $\mathbb{A} = \{a, b, c, \ldots\}$, *variables* $\mathbb{X} = \{X, Y, \ldots\}$ and a signature Σ, a finite set of function symbols with fixed arity. We follow Gabbay's *permutative convention*: atoms a, b range permutatively over $\mathbb{A}$. A *permutation* π is a bijection $\mathbb{A} \to \mathbb{A}$ such that $\mathtt{dom}(\pi) := \{a \in \mathbb{A} \mid \pi(a) \neq a\}$ is finite. The identity permutation is $\mathtt{id}$ and $\pi \circ \rho$ the composition of π and ρ.

Nominal terms are given by the grammar: $s, t := a \mid \pi \cdot X \mid [a]t \mid f(t_1, \ldots, t_n)$ where a is an *atom*, $\pi \cdot X$ is a moderated variable, $[a]t$ is the *abstraction* of a in the term t, and $f(t_1, \ldots, t_n)$ is a *function application* with $f : n \in \Sigma$. We abbreviate an ordered sequence $t_1, \ldots, t_n$ of terms by $\tilde{t}$. Permutation action on terms is given by: $\pi \cdot a = \pi(a)$, $\pi \cdot (\pi' \cdot X) = (\pi \circ \pi') \cdot X$, $\pi \cdot ([a]t) = [\pi(a)](\pi \cdot t)$, and $\pi \cdot f(t_1, \ldots, t_n) = f(\pi \cdot t_1, \ldots \pi \cdot t_n)$. *Substitutions* are finite mappings from variables to terms. A substitution σ is lifted to a map over terms by: $a\sigma = a$, $(\pi \cdot X)\sigma = \pi \cdot (X\sigma)$, $([a]t)\sigma = [a](t\sigma)$, and $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$. Note that $t(\sigma\gamma) = (t\sigma)\gamma$.

$$\frac{\pi(a) = a}{\Upsilon \vdash \pi \curlywedge a}\ (\curlywedge\mathbf{a}) \qquad \frac{\mathtt{dom}(\pi^{\rho^{-1}}) \subseteq \mathtt{dom}(\mathtt{perm}(\Upsilon|_X))}{\Upsilon \vdash \pi \curlywedge \rho \cdot X}\ (\curlywedge\mathbf{v}) \qquad \frac{\Upsilon \vdash \pi \curlywedge t_i}{\Upsilon \vdash \pi \curlywedge f(\tilde{t})}\ (\curlywedge\mathbf{f}) \qquad \frac{}{\Upsilon \vdash a \stackrel{\curlywedge}{\approx}_\alpha a}\ (\stackrel{\curlywedge}{\approx}_\alpha \mathbf{a})$$

$$\frac{\Upsilon, (c_1\ c_2) \curlywedge \mathtt{Var}(t) \vdash \pi \curlywedge (a\ c_1) \cdot t}{\Upsilon \vdash \pi \curlywedge [a]t}\ (\curlywedge\mathbf{ab}) \qquad \frac{\Upsilon \vdash t_i \stackrel{\curlywedge}{\approx}_\alpha t'_i}{\Upsilon \vdash f(\tilde{t}) \stackrel{\curlywedge}{\approx}_\alpha f(\tilde{t'})}\ (\stackrel{\curlywedge}{\approx}_\alpha \mathbf{f}) \qquad \frac{\Upsilon \vdash t \stackrel{\curlywedge}{\approx}_\alpha t'}{\Upsilon \vdash [a]t \stackrel{\curlywedge}{\approx}_\alpha [a]t'}\ (\stackrel{\curlywedge}{\approx}_\alpha [\mathbf{a}])$$

$$\frac{\mathtt{dom}((\pi')^{-1} \circ \pi) \subseteq \mathtt{dom}(\mathtt{perm}(\Upsilon|_X))}{\Upsilon \vdash \pi \cdot X \stackrel{\curlywedge}{\approx}_\alpha \pi' \cdot X}\ (\stackrel{\curlywedge}{\approx}_\alpha \mathbf{v}) \qquad \frac{\Upsilon \vdash s \stackrel{\curlywedge}{\approx}_\alpha (a\ b) \cdot t \quad \Upsilon, (c_1\ c_2) \curlywedge \mathtt{Var}(t) \vdash (a\ c_1) \curlywedge t}{\Upsilon \vdash [a]s \stackrel{\curlywedge}{\approx}_\alpha [b]t}\ (\stackrel{\curlywedge}{\approx}_\alpha \mathbf{ab})$$

Table 1: Fixed-point and equality rules. c_1 and c_2 are new atoms.

Term-equality via fixed-point constraints. Following the fixed-point approach [2], we axiomatise nominal α-equivalence in terms of both a fixed-point $\curlywedge$ and term-equality $\stackrel{\curlywedge}{\approx}_\alpha$ predicates. Intuitively, $\pi \curlywedge t$ means that π has no effect on t except by permuting abstracted names, while $s \stackrel{\curlywedge}{\approx}_\alpha t$ means that s and t are α-equivalent. For instance, $(a\ b) \curlywedge [a]a$ but not $(a\ b) \curlywedge f(a)$. In order to formally define judgement derivations for fixed-point $(\Upsilon \vdash \pi \curlywedge t)$ and term-equality $(\Upsilon \vdash s \stackrel{\curlywedge}{\approx}_\alpha t)$, we need to introduce some notation.

A fixed-point context Υ is a set of *primitive fixed-point constraints* of the form $\pi \curlywedge X$. Given two permutations π and ρ, the permutation $\pi^\rho = \rho \circ \pi \circ \rho^{-1}$ denotes the conjugate of π with respect to ρ. The set $\mathtt{Var}(\Upsilon)$ contains all the variables mentioned in the fixed-point context Υ and $\mathtt{perm}(\Upsilon|_X) := \{\pi \mid \pi \curlywedge X \in \Upsilon\}$ as the set of permutations of Υ with respect to $X \in \mathtt{Var}(\Upsilon)$. We write $\pi \curlywedge \mathtt{Var}(t)$ as an abbreviation for $\{\pi \curlywedge X \mid X \in \mathtt{Var}(t)\}$. Derivability for the judgements $\Upsilon \vdash \pi \curlywedge t$ and $\Upsilon \vdash s \stackrel{\curlywedge}{\approx}_\alpha t$ is therefore defined by the derivation rules in Figure 1.

Nominal Unification via Fixed-Point Constraints. A *nominal unification problem* Pr is a finite set of fixed-point $(\pi \curlywedge^? t)$ and equality $(s \stackrel{\curlywedge}{\approx}_\alpha^? t)$ constraints. A solution to this problem is a pair $\langle \Phi, \sigma \rangle$ consisting of a context Φ and substitution σ satisfying: (i) $\Phi \vdash \pi \curlywedge t\sigma$, if $\pi \curlywedge^? t \in \mathtt{Pr}$, and (ii) $\Phi \vdash s\sigma \stackrel{\curlywedge}{\approx}_\alpha t\sigma$, if $s \stackrel{\curlywedge}{\approx}_\alpha^? t \in \mathtt{Pr}$. As usual, the set of solutions $\mathcal{U}(\mathtt{Pr})$ for Pr is ordered via an instantiation ordering: $\langle \Phi_1, \sigma_1 \rangle \leq \langle \Phi_2, \sigma_2 \rangle$ iff there exists a substitution δ such that $\Phi_2 \vdash X\sigma_2 \stackrel{\curlywedge}{\approx}_\alpha X\sigma_1\delta$ and $\Phi_2 \vdash \Phi_1\delta$, for all $X \in \mathbb{X}$. In this case, the pair $\langle \Phi_2, \sigma_2 \rangle$ is an *instance* of the pair $\langle \Phi_1, \sigma_1 \rangle$.

In [2] a rule-based algorithm ($\mathtt{unify}_\curlywedge$) was proposed to compute solutions of these problems, if any exists. It applies the rules in Table 1 bottom-up plus rules for instantiating variables:

$$\begin{array}{lll} (\stackrel{\curlywedge}{\approx}_\alpha inst1) & \mathtt{Pr} \uplus \{\pi \cdot X \stackrel{\curlywedge}{\approx}_\alpha^? t\} \stackrel{[X \mapsto \pi^{-1} \cdot t]}{\Longrightarrow} & \mathtt{Pr}\{X \mapsto \pi^{-1} \cdot t\}, \text{ if } X \notin \mathtt{Var}(t) \\ (\stackrel{\curlywedge}{\approx}_\alpha inst2) & \mathtt{Pr} \uplus \{t \stackrel{\curlywedge}{\approx}_\alpha^? \pi \cdot X\} \stackrel{[X \mapsto \pi^{-1} \cdot t]}{\Longrightarrow} & \mathtt{Pr}\{X \mapsto \pi^{-1} \cdot t\}, \text{ if } X \notin \mathtt{Var}(t) \end{array}$$

The algorithm was shown to be terminating, sound, and complete.

A matching-in-context problem has the form $(\Phi \vdash s) \approx_? (\Upsilon \vdash t)$, it is a version of the nominal unification problem in which only one side can be instantiated (here, the right-hand side) and, in addition, the contexts Φ, Υ have to be satisfied. A solution to a matching problem is a substitution δ satisfying: $\Phi \vdash s \stackrel{\curlywedge}{\approx}_\alpha t\delta, \Upsilon\delta$. Matching-in-context, also called pattern-matching, was introduced in [5] to define nominal rewriting (using freshness constraints).

3 Nominal Disunification via Fixed-Point Constraints

This section contains the contributions that extend [3]. Next, we define the nominal disunification problem via fixed-point constraints.

Definition 3.1. A nominal disunification problem $\mathcal{P}_{\curlywedge}$ is a pair $\langle \texttt{Pr} \mid\mid D_{\curlywedge} \rangle$ of the form $\mathcal{P}_{\curlywedge} = \langle\ \texttt{Pr} \mid\mid u_1 \not\stackrel{\curlywedge}{\approx}_\alpha^{?} v_1, \ldots, u_m \not\stackrel{\curlywedge}{\approx}_\alpha^{?} v_m \rangle$, where $\texttt{Pr}$ is a nominal unification problem and $D_{\curlywedge}$ consists of a finite (possibly empty) set of nominal disequations.

In contrast with unification, where we are interested in solving equations, disunification problems enrich unification problems with disequations. The intuition is that disequations are constraints on the way we instantiate solutions of the equations we want to solve. For instance, $\mathcal{P}_{\curlywedge} = \langle X \stackrel{\curlywedge}{\approx}_\alpha^{?} f(Y) \mid\mid Y \not\stackrel{\curlywedge}{\approx}_\alpha^{?} a, Y \not\stackrel{\curlywedge}{\approx}_\alpha^{?} b \rangle$ expresses that, while solving the equation $X \stackrel{\curlywedge}{\approx}_\alpha^{?} f(Y)$, a solution must also satisfy the constraint that no instance of it is allowed to map X to $f(a)$ nor X to $f(b)$. Therefore, while $[X \mapsto f(c)]$, which is a grounding instance of the mgu $[X \mapsto f(Y)]$, validates the constraints imposed by $\mathcal{P}_{\curlywedge}$, the instance $[X \mapsto f(a)]$ does not.

The next example illustrates this principle when fixed-point constraints are taken into account.

Example 3.1. Consider $\mathcal{P}_{\curlywedge} = \langle X \stackrel{\curlywedge}{\approx}_\alpha^{?} (a\ b) \cdot Y \mid\mid [a]X \not\stackrel{\curlywedge}{\approx}_\alpha^{?} [b]Y \rangle$. The substitution $[X \mapsto (a\ b) \cdot Y]$ solves the equational part. In order to get the set of constraints imposed by $[a]X \not\stackrel{\curlywedge}{\approx}_\alpha^{?} [b]Y$, we solve the equation $[a]X \not\stackrel{\curlywedge}{\approx}_\alpha [b]X$ associated to it. The equation associated to the disequation is $[a]X \stackrel{\curlywedge}{\approx}_\alpha^{?} [b]Y$. Using the rule ($\stackrel{\curlywedge}{\approx}_\alpha \mathbf{ab}$) in Table 1, the fact that the system is syntax-directed, and satisfies the inversion property it follows that $[a]X \stackrel{\curlywedge}{\approx}_\alpha^{?} [b]Y$ *iff* $\{X \stackrel{\curlywedge}{\approx}_\alpha^{?} (a\ b) \cdot Y,\ (a\ c_1) \curlywedge^{?} Y, (c_1\ c_2) \curlywedge^{?} Y\}$, where c_1, c_2 are new names.

The solutions for $\mathcal{P}_{\curlywedge}$ are all the instances of the pair $\langle \Phi, \sigma \rangle = \langle \emptyset, [X \mapsto (a\ b) \cdot Y] \rangle$ that do not satisfy $(a\ c_1) \curlywedge^{?} Y, (c_1\ c_2) \curlywedge^{?} Y$.

The example above shows that we need the information of new names that are generated when we solve equations associated to disequations.

Definition 3.2. Let $\mathcal{P}_{\curlywedge}$ be a disunification problem. A *pair with exceptions for* $\mathcal{P}_{\curlywedge}$, denoted as $\langle \Phi, \sigma \rangle - \Theta$, consists of a pair $\langle \Phi, \sigma \rangle$ and an indexed family Θ of the form $\{\langle \nabla_l^1 \uplus \nabla_l^2, \theta_l \rangle \mid l \in I\}$, where ∇_l^2 is a (possibly empty) set of primitive fixed-point constraints involving new names, i.e., names not occurring anywhere in $\mathcal{P}_{\curlywedge}$.

The notion of pair with exceptions is key for the representation of solutions of a disunification problem: it will impose restrictions (the exceptions) on how these solutions can be instantiated. Intuitively, Θ consists of pairs of solutions of the equations associated to the disequations in $\mathcal{P}_{\curlywedge}$.

Definition 3.3. Let $\mathcal{P}_{\curlywedge}$ be a disunification problem. Let $\overline{c}$ be a set of new atoms and $\overline{X}$ be the set of variables of $\mathcal{P}_{\curlywedge}$. We denote by $\Phi^{\overline{c},\overline{X}}$ the extension of Φ with a set of primitive constraints $(c_1\ c_2) \curlywedge X$ for every pair c_1, c_2 in $\overline{c}$ and $X \in \overline{X}$. We say that

(i) $\langle \Phi, \sigma \rangle$ is an instance of a family $\Theta = \{\langle \nabla_l^1 \uplus \nabla_l^2, \theta_l \rangle \mid l \in I\}$ iff every instance of $\langle \Phi^{\overline{c},\overline{X}}, \sigma \rangle$, is an instance of some $\langle \nabla_l^1 \uplus \nabla_l^2, \theta_l \rangle \in \Theta$, where ∇_l^2 consists of primitive fixed-point constraints involving new names, $\overline{c}$ are all the new names occurring in ∇_l^2 for any l.

(ii) $\langle \Delta, \lambda \rangle$ is an instance of $\langle \Phi, \sigma \rangle - \Theta$ iff $\langle \Delta, \lambda \rangle$ is an instance of $\langle \Phi, \sigma \rangle$ but not of Θ.

(iii) A pair with exceptions $\langle\Phi,\sigma\rangle - \Theta$ is consistent iff it has at least one instance.

Lemma 3.1 (Inconsistency Lemma)**.** A pair with exceptions $\langle\Phi,\sigma\rangle - \Theta$ is inconsistent if and only if $\langle\Phi,\sigma\rangle$ is an instance of Θ.

The next result is the basis for an algorithm (Algorithm 1) to test the consistency of a pair with exceptions $\langle\Phi,\sigma\rangle - \Theta$ for $\mathcal{P}_\curlywedge$. It suffices to solve *matching-in-context* problems of the form $(\Phi^{\overline{c},\overline{X}} \vdash X\sigma) \approx_? (\nabla_l^1 \uplus \nabla_l^2 \vdash X\theta_l)$ for every variable X in $\mathcal{P}_\curlywedge$, Differently from [3], here we check the domain of the permutations in $\Phi^{\overline{c},\overline{X}}|_X$ and in an instance[1] $\langle(\nabla_l^1 \uplus \nabla_l^2)\delta\rangle_{\mathtt{nf}}|_X$ for X in $\mathcal{P}_\curlywedge$.

Corollary 3.1. Let $\langle\Phi,\sigma\rangle - \Theta$ be a pair with exceptions for $\mathcal{P}_\curlywedge$. If there is some $\langle\nabla_l^1 \uplus \nabla_l^2, \theta_l\rangle \in \Theta$ such that there exists a solution δ for the matching-in-context problems $(\Phi^{\overline{c},\overline{X}} \vdash X\sigma) \approx_? (\nabla_l^1 \uplus \nabla_l^2 \vdash X\theta_l)$, for all $X \in \mathcal{P}_\curlywedge$, then $\langle\Phi,\sigma\rangle - \Theta$ is inconsistent. Moreover, $\mathtt{dom}(\mathtt{perm}(\langle(\nabla_l^1 \uplus \nabla_l^2)\delta\rangle_{\mathtt{nf}}|_X)) \subseteq \mathtt{dom}(\mathtt{perm}((\Phi^{\overline{c},\overline{X}}|_X))$ for each X.

The corollary provides a method for checking for consistency of a pair with exceptions for a problem $\mathcal{P}_\curlywedge$:

Algorithm 1: Consistency Test

```
input: a finite pair with exceptions ⟨Φ,σ⟩ − Θ for P⅄.
output: true if the input is consistent, false, otherwise.
for ⟨∇l¹ ⊎ ∇l², θl⟩ ∈ Θ do
    if δ = matching(Φ^{c̄,X̄}, X1σ ≈? X1θl, ··· , Xnσ ≈? Xnθl)
    then
        if dom(Perm(⟨(∇l¹ ⊎ ∇l²)δ⟩nf|X)) ⊆ dom(Perm(Φ^{c̄,X̄}|X)), for all X ∈ P⅄
        then
          return false and stop
        end if
    end if
end for
return true
```

A solution for a disunification problem $\mathcal{P}_\curlywedge$ will be a pair $\langle\Phi,\sigma\rangle$ that satisfies the conjunction of constraints in $E_\curlywedge$, and the conjunction of the constraints in $D_\curlywedge$. Formally,

Definition 3.4. Let $\mathcal{P}_\curlywedge = \langle \mathtt{Pr} \mid\mid p_1 \overset{\curlywedge ?}{\not\approx}_\alpha q_1, \ldots, p_m \overset{\curlywedge ?}{\not\approx}_\alpha q_m\}\rangle$ be a nominal disunification problem. A *solution* to $\mathcal{P}_\curlywedge$ is a pair $\langle\Delta,\lambda\rangle$ of a context Δ and a substitution λ satisfying the following conditions:

1. $\langle\Delta,\lambda\rangle$ is a solution for $\mathtt{Pr}$ of $\mathcal{P}_\curlywedge$;

2. $\langle\Delta,\lambda\rangle$ satisfies the disequational part $D_\curlywedge$ of $\mathcal{P}_\curlywedge$, that is, for all grounding substitution δ:

$$\Delta \not\vdash p_1\lambda\delta \overset{\curlywedge}{\approx}_\alpha q_1\lambda\delta \wedge \ldots \wedge p_n\lambda\delta \overset{\curlywedge}{\approx}_\alpha q_n\lambda\delta.$$

Definition 3.5. We call a set S of pairs with exceptions for $\mathcal{P}_\curlywedge$ a *complete representation* of the solutions of the constraint problem $\mathcal{P}_\curlywedge$ iff S satisfies the following conditions:

1. if $\langle\Phi,\sigma\rangle - \Theta \leq \langle\Delta,\lambda\rangle$ for some $\langle\Phi,\sigma\rangle - \Theta$ in S, then $\langle\Delta,\lambda\rangle$ solves $\mathcal{P}_\curlywedge$;

2. if $\langle\Delta,\lambda\rangle$ solves $\mathcal{P}_\curlywedge$, then it is an instance of some $\langle\Phi,\sigma\rangle - \Theta$ in S;

[1] The normal form of the instance $(\nabla_l^1 \uplus \nabla_l^2)\delta$ of the context $\nabla_l^1 \uplus \nabla_l^2$ w.r.t. the rules in $\mathtt{unify}_\curlywedge$

3. $\langle\Phi,\sigma\rangle - \Theta$ is consistent for all $\langle\Phi,\sigma\rangle - \Theta$ in S.

Theorem 3.1 (Representation Theorem)**.** Let $\mathcal{P}_\curlywedge = \langle \texttt{Pr} \mid\mid D_\curlywedge\rangle$ be a nominal disunification problem. Define the family

$$\Theta := \bigcup_{p_i \not\overset{\curlywedge}{\approx}^?_\alpha q_i \in D_\curlywedge} \mathcal{U}(p_i \overset{\curlywedge}{\approx}^?_\alpha q_i).$$

Then the set $S = \{\langle\Phi,\sigma\rangle - \Theta \mid \langle\Phi,\sigma\rangle \in \mathcal{U}(\texttt{Pr}) \text{ and } \Theta \not\leq \langle\Phi,\sigma\rangle\}$ is a complete representation of solutions for the problem $\mathcal{P}_\curlywedge$.

Algorithm 2: Construction of a complete representation of solutions

input: A problem $\mathcal{P}_\curlywedge = \langle \texttt{Pr} \mid\mid D_\curlywedge\rangle$.
output: A finite set S of pair pairs with exceptions (possibly empty).
let $\langle\Phi,\sigma\rangle := \texttt{unify}_\curlywedge(\texttt{Pr})$
let
$\Theta := \bigcup_{p_i \not\overset{\curlywedge}{\approx}^?_\alpha q_i \in D_\curlywedge} \{\langle\nabla_i^1 \uplus \nabla_i^2, \theta_i\rangle = \texttt{unify}_\curlywedge(p_i \overset{\curlywedge}{\approx}^?_\alpha q_i)\}$
if $\texttt{consistent}(\langle\Phi,\sigma\rangle - \Theta)$ **then**
return $\langle\Phi,\sigma\rangle - \Theta$
else return $\emptyset$
end if

Example 3.2. Let $\mathcal{P}'_\curlywedge = \langle \underbrace{\{(a\ c_1) \curlywedge^? Y, (c_1\ c_2) \curlywedge^? Y, X \overset{\curlywedge}{\approx}^?_\alpha (a\ b)\cdot Y\}}_{\texttt{Pr}} \mid\mid [a]X \not\overset{\curlywedge}{\approx}^?_\alpha [b]Y\rangle$. Applying Algorithm 2 one has:

- $\texttt{unify}_\curlywedge(\texttt{Pr}) = \langle\{(a\ c_1) \curlywedge Y, (c_1\ c_2) \curlywedge Y\}, [X \mapsto (a\ b)\cdot Y]\rangle = \langle\Phi,\sigma\rangle$, and
- $\Theta = \{\langle\underbrace{\{(a\ c_1') \curlywedge Y, (c_1'\ c_2') \curlywedge Y\}}_{\nabla}, \underbrace{[X \mapsto (a\ b)\cdot Y]}_{\theta}\rangle\} = \texttt{unify}_\curlywedge([a]X \overset{\curlywedge}{\approx}^?_\alpha [b]Y)$ where c_1' and c_2' are new names.

Inconsistency of $\langle\Phi,\sigma\rangle - \Theta$ follows from Algorithm 1:

- $\Phi^{\overline{c},\overline{X}} = \Phi^{c_1',c_2',X,Y} = \Phi \cup \{(c_1'\ c_2') \curlywedge X, (c_1'\ c_2') \curlywedge Y\}$.
- $id = \texttt{matching}(\Phi^{c_1',c_2',X,Y}, X\sigma \approx_? X\theta, Y\sigma \approx_? Y\theta)$ and
- $\texttt{dom}(\texttt{perm}(\langle\nabla id\rangle_{\texttt{nf}}|_Y)) = \{c_1', c_2', a\} \subseteq \texttt{dom}(\texttt{perm}(\Phi^{c_1',c_2',X,Y})|_Y) = \{a, c_1, c_2, c_1', c_2'\}$.
- $\texttt{dom}(\texttt{perm}(\langle\nabla id\rangle_{\texttt{nf}}|_X)) = \emptyset \subseteq \texttt{dom}(\texttt{perm}(\Phi^{c_1',c_2',X,Y})|_X) = \{c_1', c_2'\}$.

Therefore, $S = \emptyset$ and there is no solution for $\mathcal{P}_\curlywedge$.

4 Conclusion and Future Work

This work used the fixed-point relation, intrinsic to the definition of the freshness relation, in order to extend the syntax concepts already defined in the usual nominal disunification. The fixed-point approach proved to be useful for dealing with equational theories that involve commutativity. For this reason, in future work, we intend to finalise the semantic analysis of our extension and take advantage of its finite representation of solutions to investigate disunification problems involving equational theories.

References

[1] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, and Daniele Nantes-Sobrinho. On solving nominal fixpoint equations. In *Front. of Combining Systems - 11th Int. Symp., FroCoS 2017, Proc.*, volume 10483 of *LNCS*, pages 209–226. Springer, 2017. doi:10.1007/978-3-319-66167-4_12.

[2] Mauricio Ayala-Rincón, Maribel Fernández, and Daniele Nantes-Sobrinho. On nominal syntax and permutation fixed points. *LMCS*, 16(1), 2020. doi:10.23638/LMCS-16(1:19)2020.

[3] Mauricio Ayala-Rincón, Maribel Fernández, Daniele Nantes-Sobrinho, and Deivid Vale. On solving nominal disunification constraints. In *Proc. LSFA 2019*. doi:10.1016/j.entcs.2020.02.002.

[4] Wray L. Buntine and Hans-Jürgen Bürckert. On solving equations and disequations. *J. ACM*, 41(4):591–629, 1994. doi:10.1145/179812.179813.

[5] Maribel Fernández and Murdoch Gabbay. Nominal rewriting. *Inf. Comput.*, 205(6):917–965, 2007. doi:10.1016/j.ic.2006.12.002.

[6] Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects Comput.*, 13(3-5):341–363, 2002. doi:10.1007/s001650200016.

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

[8] Manfred Schmidt-Schauß, Temur Kutsia, Jordi Levy, and Mateu Villaret. Nominal unification of higher order expressions with recursive let. In Manuel V. Hermenegildo and Pedro López-García, editors, *Logic-Based Program Synthesis and Transformation - 26th International Symposium, LOPSTR 2016, Edinburgh, UK, September 6-8, 2016, Revised Selected Papers*, volume 10184 of *Lecture Notes in Computer Science*, pages 328–344. Springer, 2016. doi:10.1007/978-3-319-63139-4_19.

[9] Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004. doi:10.1016/j.tcs.2004.06.016.

Nominal Anti-Unification with Atom-Variables

Manfred Schmidt-Schauß

Dept. Computer Science and Mathematics,
Goethe-university Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de

Abstract

Nominal unification was introduced by Urban, Pitts, and Gabbay and is an extension of first-order unification by higher-order constructs, where alpha-equality is used. The extension with atom-variables improves expressivity and applicability of nominal unification.

Anti-unification is the task of generalizing a set of expressions in the most specific way. It was extended to the nominal framework by Baumgarter, Kutsia, Levy und Villaret, who defined an algorithm solving the nominal anti-unification problem, which runs in polynomial time. Unfortunately the type of the set of solutions in nominal anti-unification (with explicit atoms) is infinitary, since every solution set can be strictly refined by adding a constraint $a\#X$ where a is a fresh atom without changing the other properties.

We use atom-variables instead of explicit atoms in our formulations, which avoids the infinitary property of solutions sets. Our formulation is strictly more general in several aspects: One aspect is that atom-variables are only names for atoms, and an extension is if it is permitted that different atom-variables are instantiated with identical atoms. These freedom in the formulation increases its application potential. We adapt their solution algorithm to atom-variables. There is a price to pay in the general case: checking freshness constraints and other logical questions of freshness constraints will require exponential time, hence it will be more complex as in the previous approach. Since it is work-in-progress, our working hypothesis is that the algorithm can be shown to be sound and complete and unitary in the case where atom-variables are names, or if the generalization variables are linear in the generalizations. There are good chances that subcases have polynomial complexity. In the general case the algorithm is presumably finitary but the complexity is strictly higher.

– work in progress –

1 Introduction

The work in [10] presents a nominal unification algorithm to unify abstract and higher-order expressions of a ground language defined by $e ::= a \mid \lambda a.e \mid f(a_1,\ldots,a_n)$ where a represents atoms (i.e. names), e expressions, λ is the usual lambda abstraction, and $f(a_1,\ldots,a_n)$ permits to form terms for function symbols in a given signature. For example, applications $(e\ e)$ in a higher-order language are represented using a binary function symbol. The abstract language also has unification variables (or expression variables X), where expressions can be substituted. The language for solutions has in addition permutations of ground atoms, and two extra terms in the grammar for e: X, and $\pi{\cdot}e$. Solutions in this formalism are substitution together with a set of constraints. Nominal unification solves equations w.r.t. α-equivalence on the ground language. The introduction of permutations is, however, not only technical, but really adds to the power of the method. The main reason is that $e_1 \sim_\alpha e_2 \Leftrightarrow \pi{\cdot}e_1 \sim_\alpha \pi{\cdot}e_2$, and that arguing and computing with permutations is smoother than arguing and computing with renamings. Nominal unification in its basic form is unitary and a polynomial (i.e.quadratic) algorithm is known for computing a unifier as well as for deciding nominal unifiability [10, 7, 4].

Anti-unification is the task, given a set of expressions, to find a most specific generalization of all the expressions in this set. The paper of Kutsia et. al. [6] investigates an anti-unification algorithm for (non-nominal) unranked terms and hedges, and later Baumgartner et. al. [2] built upon this by extending it to nominal terms. They constructed a nominal anti-unification algorithm that computes a finite set of least general generalizations (lgg) in polynomial time, however, as they argued, it is not possible to compute a single least general one. It has applications in the analysis of programs, such as finding similar expressions and clones [1, 5, 8].

In the first-order version, anti-unification simply looks for the largest common term structure and also takes care of equal variables. The problem can be solved in polynomial time and produces a unique solution.

A problem related to nominal anti-unification is anti-unification for higher-order pattern, which is unitary and where a linear time algorithm exists [3].

In the application to nominal terms, the syntax of the terms is more powerful, which means that the syntax of the generalizations is more powerful, which has to be taken into account by the search for a most specific generalization. This means freshness constraints and permutations, which are part of the syntax of nominal terms, also may appear in generalizations, which increases the representative power of this approach.

The example in [2] on infinitary nominal anti-unification is as follows: It simply says that for $f(a_1)$ and $g(a_2)$ the generalisation $(\emptyset, X)$ is appropriate, where the pair consists of a set of freshness constraints and the variable X as generalization. However, there is a a strictly decreasing chain $(\emptyset, X)$, $(\{a_3\#X\}, X)$,$(\{a_3\#X, a_4\#X\}, X)$, $\ldots$ which are also generalizations, for an infinite set of atoms $\{a_3, a_4, \ldots\}$. Somehow, the names $a_3, a_4, \ldots$ are irrelevant for the problem, but provide an argument that a complete set of least general generalizations is in general infinite. Restricting the set of available atoms to a finite set as in [2] results in nice properties of the algorithm. Although it may suffice in practice, it is not satisfactory. We show that atom-variables-as-names solve this problem by showing that infinitely descending generalisations are avoided, while keeping the good properties of the algorithm in [2], see Proposition 2.2.

Our approach is to add atom-variables, as in [9] and formulate the anti-unification problem in this language. Atoms are only used in the semantics, whereas in the expression language, only atom-variables are used.

In our approach we permit atom-variables $A_1, A_2, \ldots$ (which are not fixed). If we try to simulate the chain in the example above, the result would be $(\emptyset, X)$, $(\{A_3\#X\}, X)$, $(\{A_3\#X, A_4\#X\}, X)$, plus constraints like $A_3\#A_4$ to make the instances different. However, the set of instances is no longer a strictly decreasing chain, since every single instance is finite and the set of all instances does not change if the constraints grow longer. Hence this is no longer an example for a properly decreasing chain of generalizations. Our working hypothesis is that there is no counterexample at all, and there are always finite complete sets of generalizations.

2 Preparations for the Algorithm

The ground language NL_a is $s ::= a \mid f(s_1, \ldots, s_n) \mid \lambda a.s$, where a is a nonterminal for atoms in an infinite set of atoms, f is a function symbol in the function signature, where we assume that there is at least one (say c) of arity zero and one of arity 2. For example we use pairing as a function symbol, and an application in the lambda calculus can be represented using a binary function symbol `app`. Alpha-equivalence $\sim$ is defined on NL_a as usual.
The grammar for the expression language NL_A of expressions s and permutations π is as follows:

- $\pi ::= \emptyset \mid (\pi_1{\cdot}A_1\ \pi_2{\cdot}A_2){\cdot}\pi$
- $s ::= \pi{\cdot}A \mid \pi{\cdot}X \mid f(s_1,\ldots,s_n) \mid \lambda\pi{\cdot}A.s,$

where A is a nonterminal for atom-variables, X is a nonterminal for variables (for unification). Note that expressions fro NL_A do not contain atoms. We use the usual conventions for dealing with permutations and so-called suspensions, for example to move permutations inside terms and viewing $\emptyset{\cdot}s$ as the same term as s. We also use the convention to replace $\pi cdot A\#s$ immediately by $A\#\pi^{-1}{\cdot}s$ and then to move the permutation inside the expression. A single component $(\pi_1{\cdot}A_1\ \pi_2{\cdot}A_2$ of a permutation is also called a *swapping*. The notation W is sometimes used as an abbreviation for a suspension of atom-variables.

We also use generalized *freshness constraints* $A\#s$ and constraint sets (of freshness constraints). A constraint $A\#s$ is valid for ("ground") σ, if $A\sigma$ does not occur free in $s\sigma$.

Definition 2.1. *The representation of generalizations are terms-in-context* (∇, s) *where* ∇ *is a constraint set and* s *is an* NL_A*-expression.*
The semantics of (∇, s) *is the set of ground instances of* s *that satisfy* ∇*, i.e.,* $[\![(\nabla, s)]\!] := \{[r]_\sim \mid r \text{ is ground and } \exists\sigma : s\sigma \sim r \wedge \nabla\sigma \text{ holds }\}$.
A term-in-context (∇, t) *is* **more general** *than another term-in-context* (∇', t')*, if* $[\![(\nabla', t')]\!] \subseteq [\![(\nabla, t)]\!]$*, where we assume that the set consists of equivalence classes modulo* $\sim$*.*

The goal is to find a least generalization of sets of terms-in-context. Thus the generalization problem for two terms-in-context (∇_1, t_1) and (∇_2, t_2) is to find another (∇_3, t_3) such that $[\![(\nabla_1, t_1)]\!] \subset [\![(\nabla_3, t_3)]\!]$ and $[\![(\nabla_2, t_2)]\!] \subset [\![(\nabla_3, t_3)]\!]$, and $[\![(\nabla_3, t_3)]\!]$ is as small as possible, where the best case is that it is the smallest one. For example $(\{A_1\#A_2\}, f(A_1, \lambda A_2.A_2)$ and $(\emptyset, f(c, \lambda A_3.A_3)$, where c is a unary symbol in the signature, have as generalization $(\emptyset, f(X, \lambda A_2.A_2))$, since the constraint is irrelevant in the final expression $f(X, \lambda A_2.A_2)$.

Notations and Conventions *Head*(s) is defined as f if $s = f(\ldots)$, and λ if $s = \lambda a.s'$; if s is a suspension $\pi{\cdot}X$, then X; and if s is $\pi{\cdot}A$ then A. The notation W means a suspension of an atom variable.

2.1 Atom-Variables as Names

We investigate solutions of the form (X, ∇) and show that extending ∇ by entries of the form $A\#X$ for fresh atom-variables does mot change the set of instances.

Proposition 2.2. *Let* X *be a variable,* ∇ *be a constraint set, where we assume the atom-variables-as-names regime: for every pair* A, B *of atom-variables that is used, also* $A\#B$ *is in* ∇*. Let* A' *be a semantically fresh atom-variable, and let* $\nabla' = \nabla\ \cup \nabla''$*, where* ∇'' *consists of pairs* $A'\#A_i$ *for all atom-variables* A_i *occurring in* ∇*, and a constraint* $A'\#X$*. This implies all used different atom-variables have different instances. Then* $[\![(X, \nabla)]\!] = [\![(X, \nabla')]\!]$*.*

Proof. The only necessary argument is that there is some atom a not contained in $\sigma(X)$ nor in any other expression in ∇. Then we can change σ to σ', if necessary, to $\sigma'(A') = a$. Then the constraint is satisfied and the expression is also in $[\![(X, \nabla')]\!]$. □

3 The Algorithm AtomAntiUnification and its Rules

3.1 The Nominal Generalization Algorithm

The data structure of the algorithm AtomAntiUnification is a tuple (Γ, M, ∇, L) where

(Decomposition):
$$\frac{\{X{:}f(s_1,\ldots,s_n) \triangleq f(t_1,\ldots,t_n)\} \dot\cup \Gamma, M, \nabla, L}{\Gamma \dot\cup \{X_1{:}s_1 \triangleq t_1,\ldots,X_n{:}s_n \triangleq t_n\}, M, \nabla, L \cup \{X \mapsto f(X_1,\ldots,X_n)\}}$$ where X_i are fresh variables

(Abstraction):
$$\frac{\{X{:}\lambda W_1.s \triangleq \lambda W_2.t\} \dot\cup \Gamma, M, \nabla, L}{\Gamma \dot\cup \{Y{:}(W_1\ B)\cdot s \triangleq (W_2\ B)\cdot t\}, M, \nabla', L \cup \{X \mapsto \lambda B.Y\}}$$ where Y is a fresh variable, and B is a semantically fresh atom-variable, represented in ∇'

(Suspension):
$$\frac{\{X{:}W_1 \triangleq W_2\} \dot\cup \Gamma, M, \nabla, L \qquad \nabla \models W_1 = W_2}{\Gamma, M, \nabla, L \cup \{X \mapsto W_1\}}$$

(Merging):
$$\frac{\Gamma, \{X{:}s_1 \triangleq t_1, Y{:}s_2 \triangleq t_2\} \dot\cup M, \nabla, L \qquad EQVM(\{(s_1,t_1) \preceq (s_2,t_2)\}, \nabla) = \pi}{\Gamma, M \cup \{X{:}s_1 \triangleq t_1\}, \nabla, L \cup \{Y \mapsto \pi\cdot X\}}$$

(Merging-Sym):
$$\frac{\Gamma, \{X{:}s_1 \triangleq t_1, Y{:}s_2 \triangleq t_2\} \dot\cup M, \nabla, L \qquad EQVM(\{(s_1,t_1) \preceq (t_2,s_2)\}, \emptyset, \nabla) = \pi}{\Gamma, M \cup \{X{:}s_1 \triangleq t_1\}, \nabla, L \cup \{Y \mapsto \pi\cdot X\}}$$

(General):
$$\frac{\{X{:}s \triangleq t\} \dot\cup \Gamma, M, \nabla, L}{\Gamma, M \cup \{X{:}s \triangleq t\}, \nabla, L}$$ If $\mathit{Head}(s) \neq \mathit{Head}(t)$ and if s and t are not both suspensions of atom-variables.

(GeneralAB):
$$\frac{\{X{:}W_1 \triangleq W_2\} \dot\cup \Gamma, M, \nabla, L \qquad \nabla \not\models W_1 = W_2}{\Gamma, M \cup \{A{:}W_1 \triangleq W_2\}, \nabla, L \cup \{X \mapsto A\}}$$ A is a fresh atom-variable.

Figure 1: Rules of the algorithm AtomAntiUnification

- Γ is a set of generalization triples of the form $X{:}s \triangleq t$, where X is a (generalization-) variable, and s, t are NL_A-expressions.
- M is a set of solved generalization triples.
- ∇ is a set of freshness constraints.
- L is a substitution represented as a list of bindings.

The output is a term-in-context generated from the generated substitution, i.e. the output is $(X \circ L, \nabla)$, where X is the initial generalization variable, L is the computed substitution, and ∇ the final constraint.

We say B is a *semantically fresh* atom-variable, if it is created and constraints $A\#A'$ for different A, A' are added to ∇ that make it semantically different from all already used atom-variables.

The rules of the algorithm AtomAntiUnification are in Fig. 1.
The rules are applied until the set Γ is empty, and the merging rules (Merging) and (Merging-Sym) are no longer applicable.

$$\frac{\Gamma \dot\cup \{e \preceq e\}, \Pi, \nabla}{\Gamma, \Pi, \nabla} \qquad \frac{\Gamma \dot\cup \{(f\ s_1 \ldots s_n) \preceq (f\ s'_1 \ldots s'_n)\}, \Pi, \nabla}{\Gamma \cup \{s_1 \preceq s'_1, \ldots, s_n \preceq s'_n\}, \Pi, \nabla}$$

$$\frac{\Gamma \dot\cup \{\pi_1 \cdot X \preceq \pi_2 \cdot X\}, \Pi, \nabla \quad \nabla \models \pi_1 \approx \pi_2}{\Gamma, \Pi, \nabla} \qquad \frac{\Gamma \dot\cup \{\lambda W_1.s \preceq \lambda W_2.t\}, \Pi, \nabla \quad \nabla \models W_1 \# \lambda W_2.t}{\Gamma \cup \{(W_1\ W_2) \cdot s \preceq t\}, \Pi, \nabla}$$

$$\frac{\Gamma \dot\cup \{W_1 \preceq W_2\}, \Pi, \nabla}{\Gamma, \{W_1 \mapsto W_2\} \cup \Pi, \nabla} \qquad \frac{\emptyset, \Pi, \nabla \quad \nabla \models (\Pi \text{ is a bijection})}{\text{success: the result of the call to } EQVM \text{ is the computed permutation } \pi \text{ (see Def. 3.1)}}$$

Figure 2: Rules of the permutation matching algorithm *EQVMatch*

3.2 Equivariance Algorithm

The merge-rule as in [2], which is an equivariance problem, will be treated similarly, however, slightly generalized to atom-variables and nested permutation expression. We use a matching-like rule-based algorithm that finally is able to produce a permutation for the merge rule, if there is one at all. Instead of fixing the derivation $\nabla \vdash \ldots$, we will use the semantic variant $\nabla \models \ldots$ to be as general as possible, which will leave some open space for optimizations.

Definition 3.1. *The rules of the nondeterministic algorithm EQVMatch are in Fig. 2. The initial triple has $\emptyset$ as Π, i.e. the start triple is $(\Gamma, \emptyset, \nabla)$, where Γ and ∇ are delivered in the call to this algorithm. The rules are to be applied as long as possible. If the state is reached where Γ is empty, and ∇ implies that the mappings in Π do not collide and form an injection, then the algorithm is successful, otherwise there is no permutation returned.*
In the success case a permutation is returned after the following processing.
We exploit the guarantee by ∇ that the mappings in Π are injective. First perform the following actions on Π until no longer possible:

- *If there are two pairs $(W_1 \mapsto W_2)$, $(W'_1 \mapsto W'_2)$ in Π, and $\nabla \models W_1 = W'_1$, then remove the mapping $(W'_1 \mapsto W'_2)$ from Π.*
- *If there are two pairs $(W_1 \mapsto W_2)$, $(W'_1 \mapsto W'_2)$ in Π, and $\nabla \models W_2 = W'_2$, then remove the mapping $(W'_1 \mapsto W'_2)$ from Π. (Note that this computation is redundant).*
- *If there is a pair $(W_1 \mapsto W_2)$, and $\nabla \models W_1 = W_2$, then remove the mapping from Π.*

Then group the mappings such that a permutation can be formed. For example, if one group is a sequential subset of mappings like $a \mapsto b, b \mapsto c, c \mapsto d$ and there is no mapping for d (or the mapping $d \mapsto a$), then generate the permutation $\pi_1 := (a\ b) \cdot (b\ c) \cdot (c\ d)$. Note that we assume application from the left to atoms. For several groups generate a permutation for each group, and then compose them to a single permutation π, which is the result of the EQVMatch.

An algorithm for checking $\nabla \models \ldots$ can be done by a brute-force algorithm checking all cases of potential instantiations of atom variables with atoms (from a finite set at most as large as the set of atom variables). An This can be performed in exponential time since it is sufficient to check all possibilities of equality and disequality of atom-variables.

Example 3.2. *An example demonstrating the use of ∇ and the treatment of bindings is as follows. Let the expressions be $\lambda A_1.A_2$ and $\lambda A_2.A_1$. The generalization is $\lambda A.X$ or $\lambda A.A$ depending on ∇. The computation is roughly as follows:*
$X{:}\lambda A_1.A_2 \triangleq \lambda A_2.A_1$ as input. After an application of (Abstraction) we obtain: $Y{:}A_2 \triangleq A_1$ and

$\nabla' = \nabla \cup \{B\#A_1, B\#A_2\}$, *and* $X \mapsto \lambda B.Y$, *(and using this in the application of permutation). Now it depends on whether* $\nabla \models A_1 = A_2$ *holds.*
If $\nabla \models A_1 = A_2$, *then we apply (Suspension) and obtain* $X \mapsto \lambda B.A_1$. *If* $\nabla \not\models A_1 = A_2$, *then we apply (GeneralAB) and obtain* $X \mapsto \lambda B.A$ *for a fresh atom-variable* A.

The algorithm AtomAntiUnification is designed such that it is correct and complete. A proof of its properties, however, is future work.

Specialization atom-variables-as-names The restriction is that all atom-variables have different atoms as instances. This restriction is already mentioned in Proposition 2.2, where it is argued that it improves the view on the framewoj and algorithms in [2]. There is no need to change the algorithms in [2], however, the theoretical properteis can be improved: it will be unitary in this framework.

Specialization Linear Generalization Expression In our framework, the assumption that the only linear expressions are permitted as generalizations makes the algorithm much simpler: the subalgorithm for detecting equivariant terms-in-context is no longer required. The remaining rules do not need a guessing, and thus we can expect that it is unitary.

A first look at the complexity of the rule-based algorithm reveals that (Suspension) and (GeneralAB) are problematic, since they require to compute a consequence of ∇. This appears to be of exponential complexity. To avoid this complexity, it would be possible to omit the (GeneralAB)-rule, and always use the (Suspension)-rule without checking the condition. This restriction of the algorithm is expected to lead to y polynomial complexity.

General Case We believe that the algorithm is correct and complete. Since the guessing is finite, there will be only finitely many generalization terms. So our guess is that we can prove that generalization in the atom-variable case is finitary. The complexity of the algorithm at a first look is simply exponential. A deeper analysis may argue for a low complexity within the polynomial hierarchy.

4 Future Work

Proofs of properties of the generalization are in order, and perhaps adaptations of the algorithm according to improvements. Since it is work-in-progress, our working hypotheses are: checking freshness constraints and implications of freshness constraints will require exponential time, and the algorithm is complete and finitary.

It would be interesting to analyse the weaker variant where the generalizations are linear in the generalization variables. This would make the merge-rules superfluous, thus a source of complexity of the algorithm is removed which may result in a good trade-off between expressiveness and complexity. Further ignoring the exactness of the rules (Suspension) and (GeneralAB) may lead to a polynomial complexity.

If the algorithm is applied in a programming language environment, it would be appropriate to include constraints that reflect the so-called variable-assumption: that bound variables in program definitions are different, and that free variables are different from bound variables.

Acknowledgment

I want do thank the referees of UNIF 2021 for their very helpful comments.

References

[1] 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.

[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, 2015, Warsaw, Poland*, volume 36 of *LIPIcs*, pages 57–73. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[3] 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.

[4] Christophe Calvès and Maribel Fernández. A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306, 2008.

[5] 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, *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, volume 4830 of *Lecture Notes in Computer Science*, pages 273–282. Springer, 2007.

[6] Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for unranked terms and hedges. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPIcs*, pages 219–234. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[7] Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In Christopher Lynch, editor, *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 209–226. Schloss Dagstuhl, 2010.

[8] Jianguo Lu, John Mylopoulos, Masateru Harao, and Masami Hagiya. Higher order generalization and its application in program verification. *Ann. Math. Artif. Intell.*, 28(1-4):107–126, 2000.

[9] Manfred Schmidt-Schauß, David Sabel, and Yunus D. K. Kutz. Nominal unification with atom-variables. *J. Symb. Comput.*, 90:42–64, 2019.

[10] Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. In *17th CSL, 12th EACSL, and 8th KGC*, volume 2803 of *LNCS*, pages 513–527. Springer, 2003.

Checking Symbolic Security of Cryptographic Modes of Operation*

Hai Lin and Christopher Lynch

Clarkson University, Potsdam, NY, U.S.A
hlin@clarkson.edu clynch@clarkson.edu

Abstract

In cryptography, modes of operation are used to encrypt plaintext messages of multiple blocks. The standard notion of computational security for modes of operation is IND$-CPA security. One can use symbolic methods to reason about the security of modes of operation. In this paper, we introduce a sufficient condition (called the *uniqueness property*) for symbolic security of modes of operation. We present an algorithm for checking the uniqueness property. The algorithm can be used to automatically synthesize secure cryptographic modes of operation.

1 Introduction

Cryptographic modes of operation are ways to encrypt plaintext messages of multiple blocks using block ciphers. The standard notion of computational security is IND$-CPA security [5]. To check the IND$-CPA security of a mode of operation, an adversary interacts with some encryption oracle by sending plaintext blocks to the encryption oracle and receiving ciphertext blocks back from the encryption oracle. Multiple interactions involving multiple messages can be interleaved. A mode of operation is IND$-CPA secure if and only if the adversary cannot distinguish between the ciphertext blocks and random bits of the same length. There has been a recent trend of automatic verification and synthesis of secure cryptosystems [1, 2].

In [4], Meadows establishes the connection between the computational world and the symbolic world. It is shown that cryptographic operations (e.g. block cipher) can be modelled as function symbols, message blocks can be modelled as terms, properties of operations (e.g. exclusive-or) can be modelled as an equational theory, etc. A notion of *symbolic security* is introduced, which aims to connect computational security and symbolic security.

In this paper, we propose a sufficient condition for symbolic security, which is called the *uniqueness property*. We also propose a sound and terminating algorithm for checking the uniqueness property. Since the problem of checking symbolic security is undecidable in general [3], our algorithm is incomplete. But it can handle popular modes of operation including Cipher Feedback Mode, etc. Our proposed algorithm can be used to automatically synthesize secure cryptographic modes of operation.

The rest of this paper is organized as follows. In Section 2, we introduce some notation and describe the notion of symbolic security, which is introduced in [4]. In Section 3, we introduce the uniqueness property, and show that it implies symbolic security. In Section 4, we propose an algorithm for checking the uniqueness property. We conclude and discuss future work in Section 5.

*The work is supported by NRL under contract N00173-19-1-G012.

2 Preliminaries

In [4], a first-order signature $\Sigma = \{f/1, \oplus/2, 0/0\}$ is used to model modes of operation, where f models block cipher, 0 models message blocks of all 0's. N is a set of constants modelling blocks of random bits (e.g. initialization vectors). X is a set of variables containing *plaintext variables* modelling plaintext blocks and *ciphertext variables* modelling ciphertext blocks. Terms over $T(\Sigma \cup N, X)$ model ciphertext blocks, (possibly) in terms of other plaintext blocks and ciphertext blocks.

More precisely, let M be a mode of operation, which is defined inductively as $C_{p,i} = t_{ind}, C_{p,0} = t_0$, where $t_{ind}, t_0 \in T(\Sigma \cup N, X)$. A symbolic history H is a sequence of terms exchanged between the adversary and the encryption oracle. A symbolic history can be an interleaving of multiple sessions, each of which is used to encrypt a single message of some plaintext blocks. We use C_i to denote the ith ciphertext block sent by the encryption oracle in H[1]. We call $C_{p,i}$ a *ciphertext variable*, and use it to denote the ith ciphertext block from the pth session. We call $x_{p,i}$ a *plaintext variable*, and use it to denote the ith plaintext block from the pth session. We use r_p to denote the initialization vector of the pth session. If we *unfold* $C_{p,i}$, we get t_{ind}. We assume that t_{ind} is a term of the form $f(t_1) \oplus \ldots \oplus f(t_m) \oplus x_{p,i}$. We use *top-$f$-terms*$(C_{p,i})$ to denote $\{f(t_1), \ldots, f(t_m)\}$. Each $f(t_j)$ $(1 \leq j \leq m)$ is called an *f-rooted summand* of $C_{p,i}$. We define $size_f(C_{p,i})$ to be the number of f-rooted summands of $C_{p,i}$.

In [4], the notions of *computable substitution* and *symbolic security* are introduced. Given a symbolic history H, a substitution σ is a *computable substitution* of H if σ maps each variable x to the exclusive-or of some (0 or more) terms that appear in H before x. A mode of operation is *symbolically secure* if it does not admit any symbolic history H together with a computable substitution σ of H s.t. there exists a subsequence $C_{i_1}, \ldots, C_{i_k}$ in H s.t. $(C_{i_1} \oplus \ldots \oplus C_{i_k})\sigma =_\oplus 0$.

Example 1. *Consider the following Cipher Feedback Mode, where*

$C_{p,i} = f(C_{p,i-1}) \oplus x_{p,i}$
$C_{p,0} = r_p$, *where* r_p *is a constant.*

$H = r_1, r_2, x_{1,1}, f(r_1) \oplus x_{1,1}, x_{2,1}, f(r_2) \oplus x_{2,1}, x_{1,2}, f(f(r_1) \oplus x_{1,1}) \oplus x_{1,2}$ *is a possible symbolic history of Cipher Feedback Mode, which is the interleaving of two sessions.* $\sigma = \{x_{1,1} \mapsto 0, x_{2,1} \mapsto f(r_1), x_{1,2} \mapsto f(r_1) \oplus r_2\}$ *is a computable substitution of* H.

Definition 1. *Given a term* t, $C_Var(t)$ *denotes the set of ciphertext variables in* t. *More formally,*

- $C_Var(C_{p,i}) = \{C_{p,i}\}$, *if* $C_{p,i}$ *is a ciphertext variable.*
- $C_Var(x_{p,i}) = \emptyset$, *if* $x_{p,i}$ *is a plaintext variable.*
- $C_Var(f(t)) = C_Var(t)$.
- $C_Var(t_1 \oplus t_2) = C_Var(t_1) \cup C_Var(t_2)$.

3 Uniqueness Property

The following is a definition for the *uniqueness property*, which is a sufficient condition for symbolic security.

[1] $C_i \notin T(\Sigma \cup N, X)$, where $i = 1, 2, 3, \ldots$

Definition 2. *Let M be a mode of operation. Consider any symbolic history H of M and any computable substitution σ of H. Let $C_{p,i}$ and $C_{q,j}$ be any two ciphertext variables in H. M satisfies the* uniqueness property *if for any two distinct terms $t_1, t_2 \in$ top-f-terms$(C_{p,i}) \cup$ top-f-terms$(C_{q,j})$, $t_1\sigma \neq_\oplus t_2\sigma$.*

The following lemma states that the uniqueness property implies symbolic security.

Lemma 1. *Let M be any mode of operation. If M satisfies the uniqueness property, then M is symbolically secure.*

Proof. Let M be a mode of operation. Consider any symbolic history H of M and any computable substitution σ. Let $S : C_{p_1,i_1}, \ldots, C_{p_m,i_m}$ be a subsequence of H. By the uniqueness property, $\sum_{k=1}^{m} \oplus C_{p_k,i_k}\sigma = \textit{top-f-terms}(C_{p_m,i_m})\sigma \oplus t$ for some t. Therefore, $\sum_{k=1}^{m} \oplus C_{p_k,i_k}\sigma \neq 0$. □

4 An Algorithm for Checking the Uniqueness Property

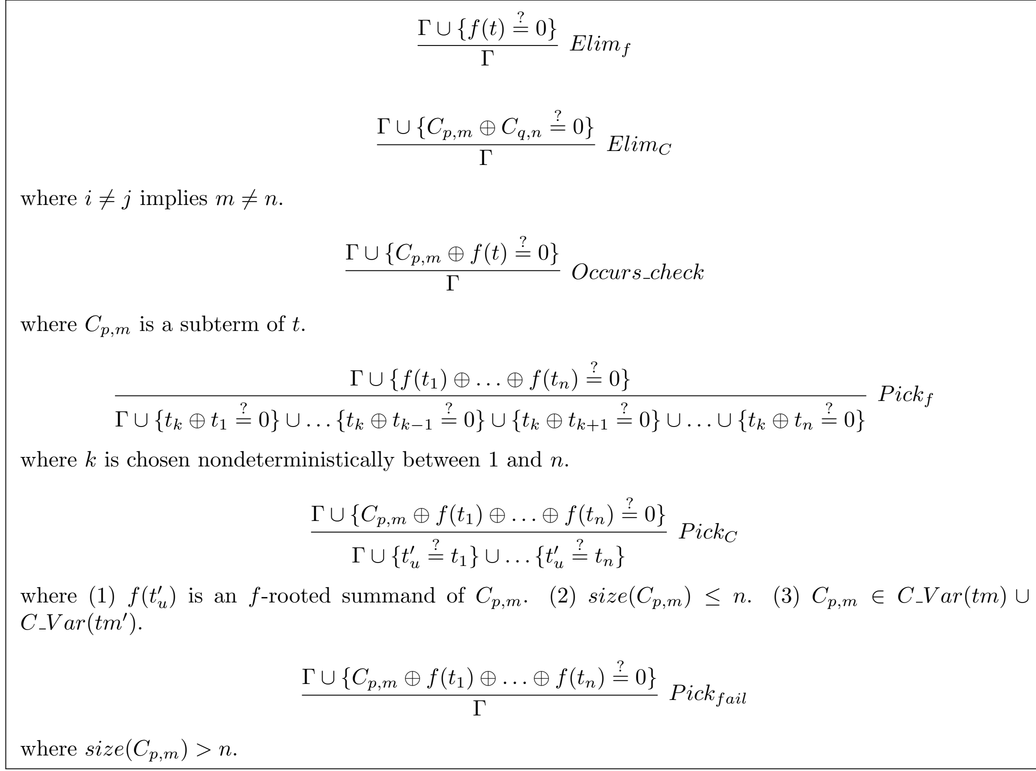
In this section, we propose an algorithm for checking the uniqueness property. Given a mode of operation M, we take an arbitrary symbolic history H of M and any two ciphertext variables $C_{p,i}$ and $C_{q,j}$ in H. We then consider two different f-rooted summands tm and tm' of $C_{p,i}$ and $C_{q,j}$. We assume that tm and tm' are the first (earliest) pair of terms in H that are unifiable modulo xor under some computable substitution, and try to derive contradiction. We start from an initial set of equations $\{tm \oplus tm' \stackrel{?}{=} 0\}$, and keep processing the equations using the inference rules in $\mathcal{I}_{i,j,tm,tm'}$ (Figure 1). Note that $\mathcal{I}_{i,j,tm,tm'}$ is parameterized by i, j, tm and tm', which are referred to by $Elim_C$ and $Pick_C$. We use $\mathcal{I}_{i,j,tm,tm'}(\{tm \oplus tm' \stackrel{?}{=} 0\})$ to represent the final result of applying $\mathcal{I}_{i,j,tm,tm'}$ to $\{tm \oplus tm' \stackrel{?}{=} 0\}$. We maintain the following invariant: If we get a set of equations Γ at any step , tm and tm' are unifiable modulo xor under some computable substitution, then at least one of the equations in Γ must hold. Intuitively, each equation in Γ represents a possibility that tm and tm' are unifiable modulo xor under some computable substitution. And Γ represents the set of all possibilities. Our goal is to derive a contradiction, which is to make Γ an empty set.

Algorithm 1 Checking Security of Modes of Operation

Input: a recursive description of some mode of operation M.

Γ = top-f-terms$(C_{p,i}) \cup$ top-f-terms$(C_{q,j})$
for each pair of distinct terms tm and tm' in Γ **do**
  if $\mathcal{I}_{i,j,tm,tm'}(\{tm \oplus tm' \stackrel{?}{=} 0\}) \neq \emptyset$ **then**
    return "unknown"
  end if
end for
return "secure"

The $Elim_f$ rule allows us to remove the possibility that an f-rooted term is 0. The $Elim_C$ rule allows us to remove the possibility that we somehow find an earlier pair of unifiable terms, since we assumed that tm and tm' are the earliest pair of unifiable terms. If we try to unify $C_{p,m}$ and a term containing $C_{p,m}$, according to the *Occurs_check* rule, that is impossible. If the xor of some f-rooted terms is 0, the $Pick_f$ rule nondeterministically picks one of them and



$$\frac{\Gamma \cup \{f(t) \stackrel{?}{=} 0\}}{\Gamma} \; Elim_f$$

$$\frac{\Gamma \cup \{C_{p,m} \oplus C_{q,n} \stackrel{?}{=} 0\}}{\Gamma} \; Elim_C$$

where $i \neq j$ implies $m \neq n$.

$$\frac{\Gamma \cup \{C_{p,m} \oplus f(t) \stackrel{?}{=} 0\}}{\Gamma} \; Occurs_check$$

where $C_{p,m}$ is a subterm of t.

$$\frac{\Gamma \cup \{f(t_1) \oplus \ldots \oplus f(t_n) \stackrel{?}{=} 0\}}{\Gamma \cup \{t_k \oplus t_1 \stackrel{?}{=} 0\} \cup \ldots \{t_k \oplus t_{k-1} \stackrel{?}{=} 0\} \cup \{t_k \oplus t_{k+1} \stackrel{?}{=} 0\} \cup \ldots \cup \{t_k \oplus t_n \stackrel{?}{=} 0\}} \; Pick_f$$

where k is chosen nondeterministically between 1 and n.

$$\frac{\Gamma \cup \{C_{p,m} \oplus f(t_1) \oplus \ldots \oplus f(t_n) \stackrel{?}{=} 0\}}{\Gamma \cup \{t'_u \stackrel{?}{=} t_1\} \cup \ldots \{t'_u \stackrel{?}{=} t_n\}} \; Pick_C$$

where (1) $f(t'_u)$ is an f-rooted summand of $C_{p,m}$. (2) $size(C_{p,m}) \leq n$. (3) $C_{p,m} \in C_Var(tm) \cup C_Var(tm')$.

$$\frac{\Gamma \cup \{C_{p,m} \oplus f(t_1) \oplus \ldots \oplus f(t_n) \stackrel{?}{=} 0\}}{\Gamma} \; Pick_{fail}$$

where $size(C_{p,m}) > n$.

Figure 1: Inference Rules $\mathcal{I}_{i,j,tm,tm'}$

lists all the possibilities that it can cancel with some other f-rooted term. The $Pick_C$ rule first unfolds $C_{p,m}$, then picks an f-rooted summand of $C_{p,m}$ and cancels it with some f-rooted term. Note that the $Pick_C$ rule rules out the possibility that two f-rooted summands of $C_{p,m}$ can cancel with each other. If the number of f-rooted summands of $C_{p,m}$ is greater than the number of f-rooted terms in an equation, the $Pick_{fail}$ rule applies.

The following is an example of using Algorithm 1 to check security of modes of operation.

Example 2. *Consider the following mode of operation, where:*

$C_{p,i} = f(C_{p,i-1}) \oplus f(f(C_{p,i-1})) \oplus x_{p,i}$
$C_{p,0} = r_p$.

- *According to Algorithm 1,* $\Gamma = \{f(C_{p,i-1}), f(f(C_{p,i-1})), f(C_{q,j-1}), f(f(C_{q,j-1}))\}$.

  There are 6 cases to consider.

  – *Case 1: Apply the inference system* $\mathcal{I}_{i,j,f(C_{p,i-1}),f(C_{q,j-1})}$ *to* $\{f(C_{p,i-1}) \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}$

$$\frac{\{f(C_{p,i-1}) \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}}{\{C_{p,i-1} \oplus C_{q,j-1} \stackrel{?}{=} 0\}} \; Pick_f$$

$$\frac{\{C_{p,i-1} \oplus C_{q,j-1} \stackrel{?}{=} 0\}}{\emptyset}\ Elim_C$$

– *Case 2: Apply the inference system* $\mathcal{I}_{i,j,f(f(C_{p,i-1})),f(f(C_{q,j-1}))}$ *to* $\{f(f(C_{p,i-1})) \oplus f(f(C_{q,j-1})) \stackrel{?}{=} 0\}$

$$\frac{\{f(f(C_{p,i-1})) \oplus f(f(C_{q,j-1})) \stackrel{?}{=} 0\}}{\{f(C_{p,i-1}) \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}}\ Pick_f$$

$$\frac{\{f(C_{p,i-1}) \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}}{\{C_{p,i-1} \oplus C_{q,j-1} \stackrel{?}{=} 0\}}\ Pick_f$$

$$\frac{\{C_{p,i-1} \oplus C_{q,j-1} \stackrel{?}{=} 0\}}{\emptyset}\ Elim_C$$

– *Case 3: Apply the inference system* $\mathcal{I}_{i,j,f(C_{p,i-1}),f(f(C_{p,i-1}))}$ *to* $\{f(C_{p,i-1}) \oplus f(f(C_{p,i-1})) \stackrel{?}{=} 0\}$

$$\frac{\{f(C_{p,i-1}) \oplus f(f(C_{p,i-1})) \stackrel{?}{=} 0\}}{\{C_{p,i-1} \oplus f(C_{p,i-1}) \stackrel{?}{=} 0\}}\ Pick_f$$

$$\frac{\{C_{p,i-1} \oplus f(C_{p,i-1}) \stackrel{?}{=} 0\}}{\emptyset}\ Occurs_check$$

– *Case 4: Apply the inference system* $\mathcal{I}_{i,j,f(C_{q,j-1}),f(f(C_{q,j-1}))}$ *to* $\{f(C_{q,j-1}) \oplus f(f(C_{q,j-1})) \stackrel{?}{=} 0\}$

$$\frac{\{f(C_{q,j-1}) \oplus f(f(C_{q,j-1})) \stackrel{?}{=} 0\}}{\{C_{q,j-1} \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}}\ Pick_f$$

$$\frac{\{C_{q,j-1} \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}}{\emptyset}\ Occurs_check$$

– *Case 5: Apply the inference system* $\mathcal{I}_{i,j,f(C_{p,i-1}),f(f(C_{q,j-1}))}$ *to* $\{f(C_{p,i-1}) \oplus f(f(C_{q,j-1})) \stackrel{?}{=} 0\}$

$$\frac{\{f(C_{p,i-1}) \oplus f(f(C_{q,j-1})) \stackrel{?}{=} 0\}}{\{C_{p,i-1} \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}}\ Pick_f$$

$$\frac{\{C_{p,i-1} \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}}{\emptyset}\ Pick_{fail}$$

- *Case 6: Apply the inference system* $\mathcal{I}_{i,j,f(f(C_{p,i-1})),f(C_{q,j-1})}$ *to* $\{f(f(C_{p,i-1})) \oplus f(C_{q,j-1}) \stackrel{?}{=} 0\}$

$$\frac{\{f(C_{q,j-1}) \oplus f(f(C_{p,i-1})) \stackrel{?}{=} 0\}}{\{C_{q,j-1} \oplus f(C_{p,i-1}) \stackrel{?}{=} 0\}}\ Pick_f$$

$$\frac{\{C_{q,j-1} \oplus f(C_{p,i-1}) \stackrel{?}{=} 0\}}{\emptyset}\ Pick_{fail}$$

- *Algorithm 1 returns "secure".*

Theorem 1 (Soundness)**.** *For any mode of operation M, if Algorithm 1 returns "secure", then M is symbolically secure.*

Theorem 2 (Termination)**.** *For any mode of operation M, Algorithm 1 always terminates.*

5 Conclusions and Future Work

There has been a recent trend of automatic verification and synthesis of secure cryptosystems. In this paper, we introduce a sufficient condition (called "uniqueness property") for symbolic security and present an algorithm for checking the uniqueness property. We plan to apply this algorithm to automatically synthesize secure cryptographic modes of operation. The idea is to generate candidate modes of operation randomly and use our algorithm to filter out those that cannot be proved to be secure.

References

[1] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. Automated analysis and synthesis of authenticated encryption schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 84–95, 2015.

[2] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. Automated analysis and synthesis of block-cipher modes of operation. In *IEEE CSF*, pages 140–152, 2014.

[3] Andrew M. Marshall, Catherine Meadows, Paliath Narendran, Veena Ravishankar, and Brandon Rozek. Algorithmic problems in synthesized cryptosystems. In *34th International Workshop on Unification*, 2020.

[4] Catherine Meadows. Symbolic security criteria for blockwise adaptive secure modes of encryption. IACR Cryptol. ePrint Arch., 2020:794, 2020.

[5] Phillip Rogaway. Nonce-based symmetric encryption. In *11th International Workshop*, pages 348–359, 2004.

Formal Analysis of Symbolic Authenticity*

Hai Lin and Christopher Lynch

Clarkson University, Potsdam, NY, U.S.A
hlin@clarkson.edu clynch@clarkson.edu

Abstract

Authenticated encryption schemes are ways of encrypting messages which simultaneously assure the privacy and authenticity of data. Designing authenticated encryption schemes can be error-prone. In this paper, we are interested in the authenticity property of authenticated encryption schemes. We introduce the notion of symbolic authenticity, and present a decision procedure for verifying symbolic authenticity. This technique can be used to automatically synthesize authenticated encryption schemes.

1 Introduction

Authenticated encryption schemes (e.g. OCB [6], XCBC [1], etc) are ways of encrypting messages which simultaneously assure the privacy and authenticity of data. It is a nontrivial task to construct authenticated encryption schemes. Automated techniques have been used to verify and synthesize authenticated encryption schemes [2].

In this paper, we are interested in the authenticity property. Roughly speaking, an authenticated encryption scheme satisfies authenticity if an adversary cannot forge any new valid ciphertext message after observing as many ciphertext messages as he wants. Motivated by the original work in [4], we propose to reason about authenticity symbolically. We introduce the notion of symbolic authenticity, and present a decision procedure for checking symbolic authenticity. The idea is that we can use function symbols to model cryptographic operations (e.g. tweakable block cipher, exclusive-or, etc.). We can use terms to model message blocks, and use an equational theory to capture the properties of cryptographic operations and the properties that valid ciphertext messages must satisfy. We reduce the problem of checking symbolic authenticity to a new unification problem modulo the equational theory. The difficulty is that the equational theory can have an unbounded number of equations.

The rest of this paper is organized as follows. In Section 2, we recall the basics of authenticated encryption. In Section 3, we introduce the notion of symbolic authenticity. We then present an algorithm for checking symbolic authenticity in Section 4. We conclude and discuss future work in Section 5.

2 Preliminaries

In cryptography, a tweakable block cipher [3] on n-bit strings with tweak space $\mathcal{T}$ and key space $\mathcal{K}$ is a map E: $\mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ s.t. $E_K(T, \cdot)$ is a permutation on $\{0,1\}^n$ for any $K \in \mathcal{K}$ and $T \in \mathcal{T}$. Roughly speaking, each combination of a key and a tweak leads to a totally independent permutation. In this paper, we consider some tweakable block cipher with a fixed key K, which is not known to the adversary.

An authenticated encryption scheme Π is a tuple $(\mathcal{E}, \mathcal{D}, \mathcal{V})$ with key space $\mathcal{K}$, message space $\mathcal{M}$, tweak space $\mathcal{T}$ and tag space $\mathcal{G}$, where $\mathcal{E}$ is an encryption algorithm, $\mathcal{D}$ is a decryption

*The work is supported by NRL under contract N00173-19-1-G012.

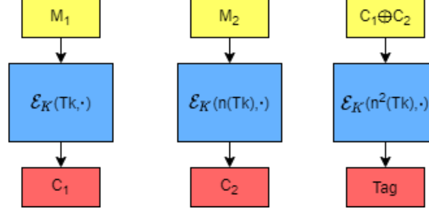


Figure 1: An Authenticated Encryption Scheme Π_1

algorithm and $\mathcal{V}$ is a verification equation, which checks authenticity. $\mathcal{E}$ uses some tweakable block cipher, and maps $(K, Tk, M) \in \mathcal{K} \times \mathcal{T} \times \mathcal{M}$ to a ciphertext $(Tk, C, Tg) \in \mathcal{T} \times \{0,1\}^* \times \mathcal{G}$. A ciphertext (C, Tg) is *valid* if and only if $\mathcal{V}_K(Tk, C, Tg)$ returns true. $\mathcal{D}$ maps $(K, Tk, C, Tg) \in \mathcal{K} \times \mathcal{T} \times \{0,1\}^* \times \mathcal{G}$ to either a message $M \in \mathcal{M}$ if (C, Tg) is valid or an error otherwise. In this paper, we only consider authenticated encryption schemes, which handle messages of fixed length.

We consider a first-order signature $\Sigma = \{e/2, d/2, n/1, \oplus/2, 0/0\}$. where e models encryption using tweakable block cipher: $e(t_1, t_2)$ is the encryption of t_2 using tweak t_1. d models decryption using tweakable block cipher: $d(t_1, t_2)$ is the decryption of t_2 using tweak t_1. In tweakable block cipher, each tweak can only be used to process a single block. If t is the tweak for processing the ith block of some message, then $n(t)$ is the tweak for processing the $i+1^{th}$ block of the same message. We use $n^k(t)$ as a shorthand for applying n to t for k times. 0 represents a block of all 0's. We use X to represent a set of variables: Tk denotes some tweak, Tg denotes some tag, and C_i denotes the ith block of some message. We use N to represent a set of constants: tk_i denotes the tweak for processing the first block of the ith message, tg_i denotes the tag of the ith ciphertext message, $m_{i,j}$ denotes the jth block of the ith plaintext message, $c_{i,j}$ denotes the jth block of the ith ciphertext message.

Example 1. *Consider the authenticated encryption scheme Π_1 in Figure 1. $\Pi_1 = (\mathcal{E}, \mathcal{D}, \mathcal{V})$, where*

$$\mathcal{E}_K(Tk, (M_1, M_2)) := (Tk, e(Tk, M_1), e(n(Tk), M_2), e(n^2(Tk), C_1 \oplus C_2)).$$
$$\mathcal{V}_K(Tk, (C_1, C_2), Tg) := (e(n^2(Tk), C_1 \oplus C_2) == Tg)$$
$$\mathcal{D}_K(Tk, (C_1, C_2), Tg) := \begin{cases} (d(Tk, C_1), d(n(Tk), C_2)) & \textit{if } \mathcal{V}_K(Tk, (C_1, C_2), Tg) == True \\ \perp & \textit{otherwise} \end{cases}$$

In order to check if an authenticated encryption satisfies authenticity, we consider a game between an adversary A and an encryption oracle $\mathcal{E}_K(\cdot, \cdot)$. The adversary queries the encryption oracle with plaintext messages, and gets back valid ciphertext messages. The adversary can choose plaintext messages adaptively based on previous queries. They can have as many rounds of interaction as they want. The adversary wins the game if and only if he can forge a new ciphertext message (Tk, C, Tg) s.t. $\mathcal{V}_K(Tk, C, Tg) == True$. An authenticated encryption scheme satisfies authenticity if and only if the adversary wins the game with negligible probability.

Example 2. *Consider the authenticated encryption scheme Π_1 in Figure 1. Suppose that there are two rounds of interaction between an adversary A and an encryption oracle $\mathcal{E}_K(\cdot, \cdot)$.*

- Round 1*: The adversary A queries the oracle with $(m_{1,1}, m_{1,2})$. $\mathcal{E}_K(\cdot, \cdot)$ replies with a valid ciphertext message $(tk_1, c_{1,1}, c_{1,2}, tg_1)$.*

- Round 2*: The adversary A queries the oracle with $(m_{2,1}, m_{2,2})$. $\mathcal{E}_K(\cdot,\cdot)$ replies with a valid ciphertext message $(tk_2, c_{2,1}, c_{2,2}, tg_2)$.*

The adversary can choose plaintext messages adaptively. For example, $m_{2,1}$ can be chosen as $c_{1,1}$, $m_{2,2}$ can be chosen as $c_{1,1} \oplus c_{1,2}$. We have the following set of equations: $E_\oplus$, E_1 and E_2. $E_\oplus$ captures the properties of exclusive-or. E_1 captures the fact that the ciphertext messages, which A receives, are valid. E_2 captures the fact that the adversary can choose plaintext messages adaptively.

$E_\oplus = \{t \oplus t = 0, t \oplus 0 = t\} \cup AC(\oplus)$
$E_1 = \{e(n^2(tk_1), c_{1,1} \oplus c_{1,2}) = tg_1, e(n^2(tk_2), c_{2,1} \oplus c_{2,2}) = tg_2\}$
$E_2 = \{d(tk_2, c_{2,1}) = c_{1,1}, d(n(tk_2), c_{2,2}) = c_{1,1} \oplus c_{1,2}\}$

After the above two rounds of interactions, the adversary can output a new valid ciphertext message: $(tk_1, c_{1,1} \oplus c_{1,2}, 0, tg_1)$. Therefore, Π_1 does not satisfy authenticity.

3 Symbolic Authenticity

In this section, we define the notion of symbolic authenticity of authenticated encryption schemes. We only consider authenticated encryption schemes, which handle messages of l blocks. We reduce the problem of checking symbolic authenticity to a new unification problem. First we formalize *verification equations* and *AE theories* w.r.t. a verification equation using the following definitions.

Definition 1. *An equation V of the form $e(n^k(Tk), t) = Tg$ is a verification equation if V satisfies the following property:*

- ***(Consistency)** (1) For all $e(s, s')$ and $e(t, t')$ that occur in V, if $s = s'$, then $t = t'$. (2) For all $d(s, s')$ and $d(t, t')$ that occur in V, if $s = s'$, then $t = t'$.*

We use $lhs(V)$ to denote $e(n^k(Tk), t)$.

The "consistency" property holds since in tweakable block cipher, each tweak can only be used to process a unique message block. We assume that, in the security game described in Section 2, the adversary receives an unbounded number of valid ciphertext messages. We define the following meta substitution ω_i^l, which will be used throughout the rest of this paper.

$$\omega_i^l = \{Tk \mapsto tk_i, C_1 \mapsto c_{i,1}, C_2 \mapsto c_{i,2}, \ldots, C_l \mapsto c_{i,l}\}$$

We can instantiate meta-substitutions by instantiating i and l. Let γ_1 and γ_2 be two meta-substitutions. $\gamma_1\gamma_2$ denotes the composition of γ_1 and γ_2. For example,
(1) $\omega_i^2 = \{Tk \mapsto tk_i, C_1 \mapsto c_{i,1}, C_2 \mapsto c_{i,2}\}$
(2) $\omega_1^2 = \{Tk \mapsto tk_1, C_1 \mapsto c_{1,1}, C_2 \mapsto c_{1,2}\}$
(3) $\omega_i^2\{C_3 \mapsto c_{i,3}\} = \{Tk \mapsto tk_i, C_1 \mapsto c_{i,1}, C_2 \mapsto c_{i,2}, C_3 \mapsto c_{i,3}\}$

Definition 2. *Let V be a verification equation. A set of ground equations E_V is an AE theory w.r.t. V if the following properties are satisfied:*

- ***(Validity)** $\forall i \in \mathcal{N}$, $V(\omega_i^l\{Tg \mapsto tg_i\}) \in E_V$.*
- ***(Consistency)** (1) For all $e(s, s')$ and $e(t, t')$ that occur in E_V, if $s = s'$, then $t = t'$. (2) For all $d(s, s')$ and $d(t, t')$ that occur in E_V, if $s = s'$, then $t = t'$.*

- ***(Stability)*** *No equation in* E_V *is of the form* $n(t) = t'$.

In Definition 2, the "Validity" property says that for all i, the ith ciphertext message $(tk_i, c_{i,1}, \ldots, c_{i,l}, tg_i)$ is valid. The "Stability" property says that the tweaks are irreducible in E_V.

Definition 3. *Let* V *be a verification equation,* E_V *is an AE theory w.r.t.* V. *Consider two terms s.t.* $t_1\omega_i^l = t_2$. t_1 *and* t_2 *are* $(\oplus, E_V)$*-unifiable under* σ *if*

- $t_1\sigma =_{\oplus, E_V} t_2$
- σ *is a* computable substitution, *meaning that each* C_j *can only be mapped to the exclusive-or of some constants and variables.*

σ is called a $(\oplus, E_V)$-unifier of t_1 and t_2. σ^* is the most general $(\oplus, E_V)$-unifier of t_1 and t_2 if for any $(\oplus, E_V)$-unifier σ of t_1 and t_2, there exists some substitution σ' s.t. $\sigma = \sigma^*\sigma'$.

To produce a valid message $m : (Tk, C, Tg)$, the adversary needs to compute some substitution σ s.t. $\mathcal{V}_K(Tk\sigma, C\sigma, Tg\sigma)$ returns true. Due to the "Validity" property, $\omega_i^l\{Tg \mapsto tg_i\}$ satisfies this requirement. But $(Tk\omega_i^l, C\omega_i^l, tg_i)$ is not new, it is the ith message that the adversary receives from the encryption oracle. Therefore, the adversary tries to find some other $(\oplus, E_V)$-unifier σ' of $lhs(V)$ and $lhs(V)\omega_i^l$. If the adversary succeeds, $(Tk\sigma', C\sigma', tg_i)$ is a new valid message.

Definition 4. *Let* S *be an authenticated encryption scheme with verification equation* V. *Let* E_V *be an AE theory w.r.t.* V. S *is symbolically authentic if the following condition holds.*

- ω_i^l *is the only* $(\oplus, E_V)$*-unifier of* $lhs(V)$ *and* $lhs(V)\omega_i^l$.

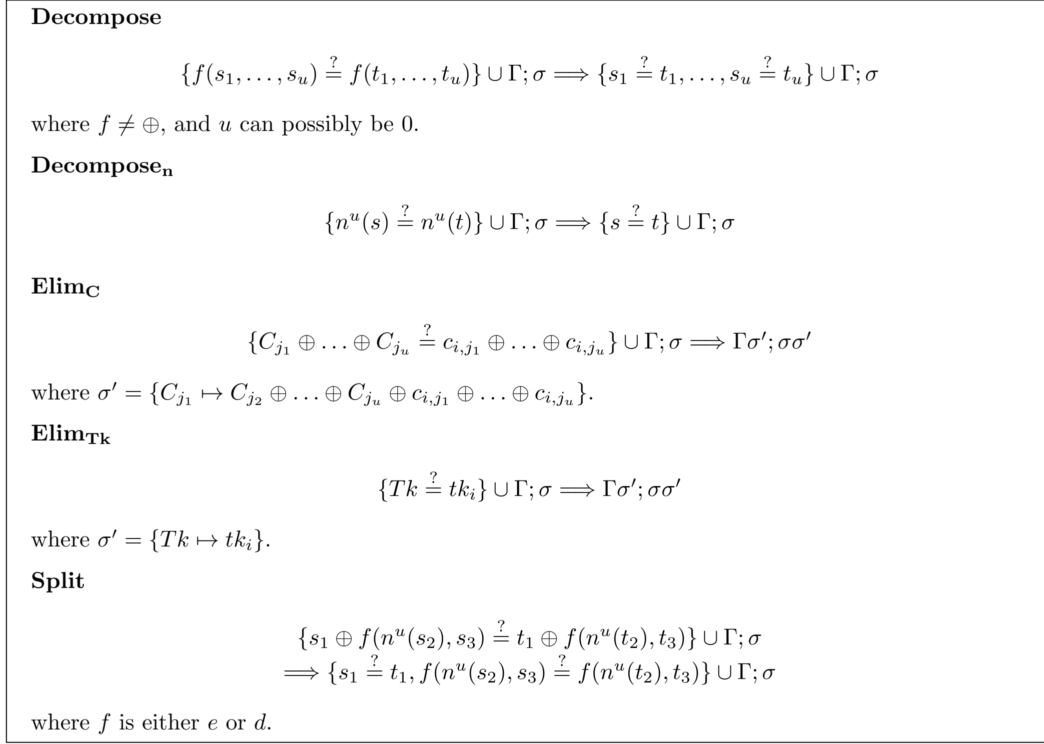
4 An Algorithm for Checking Symbolic Authenticity

Let V be a verification equation, E_V is an AE theory w.r.t. V. To check if two terms t_1 and t_2 are $(\oplus, E_V)$-unifiable, we apply the inference system $\mathcal{I}_B$ (Figure 2) to $\{t_1 \stackrel{?}{=} t_2\}$. We use $\mathcal{I}_B(t_1, t_2)$ to denote the result, which is the most general $(\oplus, E_V)$-unifier of t_1 and t_2. Let S be some authenticated encryption scheme, whose verification equation is V. Algorithm 1 checks if S satisfies authenticity, we compute the most general $(\oplus, E_V)$-unifier of $lhs(V)$ and $lhs(V)\omega_i^l$, and check if it is ω_i^l. If so, S satisfies authenticity. Otherwise, S does not satisfy authenticity. Algorithm 1 is sound, complete and terminating.

The inference rules of $\mathcal{I}_B$ are listed in Fig. 2. The *Decompose* rule is standard as in syntactic unification [5]. The $Decompose_n$ rule is an optimization rule: If we have an equation of the form $n^u(s) \stackrel{?}{=} n^u(t)$, instead of applying the *Decompose* rule u times, we can apply the $Decompose_n$ rule once. The standard *Variable Elimination* rule in syntactic unification may not lead to computable substitutions. Instead, we have the $Elim_C$ rule and the $Elim_{Tk}$ rule, which always lead to computable substitutions. The *Split* rule is the key rule in $\mathcal{I}_B$. The following example illustrates the idea behind the *Split* rule.

Example 3. *Let* V *be a verification equation, and* E_V *be an AE theory w.r.t.* V. *Consider the following inference step:*

$$\{C_1 \oplus e(tk_1, C_2) \stackrel{?}{=} c_{1,1} \oplus e(tk_1, c_{1,2})\} \stackrel{Split}{\Longrightarrow} \{C_1 \stackrel{?}{=} c_{1,1}, e(tk_1, C_2) \stackrel{?}{=} e(tk_1, c_{1,2})\}$$



Decompose

$$\{f(s_1,\ldots,s_u) \stackrel{?}{=} f(t_1,\ldots,t_u)\} \cup \Gamma;\sigma \Longrightarrow \{s_1 \stackrel{?}{=} t_1,\ldots,s_u \stackrel{?}{=} t_u\} \cup \Gamma;\sigma$$

where $f \neq \oplus$, and u can possibly be 0.

Decompose$_\mathbf{n}$

$$\{n^u(s) \stackrel{?}{=} n^u(t)\} \cup \Gamma;\sigma \Longrightarrow \{s \stackrel{?}{=} t\} \cup \Gamma;\sigma$$

Elim$_\mathbf{C}$

$$\{C_{j_1} \oplus \ldots \oplus C_{j_u} \stackrel{?}{=} c_{i,j_1} \oplus \ldots \oplus c_{i,j_u}\} \cup \Gamma;\sigma \Longrightarrow \Gamma\sigma';\sigma\sigma'$$

where $\sigma' = \{C_{j_1} \mapsto C_{j_2} \oplus \ldots \oplus C_{j_u} \oplus c_{i,j_1} \oplus \ldots \oplus c_{i,j_u}\}$.

Elim$_\mathbf{Tk}$

$$\{Tk \stackrel{?}{=} tk_i\} \cup \Gamma;\sigma \Longrightarrow \Gamma\sigma';\sigma\sigma'$$

where $\sigma' = \{Tk \mapsto tk_i\}$.

Split

$$\{s_1 \oplus f(n^u(s_2),s_3) \stackrel{?}{=} t_1 \oplus f(n^u(t_2),t_3)\} \cup \Gamma;\sigma$$
$$\Longrightarrow \{s_1 \stackrel{?}{=} t_1, f(n^u(s_2),s_3) \stackrel{?}{=} f(n^u(t_2),t_3)\} \cup \Gamma;\sigma$$

where f is either e or d.

Figure 2: Inference System $\mathcal{I}_B$

The first thing to observe is that: $\{C_1 \mapsto e(tk_1,C_2) \oplus c_{1,1} \oplus e(tk_1,c_{1,2})\}$ *is not a computable substitution. If* $(C_1 \oplus e(tk_1,C_2))\sigma =_{\oplus,E_V} c_{1,1} \oplus e(tk_1,c_{1,2})$*, there are two cases to consider:*

(1) $e(tk_1,C_2)\sigma =_{\oplus} e(tk_1,c_{1,2})$

(2) $e(tk_1,C_2)\sigma =_{\oplus} t$*, where* $t \in E_V$*. Due to the "Validity" property,* $e(tk_1,c_{1,2})$ *occurs in* E_V*. Due to the "Consistency" property,* $e(tk_1,c_{1,2})$ *is the only term in* E_V *s.t. its first argument is* tk_1*.*

In both cases, $e(tk_1,C_2)\sigma =_{\oplus} e(tk_1,c_{1,2})$*, which implies that* $C_1 =_{\oplus} c_{1,1}$*.*

Algorithm 1 Checking Symbolic Authenticity

Input: an authenticated encryption scheme S, whose verification equation is V.

if $\mathcal{I}_B(lhs(V), lhs(V)\omega_i^l) = \omega_i^l$ **then**
  return "authentic"
else
  return "inauthentic"
end if

The following example illustrates how Algorithm 1 can be used to check symbolic authenticity of Π_1 in Fig. 1.

Example 4. *The verification equation of* Π_1 *(Fig. 1) is the following:*

$$e(n^2(Tk), C_1 \oplus C_2) = Tg$$

According to Algorithm 1, we compute the most general $(\oplus, E_V)$*-meta-unifier of* $e(n^2(Tk), C_1 \oplus C_2)$ *and* $e(n^2(tk_i), c_{i,1} \oplus c_{i,2})$ *using the following inference steps:*

$$\begin{aligned}
&\{e(n^2(Tk), C_1 \oplus C_2) \stackrel{?}{=} e(n^2(tk_i), c_{i,1} \oplus c_{i,2})\}; id \\
\Longrightarrow\ &\{n^2(Tk) \stackrel{?}{=} n^2(tk_i), C_1 \oplus C_2 \stackrel{?}{=} c_{i,1} \oplus c_{i,2}\}; id && \text{(Decompose)} \\
\Longrightarrow\ &\{Tk \stackrel{?}{=} tk_i, C_1 \oplus C_2 \stackrel{?}{=} c_{i,1} \oplus c_{i,2}\}; id && \text{(Decompose}_n\text{)} \\
\Longrightarrow\ &\{C_1 \oplus C_2 \stackrel{?}{=} c_{i,1} \oplus c_{i,2}\}; \{Tk \mapsto tk_i\} && \text{(Elim}_{Tk}\text{)} \\
\Longrightarrow\ &\emptyset; \{C_1 \mapsto C_2 \oplus c_{i,1} \oplus c_{i,2}, Tk \mapsto tk_i\} && \text{(Elim}_C\text{)}
\end{aligned}$$

$\{C_1 \mapsto C_2 \oplus c_{i,1} \oplus c_{i,2}, Tk \mapsto tk_i\} \neq \omega_i^2$. *Therefore, S is not symbolically authentic. In fact, instantiating i using any natural number leads to a valid new ciphertext message. For example,* $(tk_1, C_2 \oplus c_{1,1} \oplus c_{1,2}, C_2, tg_1)$ *is a valid new ciphertext message, where C_2 can be an arbitrary message block, and C_1 is* $C_2 \oplus c_{1,1} \oplus c_{1,2}$.

5 Conclusions and Future Work

In this paper, we consider authenticity of authenticated encryption schemes. We propose the notion of symbolic authenticity and present a decision procedure for checking symbolic authenticity. We plan to apply this technique to automatically synthesize authenticated encryption schemes that satisfy authenticity. The idea is to generate candidate authenticated encryption schemes randomly and use our algorithm to filter out those that are not symbolically authentic.

References

[1] Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: XCBC encryption and xecb authentication modes. In *Fast Software Encryption (FSE) 2001*, page 92—108, 2002.

[2] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemof. Automated analysis and synthesis of authenticated encryption schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, page 84—95, 2015.

[3] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In *Advances in Cryptology—Crypto 2002*, page 31—46, 2002.

[4] Catherine Meadows. Symbolic security criteria for blockwise adaptive secure modes of encryption. IACR Cryptol. ePrint Arch., 2020:794, 2020.

[5] Alan Robinson and Andrei Voronkov. *Handbook of Automated Reasoning.* 2001.

[6] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *8th ACM Conference on Computer and Communications Security (CCS)*, page 196—205, 2001.

What, Again? Automatic Deductive Synthesis of the Unification Algorithm

Richard Waldinger, Artificial Intelligence Center, SRI International

Deductive program synthesis is an approach to automated programming in which the programming task is regarded as a problem in mathematical theorem proving. A logical statement that specifies the purpose of the desired program is treated as a theorem to be proved. The theorem expresses the existence of an output entity that satisfies the specification. The proof is restricted to be sufficiently constructive to indicate a method for finding that output, and a program that employs the method is derived from the proof. The proof also constitutes a verification of the correctness of this program. There are generally many proofs of the theorem and many corresponding programs.

The Earlier Specification

Unification [Herbrand 30] [Robinson 65] was an early target of program synthesis efforts, but the results fell short of a fully automatic derivation, one in which the program was obtained from the specification without human participation. [Manna and Waldinger 81] provided a derivation, but it was purely manual. [Paulson 85] described an interactive verification using LCF, but took the program as a given, so it could not be regarded as a synthesis. [Eriksson 84], [Nardi 89], and [Armando et al. 97] produced partial derivations interactively, not completely automatically. So, a fully automatic synthesis of a unification algorithm can still be regarded as a research goal.

Although Robinson used the term "most-general unifier," he never gave a specification for the unification algorithm or defined what it meant for one unifier to be more general than another. Since then, a theory of expressions and substitutions has been worked out (see, e.g., [Baader Snyder 01].) In [Manna and Waldinger 81] we defined a substitution θ_1 to be more general than θ_2 if, for some substitution σ, $\theta_1 \blacklozenge \sigma = \theta_2$, where $\blacklozenge$ is the composition operator. We started by specifying that the unification algorithm should find a most-general unifier for the given two terms e_1 and e_2 if the terms are unifiable; otherwise, it should return the special failure entity $\perp$.

We found that the proof would not go through unless we (by hand) strengthened the specification to require that the most-general unifier we find be *idempotent*. (An idempotent substitution θ is one such that $\theta \blacklozenge \theta = \theta$.) The strengthened specification gave us the benefit of a stronger induction hypothesis. We observed that, in the case in which θ_1 is idempotent, the condition that θ_1 is more general than θ_2 is equivalent to $\theta_1 \blacklozenge \theta_2 = \theta_2$. We shall say that a substitution θ_1 is *idempotently more general* than another substitution θ_2, denoted by $\theta_1 \succcurlyeq_{gen} \theta_2$, if $\theta_1 \blacklozenge \theta_2 = \theta_2$. In this case we shall also

say that θ_2 is an *extension* of θ_1. For our synthesis work, we have found it convenient to use this stronger notion of more-generality.

In [Waldinger 2020] we attempted to automate the derivation laid out in [Manna and Waldinger 81]. We simplified the specification by a change in nomenclature: we regarded the failure entity $\bot$ as an *improper* substitution, with the property that, for any term e, $e \blacktriangleleft \bot = blackhole$, where *blackhole* is a constant. As a consequence, the improper substitution $\bot$ "unifies" all expressions, but we don't regard two expressions as unifiable unless they have a *proper* unifier, distinct from $\bot$. (We say that such a substitution θ satisfies is-subst(θ).) It holds that $\theta \blacklozenge \bot = \bot$ and hence that $\theta \succcurlyeq_{gen} \bot$ for all substitutions θ. In other words, any substitution is idempotently more general than the failure substitution. Also, a substitution θ is idempotent if and only if $\theta \succcurlyeq_{gen} \theta$.

In the earlier paper, we took the specification for the unification algorithm to be the remarkably concise

$$\text{unify}(e_1, e_2) \Leftarrow \text{find } \theta \text{ such that mgiu}(\theta, e_1, e_2),$$

where mgiu(θ, e_1, e_2), that θ is a most-general idempotent unifier of e_1 and e_2, is taken to mean that

$$\begin{array}{l} e_1 \blacktriangleleft \theta = e_2 \blacktriangleleft \theta \text{ and} \\ (\forall \theta')\ [\text{if } e_1 \blacktriangleleft \theta' = e_2 \blacktriangleleft \theta' \text{ and} \\ \qquad\quad \text{is-subst}(\theta') \\ \qquad \text{then } \theta \succcurlyeq_{gen} \theta']. \end{array}$$

In other words, θ is a unifier and is idempotently more general than any proper unifier. This implies that the unifier is most general and idempotent. Note that, in the case in which e_1 and e_2 are not unifiable, this specification requires that θ be the failure entity $\bot$, because that is the only unifier of non-unifiable terms. The fact that the specification does not treat non-unifiability as a special case simplifies the proof and the program as well as the specification, enabling us to avoid several case analyses. The theorem prover discovered a simpler program than the one we expected, and one that was marginally more efficient because it avoided some conditional tests.

Introducing an Accumulator

The automation of the synthesis had not been completed when the work-in-progress was reported at the Workshop on Logic and Practice of Programming (LPOP 2020). During the discussion, however, the question came up whether the same techniques would allow the system to derive a more efficient unification algorithm. Unless efficiency is specified, nothing guarantees that the theorem prover will come up with an efficient algorithm. But one more efficient algorithm takes as an additional input an initial substitution θ_0, which keeps track of the partial unifier discovered so

far in the unification process.[1] We assume that the initial (accumulator) substitution is idempotent and require that the output most-general unifier be an extension of the accumulator. Our new specification is

$unify(\theta_0, e_1, e_2) \Leftarrow$ find θ such that $mgiu(\theta_0, \theta, e_1, e_2)$,

where $mgiu(\theta_0, \theta, e_1, e_2)$ is taken to mean that

$$
\begin{aligned}
&\text{if } \theta_0 \succcurlyeq_{gen} \theta_0 \\
&\text{then } \theta_0 \succcurlyeq_{gen} \theta \text{ and} \\
&\qquad e_1 \blacktriangleleft \theta = e_2 \blacktriangleleft \theta \text{ and} \\
&\qquad (\forall \theta')\,[\text{if } e_1 \blacktriangleleft \theta' = e_2 \blacktriangleleft \theta' \text{ and} \\
&\qquad\qquad \text{is-subst}(\theta') \text{ and} \\
&\qquad\qquad \theta_0 \succcurlyeq_{gen} \theta' \\
&\qquad\quad \text{then } \theta \succcurlyeq_{gen} \theta'].
\end{aligned}
$$

In other words, we seek a unifier that is an extension of the accumulator and that is idempotently more general that any unifier that is an extension of the accumulator, assuming that the accumulator is idempotent (i.e., $\theta_0 \succcurlyeq_{gen} \theta_0$). Initially we take θ_0 to be the empty substitution {}. Because {} is idempotent and is idempotently more general than any substitution, this specification reduces to our original specification for the unification algorithm. But this seemingly more complex and general specification leads to a simpler synthesis proof as well as a more efficient unification algorithm.

Mathematical Induction for Program Synthesis

As we have said, in deductive program synthesis we regard programming as a task in theorem proving. To construct a program that, for a given input a, returns an output z that satisfies a specified input-output condition, we prove the existence of an output entity z that satisfies the condition. In other words, given a specification of the form

$f(a) \Leftarrow$ find z such that $Q[a, z]$,

we attempt to prove the theorem $(\forall a)(\exists z)Q[a, z]$. The proof is restricted to be sufficiently constructive so that a program that satisfies the specification can be extracted.

---

[1] This is analogous to a derivation of a program for exponentiation: instead of deriving a program to compute $exp(a, b) \Leftarrow a^b$, we construct a more general program $exp(c, a, b) \Leftarrow c * a^b$. Here the accumulator c keeps track of intermediate values of the computation. Initially, we take c to be 1, so the value will be the desired exponentiation. Accumulators have long been employed in program transformation; e.g., see [Wegbreit 76] and [Burstall and Darlington 77]

The structure of the program reflects the structure of the proof from which it was extracted. A case analysis in the proof may produce a conditional expression in the extracted program. The use of the principle of mathematical induction in the proof may yield a recursive call in the program. A more leisurely introduction to the introduction of recursion in program synthesis is given in [Manna and Waldinger 81].

A well-founded relation is one that, like the natural numbers with <, admits no infinite decreasing sequences. Our induction is *well-founded induction*: For a given input a, we try to find an entity z that satisfies the input-output condition Q[a, z]. We may assume inductively that the program f we are trying to construct will satisfy the input-output condition for all inputs that are less than the given input a with respect to a well-founded relation $\prec_w$. In other words, we conduct the proof with the help of the induction hypothesis

if $x \prec_w a$
then $Q[x, f(x)]$.

For the unification algorithm, we assume the induction hypothesis

if $\langle \theta_0', e_1', e_2' \rangle \prec_w \langle \theta_0, e_1, e_2 \rangle$
then $mgiu(\theta_0', unify(\theta' e_1', e_2'), e_1', e_2')$.

The well-founded relation $\prec_w$ is not specified in advance; w is a variable that ranges over well-founded relations. We actually prove a theorem of form $(\exists\, w)(\forall\, a)(\exists\, z)Q[a, z]$. It is not realistic to expect the theorem prover to guess the relation w until the proof is under way. Instead, we extract the definition of the relation from the proof, by the same mechanism by which the program itself is extracted. The proof is conducted in the context of the axiomatic theory of expressions and substitutions. We provide several primitive well-founded relations, such as the size of a term and the number of variables it contains; we expect the theorem prover to discover a lexicographic combination of these relations that will allow the proof to go through, but this part of the proof has not yet been automated. In our experiments, we have temporarily provided the actual lexicographical combination of well-founded relations needed.

Our experiments are conducted using the theorem prover SNARK [Stickel et al. 00], a first-order resolution theorem prover which contains advanced capabilities for extracting answers, programs, and other information from proofs. Program synthesis is a challenging application, partly because it requires us to deal with full (universal and existential) quantification and mathematical induction. Typically, automatic theorem provers that focus on induction (e.g., ACL2) do not deal with theorems with explicit existential quantifiers, while resolution theorem provers do not deal with induction at all. Furthermore, interesting program synthesis requires case analysis---otherwise, how else do we introduce conditional programs? But resolution theorem provers are not so good at

case analysis. Furthermore, our approach requires us to prove the existence of a suitable well-founded relation, which would most easily be achieved in a higher-order-logic setting. But as far as we can tell, existing automatic higher-order-logic theorem provers do not do program extraction at all, let alone the formation of conditional programs.

To quantify over relations in a first-order setting, we *reify* relations. In other words, when we mean that, say, $x \prec_w y$, we actually write holds(w, x, y), where w is a variable that ranges over relations. Our theory also contains functions over relations; in the synthesis proof, we use the lexicographic function lex. If $\prec_{w_1}$ and $\prec_{w_2}$ are relations, their lexicographic combination $\prec_{lex(w_1, w_2)}$ is defined so that $x \prec_{lex(w_1, w_2)} y$ if and only if

$x \prec_{w_1} y$ or
$(x \preccurlyeq_{w_1} y$ and $x \prec_{w_2} y)$.

If $\prec_{w_1}$ and $\prec_{w_2}$ are well-founded, $\prec_{lex(w_1, w_2)}$ is also well-founded.

Extracted Program Fragments

In our experiments we have restricted our attention to symbolic expressions, like LISP S-expressions, in which the only function is the cons function $\bullet$. (This is not a substantive simplification—for one thing, any functional term can be encoded as an S-expression.)

While the proof is not complete, we have extracted program fragments for particular subcases. For instance, in the part of the base case in which e_1 is a variable and both e_1 and e_2 *evade* θ_0, we obtain

$$
\begin{aligned}
\text{unify}(\theta_0, e_1, e_2) \Leftarrow\ & \text{if } e_1 \in e_2 \\
& \text{then } \bot \\
& \text{else if } e_1 = e_2 \\
& \qquad \text{then } \theta_0 \\
& \qquad \text{else } \theta_0 \blacklozenge \{e_1 \leftarrow e_2\}.
\end{aligned}
$$

Here $\{e_1 \leftarrow e_2\}$ is the *replacement* substitution, which replaces all occurrences of e_1 with e_2. The famous *occurs-check* $e_1 \in e_2$, that is, e_1 is a proper-subexpression of e_2, has been introduced as a result of a case analysis in the proof. (When we say e *evades* θ, we mean $e \blacktriangleleft \theta = e$.) SNARK's proof, in clause form and with a more readable explanation, occurs at
http://www.ai.sri.com/coffee/unif2021-occurs-check-proof-explanation.pdf

In the non-atomic case, in which both e_1 and e_2 are conses, SNARK obtains the astonishingly simple tail-recursive program

$$unify(\theta_0, e_1, e_2) \Leftarrow unify(unify(\theta_0, left(e_1), left(e_2)), right(e_1), right(e_2)).$$

Here, left and right decompose conses, i.e., $left(e_1 \bullet e_2) = e_1$ and $right(e_1 \bullet e_2) = e_2$. The proof from which this program was extracted uses two instances of the induction hypothesis, one for each recursive call. While we expected the program to require conditional expressions for the cases in which the left halves or the right halves were not unifiable, SNARK observed that this was unnecessary.

While the theorem prover does not establish the complexity of the extracted program, by good luck it has obtained a more efficient program than the one in LPOP 2020. That program computed unifiers for the left and right subexpression of the arguments and then composed them, an expensive operation. The only composition the new program does is $\theta_0 \blacklozenge \{e_1 \leftarrow e_2\}$, i.e., to post-pend a single replacement to a given substitution---relatively cheap!

Concluding Remark

One might argue that synthesizing a unification algorithm is pointless, since the algorithm is already known and, in fact, the theorem prover requires a unification algorithm to conduct the proof. But, aside from its value as an exercise, developing a system that can synthesize unification algorithms may enable us on the fly to construct algorithms for new special theories, for which the unification problem is still unsolved. To do this, we would incorporate axioms for the new theory into our theory of expressions and substitutions. And we could consider related problems, such as anti-unification and matching. But first things first.

Acknowledgments: We are grateful for discussions with researchers from SRI International (including the Artificial Intelligence Seminar, the Crazy Idea Seminar, and the Coffee @ 4 meeting), the Kestrel Institute, and the LPOP 2020 workshop. We have benefited from suggestions from the workshop referees, Y. Annie Liu, David S. Warren, Alessandro Coglio, and Karthik Nukala.

References

[Armando et al. 97]
Armando, A., Smaill, A, and Green, I. (1997). *Automatic synthesis of recursive programs: the proof-planning paradigm.* In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering,* pp. 2–9. IEEE.

[Baader Snyder 01]
Baader, F, and Snyder, W. (2001). Unification Theory. In *Handbook of Automated Reasoning (1),* Robinson, J. A., and Voronkov, A. (eds.), pp. 447–533. Elsevier Science Publishers.

[Burstall and Darlington 77]
Burstall, R. M., and Darlington, J. (1977, January). A Transformation System for Developing Recursive Programs. *Journal of the Association for Computing Machinery (*24:1), pp 44-67. ACM.

[Eriksson 84]
Eriksson, L. H. (1984) . Synthesis of a unification algorithm in a logic programming calculus, *The Journal of Logic Programming* (1:1), pp. 3–18.

[Herbrand 30]
Herbrand, J. (1930). *Recherches sur la théorie de la* démonstration
PhD thesis, Université de Paris.

[Manna and Waldinger 81]
Manna, Z, and Waldinger, R. (1981). Deductive synthesis of the unification algorithm. *Science of Computer Programming* (1) pp. 5–48.

[Nardi 89]
Nardi, D. (1989). Formal synthesis of a unification algorithm by the deductive-tableau method. *The Journal of Logic Programming* (7:1), pp. 1–43.

[Paulson 85]
L. C. Paulson (1985). Verifying the unification algorithm in LCF. *Science of Computer Programming* (5), pp. 143–170.

[Robinson 65]
Robinson, J. A. (1965, January). A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery* (12:1) pp. 23–41. ACM.

[Stickel et al. 00]

Stickel, M., Waldinger, R. J., and Chaudhri, V. (2000*). A Guide to SNARK.* SRI International, Menlo Park, California, USA.

[Waldinger 2020]
Waldinger, R. (2020). Deductive Synthesis of the Unification Algorithm: The Automation of Introspection. In *Proceedings of the 2nd Workshop on Logic and Practice of Programming (LPOP),* pp. 12–21.

[Wegbreit 76]
Wegbreit, B. (1976, June). Goal-Directed Program Transformation*. IEEE Transactions on Software Engineering* (SE-2, 2) pp. 69–80. IEEE.

When First-order Unification Calls itself*

David M. Cerna[1][2]

[1] Czech Academy of Sciences Institute of Computer Science (CAS ICS),
Prague Czechia
[2] Research Institute for Symbolic Computation (RISC), Johannes Kepler University (JKU),
Linz, Austria
dcerna@{cs.cas.cz,risc.jku.at}

Abstract

We present a unification problem based on first-order syntactic unification which ask whether every problem in a particular infinite sequence of unification problems is unifiable. The restricted structure of our sequence of unification problems allows an alternative formulation of the problem as recursively calling first-order syntactic unification on certain bindings if the unifier has a particular structure. The latter formulation allows us to conjecture a sufficient condition for unifiability of the sequence based on the structure of a finite sequence of unifiers. It remains an open whether this condition is also necessary.

1 Introduction

We present a unification problem based on first-order syntactic unification which ask whether every problem in a particular type of infinite sequences of unification problems is unifiable. This problem denotes a special case of the *schematic unification* discussed in [3]. Overall this work is related to earlier results concerning propositional and first-order schemata [1, 4].

In this short paper we briefly discuss the general form of the problem before focusing on an important special case when every problem in the infinite sequence has the same right (left) side; what we refer to as *semiloop unification*. While this seems like a significant simplification, the problem remains difficult. The main difficulty is showing that it is possible to decide unifiability of the entire sequence from unifiability of a finite segment. In particular, when the solved form contains so called *recursion variables* which we discuss shortly. In our framework the existence of recursion variables in the solved form implies that a part of the unification problem was left unsolved and must be consider by the next problem in the sequence. It is not clear whether a termination condition exists as some sequences of problems exhibit complex behavior.

2 Problem Description

We assume knowledge of syntactic first-order unification (See [2]).

2.1 Variable Classes

Let $\mathcal{V}$ be a countably infinite set of variable symbols. A *variable class* is a pair $(Z, <)$, where $Z \subset \mathcal{V}$ (countably infinite) and $<$ is a strict well-founded total linear order. Associated with each variable class $(Z, <)$ is a *successor function* $Suc^{Z}_{<}(\cdot)$ *of the class* which has the following properties:

*Supported by the Linz Institute of Technology (LIT) Math$_{LP}$ project (LIT- 2019-7-YOU-213) funded by the state of upper Austria.

1) If $x \in Z$, then $x < Suc^{Z}_{<}(x)$

2) if $x, y \in Z$ and $Suc^{Z}_{<}(x) < Suc^{Z}_{<}(y)$ then $x < y$

We write $Suc(\cdot)$ or $Suc^{Z}(\cdot)$ when the parameters are clear from context.

To simplify notation we will consider classes of the form $V^{x}_{\mathbb{N}} = (\{x_i \mid i \in \mathbb{N}\}, <_{\mathbb{N}})$ where $<_{\mathbb{N}}$ is the strict well-founded total linear order of the natural numbers and $x \notin \{x_i \mid i \in \mathbb{N}\}$. The successor function associated with $V^{x}_{\mathbb{N}}$ is defined as $Suc(x_i) = x_{i+1}$. Note that this successor function satisfies the above properties when x_0 is the minimal element with respect to $<_{\mathbb{N}}$. Let $x_i \in V^{x}_{\mathbb{N}}$, then $|x_i| = i$. Unless otherwise stated, if $x \neq y$, then the classes $V^{x}_{\mathbb{N}}$ and $V^{y}_{\mathbb{N}}$ contain distinct variables.

2.2 Semiloop Unification

In addition to a standard first-order term signature Σ we require a countably infinite set of *recursion variables* $\mathcal{R}$. Recursion variables will be denoted using $\hat{\cdot}$. Note, substitutions are defined over $\mathcal{V} \cup \mathcal{R}$. By $\mathcal{T}(\Sigma, S, Z)$ we denote the term algebra whose members are constructed using the signature Σ, $Z = (\bigcup_{x \in X} V^{x}_{\mathbb{N}})$ where $X \subset \mathcal{V}$, and $S \subset \mathcal{R}$. For $t \in \mathcal{T}(\Sigma, S, Z)$, $var(t)$ denotes the set of variables occurring in t.

Given a term $t \in \mathcal{T}(\Sigma, S, Z)$, we can generate the *successor term of t modulo V*, for $V \subseteq Z$ by applying the shift operator defined recursively as follows:

- $\mathbf{s}^{V}(f(t_1, \cdots, t_n)) = f(\mathbf{s}^{V}(t_1), \cdots, \mathbf{s}^{V}(t_n))$
- for $z \in V$, $\mathbf{s}^{V}(z) = Suc(z)$
- for $\hat{a} \in S$, $\mathbf{s}^{V}(\hat{a}) = \hat{a}$

When V is clear from context we will simply write $\mathbf{s}(\cdot)$.

Definition 1. *Let $s \in \mathcal{T}(\Sigma, \{\hat{a}\}, (\bigcup_{z \in Z} V^{z}_{\mathbb{N}}))$ and $t \in \mathcal{T}(\Sigma, \emptyset, (\bigcup_{z \in Z} V^{z}_{\mathbb{N}}))$. Then we refer to s as* $(\Sigma, Z, \hat{a})$-extendable *and t as* $(\Sigma, Z, \hat{a})$-fixed. *Associated with each $(\Sigma, Z, \hat{a})$-extendable term s is an unary operator $\mathbf{ex}^{s}_{\hat{a}}(\cdot)$ defined recursively as follows:*

- $\mathbf{ex}^{s}_{\hat{a}}(f(t_1, \cdots, t_n)) = f(\mathbf{ex}^{s}_{\hat{a}}(t_1), \cdots, \mathbf{ex}^{s}_{\hat{a}}(t_n))$
- *for $z \in Z$, $\mathbf{ex}^{s}_{\hat{a}}(z) = z$*
- $\mathbf{ex}^{s}_{\hat{a}}(\hat{a}) = s$
- *for $\hat{b} \in \mathcal{R}$ such that $\hat{a} \neq \hat{b}$, $\mathbf{ex}^{s}_{\hat{a}}(\hat{b}) = \hat{b}$*

Both $\mathbf{s}(\cdot)$ and $\mathbf{ex}^{s}_{\hat{a}}(\cdot)$ may be applied to substitutions as follows: $\mathbf{s}(\sigma) = \{\mathbf{s}(y) \mapsto \mathbf{s}(t) \mid y\sigma = t\}$, and $\mathbf{ex}^{s}_{\hat{a}}(\sigma) = \{y \mapsto \mathbf{ex}^{s}_{\hat{a}}(t) \mid y\sigma = t\}$.

Definition 2. *Let s be $(\Sigma, Z, \hat{a})$-extendable, t be $(\Sigma, Z, \hat{b})$-extendable. Then we refer to the pair $\langle s, t\rangle$ as a* $(\Sigma, Z, \hat{a}, \hat{b})$-loop. *When it is clear from context we will write loop for $(\Sigma, Z, \hat{a}, \hat{b})$-loop.*

An important subclass of loops is the class of semiloops:

Definition 3. *Let s be $(\Sigma, Z, \hat{a})$-extendable and r (Σ, Z)-fixed. Then we refer to the pair $\langle s, r|$ ($|r, s\rangle$) as a* left (right) $(\Sigma, Z, \hat{a})$-semiloop. *When it is clear from context we will write semiloop for $(\Sigma, Z, \hat{a})$-semiloop.*

Note, that left and right semiloops are defined symmetrically. For the rest of this work, when discussing semiloops, we will exclusively consider left semiloops unless we state otherwise.

Definition 4. *Let $\langle s,t\rangle$ be a $(\Sigma, Z, \hat{a}, \hat{b})$-loop and $n \in \mathbb{N}$. The n-extension of $\langle s,t\rangle$, denoted by $\langle s,t\rangle_n$, may be defined recursively as follows:*

- $\langle s,t\rangle_0 = \langle \hat{a}, \hat{b}\rangle$
- $\langle s,t\rangle_{n+1} = \langle \mathbf{ex}^s_{\hat{a}}(\mathbf{s}(s')), \mathbf{ex}^t_{\hat{b}}(\mathbf{s}(t'))\rangle$ *where* $\langle s,t\rangle_n = \langle s',t'\rangle$.

In the case of semiloops application of the operators may be ignored for the fixed term, i.e. $\langle s,t|_0 = \langle \hat{a}, t|$, and $\langle s,t|_{n+1} = \langle \mathbf{ex}^s_{\hat{a}}(\mathbf{s}(s')), t|$ where $\langle s,t|_n = \langle s',t|$.

Example 1. *The following pairs of terms are $(\{h\}, \{V^x_{\mathbb{N}}, V^y_{\mathbb{N}}\}, \hat{a}, \hat{b})$-loops:*

a) $\langle h(h(x_1, h(x_1, h(x_1,x_1))), \hat{a})\ ,\ h(h(h(y_1,y_2), h(y_4,y_2)), \hat{b})\rangle$

b) $\langle h(h(x_6, h(x_1,x_6)), x_4)\ ,\ h(y_1, h(y_2,y_1))\rangle$

c) $\langle h(h(x_6, h(x_1,x_6)), \hat{a})\ ,\ h(y_1, h(y_2,,y_1))|$

Note, (c) is a semiloop.

Example 2. *Below we construct the first few extensions of the semiloop*

$$\langle s,t| = \langle h(h(x_6, h(x_1,x_6)), \hat{a})\ ,\ h(y_1, h(y_2,y_1))|$$

- $\langle s,t|_0 = \langle \hat{a}\ , t|$
- $\langle s,t|_1 = \langle h(h(x_6, h(x_1,x_6)), \hat{a})\ , t|$
- $\langle s,t|_2 = \langle h(h(x_7, h(x_2,x_7)), h(h(x_6, h(x_1,x_6)), \hat{a}))\ , t|$
- $\langle s,t|_3 = \langle (h(h(x_8, h(x_3,x_8)), h(h(x_7, h(x_2,x_7)), h(h(x_6, h(x_1,x_6)), \hat{a})))\ , t|$

When unifying the terms of a loop we will consider recursion variables as a type of variable with the following restrictions: Let σ be the m.g.u. of $s \stackrel{?}{=} t$ where $\{x \mapsto t'\} \in \sigma$, then for $\hat{a} \in \mathcal{R}$, $x = \hat{a}$ iff $t' \notin \mathcal{V}$. The idea behind this restriction is that variables have a higher precedence than recursion variables with respect to unification.

Definition 5. *Let $\langle s,t\rangle$ be a $(\Sigma, Z, \hat{a}, \hat{b})$-loop such that solving $s \stackrel{?}{=} t$ results in an m.g.u. σ. We refer to $\langle s,t\rangle$ as* extendably unifiable *if $Dom(\sigma) \cap \{\hat{a}, \hat{b}\} \neq \emptyset$.*

Example 3. *Consider the semiloop $\langle h(h(x_6, h(x_1,x_6)), \hat{a})\ ,\ h(y_1, h(y_2,y_1))|$ once again. The unification problem $h(h(x_6, h(x_1,x_6)), \hat{a}) \stackrel{?}{=} h(y_1, h(y_2,y_1))$ has a solved form $\{y_1 \mapsto h(x_6, h(x_1,x_6))\ ,\ \hat{a} \mapsto h(y_2, h(x_6, h(x_1,x_6)))\}$ and thus is extendably unifiable. A slight variation, namely $\langle h(h(x_6, h(x_1,x_6)), \hat{a})\ ,\ h(y_1,y_2)|$, is not extendably unifiable as the solved form is $\{y_1 \mapsto h(x_6, h(x_1,x_6))\ ,\ y_2 \mapsto \hat{a}\}$.*

Definition 6 (Loop Unification Problem)**.** *Let $X \subseteq Z$. Given a $(\Sigma, Z, \hat{a}, \hat{b})$-loop $\langle s,t\rangle$ such that s is $(\Sigma, (Z \setminus X), \hat{a})$-extendable and t is $(\Sigma, X, \hat{b})$-extendable, the* loop unification problem, *denoted by $s \stackrel{\circlearrowleft}{=} t$, is the problem of deciding if every extension of $\langle s,t\rangle$ is unifiable. We refer to such loops as* loop unifiable.

Example 4. *Consider:* $\langle h(h(h(x_2, h(x_1, x_1)), x_3), \hat{a})\ ,\ h(h(y_4, y_3), h(y_1, y_2))|$. *The* 0*-extension is always unifiable so we can ignore it. The* 1*-extension has the following unifier:* $\{y_3 \mapsto x_3\ ,\ y_4 \mapsto h(x_2, h(x_1, x_1))\ ,\ \hat{a} \mapsto h(y_1, y_2)\}$. *Thus, the* 1*-extension is extendably unifiable. What about the* 2*-extension?*

$$\langle h(h(h(x_3, h(x_2, x_2)), x_4), h(h(h(x_2, h(x_1, x_1)), x_3), \hat{a}))\ ,\ h(h(y_4, y_3), h(y_1, y_2))|$$

It is unified by: $\{y_3 \mapsto x_4\ ,\ y_4 \mapsto h(x_3, h(x_2, x_2))\ ,\ y_1 \mapsto h(h(x_2, h(x_1,\ x_1)), x_3)\ ,\ y_2 \mapsto \hat{a}\}$. *Notice that this extension is not extendably unifiable. From this unifier we can build a unifier for every extension greater than 2. Due to space we do not go into detail concerning the construction.*

Example 5. *Let* $\langle s, t| = \langle h(h(h(x_2, x_1), h(x_2, x_3)), \hat{a}), h(h(y_3, y_1), h(y_4, y_4))|$. *The* 1*-extension has the following unifier:* $\{y_3 \mapsto h(x_2, x_1)\ ,\ y_1 \mapsto h(x_2, x_3)\ ,\ \hat{a} \mapsto h(y_4, y_4)\}$. *Thus, the* 1*-extension is extendably unifiable. What about the* 2*-extension*

$$\langle h(h(h(x_3, x_2), h(x_3, x_4)), h(h(h(x_2, x_1), h(x_2, x_3)), \hat{a}))\ , t|?$$

It has the following unifier:$\{y_3 \mapsto h(x_3, x_2)\ ,\ y_1 \mapsto h(x_3, x_4)\ ,\ y_4 \mapsto h(h(x_2, x_1), h(x_2, x_3))\ ,\ \hat{a} \mapsto h(h(x_2, x_1), h(x_2, x_3))\}$. *The* 2*-extension is also extendably unifiable. The* 3*-extension is* $\langle h(h(h(x_4, x_3), h(x_4, x_5)), h(h(h(x_3, x_2), h(x_3, x_4)), s))\ , t|$. *Notice that the solved form contains an occurrence check on the variable* x_2 *and thus the* 3*-extension is not unifiable.* $\{y_3 \mapsto h(x_4, x_3)\ ,\ y_1 \mapsto h(x_4, x_5)\ ,\ y_4 \mapsto h(h(x_3, x_2), h(x_3, x_4))\ ,\ \hat{a} \mapsto h(x_3, x_4)\ ,\ x_3 \mapsto h(x_2, x_1)\ ,\ x_2 \mapsto h(x_2, x_3)\}$.

Definition 7. *Let* $\langle s, t\rangle$ *be a loop which is loop unifiable. We say* $\langle s, t\rangle$ *is* infinitely loop unifiable *if for every* $n \in \mathbb{N}$, $\langle s, t\rangle_n$ *is* extendably unifiable. *Otherwise, we say* $\langle s, t\rangle$ *is finitely loop unifiable.*

Example 6. *The following is a simply example of a semiloop which is infinitely loop unifiable:* $\langle h(h(x_1, x_1), \hat{a})\ ,\ h(y_1, y_1)|$. *Notice that for every extension the solved form contains either* $\hat{a} \mapsto h(y_1, y_1)$ *or* $\hat{a} \mapsto h(x_1, x_1)$. *Thus, every extension is extendably unifiable.*

Interestingly, even simple examples can lead to infinite loop unifiability:

Example 7. *Consider* $\langle h(\hat{a}, h(h(h(x_1, x_1), x_1), x_1)), h(h(h(y_1, y_1), y_1), y_1), y_1)|$. *While this may not be clear from looking at this semiloop, for* $n \geq 1$, *the solved form of the* $(3n)$*-extension contains:* $\{\hat{a} \mapsto h(h(h(h(h(x_2, x_2), x_2), x_2), h(h(h(x_2,\ x_2), x_2), x_2)), h(h(h(x_2, x_2), x_2), x_2))\}$, *the solved form of the* $(3n+1)$*-extension contains:* $\{\hat{a} \mapsto h(h(h(h(x_2, x_2), x_2), x_2), h(h(h(x_2, x_2), x_2), x_2))\}$, *the solved form of the* $(3n+2)$*-extension contains:* $\{\hat{a} \mapsto h(h(h(x_2, x_2), x_2), x_2)\}$. *This pattern repeats for all* m*-extensions where* $m > 2$.

2.3 Towards Termination Conditions

Consider the following procedure:

```
1: function LoopUnif(⟨s, t|, k)
2:     if ⟨s, t|_k is extendably unifiable then
3:         LoopUnif(⟨s, t|, k + 1)
4:     end if
5: end function
```

Question: Is termination of **LoopUnif**$(\langle s, t|, 1)$ is decidable?

In the previous section we described a few cases for which **LoopUnif**$(\langle s, t|, 1)$ does not terminate. In each of these cases exists there is cyclic behavior in the solved forms. Below, we provide a somewhat formal definition of what we mean be cyclic behavior

3 Cyclicity Conjecture

Before we state our conjecture, we need to define one additional construction.

Definition 8. *Let $\langle s, t|$ be infinitely unifiable. Then the* loop sequence of $\langle s, t|$*, denoted by* $loopSeq_{s,t}$ *is defined as follows:*

- $loopSeq_{s,t}(0) = t$
- $loopSeq_{s,t}(i + 1) = r_{i+1}$ *where* $\{\hat{a} \mapsto r_{i+1}\} \in \sigma$ *and* σ *is the unifier of* $\langle s, t|_i$

Conjecture 1 (cyclicity)**.** *Let $\langle s, t|$ be infinitely unifiable. Then there exists $i, k \in \mathbb{N}$, such that for all $i \leq j$, $loopSeq_{s,t}(j) = loopSeq_{s,t}(j + k)$*

Note that the cyclicity property provides a sufficient condition for non-termination of **LoopUnif**$(\langle s, t|, 1)$. A weaker statement would be as follows:

Conjecture 2 (weak cyclicity)**.** *Let $\langle s, t|$ be infinitely unifiable. Then there exists $i, k \in \mathbb{N}$, such that for all $i \leq j$, $|loopSeq_{s,t}(j)| = |loopSeq_{s,t}(j + k)|$*

While the weak cyclicity conjecture is less interesting, it allows us to present interesting examples succinctly. Note, in both examples below, terms with equal size are syntactically equivalent. This does not hold in the general case. For the following example *LoopUnif* does not terminate and a cycle of length 3 is found.

Example 8.

$$\langle h(x_1, h(x_4, h(x_1, x_4))), x^*) \ , \ h(y_1, h(y_4, h(y_1, y_4)))|$$

$$\begin{array}{ll}
|loopSeq_{s,t}(0)| = 5 & |loopSeq_{s,t}(1)| = 7 \\
|loopSeq_{s,t}(2)| = 6 & |loopSeq_{s,t}(3)| = 5 \\
|loopSeq_{s,t}(4)| = 7 & |loopSeq_{s,t}(5)| = 9 \\
|loopSeq_{s,t}(6)| = 8 & |loopSeq_{s,t}(7)| = 7 \\
|loopSeq_{s,t}(8)| = \mathbf{9} & |loopSeq_{s,t}(9)| = \mathbf{8} \\
|loopSeq_{s,t}(10)| = \mathbf{10} & |loopSeq_{s,t}(11)| = 9 \\
|loopSeq_{s,t}(12)| = 8 & |loopSeq_{s,t}(13)| = 10 \\
|loopSeq_{s,t}(14)| = 9 & |loopSeq_{s,t}(15)| = 8
\end{array}$$

While this example follows the cyclicity conjecture there exists very similar examples which are not infinitely unifiable but exhibit cyclicity at some points:

Example 9.

$$s = h(h(x_1, h(x_{16}, h(x_{32}, h(x_1, h(x_{16}, x_{32}))))), x^*)$$

$$t = h(y_1, h(y_{16}, h(y_{32}, h(y_1, h(y_{16}, y_{32})))))$$

$$
\begin{array}{ll}
|loopSeq_{s,t}(0)| = \mathbf{7} & |loopSeq_{s,t}(1)| = \mathbf{10} \\
|loopSeq_{s,t}(2)| = \mathbf{10} & |loopSeq_{s,t}(3)| = \mathbf{9} \\
|loopSeq_{s,t}(4)| = \mathbf{8} & |loopSeq_{s,t}(5)| = 7 \\
|loopSeq_{s,t}(6)| = 10 & |loopSeq_{s,t}(7)| = 10 \\
|loopSeq_{s,t}(8)| = 9 & |loopSeq_{s,t}(9)| = 8 \\
|loopSeq_{s,t}(10)| = 7 & |loopSeq_{s,t}(11)| = 10 \\
|loopSeq_{s,t}(12)| = 10 & |loopSeq_{s,t}(13)| = 9 \\
|loopSeq_{s,t}(14)| = 8 & |loopSeq_{s,t}(15)| = 7 \\
|loopSeq_{s,t}(16)| = 10 & |loopSeq_{s,t}(17)| = 10 \\
|loopSeq_{s,t}(18)| = \boxed{14} & |loopSeq_{s,t}(19)| = 13 \\
|loopSeq_{s,t}(20)| = 12 & |loopSeq_{s,t}(21)| = 11 \\
|loopSeq_{s,t}(22)| = 10 & |loopSeq_{s,t}(23)| = 18 \\
|loopSeq_{s,t}(24)| = 13 & |loopSeq_{s,t}(25)| = 12 \\
|loopSeq_{s,t}(26)| = 11 & |loopSeq_{s,t}(27)| = 15
\end{array}
$$

Notice that the cycle $7, 10, 10, 9, 8$ *repeats* 3 *times before breaking. The 28-extension is not unifiable.*

While this last example does not refute the conjecture, it illustrates that even if the conjecture holds our question concerning termination remains non-trivial.

References

[1] Vincent Aravantinos, Mnacho Echenim, and Nicolas Peltier. A resolution calculus for first-order schemata. *Fundam. Informaticae*, 125(2):101–133, 2013.

[2] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001.

[3] David M. Cerna, Alexander Leitsch, and Anela Lolic. Schematic refutations of formula schemata. *J. Autom. Reason.*, 65(5):599–645, 2021.

[4] Alexander Leitsch, Nicolas Peltier, and Daniel Weller. CERES for first-order schemata. *J. Log. Comput.*, 27(7):1897–1954, 2017.

A Polynomial-time Algorithm for the Common Left Multiplier Problem for Forward-closed String Rewriting Systems

(work in progress)

Wei Du, Paliath Narendran, and Bharvee Acharya

University at Albany–SUNY (USA),
e-mail: {wdu2, pnarendran, bacharya}@albany.edu

Abstract

Given a forward-closed convergent string rewriting system R and two strings α, β, the common left multiplier problem is to find a string W, which when concatenated to the left of the given strings α, β respectively will make the resulting strings $W\alpha$ and $W\beta$ joinable with respect to R. In this paper, we provide a polynomial-time algorithm for this problem.

Keywords: string-rewriting systems, forward-closed, convergent

1 Introduction

In this paper, we present a polynomial-time algorithm for the common left multiplier problem for forward-closed convergent string rewriting systems. Our work is motivated by [7], where convergent and forward-closed string rewriting systems were studied.

The common left multiplier problem for a string rewriting system R is to find, given two strings α and β, a string W such that $W\alpha$ and $W\beta$ are congruent modulo R. If R is convergent and forward-closed, then we can assume that α and β are irreducible with respect to R and the question becomes whether there is a string W such that $W\alpha$ and $W\beta$ are joinable, i.e., $W\alpha \downarrow W\beta$. Note that this can be viewed as a particular case of the unification problem, where we consider symbols in the alphabet as unary function symbols and treat concatenation as function composition in the following way: $ab(x) = b(a(x))$. The unification problem in general for forward-closed, convergent string rewriting systems (i.e., where there could be more than one equation) is NP-complete [6].

This problem was first investigated by Otto who showed decidability when R is convergent and *monadic*[1] [10]. It was also shown that the problem is in **P** in this case [9].

Here we come up with a polynomial-time algorithm for this problem when the string rewriting system is convergent and forward-closed. The algorithm is based on *narrowing*, since upward-innermost narrowing is known to terminate in the case of forward-closed convergent rewrite systems. To achieve the polynomial-time result we use search dags that we call "narrowing dag". We also utilize preprocessing steps to avoid unnecessary recomputation. The dag will generate a list of candidate strings that are possible suffixes of the left multiplier we are looking for, along with the resulting normal forms. After which, we need to effectively solve two equations to determine the unknown string W.

Proofs are omitted due to lack of space.

[1] A string rewriting system is *monadic* if and only if the length of every right-hand side is either 0 or 1

2 Definitions

We give only a few essential definitions here; for more details, the reader is referred to [1] for term rewriting systems, and to [2] for string rewriting systems.

Let R be a convergent, forward closed string rewriting system. A redex is a string of the form wl where $w \in \Sigma^*$ and l is a left-hand side of a rule. A redex is *innermost* if no proper prefix of it is a redex. If all the innermost redexes can be reduced to their normal forms in a single rewrite step then the system is called *forward-closed*. A string is irreducible (or in normal form) if and only if it contains no redex. The set of all strings that are irreducible modulo a string rewriting system R is denoted $IRR(R)$. The size of a string-rewriting system, denoted $|R|$, is the sum of the lengths of all left-hand sides and right-hand sides: $|R| = \sum_{l \to r \in R} |l| + |r|$.

For a string w, let $SUF(w)$ denote the set of all its suffixes, and $SUBST(w)$ the set of all its substrings.

Let $\mathfrak{L}$ be the set of all left-hand sides of R and

$$\Gamma = \{x \mid x \text{ is an irreducible substring of some left-hand side of } R\}$$

Let $\Delta = (\Sigma \circ \Gamma) \smallsetminus (\Gamma \cup \mathfrak{L} \circ \Sigma^*)$. Thus the strings in Δ are themselves not proper substrings of any left-hand side; but all their proper suffixes are proper substrings of some left-hand side. Besides, the strings in Δ are irreducible. Thus Δ can also be written as $\Delta = \big((\Sigma \circ \Gamma) \smallsetminus \Gamma\big) \cap IRR(R)$. Note that $|\Gamma|$ is $O(|R|^2)$ and $|\Delta|$ is $O(|\Sigma| * |R|^2)$.

For instance, for $R = \{ab \to c,\ abc \to cc\}$, we have $\Sigma = \{a, b, c\}$, then $\Gamma = \{\varepsilon, a, b, c, bc\}$ and $\Delta = \{aa, ac, ba, bb, bc, bbc, ca, cb, cc, cbc\}$.

We define *leftmost-largest* rewriting (ll-rewriting) as follows: let $\prec$ be a given total ordering on the alphabet Σ and $\prec_L$ be its shortlex extension [11, page 14]. A rewrite step $xly \to xry$ is *leftmost-largest* if and only if (a) xl is an innermost redex, (b) any other left-hand side that is a suffix of xl is a suffix of l as well, and (c) if $l \to r'$ is another rule in the rewrite system, then $r \prec_L r'$. (Condition (c) is clearly redundant if R is convergent and right-reduced.)

We generalize this a little to deal with tuples of strings. Let (x, y) be a tuple of irreducible strings. An ll-rewrite step on this tuple is defined a follows:

$$(x, y) \xrightarrow{ll} (x', y')$$

if and only if there are strings x_1, x_2, y_1, y_2 and a rule $l \to r$ such that $x = x_1x_2,\ y = y_1y_2,\ l = x_2y_1$, $x' = x_1r$, $y' = y_2$, and $xy_1 = x_1l \to x_1r = x'$ is an ll-rewrite step. (Note that neither x_2 nor y_1 can be ε.)

Lemma 2.1. *[6] Let R be a convergent, forward-closed string rewriting system and x,y be two irreducible strings. Then xy can be reduced to its normal form in at most $|y|$ ll-rewrite steps.*

Narrowing modulo a convergent string rewriting system R can be defined in the following way: let x be an irreducible string and $l \to r$ be a rule in R. Then

$$x \rightsquigarrow_{[p, l \to r]} rx_2$$

if and only if there is a non-empty string x_1 such that $x = x_1x_2$, $p = |x_2| < |x|$ and x_1 is a suffix of l.

For instance, consider the system $\{abb \rightarrow ab\}$ which is convergent (and forward-closed). Then $baa \rightsquigarrow_{[2,abb\rightarrow ab]} abaa$. Here $x_1 = b$ and $x_2 = aa$.

For a system R, we write $x \rightsquigarrow_{[p,R]} y$ if and only if there is a rule $(l \rightarrow r)$ such that $x \rightsquigarrow_{[p,l\rightarrow r]} y$. A narrowing sequence $x_1 \rightsquigarrow_{[p_1,R]} x_2 \rightsquigarrow_{[p_2,R]} \ldots \rightsquigarrow_{[p_{n-1},R]} x_n$ is *upward-innermost* if and only if $p_1 > p_2 > \ldots > p_n$. Clearly upward-innermost narrowing[2] always terminates. It has been shown that upward-innermost narrowing is complete if the rewrite system is convergent and forward-closed.

3 The Common Left Multiplier Problem

The problem we are trying to solve is the following:

> **Input:** A convergent forward-closed SRS R and two irreducible strings α, β.
>
> **Question:** Does there exist a string W such that $W\alpha \downarrow_R W\beta$?

Note that we can assume without loss of generality that the string W that we are looking for is irreducible as well. Now if $W\alpha$ and $W\beta$ have the same normal form, then by Lemma 2.1, $W\alpha$ can be reduced to its normal form in $|\alpha|$ ll-rewrite steps or less and $W\beta$ can be reduced to its normal form in $|\beta|$ ll-rewrite steps or less. Thus there must be strings $W_1, W_2, W_3, W_4, \alpha', \beta'$ such that $W = W_1W_2 = W_3W_4$, $W_2\alpha \rightarrow^! \alpha'$, $W_4\beta \rightarrow^! \beta'$, and $W_1\alpha' = W_3\beta'$ where W_2 and W_4 are the parts of W that is involved in the ll-rewrite sequences.

A nondeterministic-polynomial (**NP**) time algorithm for unification modulo forward-closed and convergent term rewriting systems is straightforward and well-known[3]. Our approach is to carefully analyze these possible ll-rewrite steps to obtain possible candidates for W_2 and W_4. To model this reduction sequence, we first define a transition relation on 3-tuples (triples) of strings from $IRR(R) \times IRR(R) \times IRR(R)$. Suppose we want to normalize $U\alpha$ where U is an irreducible string. Our aim is to apply ll-rewrite steps. The initial triple is $\big(U,\, \varepsilon,\, \alpha\big)$.

The steps are as follows:

(i) $\big(Ub,\, V,\, W\big) \;\mapsto\; \big(U,\, bV,\, W\big)$ — if V is a *proper substring* of a left-hand side and bV is irreducible

(ii) $\big(U,\, V_1V_2,\, W_1W_2\big) \;\mapsto\; \big(U,\, V_1Z,\, W_2\big)$ — if $(V_1V_2,\, W_1) \xrightarrow{ll} (V_1Z,\, \varepsilon)$ is an ll-rewrite step and step (i) is not applicable. (Note: $V_1V_2 \in IRR(R)$.)

In other words, we apply step (ii) only when the second component of the triple is *not* a proper substring of any left-hand side, i.e., $V \in IRR(R) \smallsetminus \Gamma$. One thing to note here is this: suppose we have been applying step (i) and at some point we move to step (ii). Then the point at which we move to step (ii) is when $V \in \Delta$.

[2] This concept has been further refined in [8] as BNR: *b*asic *n*arrowing with *r*ight-hand side abstracted

[3] Since forward-closed systems have the *finite variant property* [3, 4, 5]

An invariant property is that concatenation of the first and second components in the triple results in an irreducible string, i.e., XY is irreducible in all tuples (X, Y, Z). Another invariant property is that the third component is a suffix of α.

Since we do not know U in advance, we need to mimic these by applying (upward-innermost) narrowing steps to α. For this we construct what we call a "narrowing dag" or *n-dag* which resembles a finite-state automaton. The nodes of the n-dag are two-tuples (pairs) which correspond to the second and third components of the triples we introduced earlier. The start or root node is (ε, α). The transitions are defined as follows:

T_1: $(u, v) \xrightarrow{a} (au, v)$ if u is a proper substring of a left-hand side and au is irreducible.

If u is not an irreducible proper substring of a left-hand side, i.e., $u \notin \Gamma$, then either uv is irreducible in which case we halt, or it is reducible and we apply an ll-rewrite step to (u, v). (Thus if v is ε we halt.) But after applying this step, $(u, v) \xrightarrow{ll} (u', v')$, the string u' could be in Γ in which case we can introduce an ε-transition

$$(u, v) \xrightarrow{\varepsilon} (u', v')$$

If not, then we have to try applying an ll-rewrite step again, which could lead to a chain of ε-transitions. To avoid this we use the following pre-processing step:

For all suffixes α_1 of α, and all strings $w \in \Delta$, let β_1 be the result of an ll-rewrite step on $w\alpha_1$. If β_1 is not a proper substring of an lhs, then apply ll-rewrite again. Alternatively, for all tuples (u, v) in $\Delta \times SUF(\alpha)$ apply the following procedure P:

1: if uv is irreducible then return (u, v)
else let $(u, v) \xrightarrow{ll} (u', v')$
if $u' \in \Gamma$ then return (u', v')
else goto 1

Let $P(u, v)$ denote the output of this procedure on input (u, v). Thus the above ε-transition can be modified to

T_2: $(u, v) \xrightarrow{\varepsilon} P(u, v)$ if $u \notin \Gamma$ and uv is reducible.

Note that a node that has an ε-transition going out of it has no other transitions.

A table for P can be computed in polynomial time since computing each entry involves at most $|\alpha|$ ll-rewrite steps. The table has $|\Delta| * |\alpha|$ entries and $|\Delta|$ is $O(|\Sigma| * |R|^2)$ as mentioned earlier.

Lemma 3.1. *Let D_α be the n-dag for α and D_β be the n-dag for β. The common left multiplier problem for α and β has a solution if and only if there are leaf nodes (u, v) in D_α and (x, y) in D_β and paths π_1 in D_α to (u, v), π_2 in D_β to (x, y), such that either*

(a) $\pi_2 = z\pi_1$ and $xy = zuv$ for some z, or

(b) $\pi_1 = z'\pi_2$ and $z'xy = uv$ for some z'.

Lemma 3.2. *For any irreducible string α, the number of nodes in the n-dag for (ε, α) is polynomial in $|\alpha|$ and $|R|$.*

Proof-sketch: Since every node is labelled by a pair of strings, we derive an upper bound on pairs that can appear in an n-dag for $(\varepsilon,\ \alpha)$. From the transitions we can see that there are two kinds of pairs: (a) those that belong to $\Gamma \times SUF(\alpha)$ and (b) those that belong to $\Delta \times SUF(\alpha)$ and possibly one extra we could get by applying $P(u,v)$. Thus $|\Gamma \times SUF(\alpha)| + 2 * |\Delta \times SUF(\alpha)|$ is a weak upper bound for the maximum number of nodes in the n-dag. This is $O(|\Sigma| * |R|^2 * |\alpha|)$. □

Lemma 3.3. *Checking whether there is a root-to-leaf path π in an n-dag can be done in time linear in the size of the n-dag.*

Proof-sketch: Since there are no chains of ε-transitions, we can trace the string starting from the root and check whether a leaf is reached at the end. □

Since each n-dag can be constructed in polynomial time and the condition in Lemma 3.1 can be checked in linear time we can conclude that the whole algorithm runs in **P**.

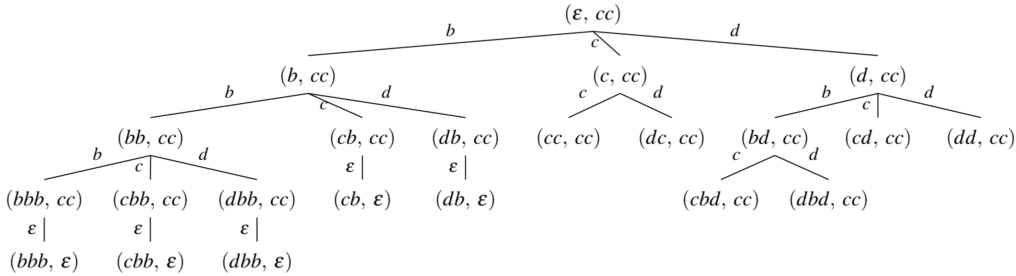
4 Some Examples

Input: A convergent forward-closed SRS $R = \{bc \to b,\ bbd \to b\}$
and two irreducible strings $\alpha = cc$, $\beta = db$.
$\Sigma = \{b,\ c,\ d\}$
$\Gamma = \{\varepsilon,\ b,\ c,\ d,\ bb,\ bd\}$
$\Delta = \{cb,\ cc,\ cd,\ db,\ dc,\ dd,\ bbb,\ cbb,\ dbb,\ cbd,\ dbd\}$
Question: Does there exist a string W such that $W\alpha \downarrow_R W\beta$?

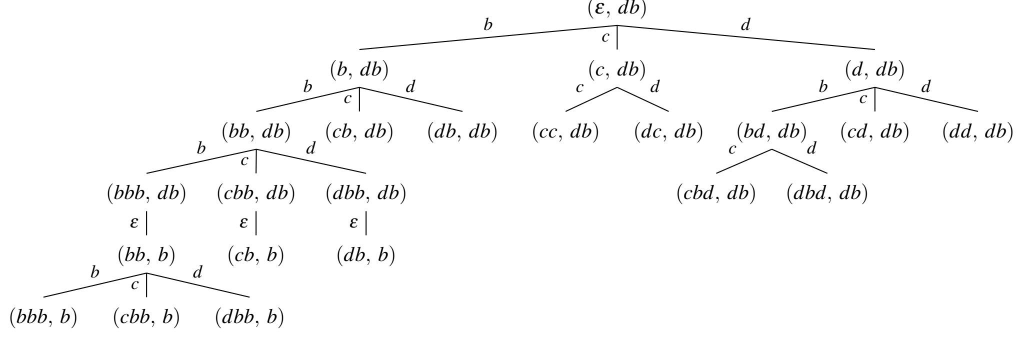
By applying pre-processing steps, we can find that $\Delta \times SUF(\alpha)$ contains the tuples $(bbb,\ cc)$, $(cbb,\ cc)$, $(dbb,\ cc)$, $(cb,\ cc)$, $(db,\ cc)$ that have chains of ε-transitions. For example,

$$(bbb,\ cc) \xrightarrow{\varepsilon} (bbb,\ c) \xrightarrow{\varepsilon} (bbb,\ \varepsilon)$$

which will be shown in the dag as: $(bbb,\ cc) \xrightarrow{\varepsilon} (bbb,\ \varepsilon)$. On the other hand, the tuples $(bbb,\ c)$, $(cbb,\ c)$, $(dbb,\ c)$, $(cb,\ c)$, $(db,\ c)$ have only one ε-transition from them. And the rest of the tuples are irreducible.



Starting from $(\varepsilon,\ cc)$, we create transitions by T_1 till we reach a level when we cannot apply T_1 any further. There we will create ε-transitions by T_2. One of the paths for α obtained using the above method is: $(\varepsilon,\ cc) \xrightarrow{b} (b,\ cc) \xrightarrow{b} (bb,\ cc) \xrightarrow{b} (bbb,\ cc) \xrightarrow{\varepsilon} (bbb,\ \varepsilon)$.



Similarly, there is a path for β:

$$(\varepsilon,\, db) \xrightarrow{b} (b,\, db) \xrightarrow{b} (bb,\, db) \xrightarrow{b} (bbb,\, db) \xrightarrow{\varepsilon} (bb,\, b) \xrightarrow{b} (bbb,\, b)$$

From these two paths we can find one common left multiplier $W = bbbb$. We can also get other solutions such as *cbbb*, *dbbb*, *cbb*, and *dbb* from these n-dags.

5 Future Work

We plan to extend this work to other classes of string-rewriting systems, such as the class of *sequentially-closed* systems introduced by Yu Zhang in [12, 13]. Another possible extension is to *context rewriting* where a *context* can be viewed as a linear term with exactly one variable occurrence.

6 Acknowledgements

We thank Dan Hono and the reviewers for their detailed comments, which helped improve the paper.

References

[1] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[2] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Texts and Monographs in Computer Science. Springer, 1993.

[3] Christopher Bouchard, Kimberly A. Gero, Christopher Lynch, and Paliath Narendran. On forward closure and the finite variant property. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2013.

[4] Hubert Comon-Lundh and Stephanie Delaune. The Finite Variant Property: How to Get Rid of Some Algebraic Properties. In J. Giesl, editor, *Term Rewriting and Applications*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer Berlin Heidelberg, 2005.

[5] Santiago Escobar, Ralf Sasse, and Jose Meseguer. Folding variant narrowing and optimal variant termination. *Journal of Logic and Algebraic Programming*, 81(7-8):898–928, 2012.

[6] Daniel S. Hono II. *On LM-systems and forward-closed string rewriting systems*. PhD thesis, Dept. of Computer Science, University at Albany—SUNY, Albany, NY, 2018.

[7] Daniel S Hono II, Paliath Narendran, and Rafael Veras. Lynch-Morawska systems on strings. In *Informal Proceedings of the 30th International Workshop on Unification (UNIF 2016)*, page 19, 2016.

[8] Dohan Kim, Christopher Lynch, and Paliath Narendran. Reviving basic narrowing modulo. In Andreas Herzig and Andrei Popescu, editors, *Frontiers of Combining Systems - 12th International Symposium, FroCoS 2019, London, UK, September 4-6, 2019, Proceedings*, volume 11715 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 2019.

[9] Paliath Narendran and Friedrich Otto. Some polynomial-time algorithms for finite monadic church-rosser thue systems. *Theor. Comput. Sci.*, 68(3):319–332, 1989.

[10] Friedrich Otto. On two problems related to cancellativity. *Semigroup Forum*, 33:331–356, 1986.

[11] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, Third edition, 2013.

[12] Yu Zhang. *Sequentially-closed and forward-closed String Rewriting Systems*. PhD thesis, Dept. of Computer Science, University at Albany—SUNY, Albany, NY, 2019.

[13] Yu Zhang, Paliath Narendran, and Heli Patel. On forward-closed and sequentially-closed string rewriting systems. In *UNIF 2019: 33rd International Workshop on Unification*, June 24, 2019.