Silvio Ghilardi and Manfred Schmidt-Schauß (Eds.)

Informal Proceedings of the 30th International Workshop on Unification (UNIF 2016)

June 26, 2016

Porto, Portugal

Preface

This volume contains the papers presented at UNIF 2016: The 30th International Workshop on Unification (UNIF 2016) held on June 26, 2016 in Porto.

There were 10 submissions. Each submission was reviewed by at least 3, and on the average 3.3, program committee members. The committee decided to accept 8 papers for publication in these Proceedings. The program also includes 2 invited talks.

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

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

- Unification algorithms, calculi and implementations
- Equational unification and unification modulo theories
- Unification in modal, temporal and description logics
- Admissibility of Inference Rules
- Narrowing
- Matching algorithms
- Constraint solving
- Combination problems
- Disunification
- Higher-Order Unification
- Type checking and reconstruction
- Typed unification
- Complexity issues
- Query answering
- Implementation techniques
- Applications of unification
- Antiunification/Generalization

This years UNIF is a satellite event of the first International Conference on Formal Structures for Computation and Deduction (FSCD). UNIF 2015 will be held on 26th June 2016 at the Department of Computer Science at the Faculty of Science of the University of Porto, Portugal.

We would like to thank all the members of the Program Committee and all the referees for their careful work in the review process. Finally, we express our gratitude to all members of the local organization of FSCD 2016, whose work has made the workshop possible. We also thanks the Department of Mathematics of the University of Milano and the Computer Science Institute of the Frankfurt University for financial support. We finally thank the Easy Chair Team for the allowing use of the online platform and facilities.

Silvio Ghilardi and Manfred Schmidt-Schauß

Program Committee

Franz Baader	TU Dresden
Philippe Balbiani	Institut de recherche en informatique de Toulouse
Wojciech Dzik	University of Silesia, Katowice, Poland
Santiago Escobar	Technical University of Valencia
Adria Gascon	SRI International
Silvio Ghilardi	Dipartimento di Matematica, Università degli Studi di Milano
Artur Jeż	University of Wroclaw, Institute of Computer Science
Temur Kutsia	RISC, Johannes Kepler University Linz
Jordi Levy	IIIA - CSIC
Christopher Lynch	Clarkson University
George Metcalfe	University of Bern
Barbara Morawska	TU Dresden
Paliath Narendran	University at Albany–SUNY
Christophe Ringeissen	LORIA-INRIA
Manfred Schmidt-Schauss	Inst. for Informatik, Goethe-University Frankfurt
Mateu Villaret	Universitat de Girona

Table of Contents

Author Index

Automated Symbolic Proofs of Security Protocols

Ralf Sasse
Institute of Information Security
Department of Computer Science
ETH Zürich

Security protocols are a mainstay of today's internet ecosystem. Increasingly many communications happen encrypted, i.e., communication uses a security protocol for key exchange and then encryption with derived keys. For human users this is most visible as transport layer security (TLS) used by all web browsers. History has shown that developing such protocols is an error-prone process, and attacks have been found even after protocols were in widespread use for years. One way to find these attacks, and to show their absence with respect to a particular abstraction, is to use automated protocol verification tools, like ProVerif [2], Maude-NPA [3], or Tamarin [7, 5, 6].

In this talk we give a brief overview of symbolic protocol analysis methods. We explain their modeling choices in general, and show Tamarin and some of its applications in particular. We give a high-level overview of Tamarin's inner workings and use. We present multiset rewriting for use in security protocol formalization. We show that symbolic methods abstract cryptographic operators with equational theories. We use constraint-solving as a decision procedure to determine whether the properties given as first-order logic formulae hold. We also consider the different stages of the tool's execution. In particular, we look in more detail at folding variant narrowing [4] as a building block of Tamarin.

We focus on extending Tamarin from trace properties to observational equivalence properties [1]. Observational equivalence expresses that two protocol runs appear the same to the adversary. Trace properties suffice for key exchange as used in TLS for online banking. Observational equivalence is necessary, e.g., for ballot privacy for electronic voting and untraceability of RFID passports. This extension of Tamarin requires a number of changes to the constraint solving. Also, normal form conditions for the dependency graph representation of protocol execution must be modified. Observational equivalence is then approximated by a novel technique, called *mirroring dependency graphs*. We finally present a number of case studies for observational equivalence.

We conclude by highlighting an important open issue in symbolic protocol analysis: find an equational theory for Diffie-Hellman (DH) exponentiation that can treat addition in the exponent, but still has the *finite variant property*. Currently used DH theories only allow multiplication in the exponent. To the best of our knowledge no representation with the *finite variant property* is known for DH with addition, but the existence of such a theory has not been disproven either.

References

[1] David Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1144–1155, New York, NY, USA, 2015. ACM.

[2] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE, 2001.

[3] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *FOSAD*, volume 5705 of *LNCS*, pages 1–50. Springer, 2007.

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

[5] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The Tamarin Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.

[6] Simon Meier, Benedikt Schmidt, Cas Cremers, Cedric Staub, Jannik Dreier, and Ralf Sasse. The Tamarin prover: source code and case studies, June 2016. available `http://tamarin-prover.github.io`.

[7] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Computer Security Foundations Symposium (CSF)*, pages 78–94. IEEE, 2012.

Unification in predicate logic

Wojciech Dzik

University of Silesia, Katowice, Poland
a joint work with
Piotr Wojtylak, University of Opole
wojciech.dzik@us.edu.pl, piotr.wojtylak@math.uni.opole.pl

Abstract

We introduce and apply 2^{nd} order unification to predicate logics which extend intuitionistic predicate logic Q-INT. We show that unification in a logic L is projective iff L contains IP.Q-LC, Gödel-Dummett's predicate logic plus Independence of Premise IP; hence, in such L each admissible rule is either derivable or passive and unification in L is unitary. We provide an explicit basis for all passive rules in Q-INT. We show that every unifiable Harrop formula is projective and we extend the classical results of Kleene on disjunction and existential quantifier under implication to projective formulas and to all extensions of Q-INT. We provide rules that are admissible in all extensions of Q-INT. After proving that L has filtering unification iff L extends Q-KC = Q-INT + $(\neg A \vee \neg\neg A)$, we show that unification in Q-LC and Q-KC is nullary and in Q-INT it is not finitary.

1 Introduction

We introduce unification in predicate logic and apply it to solve some problems such as admissibility of inference rules. Not much is known on admissible rules in intuitionistic predicate logic Q-INT [1], even less in extensions of Q-INT. In his pioneering work S.Ghilardi introduced unification in propositional logic. By application of projective unifiers he gave an elegant solution to the problem of recognizing admissible rules in intuitionistic propositional logic INT, [11], and in modal logics [12]. An inference rule is *admissible* in a logic L if adding it to L does not change the set of theorems of L. Other applications of unification in logic were also found, see e.g. [1], [10], [11], [8], [9].

Our aim is to introduce and apply unification tools to the problem of admissible rules in predicate logic, in particular in extensions of Q-INT, as well as to establish unification types of some best known predicate logics extending Q-INT.

Let us recall some notions of unification in logic: a *unifier* for A in a logic L is a substitution τ making A a theorem of L, i.e. $\tau(A) \in L$, [11]. The subsumption preorder is defined in a standard way: σ is *more general then* τ, if $\vdash_L \tau(x) \leftrightarrow \theta(\sigma(x))$, for some substitution θ. Now a formula A is *unifiable* in L (L-*unifiable*) if it has a unifier in L. A *most general unifier, mgu*, for a formula A is a unifier that is more general then any unifier for A.

If every L-unifiable formula has a mgu, then unification in L is *unitary* (type 1). Three other unification types, that is *finitary, infinitary* and *nullary (zero)*, depending on the number of maximal unifiers in the complete set of unifiers of a unification problem, are defined in a standard way. E.g. unification in a logic L is nullary if there is a unifiable formula A such that a maximal unifier for A in L does not exist. A substitution σ is a *projective* unifier for a formula A in L if it is a unifier for A, $\sigma(A) \in L$, and $A \vdash_L x \leftrightarrow \sigma(x)$. In this case a formula A is called *projective* in L, see [11], [1]. We say that a logic enjoys *projective unification* if every L-unifiable formula is projective.

Admissible rules can be defined as follows: a rule A/B is *admissible in* L if $\tau(A) \in L$ implies $\tau(B) \in L$, for every substitution τ, that is, if every unifier for A is also a unifier for B. A logic L is *Structurally Complete,* SC, if every rule A/B admissible in L is derivable, i.e. $A \vdash_L B$. Classical propositional logic is SC. A particular failure of structural completeness is caused by passive rules.

[1] e.g. [15]: "The nonderivable admissible predicate rules of intuitionistic predicate logic have not been characterized, they are known to form a complete Π^0_2 set..."

A rule A/B is *passive* L if its premise A is not unifiable in L. Passive rules are admissible in every logic. A logic L is *Almost Structurally Complete,* ASC, if every admissible rule in L which is not passive is derivable (e.g. all extensions S4.3 are ASC). Projective unification implies ASC (or SC).

Let L be a propositional logic extending INT. By Q-L we denote the least predicate logic extending L. Hnce Q-CL and Q-INT are classical and intuitionistic predicate logics, respectively. Here we concentrate on superintuitionistic predicate logics, that is on logics extending Q-INT.

We follow the above notions and definitions while introducing unification in predicate logics but there are differences and obstacles on the way. Firstly, now we use the 2^{nd} order substitutions for predicate variables see [16], [3], and they require restrictions concerning individual variables. Secondly, superintuitionistic predicate logics in general suffer from lack of adequate semantics, both algebraic and relational (Kripke). We list some important logics that are Kripke complete:
Q-INT (S. Kripke), Q-LC = Q-INT $+(A \to B) \vee (B \to A)$, the Gödel-Dummett (predicate) logic ([4]), Q-KC = Q-INT $+ \neg A \vee \neg\neg A$, De Morgan's logic ([5]); but majority of superintuitionistic predicate logics are Kripke incomplete (see papers of H. Ono, S. Ghilardi, D. Skvortsov).

Many unification results in propositional logic relay on algebraic concepts such as: finitely presented algebras, projective algebras as retracts of free algebras, congruences, etc. in varieties of algebras. For non-classical predicate logics algebraic counterparts are not known and algebraic tools can not be used.

In presenting non-classical predicate logics we follow Braüner - Ghilardi [2]

We consider a standard *first-order* (or *predicate*) language $\{\to, \wedge, \vee, \bot, \forall, \exists\}$ with *free individual variables*: $Var_f = \{a_1, a_2, a_3, \dots\}$, *bound individual variables*: $Var_b = \{x_1, x_2, x_3, \dots\}$, *predicate variables*: $Pr = \{P_1, P_2, P_3, \dots\}$. Neither function symbols nor the equality predicate $=$ occurs in the language. 0-ary predicate variables are (regarded as) propositional variables.

Elementary q-formulas are $\bot$ and $P_j(t_1, \dots, t_k)$ for $k = arn(P_j)$ and any individual variables $t_1, \dots, t_k$. *Compound q-formulas* are built up (from elementary ones) by the use of $\to, \wedge, \vee, \forall_{x_i}, \exists_{x_i}$, we put $\neg A = A \to \bot$. *Formulas* are q-formulas in which no bound variable occurs free and *sentences* are formulas without free variables. Let *q-Fm* and *Fm* denote the sets of all q-formulas and formulas, respectively. $A(a_1, \dots, a_k)$ means: all free variables of the formula A are included in $\{a_1, \dots, a_k\}$.

If u, v are individual variables (bound or free), then $A[u/v]$ is the q-formula resulting by the *substitution of the variable v for u in A*, i.e. $P(x)[x/a] = P(a)$; we write $B(a)$ instead of $B[x/a]$, and $\forall_{\overline{x}} A(\overline{x})$ for a quantifier closure of the formula A. We *rename bound variables* in a given q-formula A in a uniform way by increasing indices of bound variables with the same number n (see [16]). The formulas A and $(A)_n$ are the same, up to renaming of bound variables, and we will often write $A = (A)_n$. A (second-order) *substitution for predicate variables* is any mapping ε: *q-Fm* $\to$ *q-Fm* satisfying the following conditions:

(i) $\varepsilon[Fm] \subseteq Fm$;
(ii) $\varepsilon(P_j(t_1, \dots, t_k)) \approx (\varepsilon(P_j(x_1, \dots, x_k)))_n \quad [x_1/t_1, \dots, x_k/t_k]$
where $k = ar(P_j)$ and $n = ind(P_j(t_1, \dots, t_k))$;
(iii) $\varepsilon(\bot) = \bot$;
(iv) $\varepsilon(A \wedge B) = \varepsilon(A) \wedge \varepsilon(B)$;
(v) $\varepsilon(A \to B) = \varepsilon(A) \to \varepsilon(B)$;
(vi) $\varepsilon(A \vee B) = \varepsilon(A) \vee \varepsilon(B)$;
(vii) $\varepsilon(\forall_{x_i} A) = \forall_{x_i} \varepsilon(A)$;
$(viii)$ $\varepsilon(\exists_{x_i} A) = \exists_{x_i} \varepsilon(A)$;
(ix) $\varepsilon(P_j(x_1, \dots, x_k)) \neq P_j(x_1, \dots, x_k)$ for a finite number of variables P_j.

A *superintuitionistic predicate logic* L is any set of formulas $L \subseteq Fm$, containing all axioms of the intuitionistic propositional logic INT (meant as 1^{st}-order formulas) and containing the axioms:

$$\forall_x(A \to B(x)) \to (A \to \forall_x B(x)), \qquad \forall_x(B(x) \to A) \to (\exists_x B(x) \to A),$$
$$\forall_x B(x) \to B(a), \qquad B(a) \to \exists_x B(x);$$

closed under the inferential rules

$$MP: \frac{A \to C, A}{C} \qquad \text{and} \qquad RG: \frac{B(a)}{\forall_x B(x)} (a \text{ does not occur in} \forall_x B(x))$$

and closed under substitutions: $\varepsilon(A) \in L$, if $A \in L$, for each substitution ε.

2

2 Unifiability. A basis for passive rules

A unifier v: $Pr \to \{\bot, \top\}$ is called *ground*. Note that: (i) A is L-unifiable iff (ii) there is a ground unifier for A in L iff (iii) there is a ground unifier for A in Q–CL (or Q–INT). Note: Unifiable $\neq$ Consistent.

$(P\forall)$ denotes the rule: $\dfrac{\neg\forall_{\overline{z}}C(\overline{z}) \wedge \neg\forall_{\overline{z}}\neg C(\overline{z})}{\bot}$.

Theorem 1. *All passive rules are consequences, over* Q–INT*, of the rule* $(P\forall)$*, that is all passive rules are derivable in the extension of* Q–INT *with the rule* $(P\forall)$.

3 Projective unification and Harrop formulas

A unifier ε for a formula A in a logic L is *projective* if $A \vdash_L \varepsilon(P_i(a_1, \ldots, a_n)) \leftrightarrow P_i(a_1, \ldots, a_n)$ for each predicate variable P_i. if every L-unifiable formula has such ε, then L enjoys *projective unification*.
IP.Q-LC denotes the Gödel-Dummett predicate logic extended with the following formula called the *Independence of Premise* principle:
(IP): $(A \to \exists_x B(x)) \to \exists_x(A \to B(x))$.

Theorem 2. *A superintuitionistic predicate logic L enjoys projective unification if and only if* IP.Q-LC $\subseteq L$.

Corollary 3. *Every logic containing* IP.Q-LC *is almost structurally complete i.e. every admissible rule is either derivable or passive.*

Corollary 4. IP.Q-LC *is the least superintuitionistic logic in which* $\vee$ *and* $\exists$ *is definable by* $\{\wedge, \to, \forall\}$.

Theorem 5. *For every infinite rooted Kripke frame* $\mathcal{F} =< W, \leqslant, \mathcal{D} >$, IP *is valid in* $\mathcal{F}$ *iff* $\mathcal{F}$ *has constant domain* $\mathcal{D}$ *and* W *is well ordered. Hence* IP.Q-LC *is Kripke incomplete.*

Harrop formulae are "well-behaved" from a constructivist point of view. Neither disjunction nor existential q-formula is Harrop. Harrop q-formulas $q\text{-}Fm_H$ (or Harrop formulas Fm_H) are defined by the clauses:
1. all elementary q-formulas (including $\bot$) are Harrop; 2. if $A, B \in q\text{-}Fm_H$, then $A \wedge B \in q\text{-}Fm_H$;
3. if $B \in q\text{-}Fm_H$, then $A \to B \in q\text{-}Fm_H$; 4. if $B \in q\text{-}Fm_H$, then $\forall_{x_j} B \in q\text{-}Fm_H$.

Theorem 6. *Any unifiable Harrop's formula is projective in Q–INT.*

Theorem 7. *For any L-projective sentence A and any formulas $B_1, B_2, \exists_x C(x)$, we have*
(i) if $\vdash_L A \to B_1 \vee B_2$*, then* $\vdash_L (A \to B_1) \vee (A \to B_2)$;
(i) if $\vdash_L A \to \exists_x C(x)$*, then* $\vdash_L \exists_x(A \to C(x))$.

Example: The following non-passive rule is admissible in every superintuitionistic predicate logic:
$\neg(\exists_x P(x) \wedge \exists_x \neg P(x)) \to \exists_y Q(y)$ / $\exists_y[\neg(\exists_x P(x) \wedge \exists_x \neg P(x)) \to Q(y)]$

4 Filtering unification and unification types

In [13] a characterization is given of modal logics in which unification is *filtering*, that is, for every two unifiers for a formula A there is another unifier that is more general then both of them (type 1 or 0).

Theorem 8. *Let L be an superintuitionistic predicate logic. Unification in L is filtering iff the Stone law* $\neg\neg A \vee \neg A$ *is in L.*

Corollary 9. *For every superintuitionistic predicate logic L*
(i) if Q-KC $\subseteq L$*, then unification in L is unitary or nullary;*
(ii) if L enjoys unitary unification, then Q-KC $\subseteq L$.

Corollary 10. *Unification in* Q-LC*, as well as in* Q-KC *is nullary. Unification in* Q-INT *is either nullary or infinitary.*

Recall that unification in LC and in KC is unitary, but in INT it is finitary, see [11].
Results analogous to Theorem 2 and 3 hold in modal predicate logics extending Q-S4.

References

[1] Baader F.,Ghilardi S., Unification in Modal and Description Logics, *Logic Journal of the IGPL*, 19(5)(2011), 705-730.

[2] Braüner T., Ghilardi S., First-order Modal Logic, in Blackburn P., (ed.) *Handbook of Modal Logic* 2007 Elsevier, 549–620.

[3] Church A., Introduction to Mathematical Logic I, *Princeton University Press*, Princeton, New Jersey (1996).

[4] Corsi G. , Completeness Theorem for Dummett's LC Quantified and Some of Its Extensions, *Studia Logica* 51 (1992) 317-335, .

[5] Corsi G., Ghilardi S.,, Directed frames, *Archive for Math. Logic* 29 (1989), 53-67

[6] Dzik, W., Chains of structurally complete predicate logics with the application of Prucnal's substitution,.*Reports on Mathematical Logic*, 38 (2004) pp. 37–48.

[7] Dzik, W., Splittings of lattices of theories and unification types. In :*Contributions to general algebra* 17, (2006), Heyn, Klagenfurt, 71–81.

[8] Dzik W., Wojtylak P., Projective Unification in Modal Logic, *Logic Journal of the IGPL* 20(2012) No.1, 121–153.

[9] Dzik W., Wojtylak P., Modal consequence relations extending **S4.3**. An application of projective unification, *Notre Dame Journal of Formal Logic* (to appear in 2016).

[10] Ghilardi S., Unification through Projectivity, *Journal of Symbolic Computation* 7 (1997), 733–752.

[11] Ghilardi S., Unification in Intuitionistic Logic, *Journal of Symbolic Logic* 64(2) (1999), 859–880.

[12] Ghilardi S., Best Solving Modal Equations, *Annals of Pure and Applied Logic* 102 (2000), 183–198.

[13] Ghilardi S., Sacchetti, L., Filtering unification and most general unifiers in modal logic, *The Journal of Symbolic Logic* 69

[14] Minari, P. , Wroński A., The property (HD) in Intuitionistic Logic. A Partial Solution of a Problem of H. Ono, *Reports on Mathematical Logic* 22 (1988), 21–25.

[15] Moschovakis, J.R. *The Logic of Brouwer and Heyting* in: *Logic from Russell to Church*, Gabbay D., Woods, J. (eds.), North Holland, 2007.

[16] Pogorzelski W.A., Prucnal T. Structural Completeness of the First-order Predicate Calculus, *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21 (1975), 315-320.

[17] Wroński A., Transparent Verifiers in Intermediate Logics, *Abstracts of the 54-th Conference in History of Mathematics*, Cracow (2008).

4

Solving equations in pure double Boolean algebras

Philippe Balbiani

Institut de recherche en informatique de Toulouse

1 Introduction

Herrmann *et al.* [7] have generalized lattices of concepts to algebras of semiconcepts. Operations between semiconcepts give rise to pure double Boolean algebras [9, 10]. Such algebraic structures can be seen as the union of two Boolean algebras, the intersection of which being a lattice of concepts. In this paper, we study word problems and unification problems in several classes of pure double Boolean algebras.

2 Semiconcepts

Formal contexts are structures of the form $\mathbb{K} = (G, M, \Delta)$ where G is a nonempty set (with typical member noted g), M is a nonempty set (with typical member noted m) and Δ is a binary relation between G and M. The elements of G are called *objects*, the elements of M are called *attributes* and the intended meaning of $g \Delta m$ is: *object* g *possesses attribute* m. For all $X \subseteq G$ and for all $Y \subseteq M$, let $X^{\triangleright} = \{m \in M$: for all $g \in G$, if $g \in X$ then $g \Delta m\}$ and $Y^{\triangleleft} = \{g \in G$: for all $m \in M$, if $m \in Y$ then $g \Delta m\}$. The claims in the next lemma follow directly from the definition of the maps $\triangleright : X \mapsto X^{\triangleright}$ and $\triangleleft : Y \mapsto Y^{\triangleleft}$. See [3, 6] for details.

Lemma 1. *Let* $X_1, X_2, X \subseteq G$ *and* $Y_1, Y_2, Y \subseteq M$.

1. *The following conditions are equivalent: (a)* $X \subseteq Y^{\triangleleft}$*, (b)* $X^{\triangleright} \supseteq Y$.
2. *If* $X_1 \subseteq X_2$ *then* $X_1^{\triangleright} \supseteq X_2^{\triangleright}$ *and if* $Y_1 \supseteq Y_2$ *then* $Y_1^{\triangleleft} \subseteq Y_2^{\triangleleft}$.
3. $X \subseteq X^{\triangleright\triangleleft}$ *and* $Y^{\triangleleft\triangleright} \supseteq Y$.
4. *If there exists* $Y' \subseteq M$ *such that* $X = Y'^{\triangleleft}$ *then* $X = X^{\triangleright\triangleleft}$ *and if there exists* $X' \subseteq G$ *such that* $X'^{\triangleright} = Y$ *then* $Y^{\triangleleft\triangleright} = Y$.

Given $X \subseteq G$ and $Y \subseteq M$, the pairs $(X, X^{\triangleright})$ and $(Y^{\triangleleft}, Y)$ are called *semiconcepts of* $\mathbb{K}$. More precisely, pairs of the form $(X, X^{\triangleright})$ are called *left semiconcepts of* $\mathbb{K}$ and pairs of the form $(Y^{\triangleleft}, Y)$ are called *right semiconcepts of* $\mathbb{K}$. Remark that $(\emptyset, M)$ and $(G, \emptyset)$ are semiconcepts of $\mathbb{K}$. Let $\underline{\mathcal{H}}(\mathbb{K}) = (\mathcal{H}(\mathbb{K}), 0_l, 0_r, 1_l, 1_r, \sim_l, \sim_r, \sqcup_l, \sqcup_r, \sqcap_l, \sqcap_r)$ be the algebraic structure of type $(0, 0, 0, 0, 1, 1, 2, 2, 2, 2)$ defined by

- $\mathcal{H}(\mathbb{K})$ is the set of all semiconcepts of $\mathbb{K}$,
- $0_l = (\emptyset, M)$,
- $0_r = (M^{\triangleleft}, M)$,
- $1_l = (G, G^{\triangleright})$,
- $1_r = (G, \emptyset)$,
- $\sim_l (X, Y) = (G \setminus X, (G \setminus X)^{\triangleright})$,

– $\sim_r (X,Y) = ((M \setminus Y)^{\triangleleft}, M \setminus Y)$,
– $(X_1,Y_1) \sqcup_l (X_2,Y_2) = (X_1 \cup X_2, (X_1 \cup X_2)^{\triangleright})$,
– $(X_1,Y_1) \sqcup_r (X_2,Y_2) = ((Y_1 \cap Y_2)^{\triangleleft}, Y_1 \cap Y_2)$,
– $(X_1,Y_1) \sqcap_l (X_2,Y_2) = (X_1 \cap X_2, (X_1 \cap X_2)^{\triangleright})$,
– $(X_1,Y_1) \sqcap_r (X_2,Y_2) = ((Y_1 \cup Y_2)^{\triangleleft}, Y_1 \cup Y_2)$.

Remark that if G, M are finite then $\mathcal{H}(\mathbb{IK})$ is finite too and moreover, $\mid \mathcal{H}(\mathbb{IK}) \mid \leq 2^{|G|} + 2^{|M|}$. The set $\mathcal{H}(\mathbb{IK})$ can be ordered by the binary relation $\sqsubseteq$ defined by

– $(X_1,Y_1) \sqsubseteq (X_2,Y_2)$ iff $X_1 \subseteq X_2$ and $Y_1 \supseteq Y_2$.

Before formally introducing pure double Boolean algebras, we prove lemmas which will put the above definitions into perspective.

Lemma 2. *Let* $(X_1,Y_1),(X_2,Y_2) \in \mathcal{H}(\mathbb{IK})$.

1. *The following conditions are equivalent: (a)* $(X_1,Y_1) \sqsubseteq (X_2,Y_2)$*, (b)* $(X_1,Y_1) \sqcap_l (X_2,Y_2) = (X_1,Y_1) \sqcap_l (X_1,Y_1)$ *and* $(X_1,Y_1) \sqcup_r (X_2,Y_2) = (X_2,Y_2) \sqcup_r (X_2,Y_2)$.
2. *If* (X_1,Y_1) *is a left semiconcept then* $(X_1,Y_1) \sqsubseteq (X_2,Y_2)$ *iff* $(X_1,Y_1) \sqcap_l (X_2,Y_2) = (X_1,Y_1)$.
3. *If* (X_2,Y_2) *is a right semiconcept then* $(X_1,Y_1) \sqsubseteq (X_2,Y_2)$ *iff* $(X_1,Y_1) \sqcup_r (X_2,Y_2) = (X_2,Y_2)$.

Lemma 3. *The binary relation* $\sqsubseteq$ *is reflexive, antisymmetric and transitive on* $\mathcal{H}(\mathbb{IK})$.

We shall say that an object g is *sparse* if $\Delta(g) \neq M$ and an attribute m is *sparse* if $\Delta^{-1}(m) \neq G$. $\mathbb{IK}$ is said to be *sparse* if for all $g \in G$, g is sparse and for all $m \in M$, m is sparse. We shall say that a couple (g,g') of objects is a *cover* if $\Delta(g) \cup \Delta(g') = M$ and a couple (m,m') of attributes is a *cover* if $\Delta^{-1}(m) \cup \Delta^{-1}(m') = G$. A sparse $\mathbb{IK}$ is said to be *covered* if for all $g,g',g'' \in G$, if (g',g'') is a cover then (g,g') is a cover or (g,g'') is a cover and for all $m,m',m'' \in M$, if (m',m'') is a cover then (m,m') is a cover or (m,m'') is a cover.

3 Pure double Boolean algebras

Let $\mathcal{A} = (A, 0_l, 0_r, 1_l, 1_r, \sim_l, \sim_r, \sqcup_l, \sqcup_r, \sqcap_l, \sqcap_r)$ be an algebraic structure of type $(0,0,1,1,2,2,2,2)$. For all $a \in A$, let $\star_l a = \sim_r \sim_l a$ and $\star_r a = \sim_l \sim_r a$. $\mathcal{A}$ is said to be *concrete* iff there exists a formal context $\mathbb{IK} = (G,M,\Delta)$ and an injective homomorphism from $\mathcal{A}$ to $\underline{\mathcal{H}}(\mathbb{IK})$. We shall say that $\mathcal{A}$ is a *pure double Boolean algebra* if for all $a,b,c \in A$, it satisfies the conditions 1–13, 16–28, 31 and 32 in Fig. 1. Now, we can relate pure double Boolean algebras and concrete structures.

Proposition 1 ([7]). *The following conditions are equivalent:*

1. $\mathcal{A}$ *is concrete.*
2. $\mathcal{A}$ *is a pure double Boolean algebra.*

$\mathcal{A}$ is said to be *s-concrete* iff there exists a sparse formal context $\mathbb{IK} = (G,M,\Delta)$ and an injective homomorphism from $\mathcal{A}$ to $\underline{\mathcal{H}}(\mathbb{IK})$. We shall say that a pure double Boolean algebra $\mathcal{A}$ is *sparse* if it satisfies the conditions 14 and 29 in Fig. 1.

Proposition 2. *The following conditions are equivalent:*

1. *$\mathcal{A}$ is s-concrete.*
2. *$\mathcal{A}$ is a sparse pure double Boolean algebra.*

Proof. Simple variant of the proof of Proposition 1.

$\mathcal{A}$ is said to be *c-concrete* iff there exists a covered sparse formal context $\mathbb{K} = (G, M, \Delta)$ and an injective homomorphism from $\mathcal{A}$ to $\underline{\mathcal{H}}(\mathbb{K})$. We shall say that a sparse pure double Boolean algebra $\mathcal{A}$ is *covered* if it satisfies the conditions 15 and 30 in Fig. 1.

Proposition 3. *The following conditions are equivalent:*

1. *$\mathcal{A}$ is c-concrete.*
2. *$\mathcal{A}$ is a covered sparse pure double Boolean algebra.*

Proof. Simple variant of the proof of Proposition 1.

1. $a \sqcap_l (b \sqcap_l c) = (a \sqcap_l b) \sqcap_l c$,
2. $a \sqcap_l b = b \sqcap_l a$,
3. $\sim_l (a \sqcap_l a) = \sim_l a$,
4. $a \sqcap_l (b \sqcap_l b) = a \sqcap_l b$,
5. $a \sqcap_l (b \sqcup_l c) = (a \sqcap_l b) \sqcup_l (a \sqcap_l c)$,
6. $a \sqcap_l (a \sqcup_l b) = a \sqcap_l a$,
7. $a \sqcap_l (a \sqcup_r b) = a \sqcap_l a$,
8. $\sim_l (\sim_l a \sqcap_l \sim_l b) = a \sqcup_l b$,
9. $\sim_l 0_l = 1_l$,
10. $\sim_l 1_r = 0_l$,
11. $1_r \sqcap_l 1_r = 1_l$,
12. $a \sqcap_l \sim_l a = 0_l$,
13. $\sim_l \sim_l (a \sqcap_l b) = a \sqcap_l b$,
14. $0_l = 0_r$,
15. $\star_l \star_l a \sqcap_l \star_l a = \star_l a \sqcap_l 1_l$,
16. $a \sqcup_r (b \sqcup_r c) = (a \sqcup_r b) \sqcup_r c$,
17. $a \sqcup_r b = b \sqcup_r a$,
18. $\sim_r (a \sqcup_r a) = \sim_r a$,
19. $a \sqcup_r (b \sqcup_r b) = a \sqcup_r b$,
20. $a \sqcup_r (b \sqcap_r c) = (a \sqcup_r b) \sqcap_r (a \sqcup_r c)$,
21. $a \sqcup_r (a \sqcap_r b) = a \sqcup_r a$,
22. $a \sqcup_r (a \sqcap_l b) = a \sqcup_r a$,
23. $\sim_r (\sim_r a \sqcup_r \sim_r b) = a \sqcap_r b$,
24. $\sim_r 1_r = 0_r$,
25. $\sim_r 0_l = 1_r$,
26. $0_l \sqcup_r 0_l = 0_r$,
27. $a \sqcup_r \sim_r a = 1_r$,
28. $\sim_r \sim_r (a \sqcup_r b) = a \sqcup_r b$,
29. $1_r = 1_l$,
30. $\star_r \star_r a \sqcup_r \star_r a = \star_r a \sqcup_r 0_r$,
31. $(a \sqcap_l a) \sqcup_r (a \sqcap_l a) = (a \sqcup_r a) \sqcap_l (a \sqcup_r a)$,
32. $a \sqcap_l a = a$ or $a \sqcup_r a = a$.

Fig. 1.

4 A first-order signature

Let Ω be the first-order signature consisting of the following function symbols together with their arities: $\bot_l$ (0), $\bot_r$ (0), $\top_l$ (0), $\top_r$ (0), $\neg_l$ (1), $\neg_r$ (1), $\vee_l$ (2), $\vee_r$ (2), $\wedge_l$ (2) and $\wedge_r$ (2). Let VAR be a countable set of variables (with typical members noted x, y, etc). The set $\mathcal{T}(\Omega, VAR)$ of all *terms over Ω and VAR* (with typical members noted s, t, etc) is inductively defined as usual. We write $s(x_1, \ldots, x_n)$ to denote a term whose

variables form a subset of $\{x_1, \ldots, x_n\}$. The result of the replacement of $x_1, \ldots, x_n$ in their places in s with terms $t_1, \ldots, t_n$ will be noted $s(t_1, \ldots, t_n)$. A *substitution* is a function σ assigning to each variable x a term $\sigma(x)$. We shall say that a substitution σ is ground if for all variables x, $\sigma(x)$ is a variable-free term. For all terms $s(x_1, \ldots, x_n)$ let $\sigma(s)$ be $s(\sigma(x_1), \ldots, \sigma(x_n))$. The *composition* $\sigma \circ \tau$ of the substitutions σ and τ assigns to each variable x the term $\tau(\sigma(x))$.

5 Word problems

Let $\mathcal{C}$ be a class of pure double Boolean algebras. Now, for the *word problem* in $\mathcal{C}$: given terms s, t, decide whether $\mathcal{C} \models s = t$.

Proposition 4. *1. The word problem in the class of all pure double Boolean algebras is $PSPACE$-complete.*
2. The word problem in the class of all sparse pure double Boolean algebras is $PSPACE$-complete.
3. The word problem in the class of all covered sparse pure double Boolean algebras is NP-complete.

Proof. (1) See the proofs of Propositions 45 and 51 in [2].
(2) and (3) Simple variants of the proof of Propositions 45 and 51 in [2].

6 Unification problems

Let $\mathcal{C}$ be a class of pure double Boolean algebras. Now, for the *unification problem* in $\mathcal{C}$: given a finite set $\Sigma = \{(s_1, t_1), \ldots, (s_n t_n)\}$ of pairs of terms, decide whether there exists a substitution σ such that $\mathcal{C} \models \sigma(s_1) = \sigma(t_1)$, ..., $\mathcal{C} \models \sigma(s_n) = \sigma(t_n)$. In that case, the substitution σ is called *unifier* of Σ. Remark that if a finite set of pairs of terms possesses a unifier then it possesses a ground unifier. This follows from the fact that for all unifiers σ of a finite set Σ of pairs of terms and for all ground substitutions τ, $\sigma \circ \tau$ is a ground unifier of Σ. There are two main questions about the unification problem. Firstly, there is the question of its computability.

Proposition 5. *1. The unification problem in the class of all sparse pure double Boolean algebras is NP-complete.*
2. The unification problem in the class of all covered sparse pure double Boolean algebras is NP-complete.

Proof. Firstly, remark that, in any class of sparse pure double Boolean algebras, every variable-free term is equal either to 0_l, or to 1_r. Secondly, remark that, in any class of sparse pure double Boolean algebras, the word problem is in P when restricted to variable-free terms. Hence, in any class of sparse pure double Boolean algebras, the unification problem is in NP. As for its NP-hardness, it follows from a reduction of the satisfiability problem for Boolean formulas.

Secondly, there is the question of its type. See [1, 4, 5, 8] for details about unification types.

Proposition 6. *1. The unification type in the class of all pure double Boolean algebras is nullary.*
2. The unification type in the class of all covered sparse pure double Boolean algebras is unitary.

Proof. (1) Adapting the line of reasoning suggested by Jeřábek [8] in the case of modal logic K, we prove that $\{(\neg_r\neg_l x \vee_l x, \neg_r\neg_l x \vee_l 0_l)\}$ has no minimal complete set of unifiers in the class of all pure double Boolean algebras.
(2) Adapting the line of reasoning suggested by Baader and Ghilardi [1] or Dzik [4, 5] in the case of modal logic $S5$, we prove that every finite set of pairs of terms has a most general unifiers in the class of all covered sparse pure double Boolean algebras.

7 Conclusion

The decidability of the unification problem in the class of all pure double Boolean algebras and the unification type in the class of all sparse pure double Boolean algebras are still open.

Acknowledgements

Special acknowledgement is heartly granted to the organizers and the referees of UNIF 2016 for the feedback we have obtained from them and the colleagues of the *Institut de recherche en informatique de Toulouse* who made several helpful comments for improving the correctness and the readability of this article.

References

1. Baader, F., Ghilardi, S.: *Unification in modal and description logics.* Logic Journal of the IGPL **19** (2011) 705–730.
2. Balbiani, P.: *Deciding the word problem in pure double Boolean algebras.* Journal of Applied Logic **10** (2012) 260–273.
3. Davey, B, Priestley, H.: *Introduction to Lattices and Order.* Cambridge University Press (2002).
4. Dzik, W.: *Unitary unification of $S5$ modal logics and its extensions.* Bulletin of the Section of Logic **32** (2003) 19–26.
5. Dzik, W.: *Unification Types in Logic.* Wydawnicto Uniwersytetu Slaskiego (2007).
6. Ganter, B, Wille, R.: *Formal Concept Analysis: Mathematical Foundations.* Springer (1999).
7. Herrmann, C., Luksch, P., Skorsky, M., Wille, R.: *Algebras of semiconcepts and double Boolean algebras.* Technische Universität Darmstadt (2000).
8. Jeřábek, E.: *Blending margins: the modal logic K has nullary unification type.* Journal of Logic and Computation **25** (2015) 1231–1240.
9. Vormbrock, B.: *A solution of the word problem for free double Boolean algebras.* In Kuznetsov, S., Schmidt, S. (editors): *Formal Concept Analysis.* Springer (2007) 240–270.
10. Wille, R.: *Boolean concept logic.* In Ganter, B., Mineau, G. (editors): *Conceptual Structures: Logical, Linguistic, and Computational Issues.* Springer (2000) 317–331.

Lynch-Morawska Systems on Strings

Daniel S. Hono II[1], Paliath Narendran[1], and Rafael Veras[1]

University at Albany—SUNY (USA)
{dhono,pnarendran,rveras}@albany.edu

Abstract

We investigate properties of convergent and forward-closed string rewriting systems in the context of the syntactic criteria introduced in [7] by Christopher Lynch and Barbara Morawska (we call these *LM*-Systems). One of the criteria is that the system be *non-subterm-collapsing*, i.e., no term should be equivalent to a proper subterm of it. Since a string rewriting system can be viewed as a term-rewriting system over a signature of purely monadic function symbols, we adapt the notion of *LM*-system to the string rewriting case. We prove that the subterm-collapse problem for convergent and forward-closed string rewriting systems is effectively solvable. Therefore, there exists a decision procedure that verifies if such a system is an *LM*-System. We use the same construction to prove that the *cap problem* from the field of cryptographic protocol analysis, which is undecidable for general *LM*-systems, is decidable when restricted to the string rewriting case.

1 Introduction

In this paper we investigate the properties of convergent and forward-closed string rewriting systems. Our motivation comes from the syntactic criteria defined by Christopher Lynch and Barbara Morawska in [7]. They showed that for any term-rewriting system R that satisfies their criteria (which we call *LM*-Systems), the unification problem modulo R is solvable in polynomial time. In [5] it was shown that these conditions are tight, i.e., relaxing any of them leads to NP-hard unification problems. It was also shown in [5] that the subterm-collapse problem for term-rewriting systems that satisfy all of the other conditions of *LM*-Systems is undecidable.

In this current work, we show that the subterm-collapse problem is decidable when restricted to convergent and forward-closed string rewriting systems. These string rewriting systems can be viewed as term rewriting systems over a signature of purely monadic function symbols. We give an analogous definition of *LM*-Systems for string rewriting systems. Thus, given a forward-closed and convergent string rewriting system T there is an algorithm that decides if T is an *LM*-System.

The construction used to show the decidability of the subterm-collapse problem for forward-closed and convergent string rewriting systems is also used to show that the *cap problem*, an important problem from the field of cryptographic protocol analysis, is also decidable for such string rewriting systems. This is in contrast with some of our recent work that shows that the cap problem, which is undecidable in general, remains undecidable when restricted to general *LM*-Systems [1].

All proofs in this extended abstract have been omitted for brevity. The interested reader can consult the technical report for details[1].

[1] https://arxiv.org/abs/1604.06509

2 Definitions

We present here some notation and definitions. Only a few essential definitions are given here; for more details, the reader is referred to [3] for term rewriting systems, and to [4] for string rewriting systems.

Let Σ be a finite alphabet. As is usual, Σ^* stands for the set of all strings over Σ. The empty string is denoted by λ. For a string x, $|x|$ denotes its length and x^R denotes its reversal. A string u overlaps with a string v iff there is a non-empty *proper* suffix of u which is a prefix of v. For instance, *aba* overlaps with *acc*, but *aba* does not overlap with *cca*. However, *aba* overlaps with itself since *a* is both a prefix and a suffix of *aba*.

A string rewriting (rewrite) system (SRS) R over this alphabet is a set of *rewrite rules* of the form $l \to r$ where $l, r \in \Sigma^*$; l and r are respectively called the left- and right-hand-side (*lhs* and *rhs*) of the rule. The *rewrite relation* on strings defined by the rewrite system R is denoted $\to_R$. The reflexive and transitive closure of this relation is $\overset{*}{\to}_R$. Termination and confluence are defined as usual. An SRS R is *convergent* iff it is both terminating and confluent.

A string is irreducible with respect to R iff no rule of R can be applied to it. The set of strings that are irreducible modulo R is denoted by $IRR(R)$. Note that this set is a regular language, since $IRR(R) = \Sigma^* \setminus \{\Sigma^* l_1 \Sigma^* \cup \ldots \cup \Sigma^* l_m \Sigma^*\}$, where $l_1, \ldots, l_m$ are the lhs of the rules in R. A string w' is an *R-normal form* (or a *normal form* if the rewrite system is obvious from the context) of a string w for an SRS R if and only if $w \to_R^* w'$ and w' is irreducible. We write this as $w \to_R^! w'$. An SRS R is *right-reduced* if every right-hand side is in normal form.

String rewriting systems can be viewed as a restricted class of term rewriting systems where all functions are unary. As in [2] a string u over a given alphabet Σ is viewed as a term over one variable derived from the *reversed* string of u; i.e., if $g, h \in \Sigma$, the string gh corresponds to the term $h(g(x))$. (In other words, the unary operators defined by the symbols of a string are applied successively in the order in which these symbols appear in that string.) A string of the form wl where $w \in \Sigma^*$ and l is a left-hand side is called a *redex*. A redex is *innermost* if no proper prefix of it is a redex. The longest suffix of an innermost redex that is a left-hand side in R is called its l-part and the remaining prefix its s-part.

We will also need a special kind of normal form for strings, modulo any given SRS T. With that purpose, we define, following Sénizergues [8], a *leftmost-largest* reduction as follows: let $\succ$ be a given total ordering on the alphabet Σ and $\succ_L$ be its length + lexicographic extension[2]. 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, (i.e., l is the l-part of this redex) and (c) if $l \to r'$ is another rule in the rewrite system, then $r' \succ_L r$. A string w' is said to be a *leftmost-largest* (*ll-*) *normal form* of a string w iff $w \to^! w'$ using only leftmost-largest rewrite steps. Given a terminating system T, it holds that any string w has a *unique* normal form produced by leftmost-largest rewrite steps alone, since every rewrite step is unique; this unique normal form will be denoted as $\rho_T(w)$.

Next, we define what it means for a string $x \in \Sigma^+$ to cause a subterm-collapse.

Definition 2.1. Let R be a convergent string rewriting system. A string x is said to *cause a subterm-collapse* if and only if there is a non-empty string y such that $xy \to_R^* x$. R is subterm-collapsing if and only if there is a string that causes a subterm-collapse.

Throughout the rest of the paper, $a, b, c, \ldots, h$ will denote elements of the alphabet Σ, and strings over Σ will be denoted as l, r, u, v, w, x, y, z, along with subscripts and superscripts.

A string rewrite system T is said to be *forward-closed* iff every innermost redex can be reduced to its normal form *in one step*.

[2] Sénizergues refers to this as the *short-lex* ordering

We now give some preliminary results on convergent and forward-closed string rewriting systems. This first lemma shows that reducing all right-hand sides of rules in R will preserve the equivalence generated by R as well as the properties that we are interested in.

Lemma 2.2. *Let R be a convergent and forward-closed string rewriting system, and let $l \to r$ be a rule in R. Then $\left(R \setminus \{l \to r\}\right) \cup \{l \to \rho_R(r)\}$ is convergent, forward-closed and equivalent to R.*

The following is an easy consequence:

Corollary 2.3. *Let R be a convergent, forward-closed and right-reduced string rewriting system. Then no two distinct rules have the* same *left-hand side.*

The next preliminary result shows that we can use leftmost-largest reduction steps to reduce an innermost redex to its normal form in a single step.

Lemma 2.4. *Let R be a convergent, forward-closed and right-reduced string rewriting system, and let w be an innermost redex. Then $w \to \rho_R(w)$, i.e., w reduces to its normal form in* one *leftmost-largest reduction step.*

3 LM-Conditions for String Rewriting Systems

We now give an equivalent definition of *quasi-determinism* for string rewriting systems R. This definition is adapted from that of [7]. We also define a *right-hand side critical pair* for string-rewriting systems. Thus, we are able to formulate the conditions of [7] in the context of string rewriting systems.

A string rewriting system R is *quasi-deterministic* if and only if

1. No rule has λ as its right-hand side
2. No rule in R is *end-stable*—i.e., no rule has the same rightmost symbol on its left- and right-hand sides, and
3. R has no *end pair repetitions*—i.e., no two rules in R have the same unordered pair of rightmost symbols on their sides.

We define a *right-hand-side critical pair* as follows: if $l_1 \to r_1$ and $l_2 \to r_2$ are two distinct rewrite rules and $r_2 = xr_1$ for some x (i.e., r_1 is a suffix of r_2) then $\{xl_1, l_2\}$ is a right-hand-side critical pair. The set of all right-hand-side critical pairs is referred to as $RHS(R)$.

A string-rewriting system is *deterministic* if and only if it is non-subterm-collapsing and $RHS(R)$ is quasi-deterministic. A *Lynch-Morawska string rewriting system* or *LM-system* is a convergent right-reduced string rewriting system R which satisfies the following conditions:

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

In light of the results of [6], a convergent string rewriting system R is an LM-system if and only if $RHS(R)$ is quasi-deterministic and

(a) R is almost-left reduced [6].
(b) There are no overlaps among the left-hand sides of R.
(c) No lhs overlaps with a rhs.

We now work towards proving the main results of this paper. Namely, we will show below that the subterm-collapse problem for convergent and forward-closed string rewriting systems is decidable.

The first of our results towards the above goal is:

Lemma 3.1. *Let R be a convergent, forward-closed and quasi-deterministic string rewriting system and $x,y,z \in IRR(R)$ such that $xy \to^! z$. Then there exist irreducible strings $x = x_1,x_2,\ldots,x_n,x_{n+1}$, $y_1,y_2,\ldots,y_n,y_{n+1}$ such that*

1. *$y = y_1 \ldots y_{n+1}$,*
2. *$x_i y_i$ is an innermost redex for all $1 \leq i \leq n$,*
3. *$x_i y_i \to x_{i+1}$ for all $1 \leq i \leq n$, and*
4. *$x_{n+1} y_{n+1} = z$.*

An immediate consequence of the definition of subterm-collapse (Definition 2.1)is the following:

Lemma 3.2. *Let R be a convergent forward-closed string rewriting system and $x,y \in IRR(R)$ such that $xy \to^! x$ and $y \neq \lambda$. (Thus x causes a subterm-collapse.) Let y_1 be a prefix of y. Then xy_1 causes a subterm-collapse.*

We now prove that R is subterm-collapsing if and only if there is a right-hand side of a rule in R that causes a subterm collapse in the sense of Definition 2.1. This lemma will be key in showing the decidability of the subterm-collapse problem as it allows us only to consider right-hand sides of rules for possible sources of subterm-collapse.

Lemma 3.3. *Suppose R is a convergent forward-closed string rewriting system. Then R is subterm-collapsing if and only if and only if there is a right-hand side r that causes a subterm-collapse.*

The main lemma of this section appears below. It gives us that a certain language, parameterized by two strings $u,v \in \Sigma^*$, is a deterministic context-free language. We prove this by constructing a deterministic pushdown automaton to recognize this language.

Lemma 3.4. *Let R be a convergent forward-closed string rewriting system, $u,v \in IRR(R)$, and $\# \notin \Sigma$. Then the language*

$$\mathcal{L}_{u,v} = \left\{ w\# \mid uw \to^! v,\ w \neq \lambda \right\}$$

is a deterministic context-free language over $(\Sigma \cup \{\#\})^*$

As a consequence of the above Lemma 3.4 the subterm-collapse problem is decidable for convergent, foward-closed, string-rewriting systems.

Corollary 3.5. *The following decision problem:*

Given: A convergent, forward-closed, right-reduced SRS R.

Question: Is R subterm-collapsing?

is effectively solvable.

The construction of the DPDA in the proof of Lemma 3.4 can be carried out *in polynomial time.* Thus, not only is the above subterm-collapse problem for convergent, forward-closed string rewriting systems decidable, it is efficiently decidable. This is in contrast to the results of [5] where it was shown

that checking if a given term-rewriting system is subterm-collapsing, even when the system satisfies all of the other Lynch-Morawska conditions, is undecidable.

We can therefore conclude that the problem of verifying if a convergent and forward-closed string rewriting system (or a term rewriting system over a signature of monadic function symbols) is an *LM*-system is decidable.

As another corollary of the above result, we get that the cap problem for convergent, forward-closed, string-rewriting systems is also decidable. This problem, also known as the deduction problem, is often studied in the field of symbolic cryptographic protocol analysis.

Corollary 3.6. *The Cap Problem:*

Given: A convergent, forward-closed string-rewriting system R, a string $u \in \Sigma^+$ (representing the intruder knowledge) and a secret $v \in \Sigma^+$.

Question: Does there exists a string $w \in \Sigma^+$ (called a cap term) such that $uw \to_R^! v$?

is decidable.

Proof. The construction is essentially the same as that in the proof of Corollary 3.5. This time a *DPDA* is constructed, using Lemma 3.4, for the language $\mathcal{L}_{u,v}$. □

On the other hand, the cap problem is known to be undecidable for general term-rewriting systems, even when restricted to *LM*-Systems.

References

[1] Siva Anantharaman, Paliath Narendran, and Michael Rusinowitch. Intruders with caps. In *Proceedings of the 18th international conference on Term rewriting and applications*, pages 20–35. Springer-Verlag, 2007.

[2] Siva Anantharaman, Paliath Narendran, and Michael Rusinowitch. String rewriting and security analysis: an extension of a result of Book and Otto. *Journal of Automata, Languages and Combinatorics*, 16(2–4):83–98, 2012.

[3] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1999.

[4] Ronald V Book and Friedrich Otto. *String-rewriting systems*. Springer, 1993.

[5] Kimberly Gero, Chris Bouchard, and Paliath Narendran. Some notes on basic syntactic mutation. In Santiago Escobar, Konstantin Korovin, and Vladimir Rybakov, editors, *UNIF 2012 Post-Worskhop Proceedings. The 26th International Workshop on Unification*, volume 24 of *EPiC Series in Computing*, pages 17–27. EasyChair, 2014.

[6] Daniel S. Hono II, Namrata Galatage, Kimberly A. Gero, Paliath Narendran, and Ananya Subburathinam. Notes on Lynch-Morawska systems. Technical Report SUNYA-CS-16-01, Department of Computer Science, University at Albany—SUNY, 2016.

[7] Christopher Lynch and Barbara Morawska. Basic syntactic mutation. In *Automated Deduction (CADE-18)*, pages 471–485. Springer, 2002.

[8] Géraud Sénizergues. A polynomial algorithm testing partial confluence of basic semi-Thue systems. *Theoretical computer science*, 192(1):55–75, 1998.

Notes on Lynch-Morawska Systems

Daniel S. Hono II[1], Namrata Galatage[1], Kimberly A. Gero[2], Paliath Narendran[1], and Ananya Subburathinam[1]

[1] University at Albany—SUNY (USA)
{dhono,ngalatage,pnarendran,asubburathinam}@albany.edu
[2] The College of Saint Rose (USA)
gerok@strose.edu

Abstract

In this paper we investigate convergent term rewriting systems that conform to the criteria set out by Christopher Lynch and Barbara Morawska in their seminal paper *"Basic Syntactic Mutation."* The equational unification problem modulo such a rewrite system is solvable in polynomial-time. In this paper, we derive properties of such a system which we call an *LM*-system. We show, in particular, that the rewrite rules in an *LM*-system have no left- or right-overlaps.

We also show that despite the restricted nature of an *LM*-system, there are important undecidable problems, such as the *deduction problem* in cryptographic protocol analysis (also called *the cap problem*) that remain undecidable for *LM*-systems.

Keywords: Equational unification, Term rewriting, Polynomial-time complexity, **NP**-completeness.

1 Introduction

In this paper we investigate convergent term rewriting systems that conform to the criteria set out by Christopher Lynch and Barbara Morawska in their seminal paper *"Basic Syntactic Mutation."* In earlier work [5] it was shown that relaxing any of the conditions given by Lynch and Morawska results in a unification problem that is *NP*-hard. Thus these conditions are tight in the sense that relaxing any of them leads to an intractable unification problem (assuming $P \neq NP$). In this work, we continue our investigations on Lynch-Morawska criteria. As in [5] we consider the case where the equational theories have convergent and forward-closed term-rewriting systems, which we call *LM*-Systems. This definition differs from that of [7] in which the set of axioms E is saturated by paramodulation and do not necessarily form a convergent set of rewrite rules. All other criteria of [7] remain essentially unchanged.

We give a structural characterization of these systems by showing that if R is an *LM*-System, then there are no overlaps between the left-hand sides of any rules in R and there are no forward-overlaps between any right-hand side and a left-hand side. This characterization shows that *LM*-Systems form a very restricted subclass of term-rewriting systems. Any term-rewriting system that contains overlaps of these kinds cannot be an *LM*-System. Using these results, we show that saturation by paramodulation is equivalent to forward-closure when considering convergent term-rewriting systems that satisfy all of the remaining conditions for *LM*-Systems.

Despite their restrictive character, we show in section 4 that the cap problem, which is undecidable in general, remains undecidable when restricted to *LM*-Systems. The cap problem (also called the deduction problem) originates from the field of cryptographic protocol analysis. This result shows that *LM*-Systems are yet strong enough to encode important undecidable problems. Our proof of undecidability uses a many-one reduction from the halting problem for reversible deterministic 2-counter Minsky machines.

In the interest of brevity we have omitted many details and most of the proofs of results appearing in this paper. The interested reader can find all of the details in the technical report.[1]

[1] https://arxiv.org/abs/1604.06139

2 Notation and Preliminaries

We assume the reader is familiar with the usual notions and concepts in term rewriting systems [2] and equational unification [3]. A term t is $\bar{\varepsilon}$-*irreducible* modulo a rewriting system R if and only if every *proper* subterm of t is irreducible. The concept of *forward closure* is defined in [4]. The following theorem, shown in [4], gives a necessary and sufficient condition for a rewrite-system to be forward-closed.

Theorem 2.1. [4] *A convergent rewrite system R is forward-closed if and only if every innermost redex can be reduced to its R-normal form in one step.*

We next show that there are ways to simplify a rewrite system R while still maintaining the properties that we are interested in. More precisely, given a convergent, forward-closed rewrite system R we can reduce the right-hand sides of rules in R while maintaining convergence, forward-closure and the equational theory generated by R. Let R be a convergent rewrite system. Following [6] we define $R{\downarrow} = \{l \to r{\downarrow} \mid (l \to r) \in R\}$

Lemma 2.2. *Let R be a convergent, forward-closed rewrite system. Then $R{\downarrow}$ is convergent, equivalent to R (i.e., they generate the same congruence), and forward-closed.*

A convergent rewrite system R is *right-reduced* if and only if $R = R{\downarrow}$.

From the above lemma, it is clear that right-reduction does not affect forward-closure. However, full interreduction, where one also deletes rules whose left-hand sides are reducible by *other* rules, will not preserve forward-closure. The following example illustrates this:

$$f(x,i(x)) \to g(x) \qquad g(b) \to c \qquad f(b,i(b)) \to c$$

The last rule can be deleted since its left-hand side is reducible by the first rule. This will preserve convergence, but forward-closure will be lost since $f(b,i(b))$, an innermost redex, cannot be reduced in *one step* to c in the absence of the third rule.

However, the following lemma enables us to do a restricted deletion of superfluous rules:

Lemma 2.3. *Let R be a convergent, forward-closed, term rewriting system. Let $l_i \to r_i \in R$ for $i \in \{1,2\}$ such that $\exists p \in \mathcal{FP}os(l_1) : p \neq \varepsilon$ and $l_1|_p = \sigma(l_2)$ for some substitution σ. That is, l_1 contains a proper subterm that is an instance of the left-hand side of another rule in R. Then, $R' = R \setminus \{l_1 \to r_1\}$ is convergent, forward-closed and equivalent to R.*

We call systems that have no rules such that the conditions of Lemma 2.3 obtain *almost-left-reduced*. Note, however, that they are not fully left-reduced as there is still the possibility of overlaps at the *root*. If a convergent, forward-closed and right-reduced R is not almost-left-reduced, then the above lemma tells us that we may delete such rules and obtain an equivalent system.

3 Lynch-Morawska Conditions

In this section we define the Lynch-Morawska conditions. We also derive some preliminary results on convergent term rewriting systems that satisfy the Lynch-Morawska conditions.

A new concept introduced by Lynch and Morawska is that of a *Right-Hand-Side Critical Pair*. Since our focus is only on convergent term rewriting systems, this definition can be modified as follows:

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

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

2

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

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

1. No equation in E has a variable as its left-hand side or right-hand side,
2. No equation in E is *root-stable*—i.e., no equation has the same root symbol on its left- and right-hand side, and
3. E has no *root pair repetitions*—i.e., no two equations in E have the same pair of root symbols on their sides.

A set of equations E is *subterm-collapsing* if and only if there are terms t and u such that t is a proper subterm of u and $E \vdash t \approx u$ (or $t =_E u$). A theory E is *deterministic* if and only if it is quasi-deterministic and non-subterm-collapsing.

A *Lynch-Morawska term rewriting system* or *LM-system* is a convergent, almost-left-reduced and right-reduced term rewriting system R which satisfies the following conditions:

(i) R is non-subterm-collapsing,

(ii) R is forward-closed, and

(iii) $RHS(R)$ is quasi-deterministic.

The goal of the remainder of this section is to show that, given an *LM-system* R, there can be no overlaps between the left-hand sides of any rules in R and that there can be no forward-overlaps. These notions are defined precisely below. Further, we use those results to derive the equivalence of forward-closure and saturation by paramodulation when R is an *LM-system*.

These results show that *LM-systems* are a highly restrictive subclass of term-rewriting systems. However, in a later section, we show that there are important decision problems that remain undecidable when restricted to *LM-systems*.

The first of these results, Lemma 3.1 and its proof, are used multiple times to prove other results. It concerns how terms s and t, such that s is an innermost redex and t is $\bar{\varepsilon}$-irreducible, can be joined. It establishes that there are only two possible cases, and further, only one of these cases can hold at a time.

Lemma 3.1. *Let R be an LM-system, $s = f(s_1,\ldots,s_m)$ an innermost redex and $t = g(t_1,\ldots,t_n)$ an $\bar{\varepsilon}$-irreducible term such that $f \neq g$. Then s and t are joinable modulo R if and only if* exactly one *of the following conditions holds:*

(a) there is a unique rule $l \to r$ with root pair (f,g) and $s \xrightarrow[l\to r]{} t$, or

(b) there are unique rules $l_1 \to r_1$ and $l_2 \to r_2$ with root pairs (f,h) and (g,h) such that $s \xrightarrow[l_1\to r_1]{} \widehat{t}$ and $t \xrightarrow[l_2\to r_2]{} \widehat{t}$ for some term $\widehat{t}$.

Given Lemma 3.1, we can immediately derive two results that will be useful in proving the main result of this section.

Corollary 3.2. *Suppose $l \to r \in R$ is a rule with root-pair (f,g), $s = f(s_1,\ldots,s_m)$ and $t = g(t_1,\ldots,t_n)$, and s and t are $\bar{\varepsilon}$-irreducible. Then, $s \downarrow_R t$ if and only if $s \xrightarrow[l\to r]{} t$.*

Corollary 3.3. *Let R be an LM-System. Suppose $l \to r \in R$ is a rule with root-pair (f,g) and $s = f(s_1,\ldots,s_m)$ and $t = g(t_1,\ldots,t_n)$ be terms that are joinable. Let $\widehat{s_1},\ldots,\widehat{s_m},\widehat{t_1},\ldots,\widehat{t_n}$ be respectively the normal forms of $s_1,\ldots,s_m,t_1,\ldots,t_n$. Then*

$$f(\widehat{s_1},\ldots,\widehat{s_m}) \xrightarrow[l\to r]{} g(\widehat{t_1},\ldots,\widehat{t_n}).$$

(Thus the normal form of s and t is an instance of the right-hand side r.)

The above two corollaries, along with Lemma 3.1, allows us to state the first of the results concerning the non-overlapping property of LM-systems. The following establishes that there can be no overlaps between left-hand sides of two rules occurring at the root-position and that there can be no overlaps between a right-hand side of a rule and a left-hand side of another rule at the root position. This is achieved by showing that these terms cannot be unified.

Corollary 3.4. *Let R be an LM-System and let $l_1 \to r_1$ and $l_2 \to r_2$ be* distinct *rules in R. Then*
(a) l_1 and l_2 are not unifiable, and *(b) r_1 and l_2 are not unifiable.*

Next, we work towards showing that the other possible overlaps cannot occur either. The next lemma, and its extension, are used towards this goal. The technical result is used in the proofs of various other lemmas and corollaries.

Lemma 3.5. *Let R be an LM-System, and suppose $f(s_1,\ldots,s_m) \xrightarrow[l\to r]{} g(t_1,\ldots,t_n)$. Then the following diagram commutes:*

$$\begin{array}{ccc} f(s_1,\ldots,s_m) & \xrightarrow[l\to r]{} & g(t_1,\ldots,t_n) \\ {\scriptstyle *}\downarrow & & \downarrow{\scriptstyle *} \\ f(s'_1,\ldots,s'_m) & \dashrightarrow_{l\to r} & g(t'_1,\ldots,t'_n) \end{array}$$

where $s'_1,\ldots,s'_m,t'_1,\ldots,t'_n$ are the normal forms of $s_1,\ldots,t_n$ respectively.

The above result can be extended to the general case by induction.

Definition 3.1. *(*Non-Overlay Superpositions and Foward Overlaps*). Let R be a rewrite-system. We define the following sets:*

$$NOSUP(R) := \left\{ \sigma(l_1[l_2]_p) \;\middle|\; p \in \mathcal{FP}os(l_1) \setminus \{\varepsilon\} \text{ and } \sigma = mgu(l_1|_p =^? l_2),\, l_1 \to r_1,\, l_2 \to r_2 \in R \right\}$$

$$\mathcal{FOV}(R) := \{ \sigma(l_1 \to r_1[r_2]_p) \mid p \in \mathcal{FP}os(r_1), \text{ and } \sigma = mgu(r_1|_p =^? l_2),\, l_1 \to r_1,\, l_2 \to r_2 \in R \}$$

The previous results of this section are now brought together to prove the following main results about LM-systems concerning the status of overlaps. Namely, we show that there are no non-overlay superpositions and no forward-overlaps.

We first establish that there are no superpositions occurring between the left-hand sides of two distinct rules in R. Formally, this amounts to showing that $NOSUP(R) = \emptyset$. The main idea of the proof is that we show that such superpositions would induce critical pairs, and then these critical pairs cannot be joinable, which would contradict the confluence of our system (R is convergent by definition).

Corollary 3.6. *$NOSUP(R) = \emptyset$ for all LM-systems R.*

We now turn to the case of forward-overlaps as defined in the section on forward-closure.

Lemma 3.7. *Let R be an LM-System, then $\mathcal{FOV}(R) = \varnothing$.*

We can now state the following lemma, which follows easily from the above results concerning overlaps. First we introduce the following definition:

Definition 3.2. *A term-rewriting system R is said to be* non-overlapping *if and only if there are no left-hand side superpositions and no forward-overlaps.*

Lemma 3.8. *Every LM-system is non-overlapping.*

Proof. Follows from above results. □

Finally, using the results derived above about *LM-systems*, we show that every *LM-system* is saturated by paramodulation. It is clear that every rewrite system saturated by paramodulation is also forward-closed. The next result establishes that for *LM-systems*, these two concepts are equivalent. Specifically, an *LM-system* is trivially saturated by paramodulation as there can be no overlaps into the left-hand side of an equation nor the right-hand side of an equation.

Corollary 3.9. *Every LM-System is saturated by paramodulation.*

4 The Cap Problem Modulo LM-Systems

In this section we prove that although *LM-systems* are a restrictive subclass of term-rewriting systems there are still important problems that are undecidable when restricted to *LM-systems*. Specifically, we show that the *cap problem*[2], which has important applications in cryptographic protocol analysis, is undecidable even when the rewrite system R is an *LM-system*.

The cap problem [1] is defined as follows:

Instance: An LM-System R, a set S of ground terms representing the intruder knowledge, and a ground term M.

Question: Does there exist a cap term $C(\diamond_1, \ldots, \diamond_n)$ such that $C[\diamond_1 := s_{i_1}, \ldots, \diamond_n := s_{i_n}] \rightarrow_R^* M$?

We show that the above problem is undecidable by a many-one reduction from the halting problem for reversible deterministic 2-counter Minsky machines (which are known to be equivalent to Turing machines). The construction is extremely similar to the one given in [5]. Originally, it was used to show the undecidablility of the subterm-collapse problem for LM-Systems. The construction is modified slightly to account for the cap problem, however the majority of the reduction remains unchanged. The details of Minsky machines can be found in the technical-report.

Theorem 4.1. *The cap problem modulo LM-Systems is undecidable.*

References

[1] Siva Anantharaman, Paliath Narendran, and Michael Rusinowitch. Intruders with caps. In *Proceedings of the 18th international conference on Term rewriting and applications*, pages 20–35. Springer-Verlag, 2007.

[2] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1999.

[3] Franz Baader and Wayne Snyder. Unification theory. *Handbook of automated reasoning*, 1:445–532, 2001.

[4] 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, *FroCos*, volume 8152 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2013.

[5] Kimberly Gero, Chris Bouchard, and Paliath Narendran. Some notes on basic syntactic mutation. In Santiago Escobar, Konstantin Korovin, and Vladimir Rybakov, editors, *UNIF 2012 Post-Worskhop Proceedings. The 26th International Workshop on Unification*, volume 24 of *EPiC Series in Computing*, pages 17–27. EasyChair, 2014.

[6] Bernhard Gramlich. On interreduction of semi-complete term rewriting systems. *Theoretical Computer Science*, 258(1):435–451, 2001.

[7] Christopher Lynch and Barbara Morawska. Basic syntactic mutation. In *Automated Deduction (CADE-18)*, pages 471–485. Springer, 2002.

[2] Also known as the *deduction problem*

The Unification Type of ACUI w.r.t. the Unrestricted Instantiation Preorder is not Finitary

Franz Baader[1] and Pierre Ludmann[2]

[1] TU Dresden, Germany, franz.baader@tu-dresden.de
[2] École Normale Supérieure de Cachan, France, pierre.ludmann@ens-cachan.fr

Abstract

The unification type of an equational theory is defined using a preorder on substitutions, called the instantiation preorder, whose scope is either restricted to the variables occurring in the unification problem, or unrestricted such that all variables are considered. It is known that the unification type of an equational theory may vary, depending on which instantiation preorder is used. More precisely, it was shown that the theory ACUI of an associative, commutative, and idempotent binary function symbol with a unit is unitary w.r.t. the restricted instantiation preorder, but not unitary w.r.t. the unrestricted one. Here, we improve on this result, by showing that, w.r.t. the unrestricted instantiation preorder, ACUI is not even finitary.

Introduction

The first preorder introduced to deal with unification [Rob65] was in fact the unrestricted instantiation preorder. It was designed for syntactic unification and it worked pretty well considering that it gave a unitary unification type. Yet, when it comes to equational unification, researchers usually employ the restricted instantiation preorder, though the reason for this change was not explained in early papers.

We use the following notation to distinguish between these two preorders:

$$\sigma \leq_E^X \tau \text{ iff } \exists\lambda\forall x \in X.\ \lambda(\sigma(x)) =_E \tau(x) \quad \text{and} \quad \sigma \leq_E^\infty \tau \text{ iff } \exists\lambda\forall x \in \mathcal{V}.\ \lambda(\sigma(x)) =_E \tau(x)$$

Here, $\mathcal{V}$ is the countably infinite set of all variables, whereas X is a (usually finite) subset of it. From now on, we will use $\leq_E$ to denote the restricted preorder $\leq_E^{Var(\Gamma)}$ when the unification problem Γ is clear from the context.

Note that $\leq_E^\infty \subseteq \leq_E$, which has as an easy consequence that the unification type in the restricted case can never be worse than the type in the unrestricted case. In particular, syntactic unification is also unitary w.r.t. $\leq_E$. Hence, for the empty theory it makes no difference which instantiation preorder is used. For this reason, the distinction between these two orders was not always rigorously made clear, though it was known that the unrestricted preorder could lead to unpleasant results in weak unification [Ede85].

It took until 1991 before the first example of an equational theory for which the unrestricted and restricted unification types are different was published. That particular theory is ACUI, the theory of idempotent abelian monoids:

$$\mathsf{ACUI} \quad := \quad \{x + 0 = x, x + (y + z) = (x + y) + z, x + y = y + x, x + x = x\}$$

From [BB88] it was known that ACUI is unitary in the restricted case; in [Baa91], it was shown that its unrestricted type is not unitary. This paper thus showed that the choice of the instantiation preorder makes a difference for equational unification. However, it did not show how big that difference actually is: the unrestricted type of ACUI might be finitary, which is still a quite pleasant type from the application point of view. Here we show that this is not the case: ACUI is at least infinitary. However, the precise unrestricted unification type of ACUI is still an open problem: it is either infinitary or of type zero.

Auxiliary Results

In the following, $Var(s)$ denotes the set of variables occurring in the term s, $Dom(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ the domain of the substitution σ, and $VRan(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(\sigma(x))$ the variable range of σ.

The proof of our main result (Theorem 2) is based on the fact that ACUI is a theory in which no variable is going to pop – in or out – of a term, i.e. the set of variables of a term is stable under equality modulo the theory. As a matter of fact, this property is called regularity in unification theory.

Definition 1. An identity $s = t$ is *regular* if $Var(s) = Var(t)$. A set of identities E is *regular* if all elements of E are regular.

Regularity of the defining set of identities of an equational theory implies regularity of the whole theory.

Lemma 1 ([Yel85])**.** *E is regular iff $=_E$ is regular.*

Obviously, all the identities of ACUI are regular, which by the above lemma yields that $s =_{\mathsf{ACUI}} t$ implies $Var(s) = Var(t)$. Thus, the following lemma applies to ACUI.

Lemma 2. *Let E be a regular theory and $\Gamma = \langle s = t \rangle$ an E-unification problem s.t. $Var(s) \cap Var(t) = \emptyset$. Then the set $\mathcal{U}$ of all unifiers σ of Γ satisfying*

$$\forall y \in VRan(\sigma). \exists x, x' \in \mathcal{V} \text{ s.t. } x \neq x' \text{ and } y \in Var(\sigma(x)) \cap Var(\sigma(x'))$$

is complete w.r.t. $\leq_E^\infty$.

Proof. Let σ be a unifier of Γ that does not belong to $\mathcal{U}$, i.e., there is $y_0 \in VRan(\sigma)$ s.t. there exists a unique x_0 verifying $y_0 \in Var(\sigma(x_0))$ (note that $x_0 = y_0$ is possible). Let $\tau = \{x_0 \mapsto y_0, y_0 \mapsto x_0\}$ be the substitution that exchanges x_0 with y_0 (and is the identity if $x_0 = y_0$). Then define the new substitution σ' as $\sigma'(x) := \tau(\sigma(x))$ for $x \in \mathcal{V} \setminus \{x_0\}$ and $\sigma'(x_0) := x_0$. Note that $VRan(\sigma') \subsetneq VRan(\sigma)$: in fact, if $x_0 \in VRan(\sigma)$, then $x_0 \notin VRan(\sigma')$; otherwise, $y_0 \notin VRan(\sigma')$. Using the fact that, in any case, $x_0 \notin VRan(\sigma')$, we can show that σ is an (unrestricted) instance of σ':

$$\begin{aligned} \forall x \in \mathcal{V} \setminus \{x_0\} : \quad (\tau \circ \{x_0 \mapsto \tau \circ \sigma(x_0)\} \circ \sigma')(x) &= (\tau \circ \{x_0 \mapsto \tau \circ \sigma(x_0)\})(\sigma'(x)) \\ &= \tau(\sigma'(x)) = \tau \circ \tau \circ \sigma(x) = \sigma(x) \end{aligned}$$

Additionally, we have for the variable x_0:

$$\begin{aligned} (\tau \circ \{x_0 \mapsto \tau \circ \sigma(x_0)\} \circ \sigma')(x_0) &= (\tau \circ \{x_0 \mapsto \tau \circ \sigma(x_0)\})(x_0) \\ &= \tau(\tau \circ \sigma(x_0)) = \sigma(x_0) \end{aligned}$$

This shows $\sigma' \leq_E^\infty \sigma$. However, we need to show that σ' is a unifier of Γ in order to make the comparison useful. Because E is a regular theory, $\sigma(s) =_E \sigma(t)$ implies $Var(\sigma(s)) = Var(\sigma(t))$. Then, if x_0 occurs in one of the terms s, t, it also has to occur in the other one since it is the unique producer of y_0. This contradicts our assumption on Γ. Since, on the variables different from x_0, σ' is an instance of σ, we thus know that σ' is a unifier of Γ.

Consequently, we can replace σ with σ' and so shrink $VRan(\sigma)$. Since $VRan(\sigma)$ is finite, repeating this process will end with a unifier σ^* that satisfies the required condition and is more general w.r.t. $\leq_E^\infty$ than the original unifier σ. □

2

A second auxiliary result that will be employed in the proof of our main theorem follows easily from the order-theoretic point of view on unification types [Baa89, BS01]. Given any preorder $\preceq$ on the set U of E-unifiers of Γ, we denote with $\sim$ its induced equivalence relation, i.e., $\sigma \sim \tau$ iff $\sigma \preceq \tau$ and $\tau \preceq \sigma$. We denote the $\sim$-equivalence class of a unifier σ as $[\sigma]$ and the set of all equivalence classes of unifiers as $[U]$. The partial order induced by $\preceq$ on equivalence classes is defined as usual, i.e., $[\sigma] \leq [\tau]$ iff $\sigma \preceq \tau$. We say that $M \subseteq [U]$ is complete w.r.t. $\leq$ if every element of $[U]$ is above (w.r.t. $\leq$) some element of M.

Theorem 1 ([BS01]). *Let M be the set of $\leq$-minimal elements of $[U]$. If C is a minimal complete set of E-unifiers of Γ w.r.t. $\preceq$, then $M = \{[\sigma] \mid \sigma \in C\}$. Conversely, if M is complete in U, then any set of representatives of M is a minimal complete set of E-unifiers of Γ.*

The following easy consequence of this result holds w.r.t. any preorder on unifiers, and thus in particular both for the restricted and the unrestricted instantiation preorder.

Lemma 3. *Let C be a complete set of E-unifiers of an E-unification problem Γ. Then Γ has a minimal complete set of E-unifiers iff C contains a minimal complete set of E-unifiers of Γ.*

The Unrestricted Type of ACUI is at Least Infinitary

We will more generally show the result for regular theories satisfying certain properties, and then show that ACUI satisfies these properties. From now on we assume that

- E is a regular theory,
- $\Gamma = \langle s = t \rangle$ is an E-unification problem s.t. $Var(s) \cap Var(t) = \emptyset$,
- there is a $\leq_E^\infty$-minimal unifier σ of Γ that uses fresh variables, i.e., $VRan(\sigma) \setminus X \neq \emptyset$ where $X = Var(s) \cup Var(t)$, and
- this unifier σ belongs to the set $\mathcal{U}$ defined in the formulation of Lemma 2.

We will prove that in such a configuration, Γ, and so E, is at least infinitary w.r.t. unrestricted instantiation.

Let $x_0 \in VRan(\sigma) \setminus X$ and consider the following construction of substitutions:

$$\sigma_z := \sigma \circ (x_0 z) \quad \text{where } (x_0 z) := \{x_0 \mapsto z, z \mapsto x_0\} \text{ and } z \in \mathcal{V}.$$

We will show that, under certain conditions on z, such substitutions σ_z are $\leq_E^\infty$-minimal unifiers that are incomparable to each other w.r.t. $\leq_E^\infty$. By Theorem 1, this implies that Γ cannot have a finite minimal complete set of unifiers w.r.t. $\leq_E^\infty$ since there are infinitely many variables z satisfying these conditions.

Lemma 4. *For any $z \notin X$, σ_z is a minimal unifier of Γ w.r.t. $\leq_E^\infty$.*

Proof. Note that σ_z is a unifier of Γ because $x_0, z \notin X$. Moreover, let θ be a unifier of Γ s.t. $\theta \leq_E^\infty \sigma_z$, i.e., there is a substitution λ s.t.

$$\forall x \in \mathcal{V}. \quad \sigma_z(x) =_E \lambda \circ \theta(x)$$

Consequently, if we "multiply" from the right with $(x_0 z)$, we obtain

$$\forall x \in \mathcal{V}. \quad \sigma(x) =_E \lambda \circ (\theta \circ (x_0 z))(x).$$

Because $(\theta \circ (x_0 z))$ is a unifier of Γ with $(\theta \circ (x_0 z)) \leq_E^\infty \sigma$, minimality of σ yields $\sigma \leq_E^\infty (\theta \circ (x_0 z))$, i.e., there exists μ s.t.

$$\forall x \in \mathcal{V}. \quad (\theta \circ (x_0 z))(x) =_E \mu \circ \sigma(x).$$

Consequently, "multiplying" again from the right with $(x_0 z)$ yields

$$\forall x \in \mathcal{V}. \quad \theta(x) =_E \mu \circ \sigma_z(x).$$

Thus, we have shown that $\theta \leq_E^\infty \sigma_z$ for a unifier θ of Γ implies $\sigma_z \leq_E^\infty \theta$, which proves that σ_z is a minimal unifier of Γ w.r.t. $\leq_E^\infty$. □

Lemma 5. *For any two different variables $z, z' \notin Dom(\sigma) \cup VRan(\sigma)$, σ_z and $\sigma_{z'}$ are incomparable w.r.t. $\leq_E^\infty$.*

Proof. Assume there exists λ s.t. $\forall x \in \mathcal{V}.\ \sigma_{z'}(x) =_E \lambda \circ \sigma_z(x)$, and let $t_0 := \sigma(x_0)$.

Case 1: $Var(t_0) = \emptyset$
Then $\sigma_{z'}(z) =_E \lambda \circ \sigma_z(z)$ implies $z =_E \lambda(t_0)$. This definitely contradicts the existence of λ since it cannot create a variable from a ground term.

Case 2: $Var(t_0) \neq \emptyset$
Then there exists $y_0 \in Var(t_0)$. Note that, independent of whether $y_0 = x_0$ or $y_0 \neq x_0$, we have $y_0 \in VRan(\sigma)$. Since $\sigma \in \mathcal{U}$ by our assumptions on σ, there is $x_1 \in \mathcal{V}$ different from x_0 s.t. $\sigma(x_1) =_E t_1$ and $y_0 \in Var(t_1)$. Again, $\sigma_{z'}(z) =_E \lambda \circ \sigma_z(z)$ implies $z =_E \lambda(t_0)$. Because E is regular, this implies $Var(\lambda(y_0)) \subseteq \{z\}$.

Since $z, z' \notin Dom(\sigma) \cup VRan(\sigma)$, $x_1 \neq z$ and $x_1 \neq z'$; so $\sigma_{z'}(x_1) =_E \lambda \circ \sigma_z(x_1)$ implies $t_1 =_E \lambda(t_1)$ Again, regularity yields $Var(\lambda(y_0)) \subseteq Var(t_1)$.

Then $Var(\lambda(y_0)) = \emptyset$ as $z \notin VRan(\sigma)$ and $Var(t_1) \subseteq VRan(\sigma)$. In fact, if $x_1 \in Dom(\sigma)$, then $Var(t_1)$ is contained in $VRan(\sigma)$ by the definition of $VRan$. Otherwise, we must have $x_1 = \sigma(x_1) =_E t_1$. Since $y_0 \in Var(t_1)$, regularity of E yields $y_0 = x_1$ and $Var(t_1) = \{y_0\}$. Thus, $y_0 \in VRan(\sigma)$ yields $Var(t_1) \subseteq VRan(\sigma)$.

However, since $z =_E \lambda(t_0)$, the variable z is produced by λ from at least one variable y that occurs in t_0, i.e., there is a variable $y \in Var(t_0) \setminus \{y_0\}$ such that $Var(\lambda(y)) = \{z\}$.[1] Since $y \in VRan(\sigma)$ and $\sigma \in \mathcal{U}$, there exists $x \neq x_0$ s.t. $y \in Var(\sigma(x))$. Yet again, as $z, z' \notin Dom(\sigma) \cup VRan(\sigma)$, we know $x \neq z$ and $x \neq z'$, and thus $\sigma_{z'}(x) =_E \lambda \circ \sigma_z(x)$ implies $\sigma(x) =_E \lambda \circ \sigma(x)$. Since we know that $z \in Var(\lambda \circ \sigma(x))$, regularity yields $z \in Var(\sigma(x))$. However, this is absurd since $x \neq z$ and $z \notin VRan(\sigma)$.

To sum up, we have shown that $\sigma_z \leq_E^\infty \sigma_{z'}$ does not hold. A symmetric argument yields that $\sigma_{z'} \leq_E^\infty \sigma_z$ also does not hold. □

Since the complement of the finite set $X \cup Dom(\sigma) \cup VRan(\sigma)$ in the countably infinite set $\mathcal{V}$ of all variables is infinite, the set of unifiers of Γ must have infinitely many minimal elements. By Theorem 1, this shows that Γ cannot have a finite minimal complete set of unifiers.

Lemma 6. *The unification problem Γ does not have a finite minimal complete set of E-unifiers w.r.t unrestricted instantiation, and thus E is at least infinitary w.r.t. $\leq_E^\infty$.*

We are now ready to apply this result to ACUI.

Theorem 2. ACUI *is at least infinitary w.r.t. $\leq_{\mathsf{ACUI}}^\infty$.*

Proof. Since ACUI is regular, it is sufficient to show that there is an ACUI-unification problem Γ and a minimal unifier σ of Γ satisfying the conditions stated at the beginning of this section.

[1] Note that, if there is no such additional variable in t_0, then this already contradicts the existence of λ, making the point.

According to Corollary 3.6 in [BB88], any most general unifier (w.r.t. restricted instantiation) of the ACUI-unification problem $\Gamma = \langle x + y + z = u + v \rangle$ must use a fresh variable. Let θ be such an mgu.

If Γ does not have a minimal complete set of ACUI-unifiers w.r.t. unrestricted instantiation, then we are done. Thus, assume that Γ has a minimal complete set M w.r.t. unrestricted instantiation. By Lemma 2 and Lemma 3, we can assume without loss of generality that $M \subseteq \mathcal{U}$, and by Theorem 1 we know that the elements of M are $\leq^{\infty}_{\mathsf{ACUI}}$-minimal. Since θ is an ACUI-unifier of Γ, there is a $\sigma \in M$ such that $\sigma \leq^{\infty}_{\mathsf{ACUI}} \theta$. Since $\leq^{\infty}_{\mathsf{ACUI}} \subseteq \leq^{X}_{\mathsf{ACUI}}$, this implies that σ is also an mgu of Γ w.r.t. restricted instantiation, and thus it introduces a fresh variable.

Consequently, we have shown that all prerequisites for applying Lemma 6 are satisfied, which proves the theorem. □

Conclusion

In this paper we have shown that the gap between the unification types of equational theories w.r.t restricted and unrestricted instantiation is wider than previously known. In fact, for ACUI, which is unitary w.r.t. restricted instantiation, it was only known that the unrestricted type is at least finitary [Baa91]. Now we know that it is at least infinitary, which makes using unrestricted instantiation in this setting even less desirable.

Regarding future work, it would of course be good if we could determine the exact unification type of ACUI w.r.t. unrestricted instantiation (infinitary or type zero), but we have not been able to achieve this yet.

References

[Baa89] Franz Baader. Characterizations of unification type zero. In N. Dershowitz, editor, *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 2–14, Chapel Hill, North Carolina, 1989. Springer-Verlag.

[Baa91] Franz Baader. Unification, weak unification, upper bound, lower bound, and generalization problems. In R. V. Book, editor, *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 86–97, Como, Italy, 1991. Springer-Verlag.

[BB88] Franz Baader and Wolfram Büttner. Unification in commutative idempotent monoids. *Theoretical Computer Science*, 56(3):345–353, 1988.

[BS01] Franz Baader and Wayne Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 447–533. Elsevier Science Publishers, 2001.

[Ede85] Elmar Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1(1):31–46, 1985.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965.

[Yel85] Katherine A Yelick. A generalized approach to equational unification. Technical Report TR-344, MIT Lab. for Computer Science, 1985.

Approximately Solving Set Equations

Franz Baader[1], Pavlos Marantidis[1], and Alexander Okhotin[2]

[1] TU Dresden, Germany, firstname.lastname@tu-dresden.de
[2] Dept. of Mathematics and Statistics, University of Turku, Finland, alexander.okhotin@utu.fi

Abstract

Unification with constants modulo the theory ACUI of an associative (A), commutative (C) and idempotent (I) binary function symbol with a unit (U) corresponds to solving a very simple type of set equations. It is well-known that solvability of systems of such equations can be decided in polynomial time by reducing it to satisfiability of propositional Horn formulae. Here we introduce a modified version of this problem by no longer requiring all equations to be completely solved, but allowing for a certain number of violations of the equations. We introduce three different ways of counting the number of violations, and investigate the complexity of the respective decision problem, i.e., the problem of deciding whether there is an assignment that solves the system with at most ℓ violations for a given threshold value ℓ.

1 Unification modulo ACUI and set equations

The complexity of testing solvability of unification problems modulo the theory

$$\mathsf{ACUI} \quad := \quad \{x+0=x, x+(y+z)=(x+y)+z, x+y=y+x, x+x=x\}$$

of an associative, commutative and idempotent function symbol "+" with a unit "0" was investigated in detail by Kapur and Narendran [KN92], who show that elementary ACUI-unification and ACUI-unification with constants are polynomial whereas general ACUI-unification is NP-complete. Here we concentrate on ACUI-unification with constants, but formally introduce the problem in its disguise of testing solvability of set equations.

Given a *finite* base set B and a set of variables $\mathbf{X} = \{Z_1, \ldots, Z_N\}$ that can assume as values subsets of B, consider a *system* Σ *of set equations*, which consists of finitely many equations of the following form:

$$K \cup X_1 \cup \ldots \cup X_m = L \cup Y_1 \cup \ldots \cup Y_n, \tag{1}$$

where K, L are subsets of B and $X_1, \ldots, X_m, Y_1, \ldots, Y_n \in \mathbf{X}$.

A B-*assignment* is a mapping of subsets of B to the variables, i.e., it is of the form $\sigma\colon \mathbf{X} \to \mathfrak{P}(B)$. If there is no confusion, we will omit the prefix B- from B-assignment. Such an assignment σ is a *solution* of the system of set equations Σ if

$$K \cup \sigma(X_1) \cup \ldots \cup \sigma(X_m) = L \cup \sigma(Y_1) \cup \ldots \cup \sigma(Y_n)$$

holds for all equations of the form (1) in Σ.

Solvability of a system of set equations can be reduced in polynomial time (see below) to satisfiability of propositional Horn formulae [KN92], which can be tested in linear time [DG84].

To introduce this reduction, we define Boolean variables $p(a, X)$ for every $a \in B$ and $X \in \mathbf{X}$. The intuitive semantics of these variables is that $p(a, X)$ is true iff a is *not* in X for the given assignment.

Now, for each equation of the form (1) and each $a \in K \setminus L$ we generate the Horn clauses

$$p(a, Y_1) \wedge \ldots \wedge p(a, Y_n) \to \bot.$$

Indeed, whenever an element $a \in B$ is in K but not in L, for the equation to hold true, a must be in some Y_j. The symmetric Horn clauses are also produced, i.e., for each $a \in L \setminus K$

$$p(a, X_1) \wedge \ldots \wedge p(a, X_m) \rightarrow \bot.$$

It remains to deal with the elements $a \notin K \cup L$. First, if a belongs to none of the variables on the right-hand side, then it should not belong to any of the variables on the left-hand side, which is expressed by the Horn clauses

$$p(a, Y_1) \wedge \ldots \wedge p(a, Y_n) \rightarrow p(a, X_j) \qquad \text{for all } j = 1, \ldots, m.$$

Symmetrically, if a is not on the left-hand side, it cannot be on the right-hand side, which yields

$$p(a, X_1) \wedge \ldots \wedge p(a, X_m) \rightarrow p(a, Y_j) \qquad \text{for all } j = 1, \ldots, n.$$

The number of derived Horn clauses and their sizes are polynomial in the size of the given system Σ of set equations, where the size of Σ is the sum of the cardinality of B, the number of variables in $\mathbf{X}$, and the number of equations in Σ. The size of a Horn clause is just the number of literals occurring in it.

It is easy to see that the Horn formula obtained by conjoining all the Horn clauses derived from a system of set equations is satisfiable iff the original system of set equations has a solution (see [KN92] for details). Consequently, solvability of systems of set equations can be decided in polynomial time.

2 Minimizing the number of violated equations

We say that the B-assignment σ *violates* a set equation of the form (1) if

$$K \cup \sigma(X_1) \cup \ldots \cup \sigma(X_m) \neq L \cup \sigma(Y_1) \cup \ldots \cup \sigma(Y_n).$$

Given a base set B, a set of variables $\mathbf{X} = \{Z_1, \ldots, Z_N\}$, a system Σ of k set equations of the form (1), and a nonnegative integer ℓ, we now ask whether there exists a B-assignment σ such that at most ℓ of the equations of the system are violated by σ. We call this decision problem MinVEq-SetEq. For a given ℓ, MinVEq-SetEq(ℓ) consists of all systems of set equations for which there is a B-assignment that violates at most ℓ equations of the system.

We will show that MinVEq-SetEq is NP-complete using reductions to and from Max-HSAT. Given a Horn formula φ that is a conjunction of k Horn clauses and a nonnegative integer ℓ, Max-HSAT asks whether there is a propositional assignment τ that satisfies at least ℓ of the Horn clauses of φ. For a given ℓ, Max-HSAT(ℓ) consists of those Horn formulae for which there is a propositional assignment that satisfies at least ℓ of its Horn clauses. It is well-known that Max-HSAT is NP-complete [JS87].

Reducing MinVEq-SetEq to Max-HSAT For this purpose, we introduce new Boolean variables $good(i)$, whose rôle is to determine whether the ith equation is to be satisfied or not. We conjoin $good(i)$ to the left-hand side of each of the Horn clauses derived from the ith equation, i.e., if the ith equation is of the form (1), then we generate the following Horn clauses:

- For each $a \in K \setminus L$: $good(i) \wedge p(a, Y_1) \wedge \ldots \wedge p(a, Y_n) \rightarrow \bot$;
- For each $a \in L \setminus K$: $good(i) \wedge p(a, X_1) \wedge \ldots \wedge p(a, X_m) \rightarrow \bot$;

- For each $a \notin K \cup L$:

$$good(i) \wedge p(a, Y_1) \wedge \ldots \wedge p(a, Y_n) \rightarrow p(a, X_j) \qquad \text{for all } j = 1, \ldots, m;$$
$$good(i) \wedge p(a, X_1) \wedge \ldots \wedge p(a, X_m) \rightarrow p(a, Y_j) \qquad \text{for all } j = 1, \ldots, n.$$

- Furthermore, we add the Horn clause $\top \rightarrow good(i)$.

If k' is the number of clauses generated by the original reduction (see Section 1) and k is the number of set equations in the system Σ, then we obtain $k' + k$ Horn clauses in this modified reduction. Let $\varphi_\Sigma = C_1 \wedge \cdots \wedge C_{k'+k}$ denote the Horn formula obtained by conjoining these Horn clauses.

Intuitively, setting the Boolean variable $good(i)$ to false "switches off" the Horn clauses induced by the ith equation in the original reduction. Consequently, the satisfaction of these clauses is no longer enforced, which means that the ith equation may be violated. By maximizing satisfaction of the clauses $\top \rightarrow good(i)$, we thus minimize the number of violated set equations. More precisely, we can show the following lemma.

Lemma 1. *Let Σ be a system of set equations consisting of k equations and generating k' clauses in the reduction introduced in Section 1. Then we have*

$$\Sigma \in \textit{MinVEq-SetEq}(\ell) \quad \textit{iff} \quad \varphi_\Sigma \in \textit{Max-HSAT}((k' + k) - \ell).$$

Since Max-HSAT is in NP, this lemma implies that MinVEq-SetEq also belongs to NP.

Reducing Max-HSAT to MinVEq-SetEq Consider the Horn formula $\varphi = C_1 \wedge \ldots \wedge C_k$, where C_i is a Horn clause for $i = 1, \ldots, k$. To construct a corresponding system of set equations, we use the singleton base set $B = \{a\}$. For every Boolean variable p appearing in φ, we introduce a set variable X_p. Intuitively, a belongs to X_p iff p is set to false. Now, each Horn clause in φ yields the following set equations:

- If C_i is of the form $p_1 \wedge \ldots \wedge p_n \rightarrow p$, then the corresponding set equation is

$$X_{p_1} \cup \ldots \cup X_{p_n} \cup X_p = X_{p_1} \cup \ldots \cup X_{p_n}.$$

  Obviously, this equation enforces that a cannot belong to X_p if it does not belong to any of the variables X_{p_i}.

- If C_i is of the form $p_1 \wedge \ldots \wedge p_n \rightarrow \bot$, then the corresponding set equation is

$$X_{p_1} \cup \ldots \cup X_{p_n} = \{a\}.$$

  This equation enforces that a must belong to one of the variables X_{p_i}.

- If C_i is of the form $\top \rightarrow p$, then the corresponding set equation is

$$\emptyset = X_p.$$

  This equation ensures that a cannot belong to X_p.

Given the intuition underlying the variables X_p (a belongs to X_p iff p is set to false), it is easy to prove the following lemma.

Lemma 2. *Let $\varphi = C_1 \wedge \ldots \wedge C_k$ be a Horn formula and Σ_φ the corresponding system of set equations. Then $\varphi \in \textit{Max-HSAT}(\ell)$ iff $\Sigma_\varphi \in \textit{MinVEq-SetEq}(k - \ell)$.*

Since Max-HSAT is NP-hard, this lemma implies that MinVEq-SetEq is also NP-hard. Put together, the two lemmas yield the exact complexity of the MinVEq-SetEq problem.

Theorem 1. *MinVEq-SetEq is NP-complete. NP-hardness holds even if we restrict the cardinality of the base set B to 1.*

3 Minimizing the number of violating elements

Instead of minimizing the number of violated equations, we can also minimize the number of violating elements of B.

Given an assignment σ, we say that $a \in B$ *violates an equation* of the form (1) *w.r.t.* σ if $a \in (K \cup \sigma(X_1) \cup \ldots \cup \sigma(X_m)) \,\Delta\, (L \cup \sigma(Y_1) \cup \ldots \cup \sigma(Y_n))$, where Δ denotes the symmetric difference of two sets. We say that $a \in B$ *violates the system of set equations* Σ *w.r.t.* σ if it violates some equation in Σ w.r.t. σ. Given a base set B, a set of variables $\mathbf{X} = \{Z_1, \ldots, Z_N\}$, a system Σ of k set equations and a nonnegative integer ℓ, we now ask whether there exists a B-assignment σ such that at most ℓ of the elements of B violate Σ w.r.t. σ. We call this decision problem MinVEl-SetEq. For a given ℓ, MinVEl-SetEq(ℓ) consists of all systems of set equations for which there is a B-assignment σ such that at most ℓ of the elements of B violate Σ w.r.t. σ.

In contrast to the problem MinVEq-SetEq considered in the previous section, MinVEl-SetEq can be solved in polynomial time. In order to show this, we introduce the notion of projection. Given an element $a \in B$, the *projection of an equation* of the form (1) to a is the equation

$$(K \cap \{a\}) \cup X_1 \cup \ldots \cup X_m = (L \cap \{a\}) \cup Y_1 \cup \ldots \cup Y_n. \tag{2}$$

The *projection of a system* of set equations Σ to a, Σ^a, is the system of the projections of all equations in Σ to a. Note that, for Σ^a, we use the base set $\{a\}$. Finally, the *projection of a B-assignment* σ to a is the $\{a\}$-assignment $\sigma^a \colon \mathbf{X} \to \mathfrak{P}(\{a\})$ defined as $\sigma^a(X) = \sigma(X) \cap \{a\}$.

The following facts are easy to show:

1. The element $a \in B$ violates Σ w.r.t. σ iff σ^a does not solve Σ^a.
2. Given $\{a\}$-assignments σ_a for all $a \in B$, define the B-assignment σ as

   $$\sigma(X) = \bigcup_{a \in B} \sigma_a(X) \quad \text{for all } X \in \mathbf{X}.$$

   Then we have $\sigma^a = \sigma_a$ for all $a \in B$.
3. There is a B-assignment σ such that at most ℓ of the elements of B violate Σ w.r.t. σ iff at most ℓ of the systems of set equations Σ^a ($a \in B$) are not solvable.

Thus, to check whether $\Sigma \in$ MinVEl-SetEq(ℓ), it is sufficient to check which of the systems of set equations Σ^a for $a \in B$ are solvable. This can obviously be done in polynomial time.

Theorem 2. *MinVEl-SetEq is in P.*

4 Minimizing the number of violations

A disadvantage of the measure used in the previous section is that it does not distinguish between elements that violate only one equation and those violating many equations. To overcome this problem, we count for each violating element how many equations it actually violates. We say that $a \in B$ *violates the system of set equations* Σ *p times w.r.t.* σ if it violates p equations in Σ w.r.t. σ. Further, we say that σ *violates* Σ *q times* if $q = \sum_{a \in B} p_a$ where, for each $a \in B$, the element a violates Σ p_a times w.r.t. σ.

Given a base set B, a set of variables $\mathbf{X} = \{Z_1, \ldots, Z_N\}$, a system Σ of k equations, and a positive integer ℓ, we now ask whether there is an assignment σ that violates Σ at most ℓ times. We call this decision problem MinV-SetEq. For a given ℓ, MinV-SetEq(ℓ) consists of all

systems of set equations for which there is a B-assignment σ such that σ violates Σ at most ℓ times.

It is easy to adapt the approach used in Section 2 to solve MinVEq-SetEq to this new problem. Basically, we now introduce Boolean variables $good(i, a)$ (instead of simply $good(i)$) to characterize whether the element $a \in B$ violates the ith equation. We conjoin $good(i, a)$ to the left-hand side of each of the Horn clauses derived from the ith equation for a. Furthermore, we add the Horn clauses $\top \to good(i, a)$.

Following the earlier notation, we obtain $k' + k|B|$ Horn clauses in this modified reduction, and again use φ_Σ to denote the obtained Horn formula. The following lemma implies that MinV-SetEq is in NP.

Lemma 3. *Let Σ be a system of set equations over the base set B, consisting of k equations and generating k' clauses in the reduction introduced in Section 1. Denote with $\varphi_\Sigma = C_1 \wedge \cdots \wedge C_{k'+k|B|}$ the Horn formula derived by the modified reduction. Then we have*

$$\Sigma \in \mathit{MinV\text{-}SetEq}(\ell) \quad \textit{iff} \quad \varphi_\Sigma \in \mathit{Max\text{-}HSAT}((k' + k|B|) - \ell).$$

For base sets of cardinality 1, MinV-SetEq coincides with MinVEq-SetEq, which we have shown to be NP-hard even in this restricted setting. This shows that the complexity upper bound of NP is optimal.

Theorem 3. *MinV-SetEq is NP-complete.*

5 Conclusion

Our investigation of how to approximately solve set equations was motivated by unification modulo the equational theory ACUI. The idea is that, even if there is no unifier, there may be substitutions that almost are unifiers, i.e., that almost solve the unification problem. In some applications it may be interesting to find such approximate solutions, which violate some of the equations, but in a minimal way. We have shown that, depending on how we measure violations, the complexity of the problem may stay in P or increase to NP.

As further work, we have started to look at approximate unification modulo the equational theory ACUIh, which corresponds to unification in the description logic $\mathcal{FL}_0$ [BN01]. This sort of unification can be used to detect redundancies in ontologies, and approximate unification may allow to detect more potential cases of redundancy. Since ACUIh-unification can be reduced to solving certain language equations [BN01], we thus need to investigate approximately solving language equations. In this setting, the elements of the sets are words, i.e., structured objects, and measures for violations should take this structure into account. Our investigation of unification modulo ACUI can be seen as a warm-up exercise for this more challenging task.

References

[BN01] Franz Baader and Paliath Narendran. Unification of concept terms in description logics. *J. of Symbolic Computation*, 31(3):277–305, 2001.

[DG84] W. F. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programmming*, 1(3):267–284, 1984.

[JS87] Brigitte Jaumard and Bruno Simeone. On the complexity of the maximum satisfiability problem for Horn formulas. *Inf. Process. Lett.*, 26(1):1–4, 1987.

[KN92] D. Kapur and P. Narendran. Complexity of unification problems with associative-commutative operators. *J. Automated Reasoning*, 9:261–288, 1992.

Let's Unify With Scala Pattern Matching! *

Edmund S.L. Lam[1] and Iliano Cervesato[1]

Carnegie Mellon University Qatar
sllam@qatar.cmu.edu and iliano@cmu.edu

Abstract

Scala's pattern matching framework supports algebraic data types. It also has an extensible system of user-definable pattern extractors. These advanced features enable the development of more complex term manipulations on top of Scala's primitive pattern matching infrastructure. In this paper, we discuss the development of a lightweight library for first-order term unification on top of this extensible pattern matching framework. Together with a set of combinators for writing unification control statements that resemble Scala pattern matching, this library serves as a basis for building more complex domain specific languages (e.g., constraint solvers, logical frameworks) that rely on unification.

1 Introduction

Scala [4] is a modern programming language with extensive support for both imperative and functional programming. Specifically, Scala's pattern matching framework supports algebraic data types through *case class definitions*. It also provides an extensible system of user-definable pattern extractors. These advanced features allow the programmer to build complex pattern matching routines on top of the language's native pattern matching framework (e.g., Joins and Actors [1]). Scala does not come with native support for unification, and the only open-source unification library for it [5] provides only basic utilities, which make little use of Scala's advanced features (e.g., flexible syntax, integration with pattern matching framework). In this paper, we build first-order term unification on top of this extensible pattern matching framework. We have developed a lightweight library in Scala that allows the programmer to perform unification over first-order terms. This library defines a set of combinators to implement control statements that resemble Scala's pattern matching. For instance, Figure 1 shows a unification example in the style of Scala's case matching statements.

Here, lines 1 and 2 declare `x` and `y`, two new logical variables and line 3 defines `f` as the term `F(Const(5),x)`, where `F` is a subclass of `Term`, representing an uninterpreted binary function application. All are subclasses of the `Term` type. In line 4, term `f` calls

```
1  val x: Term = new LogVar()
2  val y: Term = new LogVar()
3  val f: Term = F(Const(5),x)
4  f unify (
5     Const(4) withMgu θ => {
6        ... //  there is no θ such that F(Const(5),x)θ = Const(4)θ
7     },
8     F(y,Const(4)) >=> {
9        ... //  executes with mgu [5/y, 4/x] implicitly applied to x and y
10    }
11 )
```

Figure 1: Code snippet highlighting unification case control statement

*This paper was made possible by grants JSREP 4-003-2-001 and NPRP 4-341-1-059, from the Qatar National Research Fund (a member of the Qatar Foundation). The statements made herein are solely the responsibility of the authors.

```
1  val x: Term = new LogVar()
2  val y: Term = new LogVar()
3  val unifA = new Unif( Const(4) )
4  val unifB = new Unif( F(y,Const(4)) )
5  val f: Term = F(Const(5),x)
6  f match {
7     case unifA(θ) => ...
8     case unifB(θ) => ...
9  }
```

Figure 2: Code snippet highlighting unification in extensible pattern matching

its `unify` method with two input alternatives: lines 5–7 attempts to unify `f` and the term `Const(4)`, and if successful, the most general unifier θ is made available in the scope of the nested code sequence in line 6. Similarly, lines 8–10 defines another case that unifies `f` and `F(y,Const(4))`, but with an alternative programming flavor: if successful, `>=>` implicitly applies the most general unifier onto the logical variables `x` and `y` (logical variables double up as mutable containers), hence the instantiation of these logical variables are side-effects of the case statement. Like a matching case statement, these alternatives are tried in top-down order and the operation throws an exception if none applies. In the present scenario, the second case statement is executed. If, rather than using the case matching format, the programmer prefers to directly integrate his/her unification code with Scala's pattern matching framework, our unification library also includes a unification pattern extractor that allows this. The code fragment in Figure 1 then assumes the form shown in Figure 2.

Through Scala's extensible pattern matching framework, lines 3 and 4 declare two instances of pattern extractor `Unif` (whose implementation is discussed in Section 3) for the term `Const(4)` and `F(y,Const(4))`. These extractors are used in the matching statement (lines 7 and 8), corresponding respectively to unification cases in Figure 1. This lightweight library serves as a foundational basis for developing more complex domain specific languages (e.g., constraint solvers, logical frameworks) that rely on unification.

2 Abstract Semantics

In this section, we give an abstract semantics for our unification construct, implemented as the `unify` control statement. It describes the behavior of this construct independently of the underlying approach to implementation (a Scala embedding here). For simplicity, we focus on a core term language consisting of constants, variables and uninterpreted function symbols. We also omit typing information. We begin by introducing some notation. We write $\vec{o}$ for a tuple of syntactic objects o. We denote the extension of a sequence $\vec{o}$ with an object o as '$o, \vec{o}$', with () indicating the empty tuple. A generic substitution is denoted θ or ϕ. We write $o\theta$ for the application of θ on object o. The composition of two substitutions θ and ϕ is denoted by '$\theta \circ \phi$'. Given two first-order terms t_1 and t_2, the meta-operation $mgu(t_1, t_2)$ returns the most general unifier (mgu) θ of t_1 and t_2 if it exists, $\bot$ if not.

Standard expressions e_α Term expressions e_t Identifiers x Function symbols f

Terms $t ::= \texttt{Const}(e_\alpha) \mid \texttt{Var}(x) \mid f(\vec{e_t})$ Substitutions $\theta, \phi ::= \cdot \mid \theta, [t/x]$

Expressions $e ::= e_\alpha \mid t \mid e_t\ \texttt{apply}\ \theta \mid e_t\ \texttt{unify}\ \{\vec{u}\}$

Unify Cases $u ::= t\ \texttt{withMgu}\ \theta\ \texttt{=>}\ e \mid t\ \texttt{>=>}\ e$

Figure 3: Unification on Scala Terms: Abstract Syntax

Figure 3 shows the abstract syntax of the fragment of interest: we focus on our term

Normal Form Expression n Standard Evaluations $\langle \Psi; e\rangle \mapsto \langle \Psi; e\rangle$

Unification Evaluations: $\langle \Psi; \theta; t\rangle \mapsto \langle \Psi; \theta; e\rangle$

$$\frac{\langle \Psi; e_\alpha\rangle \mapsto \langle \Psi'; e\rangle}{\langle \Psi; \theta; e_\alpha\rangle \mapsto \langle \Psi'; \theta; e\rangle}\ \textsf{lift} \qquad \frac{\langle \Psi; e_\alpha\rangle \mapsto \langle \Psi'; e\rangle}{\langle \Psi; \theta; \texttt{Const}(e_\alpha)\rangle \mapsto \langle \Psi'; \theta; \texttt{Const}(e)\rangle}\ \textsf{const} \qquad \frac{}{\langle \Psi; \theta, [t/x]; \texttt{Var}(x)\rangle \mapsto \langle \Psi; \theta, [t/x]; t\rangle}\ \textsf{var}$$

$$\frac{\langle \Psi; \theta; \vec{e_t}\rangle \mapsto \langle \Psi'; \theta'; \vec{e'_t}\rangle}{\langle \Psi; \theta; f(\vec{e_t})\rangle \mapsto \langle \Psi'; \theta'; f(\vec{e'_t})\rangle}\ \textsf{func} \qquad \frac{\langle \Psi; \theta; e_t\rangle \mapsto \langle \Psi'; \theta'; e'_t\rangle}{\langle \Psi; \theta; e_t, \vec{e_t}\rangle \mapsto \langle \Psi'; \theta'; e'_t, \vec{e_t}\rangle}\ \textsf{seqHead} \qquad \frac{\langle \Psi; \theta; \vec{e_t}\rangle \mapsto \langle \Psi'; \theta'; \vec{e'_t}\rangle}{\langle \Psi; \theta; n, \vec{e_t}\rangle \mapsto \langle \Psi'; \theta'; n, \vec{e'_t}\rangle}\ \textsf{seqTail}$$

$$\frac{\langle \Psi; \theta; e_t\rangle \mapsto \langle \Psi'; \theta'; e'_t\rangle}{\langle \Psi; \theta; e_t\ \texttt{apply}\ \phi\rangle \mapsto \langle \Psi'; \theta'; e'_t\ \texttt{apply}\ \phi\rangle}\ \textsf{applyExp} \qquad \frac{}{\langle \Psi; \theta; n\ \texttt{apply}\ \phi\rangle \mapsto \langle \Psi; \theta; n\phi\rangle}\ \textsf{applyEnd} \qquad \frac{\langle \Psi; \theta; e_t\rangle \mapsto \langle \Psi'; \theta'; e'_t\rangle}{\langle \Psi; \theta; e_t\ \texttt{unify}\ \{\vec{u}\}\rangle \mapsto \langle \Psi'; \theta'; e'_t\ \texttt{unify}\ \{\vec{u}\}\rangle}\ \textsf{uExp}$$

$$\frac{mgu(n,t) = \bot}{\langle \Psi; \theta; n\ \texttt{unify}\ \{t\ \texttt{withMgu}\ \phi\ \texttt{=>}\ e,\ \vec{u}\}\rangle \mapsto \langle \Psi; \theta; n\ \texttt{unify}\ \{\vec{u}\}\rangle}\ \textsf{uPure1} \qquad \frac{mgu(n,t) = \phi}{\langle \Psi; \theta; n\ \texttt{unify}\ \{t\ \texttt{withMgu}\ \phi\ \texttt{=>}\ e,\ \vec{u}\}\rangle \mapsto \langle \Psi; \theta; e\rangle}\ \textsf{uPure2}$$

$$\frac{mgu(n,t) = \bot}{\langle \Psi; \theta; n\ \texttt{unify}\ \{t\ \texttt{>=>}\ e,\ \vec{u}\}\rangle \mapsto \langle \Psi; \theta; n\ \texttt{unify}\ \{\vec{u}\}\rangle}\ \textsf{uMut1} \qquad \frac{mgu(n,t) = \phi}{\langle \Psi; \theta; n\ \texttt{unify}\ \{t\ \texttt{>=>}\ e,\ \vec{u}\}\rangle \mapsto \langle \Psi; \theta \circ \phi; e\rangle}\ \textsf{uMut2}$$

Figure 4: Unification on Scala: Abstract Semantics

language and the `unify` control statement. All other syntactic fragments of Scala (referred to as "standard expressions") are treated abstractly and denoted by e_α. Types are omitted for succinctness. Expressions that are expected to evaluate to terms are denoted by e_t. We call them *term expressions*. Terms are constants $\texttt{Const}(e_\alpha)$, logical variables $\texttt{Var}(x)$ or function applications $f(\vec{e_t})$. Scala expressions e_α are agnostic to our unification library, terms t are primitive expressions, while the expression e_t `apply` θ applies substitution θ to the result of e_t. Our main focus is on the unification construct e_t `unify` $\{\vec{u}\}$, where the *subject* (resultant term of e_t) is unified with cases expressed in $\vec{u}$. We have two forms of unification cases, shown in Section 1: t `withMgu` θ `=>` e expresses the pure (immutable) unification with t resulting in mgu θ and continuing with the body expression e with no side-effects on t (instantiations has to be done explicitly by applying θ on t, i.e., t `apply` θ). On the other hand, t `>=>` e denotes mutable unification with t: e is executed with mgu θ implicitly applied (as side-effects) to logical variables appearing in t and the unification subject.

Figure 4 defines the abstract semantics of the `unify` construct by means of state transitions of the form $\langle \Psi; \theta; e\rangle \mapsto \langle \Psi'; \theta'; e'\rangle$, called *unification evaluation*. Here, Ψ represents the abstract state of the Scala runtime, θ keeps track of the mutable changes imposed by the use of mutable unify cases, and e is the current expression being evaluated. $\langle \Psi'; \theta'; e'\rangle$ represents the resultant state. Expressions in normal form are denoted by n and the abstract evaluation of a standard expression e_α is written $\langle \Psi; e_\alpha\rangle \mapsto \langle \Psi'; e\rangle$. Such abstract evaluations are lifted into a unification evaluation by the (lift) rule. Note that the resultant state is of the form e, meaning that it may be a unification expression (a term, `unify` or `apply` expression). Rules (const), (var) and (func), together with auxiliary rules (seqHead) and (seqTail), inductively define the evaluation of terms in a standard way. The rule (applyExp) evaluates expressions e_t `apply` θ when e_t is not in normal form, while (applyEnd) defines the cases when it is. Similarly, the rule (uExp) defines the evaluation of a `unify` construct where its unification term subject is not in normal form, while

```
// type Subst = Map[LogVar[Any],Term[Any]]

//  Definitions of Terms
abstract class Term[A] {
     def mgu(other: Term[A]): Option[Subst]
     def apply(theta: Subst): Term[A]
     def unify[B](cases: UnifCase[A,B]*): B
     def withMgu[B](body: Subst => B): PureUnifCase[A,B]
     def >=>[B](body: => B): ImmUnifCase[A,B]
}
class LogVar[A] extends Term[A]
case class Const[A](n:A) extends Term[A]
case class Func[A](sym: String, args: List[Term[Any]]) extends Term[A]

//  Unify cases:  t withMgu θ => e and t >=> e
abstract class UnifCase[A,B](pat: Term[A])
class PureUnifCase[A,B](pat: Term[A], body: Subst => B) extends UnifCase[A,B](pat)
class MutUnifCase[A,B](pat: Term[A], body: => B) extends UnifCase[A,B](pat)

//  Unification Pattern Extractor
class Unif[A](pat: Term[A]) {
     def unapply(t: Term[A]): Option[Subst] = t mgu pat
}
```

Figure 5: Class Declarations of the Unification Library

all other cases — (uPure1/2) and (uMut1/2) — require otherwise. Rules (uPure1/2) handle the cases where the topmost unification case to attempt is a pure unify case statement t `withMgu` θ `=>` e: (uPure1) deals with the case where this unification attempt fails (i.e., $mgu(n,t) = \bot$) during which the remaining alternatives are attempted next, while (uPure2) defines the successful case (i.e., $mgu(n,t) = \theta$), in which the body expression e is executed next, with the implicit assignment of θ with the result of $mgu(n,t)$. Rules (uMut1/2) do the same for mutable unify cases: (uMut2) is similar to (uPure2) except that it additionally composes the substitution state θ with the resultant substitution ϕ of the unification of n and t. This models logical variable instantiation as side-effects of the case statement. Note that n is guaranteed not to contain any logical variables that appear in θ, since the corresponding (var) rule would have been applied to obtain the normal form term n itself. Hence the composition '$\theta \circ \phi$' is always well-defined.

3 Implementation

We now describe our implementation of the unification library in Scala. For brevity, we discuss only the public interfaces and combinator operations of this library. The full code is open-source and available for download at https://github.com/sllam/unifscala.

Figure 5 shows the class declarations of our unification library. Substitutions are implemented as maps from logical variables to terms and aliased as the type `Subst` for convenience. Terms are represented by the class `Term[A]`. We refer to polymorphic type `A` as the base type, and require that it is not a `Term` type itself. Terms are extended by three subclasses, namely `LogVar[A]` that represents logical variables, `Const[A]` that represents a constant value and `Func[A]` that represents a function application, where `A` is some base type. Constants and function applications are declared as 'case' classes and hence are treated as algebraic datatypes for convenient pattern matching. Logical variables are omitted from this however, since they are uniquely identified as references, rather than by their structure. Terms support a number of operations: `mgu(t)` unifies the current term with `t` and returns their most general unifier, if it exists. This method corresponds to the meta-operation $mgu(t,t')$ used in Section 2 and encapsulates the actual implementation of

first-order term unification. The method `apply(`θ`)` simply applies the substitution θ onto the current term and returns the resultant term of this application. The method `unify` implements the control statement of the same name and takes a sequence of unification cases of type `UnifCase[A,B]`, where `A` is the base type of the term subject and `B` the return value of the statement. Unification cases are constructed by two methods, namely `withMgu` and `>=>`, respectively corresponding to the syntax entities introduced in Figure 3. A term t calling `withMgu` takes a higher-order function θ `=>` e (of type `Subst => B`) and constructs the unification case t `withMgu` θ `=>` e. Similarly, t calling `>=>` takes a nullary higher-order function e (of type `=> B`) and constructs the unification case t `>=>` e. To support mutable side-effects, logical variables double as containers that can be instantiated with values. We omit these code fragments since their implementations are fairly standard.

We integrate our unification library into Scala's pattern matching framework by means of the extractor class `Unif`. Scala's extensible pattern matching framework is best described by the following classic example:

```
1  object Twice {
2    def unapply(x: Int): Option[Int] = if(x%2==0) Some(x/2) else None
3    def test(x: Int) {
4      x match {
5        case Twice(y) => println(x + "is even and twice " + y)
6        case _ => println(x + " is odd") } }
7  }
```

Here, `Twice` is declared as an object with an `unapply` method that takes an integer `x` and returns the value of half `x` only if `x` is even. This `unapply` method is implicitly called during pattern matching (in the illustrative method `test`) to extract the corresponding value `y`. As shown in Figure 5, we declare `Unif` as a class of extractors: given a term pattern `pat`, `Unif(pat)` constructs a unification pattern extractor for `pat`. The `unapply` method of this class of extractors simply takes a term `t` and attempts to unify `t` with `pat` to produce an mgu (i.e., `t mgu pat`). Looking back to Figure 2, unification pattern extractors for patterns `Const(4)` and `F(y,Const(5))` were defined in this manner and used to extract most general unifiers.

4 Conclusion

We have developed a lightweight unification library in Scala that allows the programmer to express first-order term unification natively using either Scala's pattern matching framework or with combinators that resemble Scala's `match` statement. With this unification library as the core, we intend to build more sophisticated programming constructs, for instance backtracking statements that integrate with unification and logical reasoning frameworks. Our work here contributes a step towards easier development of high-level declarative domain specific languages (e.g., [2, 3]) that rely on unification.

References

[1] P. Haller and T. van Cutsem. Implementing Joins Using Extensible Pattern Matching. In *COORDINATION'08*, pages 135–152. Springer LNCS 5052.

[2] A. S. Köksal, V. Kuncak, and P. Suter. Constraints as control. In *POPL'12*, pages 151–164.

[3] E. S.L. Lam, I. Cervesato, and N. Fatima Haque. Comingle: Distributed Logic Programming for Decentralized Mobile Ensembles. In *COORDINATION'15*, pages 51–66. Springer LNCS 9037.

[4] Martin Odersky and al. An Overview of the Scala Programming Language. Technical Report IC/2004/64, EPFL, Lausanne, Switzerland, 2004.

[5] F. Raiser. Scala-Logic Library. https://github.com/FrankRaiser/scala-logic.

Type unification for structural types in Java

– Extended Abstract –

Martin Plümicke

Baden-Wuerttemberg Cooperative State University Stuttgart/Horb
pl@dhbw.de

In the past we considered type inference for Java with generics and lambda-expressions. The base of our algorithm was a finitary type unification. The algorithm determines nominal types in subjection to a given environment. This is a hard restriction as separate compilation of Java classes without relying on type informations of other classes is impossible. Let us consider the following example:

```
import java.util.Vector;
class A { m (v) { return v.elementAt(0); } }
```

For the method `m` the type `Vector<A>` $\to$ `A` is inferred, as `Vector` is the only class in the environment. This type is not principal. The principal type of `m` would be a structural type `ST<A>`, that have a method `elementAt`: `ST<A>` $\to$ `A`.
We present an extended type unification algorithm as the base of a type inference algorithm for a Java-like language, that infers structural types without given environments.

The type unification algorithm

The *type unification problem* is given as: For a set of constraints $\{\,\theta_1 \lessdot \theta'_1, \ldots, \theta_n \lessdot \theta'_n\,\}$, where θ_i, θ'_j are type terms, a substitution σ is demanded, such that for all $1 \leqslant i \leqslant n : \sigma(\,\theta_i\,)$ is a subtype of $\sigma(\,\theta'_i\,)$. The substitution σ is called *type unifier*. The type unification algorithm is given by eight rules, that are applied most often as possible. If the result C is in solved form (all elements has either the form $T \doteq \theta$, $T \lessdot \theta$, or $\theta \lessdot T$, where T is a type variable) then C is the result otherwise the algorithm fails. We prove the termination of the algorithm and give a soundness and a completeness theorem.

Example

Extending the example from the beginning

```
class Vector<A> extends ST<A> {  given in as Standard Java }

class Main { main() { return new A<>().m(new Vector<Integer>(...)); }}
```

leads to the set of type constraints $C = \{\,$`Vector<Integer>` $\lessdot \nu_1, \nu_1 \lessdot$ `ST<`ν_2`>`$\,\}$. Applying the type unification algorithm to C results in $\{\,\nu_2 \doteq$ `Integer`, `Vector<Integer>` $\lessdot \nu_1, \nu_1 \lessdot$ `ST<Integer>`$\,\}$. The type inferred program is then given as

```
class A { ν2 m (ν1 v) { return v.elementAt(0); } }

class Main { Integer main() { return new A<>().m(new Vector<Integer>(...)); }}
```

Overlap and Independence in Multiset Comprehension Patterns*

Iliano Cervesato[1] and Edmund S.L. Lam[1]

Carnegie Mellon University
sllam@qatar.cmu.edu and iliano@cmu.edu

Abstract

Rule-based programming, a model of computation by which rules modify a global state by concurrently rewriting disjoint portions of it, is gaining popularity as a simple, effective, and declarative means for implementing concurrent [1, 4] and distributed applications [5]. The state is often a multiset of first-order facts, with the rules specifying transformations over them. A rule consists of a head pattern which identifies a fragment of the state and a body which prescribes actions, often replacing this fragment with new facts. In this paper, we study rule interaction. Specifically, we identify conditions when two rule heads overlap by possibly targeting the same facts, which in many languages would prevent applying them concurrently — non-overlapping rules are independent and can always be executed in parallel without the risk of interference. A precise way of characterizing which rules are independent and which overlap is at the basis of numerous optimizations in the runtime of rule-based languages. It is also a useful debugging tool for programmers. We carry out this study in the context of Comingle [5], a rule-based language for programming mobile distributed applications. Head patterns in Comingle consist not only of partially instantiated facts, as found in most rule-based languages, but also of constraints and multiset comprehension patterns, which adds a challenging twist to our endeavor and interesting avenues of future work.

1 Multiset Comprehension Patterns

In this section, we formalize the syntax and matching semantics of a fragment of Comingle [5]. But first, some notation. We write $\overline{o}$ for a multiset of syntactic objects o. We denote the extension of a multiset $\overline{o}$ with an object o as "$\overline{o}, o$", with $\varnothing$ indicating the empty multiset. We also write "$\overline{o}_1, \overline{o}_2$" for the union of multisets $\overline{o}_1$ and $\overline{o}_2$. We write $\vec{o}$ for a tuple of o's and $[\vec{t}/\vec{x}]o$ for the simultaneous substitution within object o of all free occurrences of variable x_i in $\vec{x}$ with the corresponding term t_i in $\vec{t}$. A generic substitution is denoted θ. Substitution implicitly α-renames bound variables as needed to avoid capture.

Syntax. The top part of Figure 1 defines the abstract syntax of Comingle's head patterns. Computation in Comingle happens by rewriting *facts* F of the form $p(\vec{t})$ where p is a predicate symbol and $\vec{t}$ is a tuple of *terms*. The semantics of Comingle is largely agnostic to the specific language of terms as long as it is predicative — in this paper, we assume a first-order term language, later extended with primitive multisets.

Comingle *head patterns*, written H, have the form $\overline{E} \mid g$. We refer to $\overline{E}$ as the *template* of the pattern and to g as its *guard*. The template $\overline{E}$ consists of *atoms* F and of *comprehension patterns* [5] of the form $\wr F \mid g\int_{\vec{x}\to ts}$. An atom F is a fact $p(\vec{t})$ that may contain variables in the

*This paper was made possible by grants NPRP 4-1593-1-260 and NPRP 4-341-1-059, from the Qatar National Research Fund (a member of the Qatar Foundation). The statements made herein are solely the responsibility of the authors.

Syntax

$$\textit{Variables: } x \qquad \textit{Terms: } t \qquad \textit{Guards: } g \qquad \textit{Predicates symbols: } p$$

$$\begin{array}{llll} \textit{Facts} & F & ::= & p(\vec{t}) \\ \textit{Expressions} & E & ::= & F \mid \wr F \mid g \int_{\vec{x} \to t} \end{array} \qquad \textit{Head Patterns} \quad H ::= \overline{E} \mid g$$

Matching:
$\overline{E} \triangleq_{\mathsf{head}} St$
$E \triangleq_{\mathsf{head}} St$

$$\frac{\overline{E} \triangleq_{\mathsf{head}} St \quad E \triangleq_{\mathsf{head}} St'}{\overline{E}, E \triangleq_{\mathsf{head}} St, St'} \; (\mathsf{h}_{mset\text{-}1}) \qquad \frac{}{\varnothing \triangleq_{\mathsf{head}} \varnothing} \; (\mathsf{h}_{mset\text{-}2}) \qquad \frac{}{F \triangleq_{\mathsf{head}} F} \; (\mathsf{h}_{fact})$$

$$\frac{\models [\vec{t}/\vec{x}]g \quad \wr F \mid g \int_{\vec{x} \to \overline{ts}} \triangleq_{\mathsf{head}} St}{\wr F \mid g \int_{\vec{x} \to \vec{t}, \overline{ts}} \triangleq_{\mathsf{head}} St, [\vec{t}/\vec{x}]F} \; (\mathsf{h}_{comp\text{-}1}) \qquad \frac{}{\wr F \mid g \int_{\vec{x} \to \varnothing} \triangleq_{\mathsf{head}} \varnothing} \; (\mathsf{h}_{comp\text{-}2})$$

Residual:
$\overline{E} \triangleq^{\neg}_{\mathsf{head}} St$
$E \triangleq^{\neg}_{\mathsf{head}} St$

$$\frac{\overline{E} \triangleq^{\neg}_{\mathsf{head}} St \quad E \triangleq^{\neg}_{\mathsf{head}} St}{\overline{E}, E \triangleq^{\neg}_{\mathsf{head}} St} \; (\mathsf{h}^{\neg}_{mset\text{-}1}) \qquad \frac{}{\varnothing \triangleq^{\neg}_{\mathsf{head}} St} \; (\mathsf{h}^{\neg}_{mset\text{-}2}) \qquad \frac{}{F \triangleq^{\neg}_{\mathsf{head}} St} \; (\mathsf{h}^{\neg}_{fact})$$

$$\frac{F' \not\sqsubseteq \wr F \mid g \int_{\vec{x} \to ts} \quad \wr F \mid g \int_{\vec{x} \to ts} \triangleq^{\neg}_{\mathsf{head}} St}{\wr F \mid g \int_{\vec{x} \to ts} \triangleq^{\neg}_{\mathsf{head}} St, F'} \; (\mathsf{h}^{\neg}_{comp\text{-}1}) \qquad \frac{}{\wr F \mid g \int_{\vec{x} \to ts} \triangleq^{\neg}_{\mathsf{head}} \varnothing} \; (\mathsf{h}^{\neg}_{comp\text{-}2})$$

where $F' \sqsubseteq \wr F \mid g \int_{\vec{x} \to ts}$ iff $F' = \theta F$ and $\models \theta g$ for some $\theta = [\vec{t}/\vec{x}]$

Head match:
$St \rightarrowtail^{H} St$

$$\frac{\theta\overline{E} \triangleq_{\mathsf{head}} St^{+} \quad \theta\overline{E} \triangleq^{\neg}_{\mathsf{head}} St^{-} \quad \models \theta g}{St^{+}, St^{-} \rightarrowtail^{\overline{E}|g} St^{-}} \; (\mathsf{match})$$

Figure 1: Matching a Rule Head

terms $\vec{t}$. Guards in patterns and comprehensions are Boolean-valued expressions constructed from terms and are used to constrain the values that the variables can assume. Just like for terms we keep guards abstract, writing $\models g$ to express that ground guard g is satisfiable. Two common types of guards are term equality $t = t'$ and multiset membership $t \in ts$. We drop the guard from patterns and comprehensions when it is the always-satisfiable constant $\top$. A comprehension pattern $\wr F \mid g \int_{\vec{x} \to ts}$ represents a multiset of all facts that match the atom F and satisfy guard g under the bindings of variables $\vec{x}$ that range over ts, a multiset of tuples called the *comprehension range*. We call F the *subject* of the comprehension. The scope of $\vec{x}$ is the atom F and the guard g. We implicitly α-rename bound variables to avoid capture.

Given two head patterns $H_1 = \overline{E}_1 \mid g_1$ and $H_2 = \overline{E}_2 \mid g_2$ without (free) variables in common, we define the *parallel composition* of H_1 and H_2, written $H_1 \parallel H_2$, as the head pattern $\overline{E}_1, \overline{E}_2 \mid g_1 \wedge g_2$.

Matching. During computation, Comingle head patterns are matched against a *state*, written St, which is a multiset of ground facts $p(\vec{t})$. We describe the successful match of a head pattern $H = \overline{E} \mid g$ against a state St by means of the judgment $St \rightarrowtail^{H} St'$, with St' collecting the portion of the state St that was not matched by the pattern.

2

Let $\overline{E}$ be a ground template — we will deal with the more general case momentarily. Intuitively, matching $\overline{E}$ against a store St means splitting St into two parts, St^+ and St^-, and checking that $\overline{E}$ matches St^+ completely. The latter is achieved by the judgment $\overline{E} \triangleq\!=_{\mathtt{head}} St^+$ defined in the upper central part of Figure 1. Rules $\mathsf{h}_{mset\text{-}*}$ partition St^+ into fragments to be matched by each atom in $\overline{E}$: plain facts F must occur identically (rule h_{fact}) while for comprehension atoms $\wr F \mid g \int_{\vec{x}\to ts}$ the state fragment must contain a distinct instance of F for every element of the comprehension range ts that satisfies the comprehension guard g (rules $\mathsf{h}_{comp\text{-}*}$).

In Comingle, comprehension patterns must match *maximal* fragments of the state. Therefore, no comprehension pattern in $\overline{E}$ should match any fact in St^-. This check is captured by the judgment $\overline{E} \triangleq\!=^{\neg}_{\mathtt{head}} St^-$ in the lower central part of Figure 1: it holds unless St^- contains a fact F' and $\overline{E}$ a pattern $\wr F \mid g \int_{\vec{x}\to ts}$ for which there exists a substitution θ such that $F' = \theta F$ and guard θg is satisfied. Rules $\mathsf{h}^{\neg}_{mset\text{-}*}$ test each individual atom and rule $\mathsf{h}^{\neg}_{fact}$ ignore facts. Rules $\mathsf{h}^{\neg}_{comp\text{-}*}$ deal with comprehensions $\wr F \mid g \int_{\vec{x}\to ts}$: they check that no fact in St^- matches any instance of F while satisfying g.

Rule (match) in Figure 1 describes head matching in Comingle. This involves identifying a ground instance of the pattern obtained by means of a substitution θ. The instantiated guard must be satisfiable ($\models \theta g$) and we must be able to partition the state into two parts St^+ and St^-. The instance of the template must match St^+ ($\theta\overline{E} \triangleq\!=_{\mathtt{head}} St^+$), while the remaining fragment St^- must not match any comprehension in it ($\theta\overline{E} \triangleq\!=^{\neg}_{\mathtt{head}} St^-$). In Comingle, rules also contain a body B, another template, and the body instance θB is then unfolded into ground facts St_B which replaces St^+ in the state. A formal description and examples of use can be found in [5].

2 Overlapping Patterns

Two head patterns $H_1 = \overline{E}_1 \mid g_1$ and $H_2 = \overline{E}_2 \mid g_2$ without variables in common *overlap* if the former consumes facts that may prevent the latter from being applicable. This happens if there is a state St such that $St \overset{H_1}{\rightarrowtail} St_1$ and $St \overset{H_2}{\rightarrowtail} St_2$ for some St_1 and St_2, but there is no St' such that $St \overset{H_1 \| H_2}{\rightarrowtail} St'$. Head patterns H_1 and H_2 (without common variables) are *independent* if they do not overlap. For example, $H_1 = p(a, X), q(X)$ and $H_2 = p(Y, Y), r(Z)$ overlap, for instance in state $p(a, a), q(a), r(b)$, while H_1 and $H_2' = p(b, Y), r(Z)$ are independent.

The above definitions do not provide any guidance as to how to determine whether two head patterns overlap (or are independent). In the rest of this section, we devise effective conditions to make this determination in some common cases. We proceed incrementally, starting from simplified patterns and progressing towards the general case, which remains unsolved in its full generality.

Multisets We start with head patterns of the form $H = \overline{F}$, featuring an empty guard and no comprehensions. Then, H_1 and H_2 overlap if and only if one contains a fact unifiable in the other template. In symbols, if $H_1 = p(\vec{t_1}), \overline{F}'_1$ and $H_2 = p(\vec{t_2}), \overline{F}'_2$, and moreover there is a substitution θ such that $\theta\vec{t_1} = \theta\vec{t_2}$.

This was the situation in our earlier example. Note that the pair of facts $p(\vec{t_1})$ and $p(\vec{t_2})$ may not be unique in H_1 and H_2. While this makes determining independence a combinatorial problem, head patterns in typical rule-based language contain just a handful of atoms.

In a traditional first-order language, the substitution θ can be conveniently chosen to be the most general unifier (mgu) of $p(\vec{t_1})$ and $p(\vec{t_2})$, but no such concept exists for more complex term languages, for example languages featuring term-level multisets [2] or higher-order terms.

Guards We next add guards, but still no comprehensions. Our head patterns then assume the form $H = \overline{F} \mid g$. Now, head patterns $H_1 = \overline{F}_1 \mid g_1$ and $H_2 = \overline{F}_2 \mid g_2$ without common variables overlap if their template contains a unifiable fact and both guards hold for the witness substitution. In symbols, H_1 and H_2 overlap if $H_1 = p(\vec{t}_1), \overline{F}'_1 \mid g_1$ and $H_2 = p(\vec{t}_2), \overline{F}'_2 \mid g_2$, and moreover there is a substitution θ such that $\theta\vec{t_1} = \theta\vec{t_2}$ and $\models \theta g_1$ and $\models \theta g_2$. For example, in a term language over integers, the patterns $H_1 = p(X) \mid X > 3$ and $H_2 = p(Y) \mid Y < 10$ overlap for the state $p(7)$ for instance, while H_1 and $H'_2 = p(Y) \mid Y < 3$ are independent.

An effective approach to finding overlaps, for instance in this last example, is to compute unifiers θ between candidate facts $p(\vec{t}_1)$ and $p(\vec{t}_2)$, and then pass the partially instantiated guards θg_1 and θg_2 to an SMT solver such as Z3 [3] to ascertain satisfiability.

Multisets of facts, possibly constrained through guards, are found in a majority of rule-based languages, for example CHR [4]. This makes the above approach widely applicable. Comprehensions in Comingle complicate this picture, however, as we explore next.

Open-ended Comprehensions We will now allow multiset comprehensions among the atoms of a head pattern, but impose the restriction that their comprehension range does not appear anywhere else in the template, in particular not in guards. Therefore, we shall accept the pattern $p(X), \wr p(x) \mid x > 0 \int_{x \to Xs}$, but not $p(X), \wr p(x) \mid x > 0 \int_{x \to Xs} \mid |Xs| = 0$ where $|M|$ returns the number of elements in multiset M.

Given two head patterns $H_1 = \overline{E}_1 \mid g_1$ and $H_2 = \overline{E}_2 \mid g_2$ with the above characteristics (and without common variables), it may come as a surprise that the procedure we just saw in the absence of comprehensions still determines overlaps. The reason is that an open-ended comprehension — one whose range we do not constrain through guards or other mechanisms — can never fail: at worst it will return an empty multiset through its range.

As a simple example, consider the templates $H_1 = p(X)$, which consists of a single fact, and $H_2 = \wr p(x) \int_{x \to Xs}$. Template H_1 succeeds in any state that contains at least one fact headed by the predicate symbol p. For example, $p(a) \xrightarrow{H_1} \varnothing$. Instead, H_2 always succeeds: indeed $p(a) \xrightarrow{H_2} \varnothing$ and $\varnothing \xrightarrow{H_2} \varnothing$. Putting them together, we have that $p(a) \xrightarrow{H_1 \| H_2} \varnothing$. Note however that H_2 matches different portions of this state when applied in isolation (it matches $p(a)$) and when composed with H_1 (it matches $\varnothing$), a behavior that does not manifest in the absence of comprehensions. This weak form of interaction, although not an overlap according to our definition, has implications when implementing Comingle [5].

General Comprehensions Constraints over comprehension ranges, whether in guards or through shared variables, significantly complicate determining whether head patterns are independent. Consider the following pair of patterns:

$$A)\quad H_1 = \wr p(x) \int_{x \to Xs}, q(Y) \mid Y \in Xs \qquad \textit{and} \qquad H_2 = p(Z)$$

H_1 and H_2 overlap in states such as $p(a), q(a)$, in which each succeeds separately, but fail when composed as H_2 grabs $p(a)$ forcing the comprehension range Xs to be instantiated to the empty multiset, thereby failing the guard.

Membership constraints are not the only guards that lead to overlap in the presence of comprehensions. Consider the following cardinality constraint:

$$B)\quad H_1 = \wr p(x) \int_{x \to Xs} \mid |Xs| > 0 \qquad \textit{and} \qquad H_2 = p(Z)$$

Here, the state $p(a)$ witnesses the overlap as the combined head patterns fails since the range Xs will be instantiated to the empty multiset, which violates the constraint.

Explicit guards are not even needed to cause overlap. Consider these head patterns:

C) $H_1 = \wr p(x) \int_{x \to Xs}, \wr q(y) \int_{y \to Xs},$ *and* $H_2 = p(Z)$

and the state $p(a), q(a)$. On it, they succeed independently, but fail when composed.

In each of these examples, comprehension ranges appear more than once in one of the head patterns. This is not a sufficient condition, however, for the presence of overlap. The following three examples use the same operations on comprehension ranges seen earlier (membership test, cardinality constraints, and variable sharing) and yet the patterns appearing on each row are independent.

D) $H_1 = \wr p(x) \mid x < 3 \int_{x \to Xs}, q(Y) \mid Y \in Xs$ *and* $H_2 = p(Z) \mid Z > 5$
E) $H_1 = \wr p(x) \int_{x \to Xs} \mid |Xs| \leq 8$ *and* $H_2 = p(Z)$
F) $H_1 = \wr p(x) \int_{x \to Xs}, \wr q(y) \mid y \in Xs \int_{y \to Ys},$ *and* $H_2 = p(Z)$

Situation (D) differs from (A) in that the two bounds are such that no fact $p(n)$ can match both patterns. Example (E) imposes an upper bound on the comprehension range that matches H_1, while (B) forced a lower bound (that H_2 encroaches upon). Finally, the second occurrence of Xs in (F) filters out values for Ys rather than requiring that some terms be present.

Negation-as-absence is a common variant of (B) whose patterns are independent:

G) $H_1 = \wr p(x) \int_{x \to Xs} \mid |Xs| = 0$ *and* $H_2 = p(Z)$

Here, H_1 asks for the state not to contain predicates of the form $p(n)$ (alternatively, this constraint could be written as the guard $Xs = \varnothing$). Here H_1 and H_2 are independent as there is no state in which both can succeed.

These examples illustrate the rich space of behaviors that constrained comprehension patterns opens to. A general algorithmic approach to determining when such patterns overlap (or are independent) has not been found yet, however. This will be the focus of future research.

3 Conclusions

In this paper, we have explored overlap and independence, two forms of head pattern interaction in rule-based languages. We specifically focused on the head constructions of Comingle [5], a language for programming mobile distributed applications with roots in advanced forms of multiset rewriting. While the form of head patterns found in traditional languages lend itself to a simple procedural characterization of these properties, a similar recipe for the full range of constructs found in Comingle has so far proved elusive. Specifically, the interaction of constraints (or guards) and multiset comprehensions was shown to be diverse and varied. In future work, we intend to study this interaction at a deeper level, and to devise effective algorithms for overlap an independence in full Comingle.

References

[1] M. P. Ashley-Rollman et al. A Language for Large Ensembles of Independently Executing Nodes. In *ICLP'09*, pages 265–280. Springer LNCS 5649.

[2] I. Cervesato. Solution Count for Multiset Unification with Trailing Multiset Variables. In *UNIF'02*.

[3] L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS'08*. Springer-Verlag.

[4] T. Frühwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[5] E. S. Lam, I. Cervesato, and N. F. Haque. Comingle: Distributed Logic Programming for Decentralized Mobile Ensembles. In *COORDINATION'15*. Springer LNCS 9037.

Universal freeness and admissibility

Michał M. Stronkowski*

Warsaw University of Technology
m.stronkowski@mini.pw.edu.pl

Abstract

We study admissibility of multi-conclusion rules algebraically. For single-conclusion consequence relations the method is well established: admissibility corresponds to validity on free algebras in quasivarieties. An algebraic counterpart of a multi-conclusion consequence relation is a universal class (and of a single-conclusion one a quasivariety). The main obstruction here is that universal classes need not have free algebras. We show how to overcome this difficulty.

We apply our result and show that Blok-Esakia isomorphism between intermediate multi-conclusion consequence relations and modal Grzegorczyk multi-conclusion consequence relations preserves structural completeness and universal completeness.

1 Introduction

Admissibility and Structural completeness are well established notions for single-conclusion consequence relations (scr) and quasivarieties. However its extension to multi-conclusion consequence relations (mcr) is recent. In the paper [9] Rosalie Iemhoff studied admissibility for multi-conclusion rules from the perspective of proof theory. She proved the following fact (see the next sections for definitions).

Theorem 1 ([9, Proposition 6])**.** *A rule Γ/Δ is admissible for a multi-conclusion consequence relation $\vdash$ if and only if for every substitution and for every finite set of formulas Σ*

$$\vdash \bigwedge \sigma(\Gamma), \Sigma \quad \textit{yields} \quad \vdash \sigma(\Delta), \Sigma.$$

A study of admissibility for multi-conclusion rules with the aid of algebraic methods was undertaken by George Metcalfe in [12]. However the investigation presented there are limited to quasivarieties. More specifically, mcrs appearing in this way are least extensions of scrs. The main obstacle here is the possibility of lack of free algebras for universal classes. Indeed, free algebras exist for every quasivariety, and the admissibility may be described as validity on free algebras. We show how to overcome this difficulty in Theorem 5. This is our main contribution in this note.

We apply our results in the context of intermediate and modal mcrs. In Theorem 11 we show that the Blok-Esakia isomorphisms preserves structural completeness and universal completeness.

2 Multi-conclusion consequence relations

Let us fix a language $\mathcal{L}$ (i.e., a set of variables and a set of symbols of operations with ascribed arities). Let **Form** be the algebra of all formulas (terms) in $\mathcal{L}$. A (multi-conclusion) *rule* (in $\mathcal{L}$) is an ordered pair, written as Γ/Δ, of finite subsets of $Form$. In case when $|\Delta| = 1$ we talk about *single-conclusion rules*. A set of inference rules, written as a relation $\vdash$, is a *multi-conclusion consequence relation*, *mcr* in short, if for every finite subsets $\Gamma, \Gamma', \Delta, \Delta'$ of $Form$, for every $\varphi \in Form$ and for every substitution $\sigma\colon$ **From** $\to$ **From** the following conditions are satisfied

- $\varphi \vdash \varphi$;

*The work was supported by the Polish National Science Centre grant no. DEC- 2011/01/D/ST1/06136.

- if $\Gamma \vdash \Delta$, then $\Gamma, \Gamma' \vdash \Delta, \Delta'$;
- if $\Gamma \vdash \Delta, \varphi$ and $\Gamma, \varphi \vdash \Delta$, then $\Gamma \vdash \Delta$;
- if $\Gamma \vdash \Delta$, then $\sigma(\Gamma) \vdash \sigma(\Delta)$.

(We adopt the common convention and omit the curly brackets for sets, write commas for unions and omit the empty set.)

A rule r is *admissible* for a mcr $\vdash$ if for every finite set Δ of formulas the condition $\vdash_r \Delta$ yields that $\vdash \Delta$. Here $\vdash_r$ is a least mcr extending $\vdash$ and containing r. A mcr is *universally complete* if the sets of its admissible and derivable rules (those which are in $\vdash$) coincide. And a mcr is *structurally complete* if the sets of its admissible single-conclusion and derivable single-conclusion rules coincide.

Note that for a logic or, more generally, for saturated mcr $\vdash$ a rule Γ/Δ is not admissible iff there is a substitution σ such that $\vdash \sigma(\Delta)$ and $\nvdash \sigma(\delta)$ for every $\delta \in \Delta$, i.e., when the disunification problem for Γ/Δ has no solution [1].

Admissibility is much more elusive than derivability. However, adding admissible rules to mcrs may strengthen their "proof power" and essentially shorten derivations of theorems [5].

3 Universal classes

We say that a class $\mathcal{U}$ of algebras in a fixed signature is universal iff it is axiomatizable by first order sentences of the form

$$(\forall \bar{x})[s_1 \approx s'_1 \wedge \cdots \wedge s_m \approx s'_m \rightarrow t_1 \approx t'_1 \vee \cdots \vee t_n \approx t'_n],$$

where n and m are natural numbers not both equal to zero and s_i, s'_i, t_j, t'_j are arbitrary terms. We call such formulas *strict universal sentences*. When $m = 0$ we talk about *multi-identities*, when $n = 1$ about *quasi-identities*, when $m = 0$ and $n = 1$ about *identities*. Recall also that a class of algebras is a *universal positive class* if it is axiomatizable by multi-identities, a *quasivariety* if it is axiomatizable by quasi-identities, and a *variety* if it is axiomatizable by identities.

A strict universal sentence u is *admissible* for an universal class $\mathcal{U}$ if the sets of multi-identities satisfied in $\mathcal{U}$ and in the class $\{\mathbf{A} \in \mathcal{U} \mid \mathbf{A} \models u\}$ coincide.

A universal class $\mathcal{U}$ is *structurally complete* if the set of admissible for $\mathcal{U}$ quasi-identities and the set of quasi-identities valid in $\mathcal{U}$ coincide. And $\mathcal{U}$ is *universally complete* if the set of admissible for $\mathcal{U}$ strict universal sentences and the set of strict universal sentences valid in $\mathcal{U}$ coincide.

It is generally accepted that universal classes are algebraic counterparts of mcrs. But, according to the best of our knowledge, in the literature there is no notion of algebraizable mcrs extending the notion of algebraizable scrs [4]. Nevertheless, in our application we restrict to the situation where the correspondence is established.

4 Characterization of admissibility

Let $\mathcal{U}$ be a universal class and $\mathbf{A}$ be an algebra in the signature of $\mathcal{U}$. A congruence α of $\mathbf{A}$ is called *$\mathcal{U}$-congruence* if $\mathbf{A}/\alpha \in \mathcal{U}$. Let $\mathrm{Con}_{\mathcal{U}}(\mathbf{A})$ be the set of all $\mathcal{U}$-congruences of $\mathbf{A}$. Clearly, $\mathrm{Con}_{\mathcal{U}}(\mathbf{A})$ is ordered by the set inclusion. Let $\mathrm{Con}^{min}_{\mathcal{U}}(\mathbf{A})$ be the set of minimal congruences in $\mathrm{Con}_{\mathcal{U}}(\mathbf{A})$. Here is the key technical lemma.

Lemma 2. *For every $\gamma \in \mathrm{Con}_{\mathcal{U}}(\mathbf{A})$ there exists $\mu \in \mathrm{Con}^{min}_{\mathcal{U}}(\mathbf{A})$ such that $\mu \subseteq \gamma$.*

Proof. We use the Zorn Lemma. Indeed, the intersection of a chain of $\mathcal{U}$-congruences of $\mathbf{A}$ is a $\mathcal{U}$-congruence. In order to see this let us consider such a chain C and let $\alpha = \bigcap C$. Let $u = (\forall \bar{x}) \bigwedge_{i \in I} s_i(\bar{x}) \approx s'_i(\bar{x}) \rightarrow \bigvee_{j \in J} t_j(\bar{x}) \approx t'_j(\bar{x})$ be a strict universal sentence valid in $\mathcal{U}$. We prove that $\mathbf{A}/\alpha \models u$.

2

Suppose that $\mathbf{A}/\alpha \models \bigwedge_{i\in I} s_i(\bar{a}/\alpha) \approx s'_i(\bar{a}/\alpha)$ for some tuple $\bar{a}$ of elements from A. Then for every $\beta \in C$ we also have $\mathbf{A}/\beta \models \bigwedge_{i\in I} s_i(\bar{a}/\beta) \approx s'_i(\bar{a}/\beta)$. Since for every $\beta \in C$ we have $\mathbf{A}/\beta \models u$, the sets

$$Ind(\beta) = \{j \in J \mid \mathbf{A}/\beta \models t_j(\bar{a}/\beta) \approx t_j(\bar{a}/\beta)\}$$

are not empty. Moreover

$$\beta \subseteq \beta' \quad \text{implies} \quad Ind(\beta) \subseteq Ind(\beta').$$

This and the finiteness of all sets $Ind(\beta)$ imply that $\bigcap_{\beta\in C} Ind(\beta) \neq \emptyset$. Take $j_0 \in \bigcap_{\beta\in C} Ind(\beta)$. Then $\mathbf{A}/\beta \models t_{j_0}(\bar{a}/\beta) \approx t'_{j_0}(\bar{a}/\beta)$ for every $\beta \in C$. Hence $\mathbf{A}/\alpha \models t_{j_0}(\bar{a}/\alpha) \approx t'_{j_0}(\bar{a}/\alpha)$. This shows that $\mathbf{A}/\alpha \models u$. □

Let $\mathbf{T}$ be an algebra of terms in the signature of $\mathcal{U}$ over a fixed denumerable set $\{x_0, x_1, x_2, \ldots\}$ of variables. Define

$$\mathcal{F}_\mathcal{U} = \{\mathbf{T}/\mu \mid \mu \in \mathrm{Con}^{min}_\mathcal{U}(\mathbf{T})\}.$$

We identify the generators $x_0/\mu, x_1/\mu, \ldots$ of $\mathbf{G} = \mathbf{T}/\mu \in \mathcal{F}_\mathcal{U}$ with $x_0, x_2 \ldots$ respectively. The family $\mathcal{F}_\mathcal{U}$ is a substitute of a free algebra in a quasivariety. In particular, we have the following fact.

Lemma 3. *Let $e = (\forall \bar{x}) \bigvee_{j\in J} t_j(\bar{x}) \approx t'_j(\bar{x})$ be a multi-identity. Then the following conditions are equivalent:*

1. *$\mathcal{U} \models e$,*
2. *$\mathcal{F}_\mathcal{U} \models e$,*
3. *for every $\mathbf{G} \in \mathcal{F}_\mathcal{U}$ there exists $j \in J$ such that $\mathbf{G} \models t_j(\bar{x}) \approx t'_j(\bar{x})$.*

Proof. The implications (1)⇒(2) and (2)⇒(3) should be clear. Let us prove that (3) yields (1). Let us consider an algebra $\mathbf{A}$ from $\mathcal{U}$ and a tuple $\bar{a}$ of its elements. We show that $\mathbf{A} \models \bigvee_{j\in J} t_j(\bar{a}) \approx t'_j(\bar{a})$.

Let $h\colon \mathbf{T} \to \mathbf{A}$ be any homomorphism such that $h(x_i) = a_i$. Then $\ker(h)$ is a $\mathcal{U}$-congruence. Thus, by Lemma 2, there exists $\mathbf{G} \in \mathcal{F}_\mathcal{U}$ and a homomorphism $g\colon \mathbf{G} \to \mathbf{A}$ such that $g(x_i) = a_i$. By the assumed condition, there exists $j \in J$ such that $\mathbf{G} \models t_j(\bar{x}) \approx t'_j(\bar{x})$. Hence $\mathbf{A} \models t_j(g(\bar{x})) \approx t'_j(g(\bar{x}))$. □

This shows that if $\mathcal{U}$ and $\mathcal{W}$ are universal classes such that $\mathcal{F}_\mathcal{U} = \mathcal{F}_\mathcal{W}$, then $\mathsf{U}^+(\mathcal{U}) = \mathsf{U}^+(\mathcal{W})$, where $\mathsf{U}^+(\mathcal{K})$ denotes a least universal positive class containing $\mathcal{K}$. The converse implication is also true.

Lemma 4. *Let $\mathcal{U}$ and $\mathcal{W}$ be universal classes. Then $\mathcal{F}_\mathcal{U} = \mathcal{F}_\mathcal{W}$ if and only if $\mathsf{U}^+(\mathcal{U}) = \mathsf{U}^+(\mathcal{W})$.*

Proof. It is enough to show that $\mathcal{F}_\mathcal{U} = \mathcal{F}_{\mathsf{U}^+(\mathcal{U})}$. Since $\mathcal{U} \subseteq \mathsf{U}^+(\mathcal{U})$, by Lemma 2, for every congruence $\gamma \in \mathrm{Con}^{min}_\mathcal{U}(\mathbf{T})$ there exists $\delta \in \mathrm{Con}^{min}_{\mathsf{U}^+(\mathcal{U})}(\mathbf{T})$ such that $\delta \subseteq \gamma$. Next we use the known fact that algebras in $\mathsf{U}^+(\mathcal{U})$ are homomorphic images of algebras from $\mathcal{U}$, see [7, Exercise 3.2.2]. Thus for every $\delta \in \mathrm{Con}^{min}_{\mathsf{U}^+(\mathcal{U})}(\mathbf{T})$ there exists $\gamma \in \mathrm{Con}_\mathcal{U}(\mathbf{T})$ such that $\gamma \subseteq \delta$. By Lemma 2, we may assume that $\gamma \in \mathrm{Con}^{min}_\mathcal{U}(\mathbf{T})$. Since $\mathrm{Con}^{min}_\mathcal{U}(\mathbf{T})$ and $\mathrm{Con}^{min}_{\mathsf{U}^+(\mathcal{U})}(\mathbf{T})$ are antichains in the lattice of all congruences of $\mathbf{T}$, we obtain that $\mathrm{Con}^{min}_\mathcal{U}(\mathbf{T}) = \mathrm{Con}^{min}_{\mathsf{U}^+(\mathcal{U})}(\mathbf{T})$. □

Theorem 5. *A strict universal sentence u is admissible for $\mathcal{U}$ if and only if $\mathcal{F}_\mathcal{U} \models u$.*

Proof. Let $\mathcal{W}$ be the class of algebras from $\mathcal{U}$ satisfying u, i.e, $\mathcal{W} = \{\mathbf{A} \in \mathcal{U} \mid \mathbf{A} \models u\}$. Then u is admissible for $\mathcal{U}$ iff $\mathsf{U}^+(\mathcal{U}) = \mathsf{U}^+(\mathcal{W})$. Now if $\mathcal{F}_\mathcal{U} \models u$, then Lemma 3 gives that

$$\mathsf{U}^+(\mathcal{U}) = \mathsf{U}^+(\mathcal{F}_\mathcal{U}) \subseteq \mathsf{U}^+(\mathcal{W}) \subseteq \mathsf{U}^+(\mathcal{U})$$

and u is admissible. For the opposite implication, assume that u is admissible. Then, by Lemma 4, $\mathcal{F}_\mathcal{U} = \mathcal{F}_\mathcal{W}$. Therefore, since $\mathcal{F}_\mathcal{W} \subseteq \mathcal{W}$ and $\mathcal{W} \models u$, we have that $\mathcal{F}_\mathcal{U} \models u$. □

For a class $\mathcal{K}$ of algebras let $\mathsf{Q}(\mathcal{K})$ be a least quasivariety containing $\mathcal{K}$ and $\mathsf{U}(\mathcal{K})$ be a least universal class containing $\mathcal{K}$.

Corollary 6. *Let $\mathcal{U}$ be a universal class. Then*

- *$\mathcal{U}$ is structurally complete if and only if $\mathsf{Q}(\mathcal{U}) = \mathsf{Q}(\mathcal{F}_\mathcal{U})$,*
- *$\mathcal{U}$ is universally complete if and only if $\mathcal{U} = \mathsf{U}(\mathcal{F}_\mathcal{U})$.*

5 Application

5.1 Intermediate and modal mcrs

We are interested in intermediate and modal mcrs. Let K denotes modal logic K and INT denotes the intuitionistic logic, both interpreted as a set of formulas. Then an *intermediate mcr* is an mcr $\vdash$ in the language of INT such that $\vdash \varphi$ for every $\varphi \in INT$ and $\varphi, \varphi \to \psi \vdash \psi$ for every formulas φ, ψ. And a *modal mcr* is an mcr $\vdash$ in the language of K such that $\vdash \varphi$ for every $\varphi \in K$ and $\varphi, \varphi \to, \psi \vdash \psi$ and $\varphi \vdash \Box\varphi$ for every formulas φ, ψ. A (general, intermediate or modal) mcr is *axiomatized* by a set R of rules if it is a least (general, intermediate or modal receptively) mcr containing R.

Intermediate and modal mcrs have algebraic semantics. Let us recall that a *Heyting algebra* is a bounded lattice endowed with the binary operation $\to$ such that $a \wedge b \leqslant c$ iff $a \leqslant b \to c$ for every triple a, b, c of its elements. It appears that the class of all Heyting algebras forms a variety which constitutes a semantics for INT. A *modal algebra* is a Boolean algebra endowed with additional unary operation $\Box$ such that for all its elements a, b we have $\Box(a \wedge b) = \Box a \wedge \Box b$ and $\Box 1 = 1$. The variety of modal algebras gives a semantics for K.

Let r be a rule $\varphi_1, \dots, \varphi_m / \psi_1, \dots, \psi_n$. By the *translation* of r we mean a strict universal sentence $\mathsf{T}(r)$ given by

$$(\forall \bar{x})[\varphi_1 \approx 1 \wedge \dots \wedge \varphi_m \approx 1 \to \psi_1 \approx 1 \vee \dots \vee \psi_n \approx 1].$$

The following completeness theorem follows from [10, Theorem 2.2], see also [2, Theorem 2.5 in Appendix] for the modal case.

Theorem 7. *Let $\vdash$ be an intermediate or modal mcr axiomatized by a set of inference rules R. Let $\mathcal{U}$ be the universal class axiomatized by $\mathsf{T}(R)$ of Heyting or modal algebras respectively. Then for every inference rule r we have $r \in \vdash$ if and only if $\mathcal{U} \models \mathsf{T}(r)$.*

The above theorem allows us to switch from mcrs to universal classes. If $\vdash$ and $\mathcal{U}$ are as Theorem 7 , we say that $\mathcal{U}$ and $\vdash$ *correspond to* each other. In particular, we have the following fact.

Corollary 8. *Let $\mathcal{U}$ be a universal class of Heyting or modal algebras and $\vdash$ be an intermediate or modal mcr respectively. Assume that $\mathcal{U}$ and $\vdash$ correspond to each other. Then*

- *$\mathcal{U}$ is structurally complete if and only if $\vdash$ is structurally complete,*
- *$\mathcal{U}$ is universally complete if and only if $\vdash$ is universally complete.*

5.2 Blok-Esakia isomorphism

A modal algebra $\mathbf{M}$ is called an *interior algebra* if for every $a \in M$ it satisfies $a \geqslant \Box a = \Box\Box a$. And an interior algebra $\mathbf{M}$ is a *Grzegorczyk algebra* if it also satisfies $\Box(\Box(a \to \Box a) \to a) \leqslant a$ for every $a \in M$ [8]. Recall that the variety of modal (interior or Grzegorczyk) algebras characterizes the modal logic K (S4 or GRZ respectivelly). The connection of Heyting algebra with interior algebra is given by the following McKinsey-Tarski theorem [11, Section 1]. Recall that open elements of an interior algebra $\mathbf{M}$ form the Heyting algebra $\mathsf{O}(\mathbf{M})$ with the order structure inherited from $\mathbf{M}$.

Theorem 9. *For every Heyting algebra $\mathbf{H}$ there is an interior algebra $\mathsf{B}(\mathbf{H})$ such that*

1. *$\mathsf{OB}(\mathbf{H}) = \mathbf{H}$;*
2. *for every interior algebra $\mathbf{M}$, if $\mathbf{H} \leqslant \mathsf{O}(\mathbf{M})$, then $\mathsf{B}(\mathbf{H})$ is isomorphic to the subalgebra of $\mathbf{M}$ generated by H;*

The algebra $\mathsf{B}(\mathbf{H})$ is called the *free Boolean extension of* $\mathbf{H}$.

Let $\mathcal{H}$ be the variety of all Heyting algebras, and $\mathcal{G}$ be the variety of all Grzegorczyk algebras. Let $\mathcal{L}_U(\mathcal{H})$ and $\mathcal{L}_U(\mathcal{G})$ be the lattices of all universal classes of Heyting algebras and Grzegorczyk algebras respectively. Let us define two operators on these lattices. For $\mathcal{U} \in \mathcal{L}_U(\mathcal{H})$ and $\mathcal{Y} \in \mathcal{L}_U(\mathcal{G})$ we put

$$\rho\colon \mathcal{L}_U(\mathcal{G}) \to \mathcal{L}_U(\mathcal{H});\ \mathcal{Y} \mapsto \{\mathsf{O}(\mathbf{M}) \mid \mathbf{M} \in \mathcal{Y}\}$$
$$\sigma\colon \mathcal{L}_U(\mathcal{H}) \to \mathcal{L}_U(\mathcal{G});\ \mathcal{U} \mapsto \mathsf{U}(\{\mathsf{B}(\mathbf{H}) \mid \mathbf{H} \in \mathcal{U}\}).$$

Since O commutes with U, $\rho(\mathcal{Y})$ belongs to $\mathcal{L}_U(\mathcal{H})$ for $\mathcal{Y} \in \mathcal{L}_U(\mathcal{G})$. The fact that $\sigma(\mathcal{U}) \in \mathcal{L}_U(\mathcal{G})$ for $\mathcal{U} \in \mathcal{L}_U(\mathcal{H})$ is much less trivial. It holds since every free Boolean extension of a Heyting algebra is a Grzegorczyk algebra [3, Corollary III.7.9], see also [14] for the new proof.

The following extension of the Blok-Esakia theorem ([3, Theorem 7.10], [8, Theorem 7.11], [6, Theorem 9.66], [15, Section 3])) was firstly observed by Emil Jeřábek in [10, Theorem 5.5]. However one can also deduce it from [3]. A detailed exposition of the algebraic proof may be found in [14].

Theorem 10. *The mappings σ and ρ are mutually inverse isomorphisms between the lattices $\mathcal{L}_U(\mathcal{H})$ and $\mathcal{L}_U(\mathcal{G})$.*

The fact that σ and ρ preserves structural completeness for varieties was proved by Vladimir Rybakov in [13, Theorem 5.4.7] and for quasivarieties by the author in [14]. Here our contribution is the following extension.

Theorem 11. *Let $\mathcal{U}$ be a universal class of Heyting algebras. Then*

- *$\mathcal{U}$ is structurally complete if and only if $\sigma(\mathcal{U})$ is structurally complete,*
- *$\mathcal{U}$ is universally complete if and only if $\sigma(\mathcal{U})$ is universally complete.*

References

[1] Franz Baader and Klaus U. Schulz. Combination techniques and decision problems for disunification. *Theoret. Comput. Sci.*, 142(2):229–255, 1995.

[2] Nick Bezhanishvili and Silvio Ghilardi. Multiple-conclusion rules, hypersequents syntax and step frames. In *Advances in modal logic. Vol. 10*, pages 54–73. Coll. Publ., London, 2014.

[3] Willem J. Blok. *Varieties of interior algebras.* PhD thesis, University of Amsterdam, 1976.

[4] Willem J. Blok and Don Pigozzi. Algebraizable logics. *Mem. Amer. Math. Soc.*, 77(396):vi+78, 1989.

[5] George Boolos. Don't eliminate cut. *J. Philos. Logic*, 13(4):373–378, 1984.

[6] Alexander Chagrov and Michael Zakharyaschev. *Modal logic*, volume 35 of *Oxford Logic Guides.* The Clarendon Press Oxford University Press, New York, 1997. Oxford Science Publications.

[7] C. C. Chang and H. J. Keisler. *Model theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam, third edition, 1990.

[8] Leo L. Esakia. On the variety of Grzegorczyk algebras. In *Studies in nonclassical logics and set theory (Russian)*, pages 257–287. "Nauka", Moscow, 1979.

[9] Rosalie Iemhoff. Consequence relations and admissible rules. *J. Philos. Logic.* Appeared online.

[10] Emil Jeřábek. Canonical rules. *J. Symbolic Logic*, 74(4):1171–1205, 2009.

[11] J. C. C. McKinsey and Alfred Tarski. On closed elements in closure algebras. *Ann. of Math. (2)*, 47:122–162, 1946.

[12] George Metcalfe. Admissible rules: from characterizations to applications. In *Logic, language, information and computation*, volume 7456 of *Lecture Notes in Comput. Sci.*, pages 56–69. Springer, Heidelberg, 2012.

[13] Vladimir V. Rybakov. *Admissibility of logical inference rules*, volume 136 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam, 1997.

[14] Michał M. Stronkowski. On the Blok-Esakia theorem for universal classes. Submitted, 2016.

[15] Frank Wolter and Michael Zakharyaschev. On the Blok-Esakia theorem. In Guram Bezhanishvili, editor, *Leo Esakia on Duality in Modal and Intuitionistic Logics*, volume 4 of *Outstanding Contributions to Logic*, pages 99–118. Springer Netherlands, 2014.