UNIF 2008

The 22nd International Workshop on Unification

Proceedings

Editor: Mircea Marin

July 18, 2008
Castle of Hagenberg, Austria

RISC-Linz
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe

UNIF 2008
The 22nd International Workshop on Unification

Mircea MARIN (editor)

Castle of Hagenberg, Austria
July 18, 2008

RISC-Linz Report Series No. 08-11

Supported by: Linzer Hochschulfonds, Upper Austrian Government, Austrian Federal Ministry of Science and Research (BMWF), Johann Radon Institute for Computational and Applied Mathematics of the Austrian Academy of Sciences (RICAM)

Preface

This report contains the preliminary proceedings of the 22nd International Workshop on Unification (UNIF 2008). The workshop was held in the Castle of Hagenberg, Austria on July 18, 2008 and collocated with the 19th International Conference on Rewriting Techniques and Applications (RTA 2008). The aim of UNIF 2008 is to bring together people interested in unification, present recent (even unfinished) work, and discuss new ideas and trends in unification and related fields. This includes scientific presentations, but also descriptions of applications and software using unification as a strong component.

This workshop is the 22nd in the series: UNIF'87 (Val D'Ajol, France), UNIF'88 (Val D'Ajol, France), UNIF'89 (Lambrecht, Germany), UNIF'90 (Leeds, England), UNIF'91 (Barbizon, France), UNIF'92 (Dagstuhl, Germany), UNIF'93 (Boston, USA), UNIF'94 (Val D'Ajol, France), UNIF'95 (Sitges, Spain), UNIF'96 (Herrsching, Germany), UNIF'97 (Orléans, France), UNIF'98 (Rome, Italy), UNIF'99 (Frankfurt, Germany), UNIF'00 (Pittsburgh, USA), UNIF'01 (Siena, Italy), UNIF'02 (Copenhagen, Denmark). UNIF'03 (Valencia, Spain). UNIF'04 (Cork, Ireland). UNIF'05 (Nara, Japan). UNIF'06 (Seattle, USA). UNIF'07 (Paris, France).

UNIF 2008 received 9 submissions, and every paper was carefully reviewed by 2 reviews. We have accepted 8 papers, which are included in this report. The submission and programme committee work was organized through the EasyChair system.

I would like to thank all authors of submitted papers, the Program Committee members, the referees for their time and effort spent in the reviewing process, and the RTA 2008 organizers for hosting our workshop. The financial support of the following sponsors is gratefully acknowledged: Linzer Hochschulfonds, Upper Austrian Government, Austrian Federal Ministry of Science and Research (BMWF), and Johann Radon Institute for Computational and Applied Mathematics of the Austrian Academy of Sciences (RICAM).

July 2008 *Mircea Marin*

Conference Organization

Programme Chair

Mircea Marin	*Japan*

Programme Committee

Rachid Echahed	*France*
Temur Kutsia	*Austria*
Paliath Narendran	*USA*
Joachim Niehren	*France*
Toshiyuki Yamada	*Japan*
Rafael del Vado Virseda	*Spain*

Local Organization

Temur Kutsia

External Reviewers

Siva Anantharaman
Francisco Lopez-Fraguas
Manuel Montenegro
Wojciech Plandowski
Carlos A. Romero-Diaz

Table of Contents

Unification modulo Homomorphic Encryption is Decidable

Siva Anantharaman[1], Hai Lin[2], Christopher Lynch[2], Paliath Narendran[3], and Michael Rusinowitch[4]

[1] Université d'Orléans (Fr.) (`siva@univ-orleans.fr`)
[2] Clarkson University, Potsdam, NY, USA ({`clynch, linh`}`@clarkson.edu`)
[3] University at Albany-SUNY, USA (`dran@cs.albany.edu`)
[4] Loria-INRIA Lorraine, Nancy (Fr.) (`rusi@loria.fr`)

1 Introduction

Several methods based on rewriting have been proposed, with success, for the formal analysis of cryptographic protocols. They all have a common starting point: it is possible to model encryption and decryption operations by collapsing (right-hand sides are variables) convergent rewrite systems, which express simply that decryption cancels encryption, when provided with the right key. We thus get the following basic convergent rewrite system – referred to as the Dolev-Yao (DY) system – where '.' is the 'pairing' operation on messages, p_1, p_2 stand for the respective projections from pairs, and '*dec*' (resp.'*enc*') stands for decryption (resp. encryption); the second argument of these latter functions are usually referred to as keys:

$$\text{(DY)} \quad \begin{array}{ll} p_1(x.y) \to x & dec(enc(x,y),y) \to x \\ p_2(x.y) \to y & enc(dec(x,y),y) \to x \end{array}$$

Various decision procedures have been designed for handling other equational properties of the cryptographic primitives [8, 5, 4]. Some works have tried to derive generic decidability results for some specific *class* of intruder theories. Delaune and Jacquemard [7] consider the class of *public collapsing* theories. These theories have to be presented by rewrite systems where the rhs of every rule is a ground term or a variable. Some other results assume that the rhs of any rule is a (proper) subterm of the lhs. A general procedure for protocol security analysis has been given in [3] for such systems, extensively using equational unification and narrowing techniques. Rewrite systems with such a 'subterm' property have been called *dwindling* in [1], where a decision procedure was given for passive deduction (i.e., detecting secrecy attacks by an intruder *not* interacting actively with the protocol sessions). The technique used is one that combines unification and narrowing with a notion of cap closure, modeling the evolution of the intruder knowledge. The algorithm presented was also shown to be complete, for passive deduction, for a larger class of rewrite systems called Δ-strong, strictly including the dwindling class. This larger class includes, in particular, the theory of *Homomorphic Encryption* (HE) that plays an important role in several protocols, obtained by just extending DY by requiring that encryption distributes over pairs, HE can be defined by the following convergent, non-dwindling system, that we shall also refer to as HE:

$$
\text{(HE)}\qquad
\begin{array}{ll}
p_1(x.y) = x & \\
p_2(x.y) = y & enc(x.y, z) = enc(x, z).enc(y, z) \\
enc(dec(x, y), y) = x & dec(x.y, z) = dec(x, z).dec(y, z) \\
dec(enc(x, y), y) = x &
\end{array}
$$

We prove the following results in this paper:

(i) Unification modulo general (convergent) Δ-strong systems is undecidable.
(ii) Unification modulo HE is decidable.

Now, it is known that for active deduction modulo an intruder theory R to be decidable – i.e. in order that such an intruder can capture any message intended as secret, by interacting actively with the protocol sessions –, it is necessary that equational unification modulo R be decidable ([6]). It follows, therefore, from the result (i) above, that active deduction modulo general convergent Δ-strong intruder systems is undecidable – although passive deduction has been shown to be decidable for such theories in [1]. As for active deduction modulo HE, it is part of our ongoing work, cf. [2]. Note on the other hand, that each homomorphism $enc(-, y)$ in HE has also an inverse homomorphism $dec(-, y)$; so, the HE-unification problem does not reduce easily to unification modulo one-sided distributivity.

This paper is structured as follows: The needed preliminaries are given in Section 2. In Section 3, we establish the result (i), via a suitable reduction from MPCP. Unification modulo HE is shown to be decidable in Section 4. The main idea consists in reducing any given HE-unification problem into one of solving a 'simpler' set of equations of the form $Z = enc(X, V)$ or $Z = dec(X, V)$, with none among its variables getting split into pairs. Solving such a set of equations is essentially the unification problem modulo the two rules for encryption and decryption:

$$
\begin{array}{c}
dec(enc(x, y), y) \to x \\
enc(dec(x, y), y) \to x
\end{array}
$$

which form a confluent, dwindling system, so has a decidable unification problem, cf. [9]. However, we propose a graph-based algorithm in this work, that is specific to HE-unification problems, and *show that even solving 'simple' HE-unification problems (without pairings) is NP-complete.*

2 Notation and Preliminaries

As usual, Σ will stand for a ranked signature, and $\mathcal{X}$ a countably infinite set of variables. $\mathcal{T} = \mathcal{T}(\Sigma, \mathcal{X})$ is the algebra of terms over this signature; terms in $\mathcal{T}$ will be denoted as $s, t, \ldots$, and variables as $u, v, x, y, z, \ldots$, all with possible suffixes. The set of all positions on any term t is denoted as $Pos(t)$; if $q \in Pos(t)$, then $t|_q$ denotes the subterm of t at position q; and the term obtained from t by replacing the subterm $t|_q$ by any given term t' will be denoted as $t[q \leftarrow t']$; a similar notation is employed also for the substitution of variables of t by terms.

We assume given a simplification ordering $\succ$ on $\mathcal{T}$ that is total on ground terms (terms not containing variables). A rewrite rule is a pair of terms (l, r) such that $l \succ r$, and is represented as usual, as $l \to r$; a rewrite system is a finite set of rewrite rules. The notions of reduction and of normalization of a term by a rewrite system are assumed known, as well as those of termination and of confluence of the reduction relation defined by such a system on terms. A rewrite system R is *convergent* iff the reduction relation it defines on the set of terms is terminating and confluent. R is said to be *dwindling* iff the right-hand-side (rhs) of every rule in R is a proper subterm of its left-hand-side (lhs).

We also assume given a proper subset $\mathcal{P}$ of symbols of Σ – referred to as the set of *public* symbols – such that $\Sigma \smallsetminus \mathcal{P}$ contains at least one ground constant; the symbols of $\Sigma \smallsetminus \mathcal{P}$ will be said to be *private*. Any convergent rewrite system R, such that the top-symbol of the lhs of every rule in R is a public symbol, will be said to be an intruder theory.

Suppose R_0 is any given convergent intruder system. An n-ary public symbol f is said to be *transparent* in/for R_0, or R_0-*transparent*, if and only if, for all $x_1, \ldots, x_n$, there exist 'context-terms' (with a single hole) $t_1(\diamond), \ldots, t_n(\diamond)$ such that $t_i[\diamond \leftarrow f(x_1, \ldots, x_n)] \to^*_{R_0} x_i$, for every $1 \leq i \leq n$. For instance, the public function '.' ("pair") is transparent for the system: $p_1(x.y) \to x, \; p_2(x.y) \to y$, where p_1 and p_2 are both public. We shall consider public constants as transparent for any intruder system R_0. A public function symbol is R_0-*resistant* (or simply *resistant* if R_0 is clear from the context) iff it is not R_0-transparent. Private functions will be considered as resistant for any intruder system R_0. By definition, an R_0-resistant term is one whose top-symbol is R_0-resistant.

Let now R be any given convergent intruder theory, containing a dwindling subsystem. We shall assume that the given simplification ordering $\succ$ containing R, is precedence based (like *rpo* or *lpo*), and is such that every private symbol is greater than any public symbol, under $\succ$. Let Δ be a subsystem consisting of (some of the) dwindling rules of R. A rewrite rule $l \to r \in R$ is said to be Δ-strong, wrt the given simplification ordering $\succ$, if and only if every Δ-resistant subterm of l is greater than r wrt $\succ$. The intruder theory R is said to be Δ-strong wrt $\succ$ if and only if every rule in $R \smallsetminus \Delta$ is Δ-strong wrt $\succ$.

The rewrite system HE, presented above, is a Δ-strong system, if Δ is taken to be the subsystem formed of the four dwindling rules to the left of HE: indeed the lpo ordering built over the precedence: $enc > dec > . > p_1 > p_2$ is ground total and contains HE; and the symbols '*dec*' and '*enc*' are both Δ-resistant.

3 Unification modulo $\boldsymbol{\Delta}$-strong Theories Is Undecidable

Proposition 1 *Unification modulo general (convergent) Δ-strong theories is undecidable.*

Proof. The proof is by reduction from a restricted version of the modified Post Correspondence Problem (MPCP).

Let $\Sigma = \{a, b\}$, and let $P = \{(\phi_i, \psi_i) \mid i = 1, \ldots, n\} \subseteq \Sigma^+ \times \Sigma^+$ be a finite sequence of non-empty strings over Σ such that the following restricted version of the Modified Post Correspondence Problem (MPCP) is undecidable:
Instance: A non-empty string $\alpha \in \Sigma^+$.
Question: Do there exist indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $\alpha\phi_{i_1}\phi_{i_2}\ldots\phi_{i_k} = \psi_{i_1}\psi_{i_2}\ldots\psi_{i_k}$?

For any string w over Σ, let $\widetilde{w}(x)$ denote the term formed by treating a and b as unary function symbols and the concatenation operator as function composition; more precisely, we set:

$$\widetilde{\lambda}(x) = x, \qquad \widetilde{au}(x) = a(\widetilde{u}(x)), \qquad \widetilde{bu}(x) = b(\widetilde{u}(x)).$$

Let f be a ternary function and $g_1, \ldots, g_n$ be distinct unary function symbols. Consider then the system $\mathcal{P}$ formed of the following rewrite rules:

$$f(\widetilde{\phi_i}(x),\, g_i(y),\, \widetilde{\psi_i}(z)) \to f(x,\, y,\, z)$$

for every pair (ϕ_i, ψ_i) of the MPCP. We also add a new unary function symbol h and the following set Δ of dwindling rules:

$$\begin{aligned} h(a(x)) &\to x \\ h(b(x)) &\to x \\ h(g_i(x)) &\to x, \quad i \in \{1, \ldots, n\}. \end{aligned}$$

The effect of this addition is that the monadic functions a, b, $g_1, \ldots, g_n$ are all Δ-transparent, whereas f is Δ-resistant. The role played by the g_i is to ensure that the rewrite system has no critical pairs. The system $\mathcal{E}$ formed of all these rewrite rules (i.e., $\mathcal{P} \cup \Delta$) is therefore convergent and Δ-strong.

It is not hard then to see that the unification problem

$$f(X,\, Y,\, \widetilde{\alpha}(X)) \;=^?_{\mathcal{E}}\; f(c,\, c,\, c)$$

has a solution iff the instance of the restricted MPCP above has a solution. The "if" part is fairly straightforward, since if $\alpha\phi_{i_1}\phi_{i_2}\ldots\phi_{i_k} = \psi_{i_1}\psi_{i_2}\ldots\psi_{i_k}$ for some indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$, then the substitution

$$\tau \;=\; \{X \leftarrow \widetilde{\phi_{i_1}}\widetilde{\phi_{i_2}}\ldots\widetilde{\phi_{i_k}}(c),\; Y \leftarrow g_{i_1}g_{i_2}\ldots g_{i_k}(c)\}$$

is a solution: indeed, $\widetilde{\alpha}(\tau(X)) = \widetilde{\alpha}\,\widetilde{\phi_{i_1}}\widetilde{\phi_{i_2}}\ldots\widetilde{\phi_{i_k}}(c) = \widetilde{\psi_{i_1}}\widetilde{\psi_{i_2}}\ldots\widetilde{\psi_{i_k}}(c)$.

On the other hand, suppose θ is a solution for the above equation. Without loss of generality it can be assumed that θ is *normalized* modulo $\mathcal{E}$. Then it must be that $f(\theta(X),\, \theta(Y),\, \widetilde{\alpha}(\theta(X))) \longrightarrow^!_{\mathcal{P}} f(c,\, c,\, c)$. Now a solution for the MPCP instance can be obtained from $\theta(Y)$. □

4 Unification modulo HE

Theorem 1. *Unification modulo the theory HE is decidable.*

For the proof, we shall be applying several reductions on the given unification problem. To start with, we shall assume (via usual arguments, and reasoning mod HE) that the given unification problem $\mathcal{P}$ is in a *standard form*, in the following sense: each of its equations to solve, modulo HE, is assumed to have one of the following forms:

$$Z = T, \quad Z = X.Y, \quad Z = enc(X, Y), \quad Z = const.,$$

where the $T, X, Y, Z, \ldots$ stand for variables, and *const* is any free ground constant. (If an equation in $\mathcal{P}$ is given in the form $U = dec(V, W)$, it is rewritten mod HE as $V = enc(U, W)$.) The equations in $\mathcal{P}$ of the first (resp. second) form are said to be '*equalities*' (resp. '*pairings*'); those of the third form are said to be of the *enc* type, and the last ones of the 'constant' type. The second arguments of *enc*, in the equations of $\mathcal{P}$, are referred to as the *keys* or *key variables* of $\mathcal{P}$.

The *conjugate* of any *enc* equation $Z = enc(X, Y)$ in $\mathcal{P}$, is defined as the equation $X = dec(Z, Y)$, said to be of the '*dec*' type. For every key variable/constant Y occurring in $\mathcal{P}$, let h_Y (resp. $\overline{h}_Y$) denote the homomorphism $enc(-, Y)$ (resp. $dec(-, Y)$) defined on terms. An *enc* equation $Z = enc(X, Y)$ can thus be written as $Z = h_Y(X)$, and its conjugate as $X = \overline{h}_Y(Z)$. Let n be the number of distinct key variables/constants appearing in $\mathcal{P}$, and let $\mathcal{H}$ stand for the set of all homomorphisms ($2n$ in number), thus associated with these key variables/constants.

We construct next a graph of dependency $G = G_{\mathcal{P}}$ between the variables of the problem $\mathcal{P}$. Its nodes will be the variables (or constants) of $\mathcal{P}$. From a node Z on G, there is an oriented arc to a node X iff the following holds:

a) $\mathcal{P}$ has an equation of the form $Z = h_i(X)$, or $Z = \overline{h}_i(X)$, for some $i \in \{1, \ldots, n\}$; the arc is then labeled with the symbol h_i (resp. with $\overline{h}_i$);
b) $\mathcal{P}$ has an equation of the form $Z = X.V$ (resp. $Z = V.X$): the arc is then labelled with p_1 (resp. with p_2).

Semantics: If G contains an edge of the form $Z \to^h X$, then Z can be evaluated by applying the homomorphism h to the evaluation of X.

Several reductions, called *trimming*, will be applied to our problems. One of them ensures that the graph of $\mathcal{P}$ is irredundant, in the sense that *variables which are 'equal' in $\mathcal{P}$ will have exactly one representative node on G* (which is why $G_{\mathcal{P}}$ has no equality edges); some others result from the so-called *Perfect Encryption* assumption. The principal aim of these reductions is to ensure that the non-key variables of $\mathcal{P}$ do *not* get split into pairs. We first define the following relations on the set of variables/constants $\mathcal{X} = \mathcal{X}(\mathcal{P})$ that appear in $\mathcal{P}$:

1. $U \sim V$ is the finest equivalence relation on X such that:
   - if $U = V \in \mathcal{P}$ then $U \sim V$;
   - if $U = enc(V, T) \in \mathcal{P}$ or $V = enc(U, T) \in \mathcal{P}$, for some T, then $U \sim V$;
   - if $\mathcal{P}$ contains two pairings of the form $W = U.X$ and $W' = V.X'$ (or of the form $W = X.U$ and $W' = X'.V$), where $W \sim W'$, then $U \sim V$.
2. $U \succ V$ iff there is a loop-free chain from U to V formed of $\sim$- or p_1/p_2-steps, at least one of them being a a p_1- or p_2- step.

Rules for Trimming: We denote by **Eq** (resp. **Pair**, **Enc**) the set of equalities (resp. pairings, the *enc*-equations) in $\mathcal{P}$, respectively.

Rule 1. (*Perfect Encryption*)

$$a)\ \frac{\mathbf{Eq};\ \mathbf{Pair};\ \mathbf{Enc} \sqcup \{Z = enc(X,Y),\ Z = enc(V,Y)\}}{\mathbf{Eq} \cup \{V = X\};\ \mathbf{Pair};\ \mathbf{Enc} \sqcup \{Z = enc(X,Y)\}}$$

$$b)\ \frac{\mathbf{Eq};\ \mathbf{Pair};\ \mathbf{Enc} \sqcup \{Z = enc(X,Y),\ Z = enc(X,T)\}}{\mathbf{Eq} \cup \{T = Y\};\ \mathbf{Pair};\ \mathbf{Enc} \sqcup \{Z = enc(X,Y)\}}$$

Rule 1'. (*Variable Elimination*)

$$\frac{\{U = V\} \sqcup \mathbf{Eq};\ \mathbf{Pair};\ \mathbf{Enc}}{\{U = V\} \cup\ [V/U](\mathbf{Eq});\ [V/U](\mathbf{Pair});\ [V/U](\mathbf{Enc})}$$

Rule 2. (*Pairing is free in HE*)

$$\frac{\mathbf{Eq};\ \mathbf{Pair} \sqcup \{Z = U_1.U_2,\ Z = V_1.V_2\};\ \mathbf{Enc}}{\mathbf{Eq} \cup \{V_1 = U_1,\ V_2 = U_2\}; \mathbf{Pair} \sqcup \{Z = U_1.U_2\};\ \mathbf{Enc}}$$

Rule 3. (*Split on Pairs*)

$$a)\ \frac{\mathbf{Eq};\ \mathbf{Pair};\ \mathbf{Enc} \sqcup \{Z = enc(X,Y)\};\quad Z = Z_1.Z_2 \in \mathbf{Pair}}{\mathbf{Eq};\ ;\ \mathbf{Pair} \sqcup \{X = X_1.X_2\};\ \mathbf{Enc} \sqcup \{Z_1 = enc(X_1,Y),\ Z_2 = enc(X_2,Y)\}}$$

$$b)\ \frac{\mathbf{Eq};\ \mathbf{Pair};\ \mathbf{Enc} \sqcup \{Z = enc(X,Y)\};\quad X = X_1.X_2 \in \mathbf{Pair}}{\mathbf{Eq};\ \mathbf{Pair} \sqcup \{Z = Z_1.Z_2\};\ \mathbf{Enc} \sqcup \{Z_1 = enc(X_1,Y),\ Z_2 = enc(X_2,Y)\}}$$

Rule 4. (*Occur check*)

$$\frac{\mathbf{Eq};\ \mathbf{Pair};\ \mathbf{Enc};\quad Z \sim Z'\ \ and\ \ Z \succ Z'}{FAIL}$$

Rule 5. (*Equate Some Keys*)

$$\frac{\mathbf{Eq};\ \mathbf{Pair};\ \mathbf{Enc};\quad U,\ V \text{ are keys of } \mathcal{P}}{\mathbf{Eq} \cup \{U = V\};\ \mathbf{Pair};\ \mathbf{Enc}}$$

The X_1, X_2 in rule 3a (resp. Z_1, Z_2 in rule 3b) are fresh variables – as indicated by the notation, the $\sqcup$ signifying *disjoint union* as is standard. Inference Rule $1'$ serves to keep the graph of the current problem irredundant. *The inference rules 1', 4 are both to be applied eagerly*; if rule 4 leads to 'FAIL', then the procedure stops. Inference rules 1 to 4 are 'mandatory' in the sense that they cannot be ignored if applicable; the last rule 5 is 'auxiliary': its role is to guess some additional equalities between the keys of $\mathcal{P}$ that have not already been 'inferred as equal' by the other rules (its role is to ensure the completeness of our method for solving $\mathcal{P}$, as we shall be seeing farther down; cf. e.g. Example 2.(i), Section 4.1). At any stage of the process, the inferences under $1a, 1b, 2$ and 5 are all to be derived 'in block' with the Variable Elimination rule $1'$.

We must show that such an inference procedure terminates on any problem given in standard form. For that purpose, we need to define certain notions.

(1): The relation $\succ$ defined above on the set of variables $\mathcal{X} = \mathcal{X}_{\mathcal{P}}$ of any problem $\mathcal{P}$, for which the above inference system *does not lead to* FAIL, is a well-defined, strict, partial order. For any such $\mathcal{P}$, and for any given $Z \in \mathcal{X}$, the *sp-depth of Z* – denoted as $spd(Z)$ – is defined as *the maximum number of p_1- or p_2- steps* from Z to all possible $X \in \mathcal{X}$, *along the loop-free chains formed of $\sim$- or enc/dec- or p_1/p_2- steps from Z to X.*

(2): Let $\mathcal{P}$ be any such given problem. We introduce a binary, infix operator '$\circ$' representing pairs (but denoted differently, to avoid confusion); and define $\mathcal{T}_p(\mathcal{P}) = \mathcal{T}_p$ as the set of all terms formed over $\mathcal{X}$, the symbol '$\circ$', and the set of all homomorphisms h_T – where T runs over all the key variables of $\mathcal{P}$.

- Any pairing $X = X_1.X_2$ in $\mathcal{P}$, is seen as a rewrite rule: $X \rightarrow X_1 \circ X_2$;
- Any equation $Z = enc(X, T)$ in $\mathcal{P}$ gives rise to two rewrite rules:
  $Z \rightarrow^{h_T} X$, and $X \rightarrow^{\overline{h}_T} Z$.

Rules of the former type will be called pairing rules; those of the latter type will be respectively called h-rules or $\overline{h}$-rules, with key T, and with target X for the first among them, and Z for the second. We define $\mathcal{R}_\mathcal{P}$ to be the rewrite system formed of all such rules. By a *critical configuration* wrt $\mathcal{R}_\mathcal{P}$, we mean any given pair of distinct rewrite rules of $\mathcal{R}_\mathcal{P}$ such that:

- both rules have the same variable $X \in \mathcal{X}_\mathcal{P}$ to their left;
- if one of them is a h-rule (resp. $\overline{h}$-rule), then the other rule must be a pairing rule or a h-rule (resp. pairing rule or a $\overline{h}$-rule);
- if both are h-rules (or $\overline{h}$-rules), they have the same key or the same target.

The common lhs variable of a critical configuration is referred to as its *peak*.

(3): For any such given problem $\mathcal{P}$, and any given critical configuration wrt $\mathcal{R}_\mathcal{P}$ with $X \in \mathcal{X}$ as its peak, let n_X stand for the number of distinct nodes on $G_\mathcal{P}$ to which there is a loop-free, non-empty chain from X *formed only of $\sim$-steps*. The *weight* of the critical configuration is then defined as the (lexicographically) ordered pair of integers $(spd(X), n_X)$.

Lemma 1. *Trimming terminates on problems given in standard form.*

Proof. Given $\mathcal{P}$ in standard form, we only need to consider the inferences other than 4 (which – applied whenever applicable – would yield 'FAIL'). We define the *measure* $m(\mathcal{P})$ of $\mathcal{P}$ as the lexicographic combination of 2 components: $m_1 = m_1(\mathcal{P})$, $m_2 = m_2(\mathcal{P})$, where:

- m_1 is the number of distinct key variables appearing in $\mathcal{P}$;
- m_2 is the *multiset of weights* of all the critical configurations over $\mathcal{R}_\mathcal{P}$.

Consider now any inference on $\mathcal{P}$, by a rule other than 4 (which – applied whenever applicable – would yield 'FAIL'). We then have the following:

- Inference rule $1b$ and 5 will lower m_1.
- Inference rules $1a, 2, 3a, 3b$ – applied in block with $1'$ – will all leave m_1 unchanged, but will lower m_2.

Indeed if some nodes "become equal" under the inferences, and if the number of keys is *not* lower for the new problem derived, then:

- either some of the critical configurations have been eliminated, while the others remain unchanged;
- or some 'current' critical configurations have been replaced by new ones.

In the latter case, for any new critical config with Y as its peak, that replaces an old one with X as peak, we should have: either $spd(Y) < spd(X)$ (Inference rules $3a, 3b$), or else $spd(Y) = spd(X)$ and $n_Y < n_X$ (Inference rules $1a, 2$). $\Box$

The problem $\mathcal{P}$ is said to be *trimmed* iff it is saturated under rules 1, 2 and 3 and 4. It is not necessary to be saturated under rule 5; this signifies that rule 5 may be applied (when it is applicable and we 'intend two key variables to be equal'), or may not applied. Therefore, there are many possible trimmings, depending on which keys variables are 'made equal' under the auxiliary inference rule 5. A trimmed problem $\mathcal{P}$ gets actually divided into two sub-problems which can be treated 'almost' separately, as we shall be seeing below: one containing only the pairings and equalities of $\mathcal{P}$, and the other containing only its *enc* equations; this latter sub-problem will be referred to as the simple kernel, or just *kernel*, of $\mathcal{P}$. *A problem $\mathcal{P}$ is said to be* simple *iff it is its own kernel.*

Example 1. i) The following problem is not in standard form:
$Z = T,\ \ Z = enc(X, Y),\ \ X = dec(T, Y),\ \ X = U.V,\ \ Y = Y_1.Y_2,\ \ \ Y_2 = a$;
we first put it in standard form:
$Z = T,\ \ \ Z = enc(X, Y),\ \ T = enc(X, Y),\ \ X = U.V,\ \ Y = Y_1.Y_2,\ \ Y_2 = a.$
Under redundancy elimination, we first get:
$T = Z,\ \ Z = enc(X, Y),\ \ X = U.V,\ \ Y = Y_1.a,\ \ Y_2 = a$;
which has one critical configuration, namely: $Z \xleftarrow{\overline{h}_Y} X \rightarrow U \circ V$.

Only a splitting inference is applicable (on Z), and the final trimmed equivalent is the following problem:

$$Z = T,\ \ Z = Z_1.Z_2,\ \ X = U.V,\ \ Y = Y_1.a,\ \ Y_2 = a,$$
$$Z_1 = enc(U, Y),\ \ Z_2 = enc(V, Y).$$

ii) The following problem:

$$Z = enc(X, Y),\ \ \ Y = enc(Z, T),\ \ \ T = enc(Z, W),\ \ \ Y = Y_1.Y_2.$$

is in standard form, but not trimmed: we have one critical configuration, namely: $Z \xleftarrow{h_T} Y \rightarrow Y_1 \circ Y_2$, with peak at Y. Now $spd(Y) = 1$, but $n_Y = 3$ (we can go from Y to T, X, Z using only *enc*/*dec* arcs); so m_2 here is $\{(1, 3)\}$, and the measure $m(\mathcal{P})$ of the problem is $(3, \{(1, 3)\})$.

Trimming needs here several splitting steps. We first write $Z = Z_1.Z_2$, and replace the second *enc* equation by the 2 equations: $Y_1 = enc(Z_1, T), Y_2 = enc(Z_2, T)$; we get a problem with two critical configurations, both with peak at Z, $spd(Z) = 1$ and $n_Z = 2$; so the measure is lowered to $(3, \{(1, 2), (1, 2)\})$. Next, we write $X = X_1.X_2$ and replace the first *enc* equation by: $Z_1 = enc(X_1, Y), Z_2 = enc(X_2, Y)$, and get a problem with measure $(3, \{(1, 1)\})$. Finally, we write $T = T_1.T_2$ and replace the last *enc* equation by: $T_1 = enc(Z_1, W), T_2 = enc(Z_2, W)$. We thus get the following trimmed equivalent, with measure $(3, \{(0, 0)\}$:

$$Z_1 = enc(X_1, Y),\ \ \ \ Z_2 = enc(X_2, Y),$$
$$Y_1 = enc(Z_1, T),\ \ \ \ Y_2 = enc(Z_2, T),$$
$$T_1 = enc(Z_1, W),\ \ \ \ T_2 = enc(Z_2, W),$$
$$Y = Y_1.Y_2,\ \ \ Z = Z_1.Z_2,\ \ \ X = X_1.X_2,\ \ \ T = T_1.T_2.$$

Remark 1. (i) The number of equations in a trimmed equivalent of a problem $\mathcal{P}$ given in standard form – derived at the end of the inference procedure, when it does not FAIL – can be exponential wrt the number of initial equations in $\mathcal{P}$; a typical illustrative example is the following:

$$\begin{array}{lll} X_1 = enc(X_2, U1) & X_{11} = enc(X_{12}, U2) & X_{111} = enc(X_{112}, U3) \\ X_1 = X_{11}.X_{12} & X_{11} = X_{111}.X_{112} & X_{111} = X_{1111}.X_{1112} \end{array}$$

A rough upper bound $N(\mathcal{P})$ for the number of equations generated by trimming $\mathcal{P}$ can be given as follows: Let Variant($\mathcal{P}$) be the set of all 'variants' of $\mathcal{P}$ obtained by adding some further equalities between the key variables; let m be the sup of the number of equations in any of these variants, and let d stand for the sup of the sp-depths of the variables in these variants. Then $N(\mathcal{P}) \leq m\, 2^d$.

(ii) In view of the lemma above (and our non-redundancy assumption on the dependency graph), a trimmed problem is essentially a problem $\mathcal{P}$ in standard form – with no critical configurations on pairings alone –, such that:

- There is no node Z on the dependency graph of $\mathcal{P}$ from which there is an *outgoing* '*enc*' or '*dec*' arc, as well as an *outgoing* arc labeled with a p_1 *or* p_2.
- If X, V are any two distinct nodes on the graph of $\mathcal{P}$, then the equality $X = V$ is not an equality in $\mathcal{P}$.

(iii) As a consequence, solving a trimmed problem $\mathcal{P}$ essentially reduces to solving its kernel $\mathcal{P}'$: any variable to the left of a pairing gets its solution by substituting from the solutions for the variables of the kernel.

(iv) Observe that the key variables in any problem $\mathcal{P}$ given in standard form, remain as they are under splitting (as the Y, T, W in Example 1.ii) above); so their number remains unaffected by trimming. $\square$

4.1 Solving a (Trimmed) Simple Problem

For solving a trimmed problem $\mathcal{P}$, we shall make an *assumption* which expresses a necessary condition for $\mathcal{P}$ to admit a *solution in normal form modulo HE*; this assumption is again based on the Perfect Encryption hypothesis:

(**SNF**): For any loop γ on $G_{\mathcal{P}}$ from some node Z to itself, the word formed by the symbols labeling the arcs composing γ must simplify to the empty word, under the following set of rules:

$$(\#) \qquad h_i\overline{h}_i \rightarrow \epsilon, \quad \overline{h}_i h_i \rightarrow \epsilon, \qquad 1 \leq i \leq n.$$

A problem $\mathcal{P}$ in trimmed form will be said to be *admissible* iff it satisfies SNF. For such problems $\mathcal{P}$, it follows in particular from SNF, that:

• there can be no loop containing an arc labeled with a p_1 or p_2, from any node to itself, on the graph $G = G_{\mathcal{P}}$ ("the pairing operator is *free* in HE").

We henceforth assume all our problems $\mathcal{P}$ to be trimmed and admissible; and $\mathcal{P}'$ will stand for the kernel of $\mathcal{P}$. Our objective now is to conceive an algorithm for solving $\mathcal{P}'$. Note that the labels on the arcs of the sub-graph G' (of $G_{\mathcal{P}}$) of dependency for the problem $\mathcal{P}'$, are all in $\mathcal{H}$. Note also that, thanks to the SNF assumption on $\mathcal{P}$, there is a uniquely determined, loop-free, oriented path from any given node on G', to any other node on G'. For solving $\mathcal{P}'$, we shall be using – but only partly – the following idea on every connected component Γ of G':

- choose an 'end-variable' V (in a sense to be formalized) on Γ, and assign some value v to V;

- then, to every other node X on Γ, assign the value derived by '*propagating that value from*' V to X; i.e., assign to X the value $\alpha(v)$ where α is the word over $\mathcal{H}$ that labels the unique oriented path from X to V.

However, such an idea can only be used with some restrictive assumptions. A first assumption we make is that a variable X of $\mathcal{P}'$ may *not* be evaluated by using a word over $\mathcal{H}$ already containing h_X or $\overline{h}_X$; this corresponds to the occur-check assumption in standard unification over the empty theory.

Definition 1 *i) An oriented (loop-free) path γ, on the dependency graph of G', is said to satisfy the condition* Occur-Check-Path – *or is said to* pass the test OCP – *if and only if the following holds:*
(**OCP**): *For any arc $U \rightarrow^h V$ on G' composing γ, with label $h = h_X$ or $\overline{h}_X$,* none of the nodes traversed by γ prior to U, is X or a factor of X for pairing.

(Remark 1.(iv) gives the reason for such a roundabout formulation for OCP: a variable can be a node as well as a key; if it can get split as a node, as a key it remains unchanged.) Unfortunately, one cannot derive a complete procedure for solving simple problems, based only on checking for OCP along all the maximal paths or their inverse paths (this is illustrated by several examples given below). The approach needs to be a little more complex.

Let $\mathcal{P}$ be any simple problem (i.e., with no pairing equations). We know that solving $\mathcal{P}$ amounts to solving the unification problem modulo the convergent system R formed of the following two rules:

$$(R): \quad enc(dec(x,y),y) \rightarrow x, \qquad dec(enc(x,y),y) \rightarrow x.$$

First, we may assume obviously that the graph of $\mathcal{P}$ is connected (apply the same reasoning on every connected component of $G_{\mathcal{P}}$). We also assume explicitly that the nodes of $G_{\mathcal{P}}$, as well the keys of $\mathcal{P}$, are *all distinct* mod R, i.e., 'unequal' modulo R; and that the solutions to $\mathcal{P}$ looked for are in R-normal form, and *discriminating* in the sense that *variables which are distinct in $\mathcal{P}$, are assigned distinct ground terms.*

We then define a relation – denoted as $\succ_k$, and called *key-dependency* – between the variables of such a $\mathcal{P}$, as follows:

• $Y \succ_k X$ iff $Y \neq X$ and the (unique) path from Y to X on the graph $G_{\mathcal{P}}$ contains an arc labeled with $h_{X'}$ or $\overline{h}_{X'}$ where X' is X or contains X as a factor for pairing.

We then get the following criterion referred to as NKDC (standing for *'No-Key-Dependency-Cycle'*), for a simple problem $\mathcal{P}$ to admit a discriminating solution (under the assumption that *its keys are to be unequal* modulo R):

• (**NKDC**): The graph $G = G_{\mathcal{P}}$ does not contain a node X such that $X \succ_k^+ X$ where $\succ_k^+$ is the transitive closure of $\succ_k$.

In intuitive terms: NKDC says that G cannot contain two nodes X, Y such that the path from X to Y fails the OCP test for X, *and* its reversed path fails the OCP test for Y.

NKDC is Necessary for Solvability: In this paragraph, θ will stand for a discriminating substitution on the set $\mathcal{X}$ of variables/constants of $\mathcal{P}$, into the algebra of terms over *enc*, *dec*, $\mathcal{X}$ and the ground constants. For any $X \in \mathcal{X}$, $\tilde{h}_X$ stands for either h_X or its conjugate $\overline{h}_X$; and $C, C', \ldots$, referred to as contexts, stand for words over the $\tilde{h}_X$. All terms are assumed to be in normal form modulo R unless otherwise mentioned.

Lemma 2. *Assume that $Y = \tilde{h}_X(C[p \leftarrow t])$ for some context C, term t, and position $p \in \{1\}^*$. Then for any normalized ground substitution θ we have that $\theta(Y)$ is either a subterm of $\theta(t)$ or $\theta(X)$ is the outermost key in the term $\theta(Y)$.*

Proof. Case i) Where the encryption keys on top of $\theta(t)$ get cancelled by the decryption keys of the context C up to $\theta(X)$: in this case, $\theta(Y)$ must be a subterm of the term $\theta(t)$.

Case ii) Where the encryption by $\theta(X)$ is not cancelled by a decryption key just below, in the term $\theta(Y)$: By assumption θ assigns different terms to different variables; so, in this case $\theta(X)$ will remain the outermost key of $\theta(Y)$. □

Lemma 3. *Assume that $Y = \tilde{h}_{X'}(C[p \leftarrow X])$ for some context C, and position $p \in \{1\}^*$, where $X = X'$ or X is a factor of X' for pairing. Then $\theta(X')$ is the outermost key of $\theta(Y)$, and $|\theta(Y)| > |\theta(X)|$.*

Proof. We apply the previous lemma with $t = X$; we will be in Case ii) of that proof, se we deduce that $\theta(X')$ is the outermost key of $\theta(Y)$. Since both $\theta(X)$, $\theta(Y)$ are in normal form, we also get the assertion on their sizes. □

Lemma 4. *If $Y = C'[p' \leftarrow \tilde{h}_{X'}(C[p \leftarrow X])]$ for some contexts C, C', and positions $p, p' \in \{1\}^*$, where $X = X'$ or X is a factor of X' for pairing, then $\theta(t)$ is a subterm of $\theta(Y)$, where $t = \tilde{h}_{X'}(C[p \leftarrow X])$.*

Proof. $\theta(X')$ is the outermost key of $\theta(t)$ by Lemma 3. And no reduction above $\theta(t)$ is possible, since θ is assumed discriminating. □

Corollary 1. *If the graph $G_{\mathcal{P}}$ does not satisfy the criterion NKDC, then there is no discriminating substitution that is a solution for the simple problem $\mathcal{P}$.*

Example 2. Consider the following problems:

(i) $\mathcal{P}_1$: $Y = enc(Z, X), \quad X = enc(Z, Y)$
(ii) $\mathcal{P}_2$: $Z = enc(X, X), \quad Z = dec(T, T)$
(iii) $\mathcal{P}_3$: $U = enc(X, Z), \quad Z = enc(U, Y), \quad Y = enc(U, X)$

(i) $\mathcal{P}_1$ does not admit any discriminating substitution as solution: indeed we have $Y \succ_k X \succ_k Y$. So, if there is a solution, it must assign the same value to X, Y; so if the keys are to be unequal, then $\mathcal{P}_1$ would be unsolvable; or else, we could have guessed the key equality $X = Y$, and reduce the problem to one single equation $X = enc(Z, X)$, which is solvable as $Z = dec(X, X)$.

(ii) Problem $\mathcal{P}_2$ is unsolvable: First, we have $X \succ_k T \succ_k X$, so $\mathcal{P}_2$ does not admit any discriminating solution; if the keys are to be unequal, one deduces then that there is no solution. On the other hand, if we had guessed $X = T$, the problem to solve would reduce to: $Z = enc(X, X)$, $Z = dec(X, X)$, for which there can be no solution at all modulo the 2-rule system R.

(iii) No discriminating solution is possible for $\mathcal{P}_3$, since $X \succ_k Z \succ_k Y \succ_k X$. And guessing an equality on the keys, such as, e.g., $Y = Z$, would transform the problem into one of the two problems just studied. □

NKDC is Sufficient for Solvability: *We assume henceforth that our simple problems $\mathcal{P}$ satisfy the criterion* NKDC. Under the already made assumption that the keys of $\mathcal{P}$ are to be unequal modulo R, and the graph of $\mathcal{P}$ is connected, we propose a procedure for finding a discriminating solution for $\mathcal{P}$. Let us first illustrate the underlying idea with an example.

Example 3. Consider the following problem:

$$(\mathcal{P}):\ X = enc(U, V),\quad U = enc(V, T),\quad V = enc(Y, U)$$

Its graph is connected, comprising a single maximal (loop-free) path γ between the end-nodes X and Y. Although it does not satisfy the OCP condition in either direction between X and Y, the path γ does satisfy NKDC: indeed, we only have two key-dependencies $X \succ_k V$ and $Y \succ_k U$; there are no cycles, and both U and V are *minimal* for the relation $\succ_k$.

This means, in intuitive terms, that the path from any given node Z towards U, or towards V, passes the OCP test for that node Z; so, either U, or V, can be chosen as a base-node, to solve for Z via propagation. Thus, if we take U to be the base-node, the following substitution is a discriminating solution for $\mathcal{P}$:

$V = \overline{h}_T(U)$, $Y = \overline{h}_U(V) = \overline{h}_U\overline{h}_T(U)$, with U and T arbitrary,
and subsequently solve for X as: $X = h_V(U)$. □

Definition 2 *Let Γ be any connected component on the graph G of a simple problem $\mathcal{P}$, and $V_0 \in \Gamma$ a node that is minimal for the key-dependency relation $\succ_k$. Then V_0 is called a* base-node for $\mathcal{P}$ on the connected component Γ.

Remark 2. A connected component Γ can have more than one base-nodes. But if Y', Y'' are two base-nodes for $\mathcal{P}$ on Γ, then they are 'equivalent' in the following sense:

- the path joining Y' to Y'' passes the OCP test for Y' as well as for Y'';
- and the keys of the arcs on this path are *not nodes outside this path.*

It follows then that any intermediary node on this path between Y' and Y'' is also a base-node for Γ.

Lemma 5. *If $\mathcal{P}$ is simple* (with keys all assumed unequal) *and satisfies NKDC, then $\mathcal{P}$ admits a discriminating solution.*

Proof. On every given connected component Γ of the graph $G = G_{\mathcal{P}}$ of $\mathcal{P}$, choose some base-node V; by definition then, for any given node X on Γ, the unique path from X to V on G must satisfy the condition OCP for X; we solve for X by propagating to X any value v that is assignable to the chosen base-variable V: i.e., we set $X = \alpha_{XV}(v)$, where α_{XV} is the word over $\mathcal{H}$ labeling the path from X to V. □

We are in a position, now, to formulate a non-deterministic decision procedure for solving any HE-unification problem, given in standard form.

4.2 Solving a Problem in Standard Form: The Algorithm $\mathcal{A}$

Given: $\mathcal{P}$ = a HE-unification problem $\mathcal{P}$, given in standard form.
G = the dependency graph for $\mathcal{P}$.

```
1a. Non-deterministically, choose a non-Failing saturation of $\mathcal{P}$ by Trimming.
1b. Replace $\mathcal{P}$ by a trimmed equivalent;
        $\mathcal{P}'$ = the kernel of $\mathcal{P}$; $G'$ = the sub-graph of $G$ for $\mathcal{P}'$.
1c. If $G$ does not satisfy SNF, or if $\mathcal{P}$ contains two equations of the form
    $Z = a$, $Z = b$, where $a, b$ are two different constants, exit with 'Fail'.
2a. Check for the criterion NKDC on every connected component of $G'$;
2b. If NKDC is unsatisfied on some component, exit with 'Fail';
3a. On each connected component $\mathit{\Gamma}$ of $G'$, choose a base-node $V_{\mathit{\Gamma}}$ for $\mathcal{P}'$
        (i.e., a minimal node for the key-dependency relation $\succ_k$).
3b. Build a substitution for the variables on each component $\mathit{\Gamma}$: assign to $V_{\mathit{\Gamma}}$
    some term, and to every other node $X \in \mathit{\Gamma}$ the value derived via propagation
    from $V_{\mathit{\Gamma}}$ to $X$. Let $\sigma'$ be the substitution, solution for $\mathcal{P}'$, thus obtained.
 4. Propagate the values deduced from $\sigma'$ to the variables (of the equalities and
    pairings) of $G$, that are not in $G'$;
      - if inconsistency, exit with 'Fail';
      - else return $\sigma$ = substitution thus obtained, as solution to $\mathcal{P}$.
```

The soundness of the algorithm $\mathcal{A}$ follows from the fact that each of its steps is syntactically coherent with $\mathcal{P}$. Completeness means that if there is a solution, then we will non-derminstically find one; completeness results from the fact that NKDC is a condition necessary for any simple problem to admit a discriminating solution, the distinct key variables of $\mathcal{P}$ being all assumed unequal modulo HE.

A Complexity Estimate: The algorithm $\mathcal{A}$ is of cost NP with respect to the number of equations of the simple kernel $\mathcal{P}'$ of the problem $\mathcal{P}$; this is so, since checking for the criterion NKDC on any component can be done in time NP, wrt the number nodes on G'. And in view of Remark 1, the overall complexity turns out to be NEXPTIME wrt the number of initial equations for any non-simple problem (given in standard form).

However, the complexity estimate is much sharper for *simple* problems:

Proposition 2 *Solving general simple unification problems is NP-complete.*

Proof. We need only to prove the NP lower bound, and that is done by reduction from the following so-called *Monotone 1-in-3 SAT* problem:

- Given a propositional formula without negation, in CNF over 3 variables, check for its satisfiability under the assumption that *exactly* one literal in each clause evaluates to true.

This problem is known to be NP-complete, cf. [10]. Now consider the simple problem (without pairings) derived from the following unification problem over the 2-rule system R, involving 3 variables x_1, x_2, x_3:

$$dec(enc(dec(enc(dec(enc(a,b),x_1),b),x2),b),x3) =^? dec(enc(a,b),c)$$

Obviously, solving this problem amounts to saying that exactly one of the three variables x_1, x_2, x_3 is assigned the term c. □

4.3 Illustrative Examples

Note that the substitution that the algorithm $\mathcal{A}$ returns as a solution for a problem $\mathcal{P}$, is built "in a lazy style" in its steps $3a$ through 4: the variables are left uninstantiated, in general; they get instantiated only if/when needed (cf. e.g., Example 4 below).

Example 4. Consider the following problem:

$(\mathcal{P}')$: $\quad Z = enc(X, Y), \quad Y = enc(Z, T), \quad T = enc(Z, W).$

The problem is simple, and its dependency graph is connected:

$$\begin{tikzcd}
& Z \arrow[rr, shift right, "h_Y"'] \arrow[dl, shift right, "\overline{h}_T"'] \arrow[dr, shift left, "\overline{h}_W"] & & X \arrow[ll, shift right, "\overline{h}_Y"'] \\
Y \arrow[ur, shift right, "h_T"'] & & T \arrow[ul, shift left, "h_W"] &
\end{tikzcd}$$

The graph does satisfy NKDC: the key-dependency relations are $X \succ_k Y \succ_k T$; so, T is the only base-node here. We solve for Z and Y, along the path from Y to T (that satisfies OCP, by definition): namely $Y \rightarrow^{h_T} Z \rightarrow^{\overline{h}_W} T$; choosing arbitrarily T, W we get $Z = \overline{h}_W(T), Y = h_T\overline{h}_W(T)$ as solutions for Z, Y; and for the variable X, connected to this path at Z, we deduce get $X = \overline{h}_Y(Z) = \overline{h}_Y\overline{h}_W(T)$. (Note: the base-node T has not been assigned any specific term here.)

Suppose now, the problem $(\mathcal{P}')$ is the kernel of a non-simple problem, e.g.:

$(\mathcal{P})$: $\quad Z = enc(X, Y), \quad Y = enc(Z, T), \quad T = enc(Z, W). \quad X = a$

Then, for the above solution for its simple kernel to be valid, we need to check if $a = \overline{h}_Y\overline{h}_W(T)$ holds; this can be done by instantiating T, now, as $h_W h_Y(a)$. $\square$

Example 5. The following simple problem is unsolvable :

$X = enc(Y, T), \quad Y = enc(Z, X), \quad Z = enc(W, V), \quad W = enc(V, S)$

Indeed, its (connected) graph fails to satisfy the NKDC criterion, so no discriminating solution can exist; on the other hand, it is easy to check that, no matter which keys and/or nodes are 'made equal', the NKDC criterion will continue to fail. $\square$

References

1. S. Anantharaman, P. Narendran, M. Rusinowitch. "Intruders with Caps". In *Proc. of the Int. Conference RTA'07*, LNCS 4533, pp.20–35, Springer-Verlag, June 2007.
2. S. Anantharaman, H. Lin, C. Lynch, P. Narendran, M. Rusinowitch. "Equational and Cap Unification for Intrusion Analysis". LIFO-Research Report, `www.univ-orleans.fr/lifo/prodsci/rapports/RR/RR2008/RR-2008-03.pdf`
3. M. Baudet. "Deciding security of protocols against off-line guessing attacks". In *Proc. of ACM Conf. on Computer and Communications Security*, 2005, pp. 16-25.
4. Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani. "An NP Decision Procedure for Protocol Insecurity with XOR". In *Proc. of the Logic In Computer Science Conference, LICS'03*, pages 261–270, 2003.

5. H. Comon-Lundh, V. Shmatikov. Intruder Deductions, Constraint Solving and Insecurity Decision in Presence of Exclusive or. In *Proc. of the Logic In Computer Science Conference, LICS'03*, pages 271–280, 2003.
6. V. Cortier, S. Delaune, P. Lafourcade. "A Survey of Algebraic Properties Used in Cryptographic Protocols". In *Journal of Computer Security* 14(1): 1–43, 2006.
7. S. Delaune, F. Jacquemard. A decision procedure for the verification of security protocols with explicit destructors. In *Proc. of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 278–287, Washington, D.C., USA, October 2004. ACM Press.
8. C. Meadows, P. Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Workshop on Issues in the Theory of Security (in conjunction with POPL'02), Portland, Oregon, USA, January 14-15*, 2002.
9. P. Narendran, F. Pfenning, R. Statman. On the unification problem for cartesian closed categories. In *Proc. of the Logic in Computer Science Conference LICS'93*, pages 57–63, 1993.
10. T. J. Schaefer. The complexity of satisfiability problems. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.

Matching linear and non-linear trace patterns with regular policies

Franz Baader[1]⋆, Andreas Bauer[2], and Alwen Tiu[2]

[1] Theoretical Computer Science, TU Dresden, Germany
baader@inf.tu-dresden.de
[2] Computer Sciences Laboratory, ANU, Canberra, Australia
{baueran,tiu}@rsise.anu.edu.au

Abstract. In this paper, we consider policies that are described by regular languages. Such regular policies L are assumed to describe situations that are problematic, and thus should be avoided. Given a trace pattern u, i.e., a sequence of action symbols and variables, were the variables stand for unknown (i.e., not observed) sequences of actions, we ask whether u potentially violates a given policy L, i.e., whether the variables in u can be replaced by sequences of actions such that the resulting trace belongs to L. We determine the complexity of this violation problem, depending on whether trace patterns are linear or not, and on whether the policy is assumed to be fixed or not.

1 Introduction

In an online transaction system, policies that define which sequences of actions (called traces in the following) are viewed as being problematic can be specified using regular languages over the alphabet of action symbols. A trace w violates the policy L if $w \in L$. Sometimes, it is not possible to observe all the actions that take place. For example, assume that an online auctioning firm such as ebay.com is trying to monitor the behaviour of its buyers and sellers w.r.t. certain security policies. Then some of the actions (like making a bid or giving a positive evaluation of the seller/buyer) can be observed by ebay, whereas other actions (like actually sending the goods or paying for received goods) are not observable. We model this with the help of trace patterns, i.e., sequences of actions and variables, where the variables stand for unknown sequences of actions. For example, $abXaY$ is a trace pattern where a, b are actions (more precisely, symbols for actions) and X, Y are variables. This trace pattern says: all we know about the actual trace is that it starts with ab, is followed by some trace w, which is followed by a, which is in turn followed by some trace u. Given such a trace pattern, all traces that can be obtained from it by replacing its variables with traces (i.e., finite sequences of actions) are possibly the actual traces. In our example, these are all the traces of the form $abwau$ where w and u are arbitrary traces. The policy L is potentially violated if one of the traces obtained by such

⋆ Supported by NICTA, Canberra Research Lab.

a substitution of the variables by traces belongs to L. In our example, $abXaY$ potentially violates $L = (ab)^*$ since replacing X by ab and Y by b yields the trace $ababab \in L$.

The trace pattern in our examples is linear since every variable occurs at most once in it. We can also consider non-linear trace patterns such as $abXaX$, where different occurrences of the same variable must be replaced by the same trace. The underlying idea is that, though we do not know which actions took place in the unobserved part of the trace, we know (from some source) that the same sequence of actions took place. It is easy to see that the policy $L = (ab)^*$ is not potentially violated by the non-linear trace pattern $abXaX$ since it is not possible to replace X by a trace w such that $abwaw \in L$.

In this paper, we will show that the complexity of the problem of deciding whether a given trace pattern potentially violates a regular policy depends on whether the trace pattern is linear or not. For linear trace patterns, the problem is decidable in polynomial time whereas for non-linear trace patterns the problem is PSpace-complete. If we assume that the size of the security policy (more precisely, of a non-deterministic finite automaton or regular expression representing it) is constant, then the problem can be solved in linear time for linear trace patterns and is NP-complete for non-linear trace patterns.

2 Preliminaries

In the following, we consider finite alphabets Σ, whose elements are called *action symbols.* A *trace* is a (finite) word over Σ, i.e., an element of Σ^*. A *trace pattern* is an element of $(\Sigma \cup \mathcal{V})^*$, i.e., a finite word over the extended alphabet $\Sigma \cup \mathcal{V}$, where $\mathcal{V}$ is a finite set of *trace variables.* The trace pattern u is called *linear* if every trace variable occurs at most once in u. A *substitution* is a mapping $\sigma : \mathcal{V} \to \Sigma^*$. This mapping is extended to a mapping $\widehat{\sigma} : (\Sigma \cup \mathcal{V})^* \to \Sigma^*$ in the obvious way, by defining $\widehat{\sigma}(\varepsilon) = \varepsilon$ for the empty word ε, $\widehat{\sigma}(a) = a$ for every action symbol $a \in \Sigma$, $\widehat{\sigma}(X) = \sigma(X)$ for every trace variables $X \in \mathcal{V}$, and $\widehat{\sigma}(uv) = \widehat{\sigma}(u)\widehat{\sigma}(v)$ for every pair of non-empty trace patterns u, v.

A *policy* is a regular language over Σ. We assume that such a policy is given either by a regular expression or by a (non-deterministic) finite automaton. For our complexity results, it is irrelevant which of these representations we actually use.

Definition 1. *Given a trace pattern u and a policy L, we say that u* potentially violates *L (written $u \lesssim L$) if there is a substitution σ such that $\widehat{\sigma}(u) \in L$. The* violation problem *is the following decision problem:*

Given: *A policy L and a trace pattern u.*
Question: *Does $u \lesssim L$ hold or not?*

If the trace pattern u in this decision problem is restricted to being linear, then we call this the linear violation problem*.*

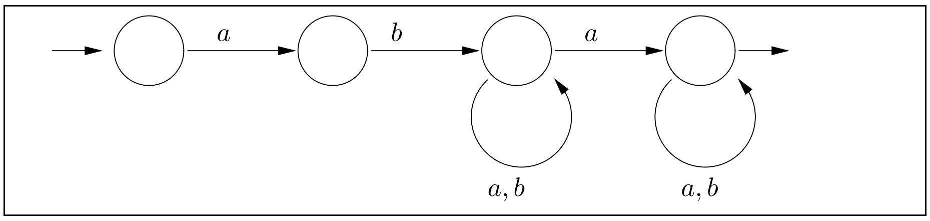

Fig. 1. A non-deterministic finite automaton accepting $ab\Sigma^*a\Sigma^*$.

We assume that the reader is familiar with regular expressions and finite automata. Given a (non-deterministic) finite automaton $\mathcal{A}$, states p, q in $\mathcal{A}$, and a word w, we write $p \to^w_{\mathcal{A}} q$ to say that there is a path in $\mathcal{A}$ from p to q with label w. The set of labels of all paths from p to q is denoted by $L_{p,q}$.

The following problem for regular languages turns out to be closely connected to the violation problem. The *intersection emptiness problem* for regular languages is the following decision problem:

Given: Regular languages $L_1, \ldots, L_n$.
Question: Does $L_1 \cap \ldots \cap L_n = \emptyset$ hold or not?

It is well-known that this problem is PSpace-complete [3, 1], independent of whether the regular languages are given as regular expressions, non-deterministic finite automata, or deterministic finite automata.

In the following, we assume that regular languages are given by a regular expression, a non-deterministic finite automaton, or a deterministic finite automaton.

3 The linear violation problem

Assume that u is a linear trace pattern and L is a regular language. Let the trace pattern u be of the form $u = u_0X_1u_1 \ldots X_mu_m$ where $u_i \in \Sigma^*$ $(i = 0, \ldots, m)$ and $X_1, \ldots, X_m$ are distinct variables. Obviously, we have

$$u \lesssim L \ \text{ iff } \ u_0\Sigma^*u_1 \ldots \Sigma^*u_m \cap L \neq \emptyset.$$

If n is the length of $u_0u_1 \ldots u_m$, then we can build a non-deterministic finite automaton $\mathcal{A}$ accepting the language $u_0\Sigma^*u_1 \ldots \Sigma^*u_m$ that has $n+1$ states. For example, given the linear trace pattern $abXaY$ from the introduction, we consider the language $ab\Sigma^*a\Sigma^*$, where $\Sigma = \{a, b\}$. Fig. 1 shows a non-deterministic finite automaton with 4 states accepting this language.[1] In addition, there is a non-deterministic finite automaton $\mathcal{B}$ accepting L such that the number of states ℓ of $\mathcal{B}$ is polynomial in the size of the original representation for L.[2] By

[1] Note that arrows without a source pointing into a state denote initial states, and arrows without a target coming out of a state denote final states.

[2] In fact, it is well-known that, given a regular expression r for L, one can construct a non-deterministic finite automaton accepting L in time polynomial in the size of r.

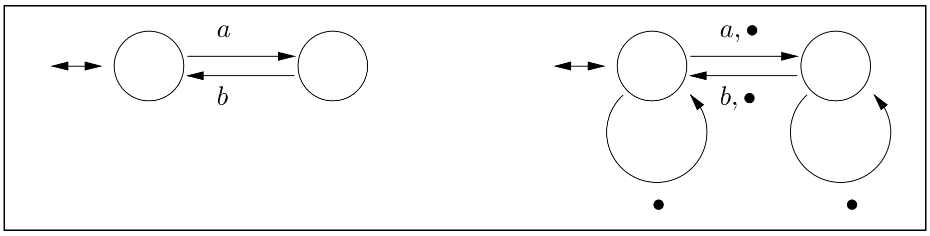

Fig. 2. A non-deterministic finite automaton $\mathcal{A}$ accepting $(ab)^*$ (left) and the corresponding automaton $\widehat{\mathcal{A}}$ (right).

constructing the product automaton of $\mathcal{A}$ and $\mathcal{B}$, we obtain a non-deterministic finite automaton accepting $u_0\Sigma^*u_1 \ldots \Sigma^*u_m \cap L$ with $(n+1)\cdot\ell$ states. Thus, emptiness of this language can be tested in time polynomial in $(n+1)\cdot\ell$, and thus in time polynomial in the size of the input u, L of our linear violation problem.

Theorem 1. *The linear violation problem can be solved in polynomial time.*

In the following, we describe an alternative way of showing this polynomiality result, which will turn out to be more convenient for proving that the problem is linear in case the size of the policy L is assumed to be constant.

Let $\bullet$ be an action symbol not contained in Σ. Given a linear trace pattern u, let $\widehat{u}$ denote the trace over the alphabet $\Sigma^\bullet := \Sigma \cup \{\bullet\}$ obtained from u by replacing every variable in u by $\bullet$. Now, assume that $\mathcal{A}$ is a non-deterministic finite automaton accepting the policy L. We transform $\mathcal{A}$ into a non-deterministic finite automaton $\widehat{\mathcal{A}}$ by adding to the transitions of $\mathcal{A}$ all transitions $(p, \bullet, q)$ such that there is $u \in \Sigma^*$ with $p \to^u_{\mathcal{A}} q$. Fig. 2 illustrates this construction for the policy $L = (ab)^*$.

The following is an easy consequence of the way $\widehat{u}$ and $\widehat{\mathcal{A}}$ have been constructed.

Lemma 1. *Let u be a linear trace pattern, and L a policy that is accepted by the non-deterministic finite automaton $\mathcal{A}$. Then,*

$$u \lesssim L \quad \textit{iff} \quad \widehat{u} \textit{ is accepted by } \widehat{\mathcal{A}}.$$

For example, we have already seen in the introduction that the linear trace pattern $u = abXaY$ violates the policy $L = (ab)^*$. The trace over $\Sigma^\bullet$ corresponding to u is $\widehat{u} = ab\bullet a\bullet$. Obviously, $\widehat{u}$ is accepted by the non-deterministic finite automaton $\widehat{\mathcal{A}}$ in Fig. 2.

The above lemma reduces the violation problem to the word problem for the automaton $\widehat{\mathcal{A}}$. It is well-know that, given a non-deterministic finite automaton $\mathcal{B}$ of size m (where the size of $\mathcal{B}$ is the sum of the number of states and the number of transitions of $\mathcal{B}$) and a word w of length n, the question whether w is accepted by $\mathcal{B}$ can be decided in $O(n \cdot m)$. This yields an alternative proof of Theorem 1. In fact, the size of $\widehat{\mathcal{A}}$ is polynomial in the size of $\mathcal{A}$ (and it can be computed in polynomial time), and the length of $\widehat{u}$ is the same as the length of u. In addition, if $\mathcal{A}$ is assumed to be constant, then the size of $\widehat{\mathcal{A}}$ is also constant (and it can be computed in constant time).

Theorem 2. *Assume that the policy is fixed. Then, the linear violation problem can be solved in time linear in the length of the input trace pattern.*

4 The general violation problem

Allowing also the use of non-linear patterns increases the complexity of the violation problem.

Theorem 3. *The violation problem is PSpace-complete.*

Proof. PSpace-hardness can be shown by a reduction of the intersection emptiness problem for regular languages. Given regular languages $L_1, \ldots, L_n$, we construct the trace pattern $u_n := \#X\#X \ldots \#X\#$ of length $2n+1$ and the policy $L(L_1, \ldots, L_n) := \#L_1\#L_2 \ldots \#L_n\#$. Here X is a variable and $\#$ is a new action symbol not occurring in any of the words belonging to one of the languages $L_1, \ldots, L_n$. Obviously, both u_n and (a representation of) $L(L_1, \ldots, L_n)$ can be constructed in time polynomial in the size of (the representation of) $L_1, \ldots, L_n$. To be more precise regarding the representation issue, if we want to show PSpace-hardness for the case where the policy is given by a regular expression (a non-deterministic finite automaton, a deterministic finite automaton), then we assume that the regular languages $L_1, \ldots, L_n$ are given by the same kind of representation. It is easy to see that the following equivalence holds:

$$L_1 \cap \ldots \cap L_n \neq \emptyset \;\text{ iff }\; u_n \lesssim L(L_1, \ldots, L_n).$$

Thus, we have shown that the intersection emptiness problem for regular languages can be reduced in polynomial time to the violation problem. Since the intersection emptiness problem is PSpace-complete (independent of whether the regular languages are given as regular expressions, non-deterministic finite automata, or deterministic finite automata), this shows that the violation problem is PSpace-hard (also independent of whether the policy is given as a regular expression, a non-deterministic finite automaton, or a deterministic finite automaton).

To show *membership* of the violation problem *in PSpace*, consider the violation problem for the trace pattern u and the policy L. Let n be the length of u and $\mathcal{A}$ a non-deterministic finite automaton accepting L. For $i \in \{1, \ldots, n\}$, we denote the symbol in $\Sigma \cup \mathcal{V}$ occurring at position i in u with u_i, and for every variable X occurring in u, we denote the set of positions in u at which X occurs with P_X, i.e., $P_X = \{i \mid 1 \leq i \leq n \wedge u_i = X\}$.

It is easy to see that $u \lesssim L$ iff there is a sequence $q_0, \ldots, q_n$ of states of $\mathcal{A}$ such that the following conditions are satisfied:

1. q_0 is an initial state and q_n is a final state;

2. for every $i \in \{1, \ldots, n\}$, if $u_i \in \Sigma$, then $q_{i-1} \rightarrow_{\mathcal{A}}^{u_i} q_i$;
3. for every variable X occurring in u, we have
$$\bigcap_{i \in P_X} L_{q_{i-1},q_i} \neq \emptyset.^3$$

In fact, if $u \lesssim L$, then there is a substitution σ with $\widehat{\sigma}(u) \in L$. Thus, $\widehat{\sigma}(u)$ is accepted by $\mathcal{A}$, which yields a sequence $q_0, \ldots, q_n$ of states of $\mathcal{A}$ such that q_0 is an initial, $q_{i-1} \rightarrow_{\mathcal{A}}^{\widehat{\sigma}(u_i)} q_i$, and q_n is a final state. In particular, this shows that the sequence satisfies Condition 1 from above. If $u_i \in \Sigma$, then $\widehat{\sigma}(u_i) = u_i$, which shows that Condition 2 from above is also satisfied. Finally, if $u_i = X \in \mathcal{V}$, then $\widehat{\sigma}(u_i) = \sigma(X)$, and thus $q_{i-1} \rightarrow_{\mathcal{A}}^{\widehat{\sigma}(u_i)} q_i$ implies that $\sigma(X) \in L_{q_{i-1},q_i}$. Since this holds for all $i \in P_X$, this shows that $\sigma(X) \in \bigcap_{i \in P_X} L_{q_{i-1},q_i}$. Consequently, Condition 3 is satisfied as well.

Conversely, assume that $q_0, \ldots, q_n$ is a sequence of states of $\mathcal{A}$ satisfying the Conditions 1–3 from above. By Condition 3, for every variable X occurring in u, there is a trace $s_X \in \bigcap_{i \in P_X} L_{q_{i-1},q_i}$. If we define the substitution σ such that $\sigma(X) = s_X$, then it is easy to see that $\widehat{\sigma}(u)$ is accepted by $\mathcal{A}$. Thus, we have $u \lesssim L$.

Based on this characterisation of "$u \lesssim L$" we can obtain a PSpace decision procedure for the violation problem as follows. This procedure is non-deterministic, which is not a problem since NPSpace = PSpace by Savitch's theorem [5]. It guesses a sequence $q_0, \ldots, q_n$ of states of $\mathcal{A}$, and then checks whether this sequence satisfies the Conditions 1–3 from above. Obviously, the first two conditions can be checked in polynomial time, and the third condition can be checked within PSpace since the intersection emptiness problem for regular languages is PSpace-complete. □

Alternatively, we could have shown membership in PSpace by reducing it to *solvability of word equations with regular constraints* [6]. In the terminology of this paper, this problem can be defined as follows:

Given: Trace patterns u, v and for every variable X occurring in u or v a regular language L_X.
Question: Is there a substitution σ such that $\widehat{\sigma}(u) = \widehat{\sigma}(v)$ and $\sigma(X) \in L_X$ for all variables X occurring in u or v?

This problem is known to be PSpace-complete [4]. The violation problem can be reduced to it (in polynomial time) as follows. Let u be a trace pattern and L a regular language. We take a new variable X (i.e., one not occurring in u), and build the the following word equation with regular constraints: the trace patterns to be unified are u and X, and the regular constraints are given as $L_X = L$ and $L_Y = \Sigma^*$ for all other variables Y. It is easy to see that $u \lesssim L$ iff there is a substitution σ such that $\widehat{\sigma}(u) = \widehat{\sigma}(X)$ and $\sigma(X) \in L$, i.e., if the constructed word equation with regular constraints has a solution.

[3] Recall that, for a given non-deterministic finite automaton $\mathcal{A}$ and states p, q in $\mathcal{A}$, we denote the set of words labeling paths from p to q in $\mathcal{A}$ by $L_{p,q}$.

It should be noted, however, the algorithm for testing solvability of word equations with regular constraints described in [4] is rather complicated and it is not clear how it could be transformed into a "practical" algorithm.

Let us now consider the complexity of the violation problem for the case where the policy is assumed to be fixed. In this case, the NPSpace algorithm described in the proof of Theorem 3 actually becomes an NP algorithm. In fact, guessing the sequence of states $q_0, \ldots, q_n$ can be realized using polynomially many binary choices (i.e., with an NP algorithm), testing Conditions 1 and 2 is clearly polynomial, and testing Condition 3 becomes polynomial since the size of $\mathcal{A}$, and thus of non-deterministic finite automata accepting the languages L_{q_{i-1},q_i}, is constant.

Theorem 4. *If the policy is assumed to be fixed, then the violation problem is in NP.*

The matching NP-hardness result of course depends on the fixed policy. For example, if $L = \Sigma^*$, then we have $u \lesssim L$ for all trace patterns u, and thus the violation problem for this fixed policy can be solved in constant time. However, we can show that there are policies for which the problem is NP-hard. Given a fixed policy L, the *violation problem for* L is the following decision problem

Given: A trace pattern u.
Question: Does $u \lesssim L$ hold or not?

Theorem 5. *There exists a fixed policy such that the violation problem for this policy is NP-hard.*

Proof. To show *NP-hardness*, we use a reduction from the well-known NP-complete problem 3SAT [1]. Let $C = c_1 \wedge \ldots \wedge c_m$ be an instance of 3SAT, and $\mathcal{P} = \{p_1, \ldots, p_n\}$ the set of propositional variables occurring in C. Every 3-clause c_i in C is of the form $c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$, where the $l_{i,j}$ are literals, i.e., propositional variables or negated propositional variables. In the corresponding violation problem, we use the elements of $\mathcal{V} := \{P_i \mid p_i \in \mathcal{P}\}$ as trace variables, and as alphabet we take $\Sigma := \{\#, \neg, \vee, \wedge, \top, \bot\}$. The positive literal p_i is encoded as the trace pattern $\#P_i\#$ and the negative literal $\neg p_i$ as $\neg\#P_i\#$. For a given literal l, we denote its encoding as a trace pattern by $\iota(l)$. 3-Clauses are encoded as "disjunctions" of the encodings of their literals, i.e., $c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ is encoded as $\iota(c_i) = \iota(l_{i,1})\vee\iota(l_{i,2})\vee\iota(l_{i,3})$, and 3SAT-problems are encoded as "conjunctions" of their 3-clauses, i.e., if $C = c_1 \wedge \ldots \wedge c_m$, then $\iota(C) = \iota(c_1)\wedge\ldots\wedge\iota(c_m)$.

Our fixed policy describes all situations that can make a 3-clause true. To be more precise, consider $\iota(c) = \iota(l_1)\vee\iota(l_2)\vee\iota(l_3)$ for a 3-clause $c = l_1 \vee l_2 \vee l_3$. If we replace the trace variables in c by either $\top$ or $\bot$, then we get a trace of the form $w_1\vee w_2\vee w_3$ where each w_i belongs to the set

$$K := \{\#\top\#,\ \#\bot\#,\ \neg\#\top\#,\ \neg\#\bot\#\}.$$

Intuitively, replacing the trace variable P_i by $\top$ ($\bot$) corresponds to replacing the propositional variable p_i by true (false). Thus, a substitution σ that replaces trace variables by $\top$ or $\bot$ corresponds to a propositional valuation v_σ. The valuation v_σ makes the 3-clause c true iff $\widehat{\sigma}(\iota(c)) = w_1{\vee}w_2{\vee}w_3$ is such that there is an $i, 1 \leq i \leq 3$, with $w_i \in \{\#\top\#, \neg\#\bot\#\}$. For this reason, we define

$$T := \{w_1{\vee}w_2{\vee}w_3 \mid \{w_1, w_2, w_3\} \subseteq K \text{ and there is an } i, 1 \leq i \leq 3, \text{ with } w_i \in \{\#\top\#, \neg\#\bot\#\}\}.$$

To make a conjunction of 3-clauses true, we must make every conjunct true. Consequently, we define our fixed policy L as

$$L_{3SAT} := (T{\wedge})^*T.$$

Since T is a finite language, L_{3SAT} is obviously a regular language. NP-hardness of the violation problem for L_{3SAT} is an immediate consequence of the following claim.

Claim For a given 3SAT problem C the following are equivalent:

1. C is satisfiable.
2. $\iota(C) \lesssim L_{3SAT}$.

To show "1 $\rightarrow$ 2," assume that the valuation v satisfies C. Consider the corresponding substitution σ_v that replaces P_i by $\top$ ($\bot$) if $v(p_i)$ is true (false). Then it is easy to see that $\widehat{\sigma}_v(\iota(C)) \in L_{3SAT}$.

Conversely, to show "2 $\rightarrow$ 1," assume that σ is a substitution such that $\widehat{\sigma}(\iota(C)) \in L_{3SAT}$. It is easy to see that the definitions of $\iota(C)$ and L_{3SAT} then ensure that σ replaces trace variables by $\top$ or $\bot$, and that the valuation v_σ corresponding to σ satisfies C. □

5 Conclusion and future work

In this paper, we have assumed that a regular policy L describes situations that are problematic, and thus should be avoided. This motivated our definition of the violation problem, which asks whether a given trace pattern u potentially violates a given policy L, i.e., whether there is a substitution σ with $\widehat{\sigma}(u) \in L$. We have seen that the complexity of the violation problem depends on whether trace patterns are linear or not, and on whether the policy is assumed to be fixed or not.

Alternatively, one could also assume that a regular policy L describes all the admissible situations. In this case, we want to know whether u always adheres to the policy L, i.e., whether $\sigma(u) \in L$ holds for all substitutions σ. Let us write $u \models L$ to denote that this is the case. Obviously, we have $u \models L$ iff not $u \lesssim \Sigma^* \backslash L$, which shows that the two problems can be reduced to each other. However, this reduction is not polynomial. In fact, there cannot be a polynomial time reduction between the two problems since the adherence problem is intractable even for linear trace pattern, for which the violation problem is tractable. To see this, consider the (linear) trace pattern X and an arbitrary regular language L over the alphabet Σ. Obviously, we have $X \models L$ iff $L = \Sigma^*$. The problem of deciding whether a regular language (given by a regular expression or a non-deterministic finite automaton) is the universal language Σ^* or not is PSpace-complete [1]. Consequently, the adherence problem is PSpace-hard even for linear trace patterns. However, the exact complexity of the problem (for linear and non-linear trace patterns) is still open.

References

1. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman & Co., New York, USA, 1979.
2. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
3. Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 254–266. IEEE Computer Society, 1977.
4. Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. In *40th Annual Symposium on Foundations of Computer Science (FOCS'99)*, pages 495–500. IEEE Computer Society, 1999.
5. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
6. Klaus U. Schulz. Makanin's algorithm for word equations - two improvements and a generalization. In *First International Workshop on Word Equations and Related Topics (IWWERT'90)*, volume 572 of *Lecture Notes in Computer Science*, pages 85–150, 1990.

Unification in the Description Logic $\mathcal{EL}$ is of Type Zero

Franz Baader and Barbara Morawska

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

Abstract. The Description Logic $\mathcal{EL}$ has recently drawn considerable attention since, on the one hand, important inference problems such as the subsumption problem are polynomial. On the other hand, $\mathcal{EL}$ is used to define large biomedical ontologies. Unification in Description Logics has been proposed as a novel inference service that can, for example, be used to detect redundancies in DL-based ontologies. We show that, w.r.t. the unification type, $\mathcal{EL}$ is less well-behaved than for the standard inference problem subsumption: it is of type zero, which in particular implies that there are unification problems that have no finite complete set of unifiers.

1 Introduction

Description logics (DLs) [6] are a successful family of logic-based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. They are employed in various application domains, such as natural language processing, configuration, databases, and biomedical ontologies, but their most notable success so far is the adoption of the DL-based language OWL [15] as standard ontology language for the semantic web.

Using a DL, the important notions of the domain can be described by *concept terms*, i.e., expressions that are built from concept names (unary predicates) and role names (binary predicates) using concept constructors. The expressivity of a particular DL is determined by which concept constructors are available in it. From a semantic point of view, concept names and concept terms represent sets of individuals, whereas roles represent binary relations between individuals. For example, using the concept name Woman, and the role name child, the concept of all *women having a daughter* can be represented by the concept term

$$\mathsf{Woman} \sqcap \exists \mathsf{child}.\mathsf{Woman},$$

and the concept of all *women having only daughters* by

$$\mathsf{Woman} \sqcap \forall \mathsf{child}.\mathsf{Woman}.$$

Knowledge representation systems based on Description Logics provide their users with various inference services that allow them to deduce implicit knowledge from the explicitly represented knowledge. For instance, the subsumption algorithm allows to determine subconcept-superconcept relationships. For example, the concept term Woman subsumes the concept term $\mathsf{Woman} \sqcap \exists \mathsf{child}.\mathsf{Woman}$ since all instances of the second term are also instances of the first term, i.e., the second term is always interpreted as a subset of the first term. With the help of the subsumption algorithm, a newly introduced concept term can automatically be placed at the correct position in the hierarchy of the already existing concept terms.

Two concept terms are *equivalent* if they subsume each other, i.e., if they always represent the same set of individuals. For example, the terms $\mathsf{Woman} \sqcap \forall \mathsf{child}.\mathsf{Woman}$ and $(\forall \mathsf{child}.\mathsf{Woman}) \sqcap \mathsf{Woman}$ are equivalent since $\sqcap$ is interpreted as set intersection, which is obviously commutative. The equivalence test can, for example, be used to find out whether a concept term representing a particular notion has already been introduced, thus avoiding multiple introduction of the same concept into the concept hierarchy. This inference capability is very important if the knowledge base containing the concept terms is very large, evolves during a long time period, and is extended and maintained by several knowledge engineers. However, testing for equivalence of concepts is not always sufficient to find out whether, for a given concept term, there already exists another concept term in the knowledge base describing the same notion. For example, assume that one knowledge engineer has defined the concept of all *women having a daughter* by the concept term

$$\mathsf{Woman} \sqcap \exists \mathsf{child}.\mathsf{Woman}.$$

A second knowledge engineer might represent this notion in a somewhat more fine-grained way, e.g., by using the term $\mathsf{Female} \sqcap \mathsf{Human}$ in place of Woman. The concept terms $\mathsf{Woman} \sqcap \exists \mathsf{child}.\mathsf{Woman}$ and

$$\mathsf{Female} \sqcap \mathsf{Human} \sqcap \exists \mathsf{child}.(\mathsf{Female} \sqcap \mathsf{Human})$$

are not equivalent, but they are meant to represent the same concept. The two terms can obviously be made equivalent by substituting the concept name Woman in the first term by the concept term $\mathsf{Female} \sqcap \mathsf{Human}$. This leads us to *unification of concept terms*, i.e., the question whether two concept terms can be made equivalent by applying an appropriate substitution, where a substitution replaces (some of the) concept names by concept terms. Of course, it is not necessarily the case that unifiable concept terms are meant to represent the same notion. A unifiability test can, however, suggest to the knowledge engineer possible candidate terms.

Unification in DLs was first considered in [10, 11] for a DL called $\mathcal{FL}_0$, which has the concept constructors *conjunction* ($\sqcap$), *value restriction* ($\forall r.C$), and the *top concept* ($\top$). It was shown that unification in $\mathcal{FL}_0$ is decidable and ExpTime-complete, i.e., given an $\mathcal{FL}_0$-unification problem, we can effectively decide whether it has a solution or not, but in the worst-case, any such decision procedure needs exponential time. This result was extended in [8] to a more expressive DL, which additional has the role constructor *transitive closure*. Interestingly, the *unification type* of $\mathcal{FL}_0$ had been determined almost a decade earlier in [2]. In fact, as shown in [10, 11], unification in $\mathcal{FL}_0$ corresponds to unification modulo the equational theory of idempotent Abelian monoids with several homomorphisms. In [2] it was shown that, already for a single homomorphism, unification modulo this theory has unification type zero, i.e., there are unification problems for this theory that do not have a minimal complete set of unifiers. In particular, such unification problems cannot have a finite complete set of unifiers.

In this paper, we consider unification in the DL $\mathcal{EL}$. The $\mathcal{EL}$-family of description logics (DLs) is a family of inexpressive DLs whose main distinguishing feature is that they provide their users with *existential restrictions* ($\exists r.C$) rather than value restrictions ($\forall r.C$) as the main concept constructor involving roles. The core language of this family is $\mathcal{EL}$, which has the top concept, conjunction, and existential restrictions as concept constructors. This family has recently drawn considerable attention since, on the one hand, the subsumption problem stays tractable (i.e., decidable in polynomial time) in situations where $\mathcal{FL}_0$, the corresponding DL with value restrictions, becomes intractable: subsumption between concept terms is tractable for both $\mathcal{FL}_0$ and $\mathcal{EL}$, but allowing the use of concept definitions or even more expressive terminological formalisms makes $\mathcal{FL}_0$ intractable [3, 16, 5], whereas it leaves $\mathcal{EL}$ tractable [4, 14, 5]. On the other hand, although of limited expressive power, $\mathcal{EL}$ is nevertheless used in applications, e.g., to define biomedical ontologies. For example, both the large medical ontology SNOMED CT [20] and the Gene Ontology [1] can be expressed in $\mathcal{EL}$, and the same is true for large parts of the medical ontology GALEN [18].

Unification in $\mathcal{EL}$ has, to the best of our knowledge, not been considered before, but matching (where one side of the equation(s) to be solved does not contain variables) has been investigated in [7, 17]. In particular, it was shown in [17] that the decision problem, i.e., the problem of deciding whether a given $\mathcal{EL}$-matching problem has a matcher or not, is NP-complete. Interestingly, $\mathcal{FL}_0$ behaves better w.r.t. matching than $\mathcal{EL}$: for $\mathcal{FL}_0$, the decision problem is tractable [9]. In this paper, we show that, w.r.t. the unification type, $\mathcal{FL}_0$ and $\mathcal{EL}$ behave the same: just as $\mathcal{FL}_0$, the DL $\mathcal{EL}$ has unification type zero.

In the next section, we define the DL $\mathcal{EL}$ and unification in $\mathcal{EL}$ more formally. In Section 3, we recall the characterisation of equivalence in $\mathcal{EL}$ from [17], and in Section 4 we use this to show that unification in $\mathcal{EL}$ has type zero. In Section 5 we point out that this result implies that unification modulo the equational theory of semilattices with monotone operators is of unification type zero.

More information about Description Logics can be found in [6], and about unification theory in [12, 13].

2 Unification in $\mathcal{EL}$

First, we define the syntax and semantics of $\mathcal{EL}$-concept terms as well as the subsumption and the equivalence relation on these terms.

Starting with a set N_{con} of concept names and a set N_{role} of role names, *$\mathcal{EL}$-concept terms* are built using the concept constructors top concept ($\top$), conjunction ($\sqcap$), and existential restriction ($\exists r.C$). The semantics of $\mathcal{EL}$ is defined in the usual way, using the notion of an interpretation $\mathcal{I} = (\mathcal{D}_\mathcal{I}, \cdot^\mathcal{I})$, which consists of a nonempty domain $\mathcal{D}_\mathcal{I}$ and an interpretation function $\cdot^\mathcal{I}$ that assigns binary relations on $\mathcal{D}_\mathcal{I}$ to role names and subsets of $\mathcal{D}_\mathcal{I}$ to concept terms, as shown in the semantics column of Table 1.

Name	Syntax	Semantics
concept name	A	$A^\mathcal{I} \subseteq \mathcal{D}_\mathcal{I}$
role name	r	$r^\mathcal{I} \subseteq \mathcal{D}_\mathcal{I} \times \mathcal{D}_\mathcal{I}$
top-concept	$\top$	$\top^\mathcal{I} = \mathcal{D}_\mathcal{I}$
conjunction	$C \sqcap D$	$(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$
existential restriction	$\exists r.C$	$(\exists r.C)^\mathcal{I} = \{x \mid \exists y : (x, y) \in r^\mathcal{I} \land y \in C^\mathcal{I}\}$
subsumption	$C \sqsubseteq D$	$C^\mathcal{I} \subseteq D^\mathcal{I}$
equivalence	$C \equiv D$	$C^\mathcal{I} = D^\mathcal{I}$

Table 1. Syntax and semantics of $\mathcal{EL}$

The concept term C *is subsumed by* the concept term D (written $C \sqsubseteq D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ holds for all interpretations $\mathcal{I}$. We say that C *is equivalent to* D (written $C \equiv D$) iff $C \sqsubseteq D$ and $D \sqsubseteq C$, i.e., iff $C^\mathcal{I} = D^\mathcal{I}$ holds for all interpretations $\mathcal{I}$.

In order to define unification of concept terms, we must first introduce the notion of a substitution operating on concept terms. To this purposes, we partition the set of concepts names into a set N_v of concept variables (which may be replaced by substitutions) and a set N_c of concept constants (which must not be replaced by substitutions). Intuitively, N_v are the concept names that have possibly been given another name or been specified in more detail in another concept term describing the same notion. The elements of N_c are the ones of which it is assumed that the same name is used by all knowledge engineers (e.g., standardised names in a certain domain).

A *substitution* σ is a mapping from N_v into the set of all $\mathcal{EL}$-concept terms. This mapping is extended to concept terms in the obvious way, i.e.,

- $\sigma(A) := A$ for all $A \in N_c$,
- $\sigma(\top) := \top$,
- $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$, and
- $\sigma(\exists R.C) := \exists R.\sigma(C)$.

Definition 1. *An $\mathcal{EL}$-*unification problem *is of the form $C \equiv^? D$, where C, D are $\mathcal{EL}$-concept terms. The substitution σ is a* unifier *(or* solution*) of this problem iff $\sigma(C) \equiv \sigma(D)$. In this case, the problem is called* solvable*, and the concept terms C and D are called* unifiable.

As usual, unifiers can be compared using the instantiation preorder $\trianglelefteq$. Let $C \equiv^? D$ be an $\mathcal{EL}$-unification problem, V the set of variables occurring in C, D, and σ, θ two unifiers of this problem. We define

$$\sigma \leq^{\bullet} \theta \text{ iff there is a substitution } \lambda \text{ such that } \theta(X) \equiv \lambda(\sigma(X)) \text{ for all } X \in V.$$

If $\sigma \leq^{\bullet} \theta$, then we say that θ is an *instance* of σ.

Definition 2. *Let $C \equiv^{?} D$ be an $\mathcal{EL}$-unification problem. The set of substitutions M is called a* complete set of unifiers *for $C \equiv^{?} D$ iff it satisfies the following two properties:*

1. *every element of M is a unifier of $C \equiv^{?} D$;*
2. *if θ is a unifier of $C \equiv^{?} D$, then there exists a unifier $\sigma \in M$ such that $\sigma \leq^{\bullet} \theta$.*

The set M is called a minimal complete set of unifiers *for $C \equiv^{?} D$ iff it additionally satisfies*

3. *if $\sigma, \theta \in M$, then $\sigma \leq^{\bullet} \theta$ implies $\sigma = \theta$.*

The unification type of a given unification problem is determined by the existence and cardinality of such a minimal complete set.

Definition 3. *Let $C \equiv^{?} D$ be an $\mathcal{EL}$-unification problem. This problem has type* unitary *(*finitary, infinitary*) iff it has a minimal complete set of unifiers of cardinality* 1 *(finite cardinality, infinite cardinality). If $C \equiv^{?} D$ does not have a minimal complete set of unifiers, then it is of type* zero.

Note that the set of all unifiers of a given $\mathcal{EL}$-unification problem is always a complete set of unifiers. However, this set is usually infinite and redundant (in the sense that some unifiers are instances of others). For a unitary or finitary $\mathcal{EL}$-unification problem, all unifiers can be represented by a finite complete set of unifiers. For problems of type infinitary or zero, this is no longer possible. In fact, if a problem has a finite complete set of unifiers M, then it also has a finite *minimal* complete set of unifiers, which can be obtained by iteratively removing redundant elements from M, i.e., by removing the unifier $\theta \in M$ if it is an instance of another unifier in M. For an infinite complete set of unifiers, this approach of removing redundant unifiers may be infinite, and the set reached in the limit need no longer be complete. This is what happens for problems of type zero. The difference between infinitary and type zero is that a unification problem of type zero cannot even have a non-redundant complete set of unifiers, i.e., every complete set of unifiers must contain different unifiers σ, θ such that $\sigma \leq^{\bullet} \theta$.

When we say that *$\mathcal{EL}$ has unification type zero*, we mean that there exists an $\mathcal{EL}$-unification problem that has type zero. Before we can prove that this is indeed the case, we must first have a closer look at equivalence in $\mathcal{EL}$.

3 Equivalence in $\mathcal{EL}$

In order to characterise equivalence of $\mathcal{EL}$-concept terms, the notion of a reduced $\mathcal{EL}$-concept term is introduced in [17]. A given $\mathcal{EL}$-concept term can be transformed into an equivalent reduced term by applying the following rules modulo associativity and commutativity of conjunction:

$C \sqcap \top \to C$	for all $\mathcal{EL}$-concept terms C
$A \sqcap A \to A$	for all concept names $A \in N_{con}$
$\exists r.C \sqcap \exists r.D \to \exists r.C$	for all $\mathcal{EL}$-concept terms C, D with $C \sqsubseteq D$

Obviously, these rules are equivalence preserving. We say that the $\mathcal{EL}$-concept term C is *reduced* if none of the above rules is applicable to it (modulo associativity and commutativity of $\sqcap$). The $\mathcal{EL}$-concept term D is a *reduced form* of C if D is reduced and can be obtained from C by applying the above rules (modulo associativity and commutativity of $\sqcap$). The following theorem is shown in [17] (see Theorem 6.3.1 on page 181).

Theorem 1. *Let C, D be reduced $\mathcal{EL}$-concept terms. Then $C \equiv D$ iff C is identical to D up to associativity and commutativity of $\sqcap$.*

As an easy consequence of this theorem, we obtain:

Corollary 1. *Let C, D be $\mathcal{EL}$-concept terms, and $\widehat{C}, \widehat{D}$ reduced forms of C, D, respectively. Then $C \equiv D$ iff $\widehat{C}$ is identical to $\widehat{D}$ up to associativity and commutativity of $\sqcap$.*

The following two lemmas, which are easy consequences of this corollary, will be used in our proof that $\mathcal{EL}$ has unification type zero.

Lemma 1. *Assume that C, D are reduced $\mathcal{EL}$-concept terms such that $\exists r.D \sqsubseteq C$. Then C is either $\top$, or of the form $C = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ where*

- *$n \geq 1$;*
- *$C_1, \ldots, C_n$ are reduced and pairwise incomparable w.r.t. subsumption;*
- *$D \sqsubseteq C_1, \ldots, D \sqsubseteq C_n$.*

Proof. We have $\exists r.D \sqsubseteq C$ iff $C \sqcap \exists r.D \equiv \exists r.D$. Since $\exists r.D$ is reduced, any reduced form of $C \sqcap \exists r.D$ must be identical (up to associativity and commutativity of $\sqcap$) to $\exists r.D$. If $C \neq \top$, then the only rule that can be applied to reduce $C \sqcap \exists r.D$ is the third one. It is easy to see that we can only obtain $\exists r.D$ by applying this rule if C is of the form $C = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ where $D \sqsubseteq C_1, \ldots, D \sqsubseteq C_n$. Since C was assumed to be reduced, the terms $C_1, \ldots, C_n$ must also be reduced and pairwise incomparable w.r.t. subsumption. □

Conversely, we also have:

Lemma 2. *If C, D are $\mathcal{EL}$-concept terms such that $C = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ and $D \sqsubseteq C_1, \ldots, D \sqsubseteq C_n$, then $\exists r.D \sqsubseteq C$.*

Proof. Assume that C, D satisfy the prerequisites of the lemma. Then it is easy to see that $\exists r.D$ is a reduced form of $C \sqcap \exists r.D$, and thus $C \sqcap \exists r.D \equiv \exists r.D$. □

4 An $\mathcal{EL}$-unification problem of type zero

To show that $\mathcal{EL}$ has unification type zero, we must exhibit an $\mathcal{EL}$-unification problem that has this type.

Theorem 2. *Let X, Y be variables. The $\mathcal{EL}$-unification problem $X \sqcap \exists r.Y \equiv \exists r.Y$ has unification type zero.*

Proof. It is enough to show that any complete set of unifiers for this problem is redundant, i.e., contains two different unifiers that are comparable w.r.t. the instantiation preorder. Thus, let M be a complete set of unifiers for $X \sqcap \exists r.Y \equiv \exists r.Y$.

First, note that M must contain a unifier that maps X to an $\mathcal{EL}$-concept term not equivalent to $\top$ or $\exists r.\top$. In fact, consider a substitution τ such that $\tau(X) = \exists r.A$ and $\tau(Y) = A$. Obviously, τ is a unifier of $X \sqcap \exists r.Y \equiv \exists r.Y$. Thus, M must contain a unifier σ such that $\sigma \lessdot \tau$. In particular, this means that there is a substitution λ such that $\exists r.A = \tau(X) \equiv \lambda(\sigma(X))$. Obviously, $\sigma(X) \equiv \top$ ($\sigma(X) \equiv \exists r.\top$) would imply $\lambda(\sigma(X)) \equiv \top$ ($\lambda(\sigma(X)) \equiv \exists r.\top$), and thus $\exists r.A \equiv \top$ ($\exists r.A \equiv \exists r.\top$), which is, however, not the case.

Thus, let $\sigma \in M$ be such that $\sigma(X) \not\equiv \top$ and $\sigma(X) \not\equiv \exists r.\top$. Without loss of generality, we assume that $C := \sigma(X)$ and $D := \sigma(Y)$ are reduced. Since σ is a unifier of $X \sqcap \exists r.Y \equiv \exists r.Y$, we have $\exists r.D \sqsubseteq C$. Consequently, Lemma 1 yields that C is of the form $C = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ where $n \geq 1$, $C_1, \ldots, C_n$ are reduced and pairwise incomparable w.r.t. subsumption, and $D \sqsubseteq C_1, \ldots, D \sqsubseteq C_n$.

We use σ to construct a new unifier $\widehat{\sigma}$ as follows:

$$\widehat{\sigma}(X) := \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n \sqcap \exists r.Z$$
$$\widehat{\sigma}(Y) := D \sqcap Z$$

where Z is a new variable (i.e., one not occurring in C, D). Lemma 2 implies that $\widehat{\sigma}$ is indeed a unifier of $X \sqcap \exists r.Y \equiv \exists r.Y$.

Next, we show that $\widehat{\sigma} \lessdot \sigma$. To this purpose, we consider the substitution λ that maps Z to C_1, and does not change any of the other variables. Then we have $\lambda(\widehat{\sigma}(X)) = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n \sqcap \exists r.C_1 \equiv \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n = \sigma(X)$ and $\lambda(\widehat{\sigma}(Y)) = D \sqcap C_1 \equiv D = \sigma(Y)$. Note that the second equivalence holds since we have $D \sqsubseteq C_1$.

Since M is complete, there exists a unifier $\theta \in M$ such that $\theta \lessdot \widehat{\sigma}$. Transitivity of the relation $\lessdot$ thus yields $\theta \lessdot \sigma$. Since σ and θ both belong to M, we have completed the proof of the theorem once we have shown that $\sigma \neq \theta$. Assume to the contrary that $\sigma = \theta$. Then we have $\sigma \lessdot \widehat{\sigma}$, and thus there exists a substitution μ such that $\mu(\sigma(X)) \equiv \widehat{\sigma}(X)$, i.e.,

$$\exists r.\mu(C_1) \sqcap \ldots \sqcap \exists r.\mu(C_n) \equiv \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n \sqcap \exists r.Z. \qquad (1)$$

Recall that the concept terms $C_1, \ldots, C_n$ are reduced and pairwise incomparable w.r.t. subsumption. In addition, since $\sigma(X) = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ is reduced and not equivalent to $\exists r.\top$, none of the concept terms $C_1, \ldots, C_n$ can be equivalent to $\top$. Finally, Z is a concept name that does not occur in $C_1, \ldots, C_n$. All this implies that $\exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n \sqcap \exists r.Z$ is reduced. Obviously, any reduced form for $\exists r.\mu(C_1) \sqcap \ldots \sqcap \exists r.\mu(C_n)$ is a conjunction of at most n existential restrictions. Thus, Corollary 1 shows that the above equivalence (1) actually cannot hold.

To sum up, we have shown that M contains two distinct unifiers σ, θ such that $\theta \lessdot \sigma$. Since M was an arbitrary complete set of unifiers for $X \sqcap \exists r.Y \equiv \exists r.Y$, this shows that this unification problem cannot have a minimal complete set of unifiers. □

Note that, in this proof, the availability of $\top$ in $\mathcal{EL}$ was not needed.[1] For this reason, the result still holds if, instead of $\mathcal{EL}$, we consider its sublanguage that has only conjunction and existential restriction as concept constructor.

5 Unification in semilattices with monotone operators

Of course, unification types were originally not introduced for Description Logics, but for equational theories. In this section, we show that the above result for unification in $\mathcal{EL}$ can actually be viewed as a result for an equational theory.

As shown in [19], the equivalence problem for $\mathcal{EL}$-concept terms corresponds to the word problem for the equational theory of semilattices with monotone operators. In order to define this theory, we consider a signature Σ_{SLmO} consisting of a binary function symbol $\wedge$, a constant symbol 1, and finitely many unary function symbols $f_1, \ldots, f_n$. Terms can then be built using these symbols and additional variable symbols and free constant symbols.

Definition 4. *The equational theory of* semilattices with monotone operators *is defined by the following identities:*

$$SLmO := \{x \wedge (y \wedge z) = (x \wedge y) \wedge z, \quad x \wedge y = y \wedge x, \quad x \wedge x = x\} \cup \{f_i(x \wedge y) \wedge f_i(y) = f_i(x \wedge y) \mid 1 \leq i \leq n\}$$

A given $\mathcal{EL}$-concept term C using only roles $r_1, \ldots, r_n$ can be translated into a term t_C over the signature Σ_{SLmO} by replacing each concept constant A by a corresponding free constants a, each concept variable X by a corresponding variable x, $\top$ by 1, $\sqcap$ by $\wedge$, and $\exists r_i$ by f_i. For example, $C = A \sqcap \exists r_3.(X \sqcap B)$ is translated into $t_C = a \wedge f_3(x \wedge b)$. Conversely, any term over the signature Σ_{SLmO} can be translated back into an $\mathcal{EL}$-concept term.

Lemma 3. *Let C, D be $\mathcal{EL}$-concept term using only roles $r_1, \ldots, r_n$. Then $C \equiv D$ iff $t_C =_{SLmO} t_D$.*

As an immediate consequence of this lemma, we have that unification in the DL $\mathcal{EL}$ corresponds to unification modulo the equational theory $SLmO$. Thus, Theorem 2 implies that $SLmO$ has unification type zero.

[1] The only place where we have said anything about $\top$ was when we excluded that $\sigma(X)$ is equivalent to $\top$ or $\exists r.\top$.

Corollary 2. *The equational theory of semilattices with monotone operators has unification type zero.*

Since the unification problem introduced in Theorem 2 contains only one role r, this is already true in the presence of a single monotone operator.

6 Future work

In this paper we have determined the unification type of $\mathcal{EL}$. Currently, we are investigating the decidability issue, i.e., given an $\mathcal{EL}$-unification problem, is it decidable whether this problem has a unifier or not. We conjecture that this problem is decidable (more precisely, NP-complete); however, the proofs of completeness and termination for the algorithm(s) that we have devised so far still have some "holes."

References

1. M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, J. M. Butler, H.and Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. *Nat Genet*, 25(1):25–29, 2000.
2. Franz Baader. Unification in commutative theories. *J. of Symbolic Computation*, 8(5):479–497, 1989.
3. Franz Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proc. of the 8th Nat. Conf. on Artificial Intelligence (AAAI'90)*, pages 621–626, Boston (Ma, USA), 1990.
4. Franz Baader. Terminological cycles in a description logic with existential restrictions. In Georg Gottlob and Toby Walsh, editors, *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 325–330, Acapulco, Mexico, 2003. Morgan Kaufmann, Los Altos.
5. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, Edinburgh (UK), 2005. Morgan Kaufmann, Los Altos.
6. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
7. Franz Baader and Ralf Küsters. Matching in description logics with existential restrictions. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000)*, pages 261–272, 2000.
8. Franz Baader and Ralf Küsters. Unification in a description logic with transitive closure of roles. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, (LPAR 2001)*, Lecture Notes in Artificial Intelligence, Havana, Cuba, 2001. Springer-Verlag.
9. Franz Baader, Ralf Küsters, Alex Borgida, and Deborah L. McGuinness. Matching in description logics. *J. of Logic and Computation*, 9(3):411–447, 1999.

10. Franz Baader and Paliath Narendran. Unification of concept terms in description logics. In H. Prade, editor, *Proc. of the 13th Eur. Conf. on Artificial Intelligence (ECAI'98)*, pages 331–335. John Wiley & Sons, 1998.
11. Franz Baader and Paliath Narendran. Unification of concepts terms in description logics. *J. of Symbolic Computation*, 31(3):277–305, 2001.
12. Franz Baader and Jörg H. Siekmann. Unification theory. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 41–125. Oxford University Press, Oxford, UK, 1994.
13. 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.
14. Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In Ramon López de Mántaras and Lorenza Saitta, editors, *Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004)*, pages 298–302, 2004.
15. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
16. Yevgeny Kazakov and Hans de Nivelle. Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*. CEUR Electronic Workshop Proceedings, http://CEUR-WS.org/Vol-81/, 2003.
17. Ralf Küsters. *Non-standard Inferences in Description Logics*, volume 2100 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.
18. Alan Rector and Ian Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, Menlo Park, California, 1997. AAAI Press.
19. Viorica Sofronie-Stokkermans. Locality and subsumption testing in $\mathcal{EL}$ and some of its extensions. In *Proc. of the 2008 Description Logic Workshop (DL 2008)*. CEUR Electronic Workshop Proceedings, volumne 353, 2008.
20. Kent Spackman. Managing clinical terminology hierarchies using algorithmic calculation. *Journal of the American Medical Informatics Association*, Fall Symposium Special Issue, 2000.

Equational Unification by Variant Narrowing (Extended Abstract)

Santiago Escobar[1], José Meseguer[2] and Ralf Sasse[2]

[1] Universidad Politécnica de Valencia, Spain. `sescobar@dsic.upv.es`
[2] University of Illinois at Urbana-Champaign, USA.
{`meseguer`,`rsasse`}`@cs.uiuc.edu`

Abstract. Narrowing is a well-known complete procedure for equational E-unification when E can be decomposed as a union $E = \varDelta \uplus B$ with B a set of axioms for which a finitary unification algorithm exists, and $\varDelta$ a set of confluent, terminating, and B-coherent rewrite rules. However, when $B \neq \emptyset$, effective narrowing strategies such as basic narrowing easily fail to be complete and cannot be used. This poses two challenges to narrowing-based equational unification: (i) finding effective narrowing strategies that are complete modulo B under mild assumptions on B, and (ii) finding sufficient conditions under which such narrowing strategies yield *finitary* E-unification algorithms. Inspired by Comon and Delaune's notion of E-variant for a term, we propose a new narrowing strategy called *variant narrowing* that has a search space potentially much smaller than full narrowing, is complete, and yields a finitary E-unification algorithm when E has the finite variant property.

1 Introduction

Equational unification is the solving of existentially quantified problems $\exists \boldsymbol{x}\; t =_E t'$ modulo an equational theory E. If the equations E are convergent, it is well-known that narrowing provides a complete unification procedure for E-unification [4]. This result extends to narrowing modulo a set B of equational axioms. That is, if $E = \varDelta \uplus B$, where $\varDelta$ is a set of oriented equations that are convergent and coherent modulo B, then narrowing with $\varDelta$ modulo B is also a complete E-unification procedure [5]. In practice, however, full narrowing, i.e., considering all narrowing sequences, can be highly inefficient. This has led to the search for complete narrowing strategies that have a much smaller search space; and to conditions under which narrowing terminates, so that a finitary unification algorithm can be obtained. Hullot's basic narrowing [4] is one such strategy, which is complete (for normalized substitutions, see [4], though it does also produce non-normalized substitutions) and terminates under suitable conditions. The problem, however, is that basic narrowing is complete for $B = \emptyset$, but is *incomplete* for a general set B of axioms, and in particular for associativity-commutativity (AC) (see [8,1]).

In [3] we have addressed the problem of finding complete narrowing procedures modulo B, under minimal assumptions on B, which have a much smaller search space than full narrowing, and for which finitary unification conditions can be given. Specifically, inspired by the notion of E-variant of a term due to Comon and Delaune [1], we have proposed a new narrowing method called *variant narrowing* with the following properties: (i) it only uses substitutions in normal form modulo B; (ii) it is complete under very general assumptions on B and Δ (and avoids many wasteful narrowing sequences that would be created by full narrowing); and (iii) if Δ has the finite variant property modulo B, it can be used to both compute all the finite variants of a term in a very space-efficient way, and to obtain a *finitary* E-unification algorithm.

Indeed, when Δ has the finite variant property modulo B, we have showed in [3] how variant narrowing can be specialized into two terminating algorithms, one for computing the finite set of variants of any term, and another optimized one for providing a finitary E-unification algorithm that computes a complete and minimal set of E-unifiers. Moreover, in [2] we have developed checkable conditions for a theory to have the finite variant property.

We assume some knowledge on term rewriting, narrowing and rewriting logic [7,6]. We adopt an order-sorted, typed setting, and assume throughout that the rules Δ are confluent, terminating and sort decreasing modulo regular axioms B as detailed in [3]. We do not explicitly mention such conditions in definitions and theorems.

2 Variants

We illustrate the major points with the following running example.

Example 1. Let us consider the following equational theory for the exclusive or operator and the cancellation equations for public encryption/decryption. The exclusive or symbol $\oplus$ has associative and commutative (AC) properties with 0 as its unit. The symbol pk is used for public key encryption and the symbol sk for private key encryption. The equational theory (Σ, E) is decomposed as a rewrite theory (Σ, B, Δ), with B the AC axioms for $\oplus$ and Δ the rules below. It has been shown to satisfy the finite variant property in [2].

$$\begin{array}{lll} X \oplus 0 \to X & X \oplus X \oplus Y \to Y & pk(K, sk(K, M)) \to M \\ X \oplus X \to 0 & & sk(K, pk(K, M)) \to M \end{array}$$

Definition 1 (Variants). [1] *Given a term t and an order-sorted equational theory E, we say that (t', θ) is an* E-variant *of t if $t\theta =_E t'$, where $Dom(\theta) \subseteq Var(t)$ and $Ran(\theta) \cap Var(t) = \emptyset$.*

Definition 2 (Minimal and complete set of variants). [1] *Let (Σ, B, Δ) be a decomposition of an equational theory (Σ, E). A* minimal and complete set of E-variants *(up to renaming) of a term t, denoted (in the case it is finite) $FV_{\Delta,B}(t)$, is a set S of E-variants of t such that, for each substitution σ, there is a variant $(t', \rho) \in S$ and a substitution θ such that: (i) t' is Δ, B-irreducible, (ii) $(t\sigma){\downarrow_{\Delta,B}} =_B t'\theta$, (iii) $(\sigma{\downarrow_{\Delta,B}})|_{Var(t)} =_B (\rho\theta)|_{Var(t)}$, and (iv) (t', ρ) is minimal,* *i.e., there is no $(t'', \rho') \in S$ and τ such that $\rho|_{Var(t)} =_B (\rho'\tau)|_{Var(t)}$ and $t' =_B t''\tau$.*

The finite variant property is integral to our approach. Checkable conditions have been developed in [2]. Next we present a result that allows computation of the set of finite variants of a term by narrowing.

Proposition 1 (Computing the Finite Variants I). [3] *Let (Σ, B, Δ) be a finite variant decomposition of an order-sorted equational theory (Σ, E). Let $t \in \mathcal{T}_\Sigma(\mathcal{X})$ and $\#_{\Delta,B}(t) = n$ (a bound on the number of rewrite steps necessary to reach a normal-form from $t\sigma$ independent of σ). Then, $(s, \sigma) \in FV_{\Delta,B}(t)$ if and only if there is a narrowing derivation $t \overset{\sigma}{\rightsquigarrow}{}^{\leq n}_{\Delta,B} s$ such that s is $\to_{\Delta,B}$-irreducible, σ is $\to_{\Delta,B}$-normalized, and there are no term s' and $\to_{\Delta,B}$-normalized substitutions σ', τ such that $t \overset{\sigma'}{\rightsquigarrow}{}^{\leq n}_{\Delta,B} s'$, $(\sigma'\tau)|_{Var(t)} =_B \sigma|_{Var(t)}$, and $s =_B s'\tau$.*

Example 2. Let us use the theory given in Example 1. For $s = X \oplus sk(K, pk(K, Y))$ we get the following seven narrowing sequences that will make up a minimal and complete set of E-variants: (i) $s \overset{id}{\rightsquigarrow}{}^*_{\Delta,B} X \oplus Y$, (ii) $s \rightsquigarrow^*_{\{X \mapsto 0, Y \mapsto Z\},\Delta,B} Z$, (iii) $s \rightsquigarrow^*_{\{X \mapsto Z, Y \mapsto 0\},\Delta,B} Z$, (iv) $s \rightsquigarrow^*_{\{X \mapsto Z \oplus U, Y \mapsto U\},\Delta,B} Z$, (v) $s \rightsquigarrow^*_{\{X \mapsto U, Y \mapsto Z \oplus U\},\Delta,B} Z$, (vi) $s \rightsquigarrow^*_{\{X \mapsto U, Y \mapsto U\},\Delta,B} 0$, and (vii) $s \rightsquigarrow^*_{\{X \mapsto U \oplus Z_1, Y \mapsto U \oplus Z_2\},\Delta,B} Z_1 \oplus Z_2$. Therefore $(X \oplus Y, id)$, $(Z, \{X \mapsto 0, Y \mapsto Z\})$, $(Z, \{X \mapsto Z, Y \mapsto 0\})$, $(Z, \{X \mapsto Z \oplus U, Y \mapsto U\})$, $(Z, \{X \mapsto U, Y \mapsto Z \oplus U\})$, $(0, \{X \mapsto U, Y \mapsto U\})$, and $(Z_1 \oplus Z_2, \{X \mapsto U \oplus Z_1, Y \mapsto U \oplus Z_2\})$ are the E-variants; indeed they are a minimal set. The alternative narrowing sequence (viii) $s \rightsquigarrow^*_{\{X \mapsto U, Y \mapsto U \oplus Z_1 \oplus Z_2\},\Delta,B} Z_1 \oplus Z_2$ is an instance of (v), simply by considering the substitution $\{Z \mapsto Z_1 \oplus Z_2\}$.

3 Variant Narrowing

Let us first motivate why an alternative narrowing strategy is necessary for confluent and terminating rewrite theories with rules Δ modulo axioms B. Applying narrowing $\rightsquigarrow_{\Delta,B}$ to perform $(\Delta \uplus B)$-unification without any restriction is very wasteful, because as soon as a rewrite step $\to_{\Delta,B}$ is enabled in a term that has also narrowing steps $\rightsquigarrow_{\Delta,B}$, that rewrite step should be taken before any further narrowing steps are applied, thanks to confluence modulo B. This idea is consistent with the implementation of rewriting logic [9] and, therefore, the relation $\to^!_{\Delta,B}; \rightsquigarrow_{\Delta,B}$ makes sense as an optimization of $\rightsquigarrow_{\Delta,B}$. However, this is still a naive approach, since a rewrite step and a narrowing step satisfy a more general property which is the reason for being able to take the rewrite step and avoiding the narrowing step. Namely, if two narrowing steps $t \overset{\sigma_1}{\rightsquigarrow}_{\Delta,B} t_1$ and $t \overset{\sigma_2}{\rightsquigarrow}_{\Delta,B} t_2$ are possible and we have that $\sigma_1 \leq_B \sigma_2$ (i.e., σ_1 is more general than σ_2), then it is enough to take only the narrowing step using σ_1.

Definition 3 (Equivalence classes for narrowing steps). [3] *Let $\mathcal{R} = (\Sigma, B, \Delta)$ be an order-sorted rewrite theory. Let us consider two narrowing steps $\alpha_1 : t \overset{\sigma_1}{\rightsquigarrow}_{\Delta,B} s_1$ and $\alpha_2 : t \overset{\sigma_2}{\rightsquigarrow}_{\Delta,B} s_2$. We write $\alpha_1 \preceq_B \alpha_2$ if $\sigma_1|_{Var(t)} \leq_B \sigma_2|_{Var(t)}$*

and $\alpha_1 \prec_B \alpha_2$ if $\sigma_1|_{Var(t)} <_B \sigma_2|_{Var(t)}$ (i.e., σ_1 is strictly more general than σ_2). We write $\alpha_1 \simeq_B \alpha_2$ if $\sigma_1|_{Var(t)} \simeq_B \sigma_2|_{Var(t)}$. The relation $\alpha_1 \simeq_B \alpha_2$ between two narrowing steps from t defines a set of equivalence classes between such narrowing steps. In what follows we will be interested in choosing a unique representation $\underline{\alpha} \in [\alpha]_{\simeq_B}$ in each equivalence class of narrowing steps from t. Therefore, $\underline{\alpha}$ will always denote a chosen unique representative $\underline{\alpha} \in [\alpha]_{\simeq_B}$.

Definition 4 (Variant Narrowing). [3] *Let $\mathcal{R} = (\Sigma, B, \Delta)$ be an order-sorted rewrite theory. We define $t \overset{p;\sigma}{\rightsquigarrow}_{\underline{\Delta},B} s$ as $\underline{\alpha} : t \overset{p;\sigma}{\rightsquigarrow}_{\Delta,B} s$ such that σ is Δ, B-normalized if $\sigma|_{Var(t)}$ is not a renaming, $\underline{\alpha}$ is minimal w.r.t. the order $\preceq_B$, and $\underline{\alpha}$ is a chosen unique representative of its $\simeq_B$-equivalence class.*

Note that the relation $\to^!_{\Delta,B}; \rightsquigarrow_{\Delta,B}$ is (appropriately) simulated by $\rightsquigarrow_{\underline{\Delta},B}$, since in the relation $\rightsquigarrow_{\underline{\Delta},B}$ rewriting steps are always given priority over narrowing steps. The normalization with our variant narrowing is unique as shown by the following result.

Lemma 1 (Normalization of Variant Narrowing). [3] *Let $\mathcal{R} = (\Sigma, B, \Delta)$ be an order-sorted rewrite theory. Let $t \in \mathcal{T}_\Sigma(\mathcal{X})$. If t is not Δ, B-irreducible, then, relative to the unique choice of $\underline{\alpha} \in [\alpha]_{\simeq_B}$ in Definition 3, there is a unique $\rightsquigarrow_{\underline{\Delta},B}$-narrowing sequence from t such that $t \overset{id}{\rightsquigarrow}{}^*_{\underline{\Delta},B} t{\downarrow}_{\Delta,B}$.*

The following result ensures that variant narrowing is complete.

Theorem 1 (Completeness of Variant Narrowing). [3] *Let $\mathcal{R} = (\Sigma, B, \Delta)$ be an order-sorted rewrite theory. If $t \overset{\sigma}{\rightsquigarrow}{}^*_{\Delta,B} t\sigma{\downarrow}_{\Delta,B}$ with σ Δ, B-normalized, and there are no substitutions ρ, ρ' such that $t \overset{\rho}{\rightsquigarrow}{}^*_{\Delta,B} t\rho{\downarrow}_{\Delta,B}$, $t\sigma{\downarrow}_{\Delta,B} =_B (t\rho{\downarrow}_{\Delta,B})\rho'$, $\sigma|_{Var(t)} =_B (\rho\rho')|_{Var(t)}$, and $\rho' \neq id$, then $t \overset{\sigma}{\rightsquigarrow}{}^*_{\underline{\Delta},B} t\sigma{\downarrow}_{\Delta,B}$.*

This result allows more efficient computation of the set of finite variants, based on variant narrowing.

Theorem 2 (Computing the Finite Variants II). [3] *Let (Δ, B) be a finite variant decomposition of an order-sorted equational theory (Σ, E). Let $t \in \mathcal{T}_\Sigma(\mathcal{X})$ and $\#_{\Delta,B}(t) = n$ Then $(s, \sigma) \in FV_{\Delta,B}(t)$ if and only if there is a variant narrowing derivation $t \overset{\sigma}{\rightsquigarrow}{}^{\leq n}_{\underline{\Delta},B} s$ such that s is $\to_{\Delta,B}$-irreducible and σ is $\to_{\Delta,B}$-normalized.*

4 Equational Unification

Our main point is that we can now compute unifiers modulo E, with (Σ, E) decomposed as (Σ, B, Δ), by computing E-variants and then solving the simpler unification problem modulo B on the variants.

Theorem 3 (Finite Variant unification procedure). [3] *Let $\mathcal{R} = (\Sigma, B, \Delta)$ be an order-sorted rewrite theory that has the finite variant property. To obtain the finitary and complete set of $\Delta \uplus B$-unifiers of two terms t and t'*

we (i) compute their E-variants, say $FV_{\Delta\uplus B}(t) = \{(t_1,\sigma_1),\ldots,(t_n,\sigma_n)\}$ *and* $FV_{\Delta\uplus B}(t') = \{(t'_1,\sigma'_1),\ldots,(t'_m,\sigma'_m)\}$*, and then (ii) try B-unification on each pair* t_i*,* t'_j *for all* $1 \leq i \leq n$ *and* $1 \leq j \leq m$*. Then,* $\theta \in CSU_{\Delta\uplus B}(t = t')$ *if and only if there are* $1 \leq i \leq n$*,* $1 \leq j \leq m$ *and two substitutions* ρ, ρ' *such that* $\rho \in (\sigma_i \cap_B \sigma'_j)$*,* $\rho' \in CSU_B(t_i = t'_j)$*, and* $\theta =_B \rho\rho'$*; where the meet* $\sigma \cap_B \sigma'$ *of two substitutions* σ, σ' *is the set of most general substitutions* τ *such that there are minimal* ρ *and* ρ' *such that* $\sigma\rho =_B \sigma'\rho'$*, and* $\tau = \sigma\rho$*. The set* $CSU_{\Delta\uplus B}(t = t')$ *of unifiers is minimal if whenever* θ *is non-normalized or has an alternative, more general one, we discard it.*

Example 3. To bring our running exclusive or and encryption example to conclusion, let us compute the unifiers of two terms by the method given in Theorem 3. For the term $s = X \oplus sk(K, pk(K, Y))$ we have $(X \oplus Y, id)$, $(Z, \{X \mapsto 0, Y \mapsto Z\})$, $(Z, \{X \mapsto Z, Y \mapsto 0\})$, $(Z, \{X \mapsto Z \oplus U, Y \mapsto U\})$, $(Z, \{X \mapsto U, Y \mapsto Z \oplus U\})$, $(0, \{X \mapsto U, Y \mapsto U\})$, and $(Z_1 \oplus Z_2, \{X \mapsto U \oplus Z_1, Y \mapsto U \oplus Z_2\})$ as E-variants as shown in Example 2. Considering $s' = a \oplus b$ with a, b constants, we have that $(a \oplus b, id)$ is a minimal and complete set of E-variants for s'. Then the E-unification question of $s =_E s'$ can be answered by considering the following combination of E-variants. First, $0 =_B a \oplus b$ has no solution. Second, $X \oplus Y =_B a \oplus b$ has two solutions $\{X \mapsto a, Y \mapsto b\}$ and $\{X \mapsto b, Y \mapsto a\}$. Third, $Z =_B a \oplus b$ has only one solution $\{Z \mapsto a \oplus b\}$ so we get four solutions by combining it with the one in the variants, namely $\{X \mapsto 0, Y \mapsto a \oplus b\}$, $\{X \mapsto a \oplus b, Y \mapsto 0\}$, $\{X \mapsto a \oplus b \oplus U, Y \mapsto U\}$, and $\{X \mapsto U, Y \mapsto a \oplus b \oplus U\}$. Fourth, $Z_1 \oplus Z_2 =_B a \oplus b$ has the two solutions $\{Z_1 \mapsto a, Z_2 \mapsto b\}$ and $\{Z_1 \mapsto b, Z_2 \mapsto a\}$ and by combination we get $\{X \mapsto U \oplus a, Y \mapsto U \oplus b\}$) and $\{X \mapsto U \oplus b, Y \mapsto U \oplus a\}$).

References

1. H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *RTA*, LNCS 3467:294–307. Springer, 2005.
2. S. Escobar, J. Meseguer, and R. Sasse. Effectively checking the finite variant property. In *RTA*, LNCS to appear, Springer, 2008.
3. S. Escobar, J. Meseguer, and R. Sasse. Variant narrowing and equational unification. In *WRLA*, ENTCS to appear, Elsevier, 2008.
4. J.-M. Hullot. Canonical forms and unification. In *CADE*, LNCS 87:318–334. Springer, 1980.
5. J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *ICALP*, LNCS 154:361–373. Springer, 1983.
6. J. Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.
7. TeReSe, editor. *Term Rewriting Systems.* Cambridge University Press, Cambridge, 2003.
8. E. Viola. E-unifiability via narrowing. In *ICTCS*, LNCS 2202:426–438. Springer, 2001.
9. P. Viry. Equational rules for rewriting logic. *Theor. Comput. Sci.*, 285(2):487–517, 2002.

First-Order Unification Using Variable-Free Relational Algebra*

Emilio Jesús Gallego Arias[1], James Lipton[2], Julio Mariño[1], and Pablo Nogueira[1]

[1] Universidad Politécnica de Madrid
{egallego,xmc,pablo}@babel.ls.fi.upm.es
[2] Wesleyan University
jlipton@wesleyan.edu

Abstract We present a new framework for the representation and resolution of first-order unification problems and their abstract syntax in a variable-free relational formalism which is a variant of Tarski-Givant relational algebra and Freyd's allegories restricted to the fragment necessary to compile and execute logic programs. A decision procedure for validity of relational terms is developed, which corresponds to solving the original unification problem. The decision procedure is presented as a conditional relational-term rewriting system. A more efficient version can be obtained by tailoring certain rewriting mechanisms. There are advantages over classical unification approaches. First, inconvenient and underspecified meta-logical procedures (name clashes, substitution, etc) are captured algebraically within the framework. Second, interesting algebraic properties usually living in the meta-level spring up. Third, other unification problems are seamlessly accommodated, for instance, unification for terms with a variable-restriction operator.

1 Introduction

Tarski and Givant [27] observe that binary relations equipped with projection operators faithfully capture all of classical first-order logic. Freyd and Scedrov [12] show that so-called tabular allegories satisfying certain conditions faithfully capture all of higher-order intuitionistic logic. Both of these formalisms effectively eliminate logical variables in different ways and can provide an algebraic semantics to logic programming. A restricted variant of these calculi [4,18] has been used as an alternative means of logic-program transformation and execution by direct rewriting of relational, variable-free representations. The variant is restricted to carve out an efficient *executable* fragment. The relational representation provides a new abstract syntax for logic programs and a formal treatment of logic variables and unification, where expensive meta-logical procedures (name

* Work supported by CAM grants 3060/2006 (European Social Fund), S-0505/TIC/0407 (PROMESAS), and MEC grant TIN2006-15660-C02-02 (DESAFIOS).

clashes, substitution, etc) are now captured by explicit algebraic manipulation of relations.

However, in the aforementioned work unification is incorporated metalogically as a black box via intersection of relational terms, with execution details left unspecified. It is only shown that unification is sound with respect to the chosen representation. In this paper we present a unification algorithm that acts directly on the representation via rewriting. There are advantages over classical unification approaches. First, interesting algebraic properties usually living in the meta-level spring up. Second, other unification problems are seamlessly accommodated because the relational framework can host the whole set of formulas and theories over the Herbrand universe and therefore making small additions is easy and non-disruptive. For example, we can incorporate existential quantification inside equality formulas which gives rises to unification between terms with a variable restriction operator ν that allows us to formalise renaming apart. To help the reader understand our algorithm we shall draw analogies with the classic non-deterministic one described in [22] and referred to here as ND. This algorithm uses multisets of equations but an efficient, deterministic version with complexity $O(n \log n)$ is possible. We believe that, with minimal modifications (tailoring the rewriting mechanism), an efficient version of our algorithm with similar complexity is possible.

The structure of the paper is as follows. Section 2 contains a summary of the relational translation of Herbrand terms and the unification problem. Section 3 defines the concept of solved form. Section 4 formally defines and valides our decision procedure and compares it with a classical unification algorithm. Section 5 contains the restriction example. Section 6 discusses related work and Sect. 7 concludes and discusses future work.

2 Term Encoding in Relational Algebra

2.1 Relational Signature and Theories

Logic programs are represented in [4] as relations in two relational theories, namely, Distributive Relational Algebra (**DRA**, Fig. 1) and $\mathsf{Rel}\Sigma$ (Fig. 2), the latter a specialised theory which depends on the signature Σ of the logic program and captures algebraically Clark's Equality Theory [19], the domain closure axiom (shown in the last line: every term is either a constant or an application of some term former to terms), the occurs check (fifth line: no proper subterm of a term may be equal to the whole term), and open sequences of terms (second line: formalizes projection operations, sixth line: rules out the use of open sequences in term forming operations). The equation $RS \cap T \subseteq (R \cap TS^{\circ})S$ is the *modular* law (recall $X \subseteq Y$ means $X \cap Y = X$). The fixed point equation (**fp**) of **DRA** is irrelevant here and has been included only for completeness. The following paragraphs provide more detail and assume a permutative convention of symbols, i.e., f, g are different and so are i, j, etc. The first-order signature of a logic program $\Sigma = \langle \mathcal{C}_{\Sigma}, \mathcal{F}_{\Sigma} \rangle$ is given by $\mathcal{C}_{\Sigma}$, the set of constant symbols, and $\mathcal{F}_{\Sigma}$, the set of term formers or functors. Function $\alpha : \mathcal{F}_{\Sigma} \to \mathbb{N}$ returns the arity

$$\begin{gathered}
R \cap R = R \qquad R \cap S = S \cap R \qquad R \cap (S \cap T) = (R \cap S) \cap T \\
R\,id = R \qquad R\mathbf{0} = \mathbf{0} \qquad \mathbf{0} \subseteq R \subseteq \mathbf{1} \\
R \cup R = R \qquad R \cup S = S \cup R \qquad R \cup (S \cup T) = (R \cup S) \cup T \\
R \cup (S \cap R) = R = (R \cup S) \cap R \\
R(S \cup T) = RS \cup RT \qquad\qquad (S \cup T)R = SR \cup TR \\
R \cap (S \cup T) = (R \cap S) \cup (R \cap T) \\
(R \cup S)^\circ = R^\circ \cup S^\circ \qquad (R \cap S)^\circ = S^\circ \cap R^\circ \\
R^{\circ\circ} = R \qquad (RS)^\circ = S^\circ R^\circ \\
R(S \cap T) \subseteq RS \cap RT \qquad RS \cap T \subseteq (R \cap TS^\circ)S \\
id \cup di = \mathbf{1} \qquad id \cap di = \mathbf{0} \qquad \mathbf{fp}x.\mathcal{E}(x) = \mathcal{E}(\mathbf{fp}x.\mathcal{E}(x)) \\
T \cap RS = R(R^\circ T \cap S) \cap T
\end{gathered}$$

Figure 1: The equational theory **DRA**.

$$\begin{gathered}
\mathbf{1}(a,a)\mathbf{1} = \mathbf{1} \qquad (a,a)R(a,a) = (a,a) \cap R \qquad (a,a) \subseteq id \\
hd(hd)^\circ \cap tl(tl)^\circ \subseteq id \qquad (hd)^\circ hd = (tl)^\circ tl = id \qquad (hd)^\circ tl = \mathbf{1} \\
id_f \overset{\text{def}}{=} \textstyle\bigcap_{1 \leq i \leq n} f_i^n (f_i^n)^\circ \subseteq id \qquad (f_j^n)^\circ f_i^n = \mathbf{1} \quad (i \neq j) \\
(f_i^n)^\circ f_i^n = id \qquad (f_i^n)^\circ g_j^m = \mathbf{0} \\
(f_1)_{i_1}^{n_1} (f_2)_{i_2}^{n_2} \cdots (f_k)_{i_k}^{n_k} \ \cap \ id = \mathbf{0} \\
hd^\circ f_i^n = 0 = tl^\circ f_i^n \qquad f_i^n hd = 0 = f_i^n tl \qquad hd \cap id = 0 = tl \cap id \\
id = \bigcup\{(a,a) : a \in \mathcal{C}_\Sigma\} \ \cup \ \bigcup\{id_f : f \in \mathcal{F}_\Sigma\}
\end{gathered}$$

Figure 2: The equational theory $\mathsf{Rel}\Sigma$.

of its functor argument. The set of logic-program variables is $\mathcal{X}$ and $x_i \in \mathcal{X}$. We write $\mathcal{T}_\Sigma(\mathcal{X})$ for the set of open terms over Σ. The set of open sequences $\mathcal{T}_\Sigma^+(\mathcal{X})$ over $\mathcal{T}_\Sigma$ is defined with the addition to the signature of a right-associative list-cons-like operator. We write $[t_1, [t_2, \ldots, [t_n, \boldsymbol{x}]]]$, or alternatively $[t_1, \ldots, t_n]\boldsymbol{x}$ with abbreviation $\boldsymbol{tx}$, for an open sequence of $t_1, \ldots, t_n$ terms where $\boldsymbol{x}$ is a variable standing for an arbitrary open sequence. Thus, the open sequence $[t_1, t_2]\boldsymbol{x}$ denotes all term sequences beginning with t_1, followed by t_2, and followed by an arbitrary term sequence.

The relational language R is built from a countable set of relation variables $R, S, T, \ldots \in R_{var}$ (not to be confused with first-order logic-variables), a set of relational constants R_Σ build from Σ, and a fixed set of relational constants and operators detailed below. Let us begin with R_Σ. Each constant $a \in \mathcal{C}_\Sigma$ defines a constant $(a,a) \in \mathsf{R}_\Sigma$. The set-theoretic interpretation is a singleton relation, that is, $[\![(a,a)]\!] = \{(a,a)\}$. Each functor $f \in \mathcal{F}_\Sigma$ of arity n defines n constants $f_1^n, \ldots, f_n^n$ in R_Σ. Their set-theoretic interpretation is the relation of the term former with its ith argument, that is, $[\![f_i^n]\!] = \{(f(\ldots, t_i, \ldots),\ t_i) \mid t_i \in \mathcal{T}_\Sigma\}$. Summarising:

$$\mathsf{R}_\Sigma \ = \ \{f_i^n \mid f \in \mathcal{F}_\Sigma,\ \alpha(f) = n,\ 1 \leq i \leq n\} \cup \{(a,a) \mid a \in \mathcal{C}_\Sigma\}$$

The full relational language R is given by the following grammar:

$$\begin{aligned} \mathsf{R}_{atom} &::= \mathsf{R}_{var} \mid \mathsf{R}_{\Sigma} \mid id \mid di \mid \mathbf{1} \mid \mathbf{0} \mid hd \mid tl \\ \mathsf{R} &::= \mathsf{R}_{atom} \ \mid\ \mathsf{R}^{\circ} \ \mid\ \mathsf{R} \cup \mathsf{R} \ \mid\ \mathsf{R} \cap \mathsf{R} \ \mid\ \mathsf{RR} \ \mid\ \mathbf{fp}\ \mathsf{R}_{var} \ .\ \mathsf{R} \end{aligned}$$

The constants $\mathbf{1}, \mathbf{0}, id, di$ respectively denote universal relation, empty relation, identity relation, and identity's complement. Operators $\cup$ and $\cap$ represent union and intersection whereas juxtaposition RR represents relation composition. R° is the converse of R, i.e., the relation obtained by swapping domain and codomain. Set-theoretically, R's domain is $\{x|(x, \_) \in R\}$ and its codomain is $\{x|(\_, x) \in R\}$. The constants hd and tl denote the relation of an open sequence with its head and tail. Using the notation R^k as shorthand for R composed with itself k times, we define the ith projection relation $P_i = tl^{(i-1)}hd$, where $i \geq 1$, and the ith partial projection relation $Q_i = \bigcap_{j \neq i} P_j(P_j)^{\circ}$.

2.2 Translation and Unification

Every first-order formula over $\mathcal{H}$ (the Herbrand universe over Σ with equality) that is built from conjunction and existential quantification can be represented with a relation expression. We refer the reader to [18] for full details.

The translation K from terms $t \in \mathcal{T}_{\Sigma}(\mathcal{X})$ to terms in R is define in a way that every ground instance of t is in $[\![K(t)]\!]$. We also define the set of relational terms in K's image inductively and call them $\mathbf{U}$-terms. The most general unifier of two terms $t_1, t_2 \in \mathcal{T}_{\Sigma}(\mathcal{X})$ is then represented by $K(t_1) \cap K(t_2)$.

Definition 1. *$K : \mathcal{T}_{\Sigma}(\mathcal{X}) \to \mathsf{R}$ is defined by induction on $\mathcal{T}_{\Sigma}(\mathcal{X})$ terms:*

$$\begin{aligned} K(a) &= (a, a)\mathbf{1} \\ K(x_i) &= (P_i)^{\circ} \\ K(f(t_1, \ldots, t_n)) &= \bigcap_{i \leq n} f_i^n K(t_i) \end{aligned}$$

For example, $K(f(x_1, g(a, x_2)))$ yields $f_1^2 P_1^{\circ} \cap f_2^2(g_1^2(a, a)\mathbf{1} \cap g_2^2 P_2^{\circ})$.

Lemma 1. *For all terms t_1, t_2 in $\mathcal{T}_{\Sigma}(\mathcal{X})$ with variables $x_1, \ldots, x_n$ and an arbitrary open sequence $\boldsymbol{u}$:*

$$(t_1\sigma, [a_1, \ldots, a_n]\boldsymbol{u}) \in [\![K(t_1) \cap K(t_2)]\!] \quad \textit{iff} \quad \mathcal{H} \models t_1\sigma = t_2\sigma$$

for all grounding substitutions $\sigma = a_1, \ldots, a_n / x_1, \ldots, x_n$.

This result is proved in [18]. In words, given an open term t all pairs relating ground instances of t with grounding elements $a_1, \ldots, a_n$ belong to $[\![K(t)]\!]$.

Definition 2 (U-terms). *The set of* $\mathbf{U}$*-terms (relational terms in K's image) is defined inductively:*

– $\mathbf{0} \in \mathbf{U}$, *$(a, a)\mathbf{1} \in \mathbf{U}$ for every $(a, a) \in \mathsf{R}_{\Sigma}$, and $P_i^{\circ} \in \mathbf{U}$ for every $i \in \mathbb{N}$.*

- *If* $R \in \mathbf{U}$ *then* $f_i^n R \in \mathbf{U}$ *for every* $f_i^n \in \mathsf{R}_\Sigma$.
- *If* $R_1, \ldots, R_n \in \mathbf{U}$ *then* $R_1 \cap \cdots \cap R_n \in \mathbf{U}$.

The unification problem in this relational setting is to decide, given terms $t_1, t_2 \in \mathcal{T}_\Sigma(\mathcal{X})$, whether $[\![K(t_1) \cap K(t_2)]\!] = \emptyset$, more precisely:

$$[\![K(t_1) \cap K(t_2)]\!] = \begin{cases} [\![K(t_1\sigma)]\!] & \text{where } \sigma = \mathbf{mgu}(t_1, t_2) \text{ if exists} \\ \emptyset & \text{otherwise} \end{cases} \tag{1}$$

3 Solved Forms for U-Terms

It is standard practice in decision problems to use a *solved form* which has a trivial decision procedure. The decision problem is then reduced to developing a method for producing solved forms [5]. At the core of our decision procedure is the theory $\mathsf{Rel}\Sigma$, for it provides a strong enough axiomatization of the original underlying algebra of finite trees.

Given the term $T = R \cap S$, we informally say that R is constrained by S in T and vice versa. We also say that R is obtained from $R \cap S$ by dropping the constraint S.

Definition 3 (P-constraint completeness). *A term R is* P-constraint complete *or $\Xi(R)$ iff for all subterms t_1 and t_2 of R of the form:*

$$t_1 = P_i^\circ \cap R_1 \cap \cdots \cap R_m$$
$$t_2 = P_j^\circ \cap S_1 \cap \cdots \cap S_n$$

if $i = j$ then the equality $t_1 = t_2$ holds modulo $\cap$-commutativity.

This formally captures the notion that every P_i° appearing in a term must have the same set of constraints. Some examples of P-constraint-complete terms are: $P_1^\circ \cap P_2^\circ$, $f_1^2(P_1^\circ \cap R) \cap g_1^2(R \cap P_1^\circ)$ and $f_1^2(P_1^\circ \cap R) \cap g_1^2(S \cap P_2^\circ)$. Some non-P-constraint-complete terms are: $P_1^\circ \cap f_1^1 P_1^\circ$ and $f_1^2(P_1^\circ \cap R) \cap g_1^2(S \cap P_1^\circ)$.

Definition 4 (Indexed P-constraint completeness). *A term R is* P-constraint complete on i *or $\Xi_i(R)$ iff for all subterms t_1, t_2 of R of the form:*

$$t_1 = P_j^\circ \cap R_1 \cap \cdots \cap R_m$$
$$t_2 = P_k^\circ \cap S_1 \cap \cdots \cap S_n$$

if $i = j = k$ then the equality $t_1 = t_2$ holds modulo $\cap$-commutativity.

Definition 5 (Solved form). $\mathbf{U}$*-terms in solved form are inductively defined for all a, i, j, f, g as:*

R			*for* $R \in \{(a,a)\mathbf{1}, P_i^\circ, \mathbf{0}\}$
R	$\cap$	S	*for* $R, S \in \{(a,a)\mathbf{1}, P_i^\circ\}$
$(a,a)\mathbf{1}$	$\cap$	$f_i^j R$	
$f_i^n R$			*if* R *in solved form.*
P_i°	$\cap$	$f_j^n R$	*if* R *in solved form.*
$f_i^m R$	$\cap$	$g_j^n S$	*if* $f \neq g$.
$f_i^n R$	$\cap$	$f_j^n S$	*if* R, S *in solved form and* $\Xi(R \cap S)$.
R_1	$\cap \cdots \cap$	R_n	*if every pair* $R_i \cap R_j, i \neq j$ *in solved form.*

If a term t is in solved form we say $solved(t)$.

Definition 6 (Unsolved form). *Unsolved forms are logical negations of solved forms, that is, $unsolved(t) = \neg solved(t)$.*

$f_i^n R$			*if R not in solved form.*
P_i°	$\cap$	$f_j^n R$	*if R not in solved form.*
$f_i^n R$	$\cap$	$f_i^n S$	
$f_i^n R$	$\cap$	$f_j^n S$	*if $\neg \Xi(R \cap S)$ or any of R, S not in solved form.*
R_1	$\cap \cdots \cap$	R_n	*if $R_i \cap R_j$ not in solved form for some $i, j, i \neq j$.*

Lemma 2. *Let $\mathbf{U_S} = \{x \in \mathbf{U} \mid solved(x)\}$ and $\mathbf{U_N} = \{x \in \mathbf{U} \mid \neg solved(x)\}$. By definition, $\mathbf{U_S}$ and $\mathbf{U_N}$ partition $\mathbf{U}$.*

The decision procedure for solved forms is an exhaustive check for incompatible intersections.

Lemma 3 (Validity of solved forms). *For any term $t \in \mathbf{U_S}$, we can always decide whether $t = \mathbf{0}$ or, what is the same, whether $[\![t]\!] = \emptyset$.*

Proof. By cases on t:

- R and $R \cap S$ for $R, S \in \{(a, a)\mathbf{1}, P_i^\circ\}$: Follows directly by term semantics.
- $(a, a)\mathbf{1} \cap f_i^j R$: Always $\mathbf{0}$, because the relation domains are disjoint.
- $f_i^n R$: We check R for validity.
- $P_i^\circ \cap f_j^n R$: We check R for validity.
- $f_i^m R \cap g_j^n S$: Always $\mathbf{0}$, because relation domains are disjoint ($f \neq g$).
- $f_i^n R \cap f_j^n S$: We check R and S for validity. This check is enough because no term $f_i^n R \cap f_i^n S$ is in solved form. Consequently, the domain of the resulting relations is known. Given that $\Xi(R \cap S)$ holds, the codomain is also known, as every projection P_i° into the codomain shares the same set of constraints, so validity of the codomain is effectively reduced to validity of constraints.
- $R_1 \cap \cdots \cap R_n$: Every pair $R_i \cap R_j$ with $i \neq j$ is in solved form, thus the term's validity depends on the pairs' validity.

4 The Decision Procedure

This section defines a decision procedure for computing normal forms of $\mathbf{U}$-terms. The core of the procedure consists of two term rewriting systems whose effect can be explained by analogy with the classic non-deterministic algorithm (ND) [22]. The first rewriting system (system $\to_{\mathcal{L}}$ defined in Sec. 4.2) performs what we call left-factoring, analogous to generation of new equations from a common root term in ND, which is now algebraically understood as the distribution of composition over intersection. The following diagram illustrates ($\mathcal{E}$ stands for a multiset of equations):

$$\{f(t_1, ..., t_n) = f(u_1, ..., u_n), \mathcal{E}\} \ \Rightarrow\ \{t_1 = u_1, ..., t_n = u_n, \mathcal{E}\}$$
$$f_i^n R \cap f_i^n S \to_{\mathcal{L}} f_i^n (R \cap S)$$

The other step in ND, equation elimination, is now algebraically understood as constraint propagation (system $\rightarrow_{\mathcal{R}}$ defined Sec. 4.4):

$$\{x = t, \mathcal{E}\} \;\Rightarrow\; \{\mathcal{E}[t/x]\} \qquad \text{when } x \notin t$$
$$F(P_i \cap R) \cap G(P_i \cap S) \rightarrow_{\mathcal{R}} F(P_i \cap R \cap S) \cap G(P_i \cap S \cap R)$$

System $\rightarrow_{\mathcal{R}}$ is conditional on what we call *functorial compatibility*, a novel way of performing occurs checks which was motivated by $\mathsf{Rel}\Sigma$'s axiom $f_i^n \cap id = \mathbf{0}$. The two rewriting systems are carefully composed with the help of a constraint-propagation one (system $\rightarrow_{\mathcal{S}}$ defined in Sect. 4.3) to guarantee termination.

4.1 Rewriting Preliminaries

The representation of $\mathbf{U}$-terms in our rewriting systems is given by the following term-forming operations: $\mathsf{c} : \mathcal{C}_\Sigma \rightarrow \mathbf{U},\ \mathsf{t} : (\mathcal{F}_\Sigma \times \mathbb{N} \times \mathbb{N}) \rightarrow \mathbf{U},\ \mathsf{P} : \mathbb{N} \rightarrow \mathbf{U},\ \odot : (\mathbf{U}_1 \times \cdots \times \mathbf{U}_n) \rightarrow \mathbf{U}$, and $\cap : (\mathbf{U}_1 \times \cdots \times \mathbf{U}_n) \rightarrow \mathbf{U}$, with $n \geq 2$ and n-ary $\odot$ and $\cap$. In addition to the above ground representation, we define patterns of $\mathbf{U}$-terms in a standard way using a set of variables. Let i, j, k etc, range over $\mathbb{N}$. Let a, b, c etc, range over $\mathcal{C}_\Sigma$. Let f, g, h etc, range over $\mathcal{F}_\Sigma$. Let R, S, T etc, range over $\mathbf{U}$-terms. We write $(a, a)\mathbf{1}$ for $\mathsf{c}(a)$, write f_i^j for $\mathsf{t}(f, i, j)$, write P_i° for $\mathsf{P}(i)$, write $R_1 \ldots R_n$ for $\odot(R_1, \ldots, R_n)$, and write $R_1 \cap \cdots \cap R_n$ for $\cap(R_1, \ldots, R_n)$.

Rewrite rules are of the form $\rho : l \rightarrow r$ with ρ the rule's name, l and r patterns, and l not a variable. Conditional rewrite rules are of the form $\rho : l \rightarrow r \Leftarrow C$ belonging to type III CTRS [16] (see Def. 9). We write $\rightarrow^!$ for the normalization relation derived from a terminating and confluent (convergent) relation $\rightarrow$. We write $\circ$ for composition of rewriting relations. We use $\equiv$ for syntactic identity (modulo AC).

We use A and AC rewriting for $\odot$ and $\cap$. More precisely, we define the equational theories $A_\odot = \{R(ST) = (RS)T\}$ and $AC_\cap = \{R \cap (S \cap T) = (T \cap S) \cap T,\ R \cap S = S \cap R\}$. The AC rewriting used is described in [8, p577–581]: associative term formers are flattened and rewrite rules are extended with dummy variables to take into account the arity of term-forming operations. Matching efficiency can be improved by using ordered rewriting.

A rewrite rule $\rho : l \rightarrow r$ matches a term t iff $l\sigma = t$ modulo $A_\odot$ and $AC_\cap$. We allow subterm matching: if there exists a position p such that $l\sigma = t_{|p}$ (modulo previous AC), then t reduces to $t[r\sigma]_{|p}$. When matching succeeds, t rewrites to $r\sigma$. Importantly, we also allow matching over *functorial variables* which are compositions of f_i^n terms. We use F, G, H etc, for functorial variables. The expression *length*(F) delivers the length of functorial variable F. For example, the term $r_1^2 s_2^2((a, a)\mathbf{1} \cap (b, b)\mathbf{1} \cap t_1^1 P_1^\circ)$ matches $F(R \cap GS)$ with $\sigma = \{F \mapsto r_1^2 s_2^2,\ R \mapsto (a, a)\mathbf{1} \cap (b, b)\mathbf{1},\ G \mapsto t_1^1,\ S \mapsto P_1^\circ\}$. Matching over functorial variables is a sort of specialized list-matching.

4.2 Left-Factoring Rewriting System

Definition 7. *The rewriting system $\mathcal{L}$ consists of the rewrite rule:*

$$f_i^n R \cap f_i^n S \rightarrow_{\mathcal{L}} f_i^n (R \cap S)$$

Lemma 4. *$\mathcal{L}$ is sound.*

Proof. Soundness is a consequence of the following equation:

$$RS \cap RT = R(S \cap T) \tag{2}$$

which holds in **DRA** when R is functional, and every f_i^n is:

$$R(S \cap T) \supseteq_{\textbf{[by modular law]}} R(S \cap R^\circ RT) \supseteq_{\textbf{[by } R^\circ R \subseteq id\textbf{]}} RS \cap RT$$

Conversely, $RS \cap RT \supseteq R(S \cap T)$ by **DRA**.

Lemma 5. *$\mathcal{L}$ is terminating and confluent.*

Proof. To prove termination it suffices to give a lexicographic path ordering on terms [7]. The ordering is $\cap \succ \odot$. The system has no critical pairs, so it is locally confluent, local confluence plus termination implies confluence [17].

4.3 Split-Rewriting System

Definition 8. *The rewriting system $\mathcal{S}$ consists of the rewrite rule:*

$$F(R \cap G(P_i^\circ \cap S)) \rightarrow_{\mathcal{S}} FR \cap FG(P_i^\circ \cap S)$$

Lemma 6. *$\mathcal{S}$ is sound, terminating and confluent.*

Proof. Soundness and closure properties are immediate from (2). Termination is proven giving the lexicographic path ordering $\odot \succ \cap$ and confluence follows from the fact that $\rightarrow_{\mathcal{S}_i}$ is terminating and locally confluent.

We will also use a parametrized version of $\mathcal{S}$, written $\mathcal{S}_i$ where i is fixed.

4.4 Constraint-Propagation Rewriting System

This purpose of this system is to propagate constraints over P_i° terms. Constraint propagation has two main technical difficulties, namely occurs check, which in our setting might lead to infinite rewriting, and propagation of constraints before checking for term clashes. Both problems can be dealt with using a decidable notion of functorial compatibility:

Definition 9. *The convergent rewriting relation $\rightarrow_\Delta$ is defined as:*

$$(f_i^n)^\circ f_i^n \rightarrow_\Delta id \qquad (f_i^n)^\circ g_j^m \rightarrow_\Delta \mathbf{0} \qquad (f_i^n)^\circ f_j^n \rightarrow_\Delta \mathbf{1} \qquad id f_i^n \rightarrow_\Delta f_i^n$$
$$(f_i^n)^\circ \mathbf{1} \rightarrow_\Delta \mathbf{1} \qquad \mathbf{1} f_i^n \rightarrow_\Delta \mathbf{1} \qquad (f_i^n)^\circ \mathbf{0} \rightarrow_\Delta \mathbf{0} \qquad \mathbf{0} f_i^n \rightarrow_\Delta \mathbf{0}$$

Definition 10 (Functorial delta). *Given functorials F and G, we define $\Delta(F, G)$ as follows:*

$$\Delta(F,G) = S, \quad F^\circ G \rightarrow_\Delta^! S \quad \text{if } length(G) \geq length(F)$$
$$\Delta(F,G) = S, \quad G^\circ F \rightarrow_\Delta^! S \quad \text{if } length(G) < length(F)$$

Lemma 7.

$$\Delta(F,G) = \begin{cases} \mathbf{0} & \textit{if } [\![F \cap G]\!] = \emptyset \\ id & \textit{if } F \equiv G. \\ S & \textit{if } G \equiv FS \textit{ and } \Delta(F,G) = id \\ \mathbf{1} & \textit{otherwise.} \end{cases}$$

Proof. By induction on functorial terms.

Definition 11 (Syntactic difference). *The syntactic difference $\Theta(R_1 \cap \cdots \cap R_m, S_1 \cap \cdots \cap S_n)$ between two arbitrary-length intersection of terms is defined as the term $\bigcap_{i \in D} S_i$, such that $i \in D$ iff there is no term R_j such that $S_i \equiv R_j$.*

Lemma 8. $R \cap \Theta(R,S) \equiv S \cap \Theta(S,R)$

Proof. By induction on the length of R.

Definition 12. *The rewriting system $\mathcal{R}$ consists of the rewrite rules:*

$R0 : P_i^\circ \cap F(P_i^\circ \cap S) \to_{\mathcal{R}} \mathbf{0}$
$R1 : F(P_i^\circ \cap R) \cap G(P_i^\circ \cap S) \to_{\mathcal{R}} \mathbf{0} \quad \Leftarrow \Delta(F,G) = \mathbf{0} \vee \Delta(F,G) = f_i^n \dots g_j^m$
$R2 : F(P_i^\circ \cap R) \cap G(P_i^\circ \cap S) \to_{\mathcal{R}} F(P_i^\circ \cap R \cap \Theta(R,S)) \cap G(P_i^\circ \cap S \cap \Theta(S,R))$
$\qquad \Leftarrow (\Theta(R,S) \neq \{\} \vee \Theta(S,R) \neq \{\}) \wedge (\Delta(F,G) = \mathbf{1} \vee \Delta(F,G) = id)$

Notice Θ helps us deal conveniently with the equivalence of terms $R \cap S$ and $S \cap R$. We will also use a parametrized version of $\mathcal{R}$, written $\mathcal{R}_i$, where i is fixed.

Lemma 9. *$\mathcal{R}$ is sound.*

Proof. $R0$ is sound beacuse every term matching the left hand side can be factored into a term of the form $id \cap f_i^n \cdots g_j^m$ using $RP_i^\circ \cap SP_i^\circ = (R \cap S)P_i^\circ$ which is a version of Eq. (3) below. $R1$ is sound for two reasons:

1. If $\Delta(F,G) = \mathbf{0}$ then F and G are incompatible and the left hand side rewrites to $\mathbf{0}$ by left-factoring.
2. If $\Delta(F,G) = f_i^n \cdots g_j^m$ then there is a common prefix F' of F and G such that $F \cap G = F'(id \cap f_i^n \cdots g_j^m)$ and the right-hand-side rewrites to $\mathbf{0}$ by $\mathsf{Rel}\Sigma$'s occurs-check axiom.

That $R2$ is sound follows from the equation:

$$F(P_i^\circ \cap R) \cap GP_i^\circ = F(P_i^\circ \cap R) \cap G(P_i^\circ \cap R) \tag{3}$$

Recall that a relation is injective when $RR^\circ \subseteq id$. Recall Lemma 4 which states that $RT \cap ST = (R \cap S)T$ for T injective. Using these facts we prove the $\subseteq$ direction:

$F(P_i^\circ \cap R) \cap GP_i^\circ =_{\textbf{[left+right factoring]}} (F \cap G)P_i^\circ \cap FR \subseteq_{\textbf{[modular law]}}$
$(F \cap G)((F^\circ \cap G^\circ)FR \cap P_i^\circ) \subseteq_{\textbf{[by } (F^\circ \cap G^\circ)F \subseteq F^\circ F = id\textbf{]}} (F \cap G)(R \cap P_i^\circ)$
$=_{\textbf{[injectivity of } (R \cap P_i^\circ)\textbf{]}} F(R \cap P_i^\circ) \cap G(R \cap P_i^\circ)$

The $\supseteq$ direction is easier:

$$F(P_i^\circ \cap R) \cap GP_i^\circ \supseteq_{\textbf{[monotonicity of } \cap\textbf{]}} F(P_i^\circ \cap R) \cap G(P_i^\circ \cap R)$$

The following example illustrates why the compatibility check is needed to ensure termination. Take the term $P_1^\circ \cap f_1^1(P_1^\circ \cap R)$. If we propagate restrictions by orienting (3) we get an infinite rewrite: $P_1^\circ \cap f_1^1(P_1^\circ \cap R) \to P_1^\circ \cap f_1^1(R \cap P_1^\circ \cap f_1^1(P_1^\circ \cap R)) \to P_1^\circ \cap f_1^1(R \cap P_1^\circ \cap f_1^1(P_1^\circ \cap R \cap f_1^1(P_1^\circ \cap R))) \ldots$

Lemma 10. *$\mathcal{R}$ terminates.*

Proof. Rules $R0$ and $R1$ rewrite to $\mathbf{0}$. For rule $R2$ we define a well-founded order that contains the relation induced by $R2$:

Definition 13. *Let $\succ_C$ be the order relation:*

$$F(P_i^\circ \cap R) \cap G(P_i^\circ \cap S) \;\succ_C\; F(P_i^\circ \cap R \cap \Theta(R,S)) \cap G(P_i^\circ \cap S \cap \Theta(S,R))$$

where $\Theta(R,S) \neq \{\}$ and $\Theta(S,R) \neq \{\}$.

Lemma 11. *$\succ_C$ is well-founded.*

Proof. Suppose $\succ_C$ is not well-founded. There exists an infinite chain with link $F(P_i^\circ \cap R \cap \Theta(R,S)) \cap G(P_i^\circ \cap S \cap \Theta(S,R)) \;\succ_C\; F(P_i^\circ \cap R \cap \Theta(R,S) \cap \Theta(R \cap \Theta(R,S), S \cap \Theta(S,R)))$ in which $\Theta(R \cap \Theta(R,S), S \cap \Theta(S,R)) \neq \{\}$. By Lemma 8, $R \cap \Theta(R,S) = S \cap \Theta(S,R)$, therefore $\Theta(T,T) \neq \{\}$ which is a contradiction.

Lemma 12. *$\mathcal{R}$ is confluent (modulo commutativity).*

Proof. We give the same lexicographic path ordering as Lemma 5 and $\mathcal{R}$ has no overlapping rules.

4.5 The Algorithm

Unfortunately, a naïve application of the previous rewriting rules does not necessarily reach a solved form. Take for example the term $r_1^2(P_1^\circ \cap s_1^1(P_2^\circ \cap (a,a)\mathbf{1})) \cap r_2^2(P_2^\circ \cap s_1^1(P_1^\circ \cap (b,b)\mathbf{1}))$. If we apply the rewriting strategy $\to_{\mathcal{S}}^! \circ \to_{\mathcal{R}}^! \circ \to_{\mathcal{L}}^!$ we do not reach a solved form because $\to_{\mathcal{S}}^!$ destroys constraints over P_1°, impeding the constraint propagation step to work properly. One solution is to complete the constraints for each P_i° one at a time.

Lemma 13. *Given a rewriting $t \to_{\mathcal{S}_i}^! t'$, every P_i° in t' occurs at* top level, *i.e., t' has the form:*

$$P_i^\circ \cap F(P_i^\circ \cap R) \cap \cdots \cap G(P_i^\circ \cap S) \cap \ldots$$

Proof. If the P_i° term occurs deeper in the term, t' has the form:

$$F(R \cap G(S \cap \cdots H(P_i^\circ \cap T))) \cap \ldots$$

which is not a normal form for $\to_{\mathcal{S}_i}^!$.

Definition 14 (Individual constraint propagation). *The rewrite relation $\to_{\mathcal{U}_i}$ parametrized on i is defined as:*

$$\to_{\mathcal{L}}^! \circ \to_{\mathcal{S}_i}^! \circ \to_{\mathcal{R}_i}^! \circ \to_{\mathcal{L}}^!$$

$dep(R \cap S) = dep(R) \cup dep(S)$	$dep'(P_j^\circ) = \{j\}$
$dep(f_i^n R) = dep'(R)$	$dep'(R \cap S) = dep'(R) \cup dep'(S)$
	$dep'(f_i^n R) = dep'(R)$

Figure 3: Definition of $dep : \mathbf{U} \to \mathcal{P}(\mathbb{N})$

Lemma 14. *$\Xi_i(t')$ holds for every term t and reduction $t \to_{\mathcal{U}_i} t'$.*

Proof. Assume a term t exists such that $t \to_{\mathcal{U}_i} t'$ and $\neg\Xi_i(t')$. There are subterms of t' of the form $P_i^\circ \cap R$ and $P_i^\circ \cap S$, with $\Theta(R, S) \neq \{\}$ or $\Theta(S, R) \neq \{\}$. Let $t \to_{\mathcal{L}}^! u$, then no subterm of the form $FR \cap FS$ exists in u. Let $u \to_{\mathcal{S}_i}^! v$, then every P_i° in v is of the form described in Lemma 13. Let $v \to_{\mathcal{R}_i}^! w$, then $\Xi_i(w)$ because w is a normal form of $\to_{\mathcal{R}_i}$ and $P_i^\circ \cap R$ and $P_i^\circ \cap S$ with $\Theta(R, S) \neq \{\}$ and $\Theta(S, R) \neq \{\}$ cannot occur at the top-level, and every P_i° is at the top level. Let finally $w \to_{\mathcal{L}}^! t'$. For $\to_{\mathcal{L}}^!$ to break P-constraint completeness, w must have a term of the form $FP_i^\circ \cap FR$ such that after $\to_{\mathcal{L}}^!$, R becomes a new constraint on P_i°. However this cannot happen, for u had no such terms and v had only terms of the form $F(P_i \cap R)^\circ \cap FG(P_i^\circ \cap R)$, which are rewritten to $\mathbf{0}$ by $R1$.

Lemma 15. *Given t such that $\Xi_i(t)$ holds then $t \to_{\mathcal{S}_i}^! t'$ and $\Xi_i(t')$.*

Proof. Follows from $\to_{\mathcal{S}_i}$ rules because if $\Xi(t)$ then no term of the form $F(P_i^\circ \cap GR)$ with P_i° in R can occur in t. Such occurrence is the necessary condition for $\to_{\mathcal{S}_i}^!$ to destroy P-constraint completeness.

Lemma 16. *Given t such that $\Xi_i(t)$ holds then $t \to_{\mathcal{R}_i}^! t$.*

Proof. Follows from the definition of $\to_{\mathcal{R}_i}$.

Lemma 17. *$\to_{\mathcal{U}_i}$ reaches a fixpoint, in other words, given a term t then $t \to_{\mathcal{U}_i} t' \to_{\mathcal{U}_i} t''$ and $t' = t''$.*

Proof. $\Xi_i(t')$ holds by Lemma 14. In the second $\to_{\mathcal{U}_i}$ step, $t' \to_{\mathcal{L}}^! t'$ holds. By Lemma 15, $t' \to_{\mathcal{S}_i}^! u$ and $\Xi_i(u)$. By Lemma 16, we have $u \to_{\mathcal{R}_i}^! u$. Finally, we need to prove $t' \to_{\mathcal{S}_i}^! u \to_{\mathcal{L}}^! t'$. This is proven by the fact that $\to_{\mathcal{L}}^!$ *undoes* everything $\to_{\mathcal{S}_i}^!$ does on already factorized terms, which is the case of t'. Given rewriting rules $F(R \cap G(P_i^\circ \cap S)) \to_{\mathcal{S}_i} FR \cap FG(P_i^\circ \cap S)$ and $FT \cap FU \to_{\mathcal{L}} F(T \cap U)$, their composition happens with substitution $\{T \mapsto R,\ U \mapsto G(P_i^\circ \cap S)\}$, resulting in the rewriting rule $F(R \cap G(P_i^\circ \cap S)) \to_{\mathcal{L} \circ \mathcal{S}_i} F(R \cap G(P_i^\circ \cap S))$ which is the identity.

Definition 15 (Solved form algorithm). *Given a term $t \in \mathbf{U}$, containing P_i° terms with $i \in \{1 \ldots n\}$ and $n \geq 1$, we define the rewriting relation $\to_{\mathcal{U}_n}$ as $\to_{\mathcal{U}_1} \circ \cdots \circ \to_{\mathcal{U}_n}$, that is:*

$$\to_{\mathcal{L}}^! \circ \to_{\mathcal{S}_1}^! \circ \to_{\mathcal{R}_1}^! \circ \to_{\mathcal{L}}^! \circ \to_{\mathcal{S}_2}^! \circ \to_{\mathcal{R}_2}^! \circ \to_{\mathcal{L}}^! \circ \cdots \circ \to_{\mathcal{S}_n}^! \circ \to_{\mathcal{R}_n}^! \circ \to_{\mathcal{L}}^!$$

For $n = 0$, there are no P_i° in t and we define $\to_{\mathcal{U}_0}$ as $\to_{\mathcal{L}}^!$.

Definition 16 (P-dependency). *Given a term $P_i^\circ \cap R$, we say i P-depends on j if $j \in dep(R)$ where $dep : \mathbf{U} \to \mathcal{P}(\mathbb{N})$ is defined in Fig. 3. We can build the P-dependency for a term t taking all its subterms in the form $P_i^\circ \cap R$.*

Lemma 18 (Constraint destruction). *Given $\Xi_j(t)$, $t \to_{\mathcal{U}_i} t'$ can make $\neg\Xi_j(t')$ iff j P-depends on i and $i \neq j$.*

Proof. The only way $\to_{\mathcal{U}_i}$ can add a new constraint to a P_j° by constraint propagation is if the term is in the form $P_j^\circ \cap FR$ and P_i° occurs in R which is precisely the definition of P-dependency.

Lemma 19 (Occurs check). *If i P-depends on i in a term t then $t \to_{\mathcal{U}_i} \mathbf{0}$.*

Proof. Such P-dependency means $P_i^\circ \cap FR$ and P_i° occurs in R which gets rewritten to $\mathbf{0}$ by $\to^!_{\mathcal{R}_i}$.

Lemma 20. *There is a finite k such that $t \to^k_{\mathcal{U}_n} t'$ and $\Xi(t')$.*

Proof. The case $n = 0$ follows from Lemma 5 with $k = 1$. The case $n = 1$ follows from Lemma 17 with $k = 1$. In the case $n > 1$, for a step $u \to_{\mathcal{U}_i} u'$ then $\Xi_i(u')$ by Lemma 14. However, by Lemma 18 such a step can make $\neg\Xi_j(u')$ hold iff j P-depends on i. Suppose the P-dependency graph of t is acyclic. Then there is a set of terminal edges E such that forall $l \in E$, Ξ_l holds after $\to_{\mathcal{U}_l}$ and no other step $\to_{\mathcal{U}_i}$ can make Ξ_l false, for they depend on no j. Once the term is Ξ_l for every $l \in E$, E can be removed from the graph. The process can be repeated with the new set E' of terminal edges a finite number of times, for the graph is acyclic and the number of edges is finite. Finally if the P-dependency graph of the original term is cyclic, then by Lemma 19 the term would get rewritten to $\mathbf{0}$ in the $\to_{\mathcal{U}_i}$ iteration corresponding to any i on the cycle. Thus when the process ends, $\Xi_l(t')$ for all $l \in \{1 \dots n\}$ which is equivalent to $\Xi(t')$.

Lemma 21. *$\to_{\mathcal{U}_n}$ reaches a fixpoint.*

Proof. The proof is similar to the one in Lemma 17, but using Lemma 20, for once $\Xi(t)$ holds, $\to_{\mathcal{U}_n}$ application does not modify t.

Lemma 22. *The fixpoint for $\to_{\mathcal{U}_n}$ is in solved form.*

Proof. We check by induction that no unsolved term can be a fixpoint:

- $P_i^\circ \cap f_j^n R$, if $P_i^\circ \notin R$ and R not in solved form: If R is in unsolved form then one of the cases below applies. Otherwise, if P_i° is a subterm of R when $\to^!_{\mathcal{R}_i}$ fires, it rewrites to $\mathbf{0}$.
- $f_i^n R \cap f_i^n S$: Terms of this form match the left side of $\to_{\mathcal{L}}$.
- $f_i^n R \cap f_j^n S$, if $\neg\Xi(R \cap S)$ or any of R, S are unsolved form: It either contradicts Lemma 21 or one of these cases apply.
- $R_1 \cap \cdots \cap R_n$, if some pair $R_i \cap R_j, i \neq j$ is in unsolved form: If the pair R_i, R_j is in unsolved form one of the cases above apply.

Definition 17 (Relational unification). *We define the relation between* **U***-terms with maximum index n, denoted* $t \rightarrow_{\{UNIF,n\}} t'$*, as follows: If* $t \rightarrow^{!}_{\mathcal{U}_n} s$ *and s is not valid then* $t' = \mathbf{0}$*; otherwise,* $t' = s$.

Theorem 1. *Terms* t_1, t_2 *with n different variables are not unifiable iff* $K(t_1) \cap K(t_2) \rightarrow_{\{UNIF,n\}} \mathbf{0}$.

Proof. By Lemma 22 we know that $\rightarrow^{!}_{\mathcal{U}_n}$ brings any **U**-term to solved form. As we have a complete decision procedure for solved terms by Lemma 3, we can decide if $K(t_1) \cap K(t_2)$ is $\mathbf{0}$, which means by Eq. 1 that t_1, t_2 are not unifiable.

Definition 18 (Relational unifiers). *We say a term t in solved form has a unifier R for i if* $P_i^\circ \cap R$ *is a subterm of t.*

For example, suppose we have the term $P_1^\circ \cap (a, a)\mathbf{1} \cap P_2^\circ$. This means in $\mathcal{T}_\Sigma(\mathcal{X})$ that $x_1 = x_2 = a$.

5 Extending the Framework

Our framework can be seamlessly extended in order to formalize and decide other notions of unification. For instance, a common unification pattern found in logic programming is unification among *renamed apart* terms. Consider a restriction operator ν such that $\nu x_1.t = x_1$ is equivalent to $t\sigma = x_1$, where σ is a renaming apart of x_1, and $\nu x_1.t = x_1$ is equivalent to $t = x_1$ iff x_1 does not occur in t.

This variable-restriction concept can be faithfully represented in our framework using the partial projection relation Q_i, which relates an open sequence with the ones where the ith element may be any term. (In [18] Q_i is understood as an existential quantifier.) Our framework is modified as follows. First, extend K with case $K(\nu x_i.t) = K(t)Q_i$. In words, the ith position of $K(\nu x_i.t)$'s codomain is free, whereas for $K(t)$ it contains the set of possible groundings for x_i. The new definition of K extends **U**, so a new decision procedure is needed. It is defined in modular fashion by adding a new rewriting subsystem for Q_i elimination. Some rules of this system are $P_j Q_i \rightarrow P_j$ and $(f_i^n P_j \cap R)Q_i \rightarrow f_i^n P_j Q_i \cap R Q_i$. The full set of rules is much more conveniently expressed by using the framework outlined in Sec. 7 and is not presented here.

6 Related Work

Unification was first proposed in [24] and axiomatized in [20,21]. The most remarkable classical algorithms for first-order unification are surveyed in [1], such as [22,23] which define fast and well-understood algorithms targeted to implementors using imperative languages. Other interesting approaches to unification include categorical views [13,26], unification for lambda terms [15,10] and a remarkable one [9], which studies unification in the typed combinatorial paradigm [6]. Nominal unification [28] is an alternative approach to meta-theory formalization that uses nominal logic. C-expressions [3,2] also provide a combinatorial unification algorithm, based on applicative terms instead of relations. Unification using explicit substitutions is handled in the higher-order case in [11].

7 Conclusions and Future Work

We have presented an algorithm for first-order unification using rewriting in variable-free relational algebra. The simple systems $\rightarrow_{\mathcal{L}}$ and $\rightarrow_{\mathcal{R}}$ suffice to decide unification for occurs-check-free pairs of terms. Function Δ and the split-rewriting system are introduced to deal with occurs check at the expense of losing simplicity. We can regain simplicity by using a more powerful notion of rewriting and matching, obtaining as a result an efficient decision procedure largely in the spirit of [22]. Furthermore, a dual right-factoring version of the algorithm exists where constraints are accumulated over f_i^n terms and factorization happens for P_i° terms, using $FP_i^\circ \cap GP_i^\circ = (F \cap G)P_i^\circ$. We plan to relate this kind of duality with other duality/symmetry notions such as deep inference [14].

In the relational world we have no variables and no substitution notion. This raises an interesting paradox: the heart of unification is the computation of a substitution acting on variables. How can we unify in such a setting? The answer is that substitution gets replaced by constraint propagation and occurs check translates to functorial compatibility in such a way that *pure* (conditional) rewriting is enough!

Some mathematical advantages of performing unification in the relational setting are those of the abstract syntax program: to reflect algebraically all the issues that most algorithms have to handle at the meta-level, such as contexts, multi-equations, and substitution in order to make them as declarative as the object language. Performing unification module additional theories is possible in our framework, as we have shown in Sec. 5. Representation of the unification problem itself is usually straightforward, whereas building a decision procedure for it will depend on the added theory. See [18] on how to represent disunification problems with relations.

References

1. Franz Baader and Wayne Snyder. Unification theory. In Robinson and Voronkov [25], pages 445–532.
2. Marco Bellia and M. Eugenia Occhiuto. C-expressions: A variable-free calculus for equational logic programming. *Theor. Comput. Sci.*, 107(2):209–252, 1993.
3. Marco Bellia and M. Eugenia Occhiuto. Lazy linear combinatorial unification. *Journal of Symbolic Computation*, 27(2):185–206, 1999.
4. Paul Broome and James Lipton. Combinatory logic programming: computing in relation calculi. In *ILPS '94: Proceedings of the 1994 International Symposium on Logic programming*, pages 269–285, Cambridge, MA, USA, 1994. MIT Press.
5. Hubert Comon. Disunification: A survey. In *Computational Logic - Essays in Honor of Alan Robinson*, pages 322–359, 1991.
6. Haskell B. Curry and Robert Feys. *Combinatory Logic*, volume 1. North Holland, 1958. Second edition, 1968.
7. Nachum Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17:279–301, 1982.
8. Nachum Dershowitz and David A. Plaisted. Rewriting. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 535–610. Elsevier and MIT Press, 2001.

9. Daniel J. Dougherty. Higher-order unification via combinators. *Theoretical Computer Science*, 114(2):273–298, 1993.
10. Gilles Dowek. Higher-order unification and matching. In Robinson and Voronkov [25], pages 1009–1062.
11. Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher order unification via explicit substitutions. *Inf. Comput.*, 157(1-2):183–235, 2000.
12. P. J. Freyd and A. Scedrov. *Categories, Allegories.* North Holland Publishing Company, 1991.
13. Joseph Goguen. What is unification? A categorical view of substitution, equation and solution. In Maurice Nivat and Hassan Aït-Kaci, editors, *Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques*, pages 217–261. Academic, 1989.
14. Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1):1–64, 2007. http://cs.bath.ac.uk/ag/p/SystIntStr.pdf.
15. Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975.
16. J. W. Klop. Term rewriting systems. In Abramsky, Gabbay, and Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Clarendon Press, 1992.
17. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 342–376. Springer, Berlin, Heidelberg, 1983.
18. Jim Lipton and Emily Chapman. Some notes on logic programming with a relational machine. In Ali Jaoua, Peter Kempf, and Gunther Schmidt, editors, *Using Relational Methods in Computer Science*, Technical Report Nr. 1998-03, pages 1–34. Fakultät für Informatik, Universität der Bundeswehr München, July 1998.
19. J. W. Lloyd. *Foundations of logic programming.* Springer-Verlag New York, Inc., New York, NY, USA, 1984.
20. Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *LICS*, pages 348–357. IEEE Computer Society, 1988.
21. A. I. Mal'cev. On the elementary theories of locally free universal algebras. *Soviet Math*, pages 768–771, 1961.
22. Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.
23. Mike Paterson and Mark N. Wegman. Linear unification. *J. Comput. Syst. Sci.*, 16(2):158–167, 1978.
24. John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
25. John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes).* Elsevier and MIT Press, 2001.
26. D. E. Rydeheard and R. M. Burstall. A categorical unification algorithm. In *Proceedings of a tutorial and workshop on Category theory and computer programming*, pages 493–505, New York, NY, USA, 1986. Springer-Verlag New York, Inc.
27. Alfred Tarski and Steven Givant. *A Formalization of Set Theory Without Variables*, volume 41 of *Colloquium Publications.* American Mathematical Society, Providence, Rhode Island, 1987.
28. Christian Urban, Andrew M. Pitts, and Murdoch Gabbay. Nominal unification. In Matthias Baaz and Johann A. Makowsky, editors, *CSL*, volume 2803 of *Lecture Notes in Computer Science*, pages 513–527. Springer, 2003.

Matching with Membership Constraints for Hedge and Context Variables

Mircea Marin[1] and Temur Kutsia[2]

[1] Graduate School of Systems and Information Engineering
University of Tsukuba
Tsukuba 307-8573, Japan
mmarin@cs.tsukuba.ac.jp
[2] Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria
tkutsia@risc.uni-linz.ac.at

Abstract. Pattern matching with membership constraints for hedge and context variables is a desirable capability for the analysis and decomposition of structures that can be presented as hedges in an algebra with flexible arity function symbols. We distinguish two kinds of patterns: for hedges and for contexts, i.e., sequences of terms with one or more occurrences of a placeholder for a nonempty hedge. Our patterns are a generalization of regular expressions where, besides regular operators we also employ hedge variables and context variables whose admissible bindings are subjected to membership constraints.
We propose a matching algorithm that is sound and complete under some reasonable restrictions on the structure of the matching problem.

1 Introduction

Regular expression patterns are a useful capability for the analysis and decomposition of structured and semistructured data. They were first introduced in traditional string manipulation languages such as Perl [11], and Unix tools such as `sed` and `awk`. The use of this mechanism in the analysis and decomposition of tree-like data structures is a more recent development, witnessed by languages for XML processing such as XDuce [4, 5], CDuce [1] and XQuery [10], languages for technical computing and symbolic computation such as Mathematica [12], etc.

Regular hedge expression patterns were proposed by Hosoya [4, 5] as an extension of ML-style patterns, and introduced in XDuce to support data extraction on sequences of terms (*a.k.a.* hedges). In essence, such a pattern is an regular hedge expression with variable-annotated subexpressions, such that: (1) all variable annotations are distinct; and (2) no variable annotations are allowed below a repetition operator. Names can be defined for name-free regular expression patterns (*a.k.a.* regular expression types) and stored in a global environment. This mechanism enables mutually recursive definitions of regular expression types.

We can view matching a sequence of ground terms against a regular expression pattern as *solving* a system of equations, definitions, and membership constraints. For example, matching a ground hedge v against $\texttt{g}(x \in \mathrm{h}(y \in \texttt{g}(\texttt{N}\ \texttt{a}*)))$ where $\texttt{N}$ is recursively defined by $\texttt{N} = \mathrm{h}(\texttt{N}\ \texttt{N}) + \texttt{b}$ amounts to solving the following system of equations, definitions and membership constraints:

$$\{\texttt{g}(x) = v,\ x \in \mathrm{h}(y),\ y \in \texttt{g}(\texttt{N}\ \texttt{a}*),\ \texttt{N} = \mathrm{h}(\texttt{N}\ \texttt{N}) + \texttt{b}\}$$

with constrained hedge variables x, y. For regular hedge extension patterns, the system of constraints is of the form $\langle E \mid \mathcal{E}\rangle$ where (1) E is a set of equations of the form $\omega = v$ with ω a regular expression type that may have hedge variables and type names as subexpressions[3], (2) $\mathcal{E}$ is a set of pattern definitions of the form $\texttt{N} = \omega$, or membership constraints of the form $x \in \tau$ with τ is regular expression type, such that every hedge variable occurs at most once in $\mathcal{E}$.

In this paper we generalize in several ways the constrained equational logic to enable more expressive patterns for the analysis and decomposition of hedges:

1. We extend the constraint logic with a special kind of second order variables, called *context variables.*
2. Membership constraints can be specified for both hedge variables and for context variables.
3. We allow variables under the repetition operator $*$.
4. We drop the restriction of having types as right-hand sides of membership constraints, and allow patterns instead. A new class of patterns, called *regular context patterns*, is introduced for the spefication of membership constraints for context variables.
5. We allow multiple occurrences of variables in patterns.

Our constraint logic bears some similarity to the order-sorted equational logic proposed by Comon [2, 3], but there are notable differences: (1) Comon's equations are between terms, whereas we can have equations between hedges; (2) Comon's membership constraints require regular types instead of patterns; (3) Comon's context variables denote terms with one hole placeholder for term, whereas our context variables denote hedges with at least one hole placeholder for hedges. These generalizations are reasonable, since we consider only matching problems, whereas Comon considers unification problems, where the right hand sides of equations are not necessarily ground expressions.

The matching algorithm described here is a further development of our former matching algorithms with regular constraints [6, 7]:

– Single-hole contexts have been generalized to multi-hole contexts,
– We introduced pattern names, to avoid the usage of intricate regular expressions for context composition and compositional iteration.

The paper is organized as follows. Section 2 describes our framework: hedges, contexts, patterns, membership constraints, and matching problems. In Sect. 3

[3] However, variables are not allowed under the repetition operator $*$.

we present a transformation system $\mathcal{M}$ as the underlying computational model of a matching procedure, and prove that it is sound, terminating and complete. Section 4 concludes and draws directions of future work.

2 Preliminaries

In general, if A is a set, then we write $A^\star$ for the monoid of strings of elements of A with neutral element ε. We use space as both the "cons" operation for adding an element to either end of a string and as the "append" operation on strings. For example, if h_1 is the string $a_1\ a_2\ a_3$, and h_2 is the string $a_4\ a_5$, then $a\ h_1$ denotes the string $a\ a_1\ a_2\ a_3$, while $h_1\ a$ denotes the string $a_1\ a_2\ a_3\ a$ and $h_1\ h_2$ denotes the string $a_1\ a_2\ a_3\ a_4\ a_5$. (There is no confusion of using space for both "cons" and "append" operations, as long as we do not need to talk about strings of strings.) Also, we write $A \uplus B$ for the union of two disjoint sets A and B, and $|A|$ for the number of elements of a finite set A.

We assume the existence of three mutually disjoint sets of symbols:

1. a nonempty set $\mathcal{F}$ of *function symbols*, denoted by f, g, a, b, possibly subscripted,
2. a countably infinite set $\mathcal{V}_\mathrm{h}$ of *hedge variables*, denoted by x, y, z, possibly subscripted,
3. a countably infinite set $\mathcal{V}_\mathrm{c}$ of *context variables*, denoted by $\overline{C}$, possibly subscripted. We define $\mathcal{V} := \mathcal{V}_\mathrm{h} \uplus \mathcal{V}_\mathrm{c}$, and call its elements *variables*,

and the *hole* symbol $\bullet \notin \mathcal{F} \cup \mathcal{V}$. Given $\mathcal{X} \subseteq \mathcal{V}$, we define the syntactic categories $\mathcal{HE}(\mathcal{F}, \mathcal{X})$ of *hedge elements*, $\mathcal{H}(\mathcal{F}, \mathcal{X}) := \mathcal{HE}(\mathcal{F}, \mathcal{X})^\star$ of *hedges*, and $\mathcal{H}_s(\mathcal{F}, \mathcal{X}) := \mathcal{H}(\mathcal{F}, \mathcal{X}) \setminus \mathcal{V}_\mathrm{h}^\star$ of *strict hedges* with variables from $\mathcal{X}$ as follows:

$$\begin{array}{ll} e ::= x \mid f(h) \mid \overline{C}(p) & \textit{hedge elements} \\ h \ \in\ \mathcal{HE}(\mathcal{F}, \mathcal{X})^\star & \textit{hedges} \\ p \ \in\ \mathcal{HE}(\mathcal{F}, \mathcal{X})^\star \setminus \mathcal{V}_\mathrm{h}^\star & \textit{strict hedges} \end{array}$$

where $\overline{C} \in \mathcal{X} \cap \mathcal{V}_\mathrm{c}$, and $x \in \mathcal{X} \cap \mathcal{V}_\mathrm{h}$. Thus, a hedge is a string of hedge elements, and a strict hedge is a hedge which is not a string of hedge variables. For example, ε and $x\ y\ z$ are hedges, whereas $\overline{C}(f(\varepsilon))$ and $x\ f(g(\varepsilon))$ are strict hedges. An expression $f(h)$ is called a *term*, and $\overline{C}(p)$ is a *flex hedge*. A term $f(\varepsilon)$ is abbreviated by f. A *value* is a hedge without occurrences of variables, i.e., an element of $\mathcal{H}(\mathcal{F}, \emptyset)$. We denote values by v. Any value v is of the form $f_1(v_1)\ \ldots\ f_n(v_n)$ with $n \geq 0$, $f_i \in \mathcal{F}$ and $v_i \in \mathcal{H}(\mathcal{F}, \emptyset)$ $(1 \leq i \leq n)$. The *size* of v, denoted by $|v|$, is defined recursively by $|v| := n + \sum_{i=1}^{n} |v_i|$, where $v = f_1(v_1)\ \ldots\ f_n(v_n)$.

A *context* $c \in \mathcal{C}(\mathcal{F}, \mathcal{X})$ is the result of replacing with $\bullet$ one or more hedge sub-elements of a hedge. A *context element* is either $\bullet$ or $f(c)$ or $\overline{C}(c)$. For example, $h = x\ y\ \overline{C}(a)$ is a strict hedge; $\bullet\ \bullet\ \overline{C}(a)$ and $x\ y\ \overline{C}(\bullet)$ are contexts obtainable from h by replacing certain sub-elements with $\bullet$; and $f(h)$ and $f(h\ h)$ are terms. We abbreviate $\mathcal{H}(\mathcal{F}, \mathcal{V})$, $\mathcal{C}(\mathcal{F}, \mathcal{V})$, $\mathcal{H}(\mathcal{F}, \emptyset)$ and $\mathcal{C}(\mathcal{F}, \emptyset)$ by $\mathcal{H}$, $\mathcal{C}$, $\mathcal{H}_0$ and $\mathcal{C}_0$ respectively. From now on we assume that h stands for hedge, c for context, and t for term.

We consider two main operations with contexts:

1. If $c, c_1 \in \mathcal{C}(\mathcal{F}, \mathcal{X})$ and $h \in \mathcal{H}(\mathcal{F}, \mathcal{X})$, then $c[h]$ is the hedge obtained by replacing all occurrences of $\bullet$ in c with h, and $c[c_1]$ is the context obtained by replacing all occurrences of $\bullet$ in c with c_1. This operation views c as a mapping $\lambda \bullet .c$ from $\mathcal{H}(\mathcal{F}, \mathcal{X})$ to $\mathcal{H}(\mathcal{F}, \mathcal{X})$.
2. If $c, c_1, \ldots, c_n \in \mathcal{C}(\mathcal{F}, \mathcal{X})$, $h_1, \ldots, h_n \in \mathcal{H}(\mathcal{F}, \mathcal{X})$ and c has n hole occurrences, then $c\langle h_1, \ldots, h_n\rangle$ is the hedge obtained by replacing holes of c with $h_1, \ldots, h_n$, and $c\langle c_1, \ldots, c_n\rangle$ is the context obtained by replacing the holes of c with $c_1, \ldots, c_n$. The replacement is performed in the order of a leftmost innermost traversal of c. This operation views c as a linear mapping $\lambda \bullet_1 \ldots \bullet_n .c\langle \bullet_1, \ldots, \bullet_n\rangle$ from $\mathcal{H}(\mathcal{F}, \mathcal{X})^n$ to $\mathcal{H}(\mathcal{F}, \mathcal{X})$.

A *substitution* is a mapping $\theta : \mathcal{V} \to \mathcal{H} \cup \mathcal{C}$ that maps hedge variables to hedges and context variables to contexts, such that its domain $dom(\theta) := \{x \in \mathcal{V}_\text{h} \mid \theta(x) \neq x\} \cup \{\overline{C} \in \mathcal{V}_\text{c} \mid \theta(\overline{C}) \neq \overline{C}(\bullet)\}$ is a finite set. As usual, we represent a substitution θ as the set $\{X \mapsto \theta(X) \mid X \in dom(\theta)\}$. Two substitutions θ_1 and θ_2 are *compatible*, notation $\theta_1 \sim \theta_2$, if $\theta_1(X) = \theta_2(X)$ for all $X \in dom(\theta_1) \cap dom(\theta_2)$. Sometimes, we will make use of the notation $\theta|_V$ for the restriction of θ to $dom(\theta) \cap V$, and $\theta|_{-V}$ for the restriction of θ to $dom(\theta) \setminus V$.

If e is a hedge element or context element, then the *instantiation* of $u \in \mathcal{H} \cup \mathcal{C}$ with a substitution θ, denoted by $u\theta$, is defined by structural induction as follows:

$$\begin{array}{lll} \varepsilon\theta = \varepsilon & \bullet\theta = \bullet & x\theta = \theta(x) \\ f(u)\theta = f(u\theta) & (e\ u)\theta = (e\theta)\ (u\theta) & \overline{C}(u)\theta = \theta(\overline{C})[u\theta] \end{array}$$

If θ_1 and θ_2 are substitutions, then $\theta_1\theta_2$ is the mapping defined by

– $\theta_1\theta_2(X) = (X\theta_1)\theta_2$ for all $X \in \mathcal{V}$, and
– $dom(\theta_1\theta_2) = \{x \in \mathcal{V}_\text{h} \mid (x\theta_1)\theta_2 \neq x\} \cup \{\overline{C} \in \mathcal{V}_\text{c} \mid (\overline{C}\theta_1)\theta_2 \neq \overline{C}(\bullet)\}$.

Note that $h\theta \in \mathcal{H}$ and $c\theta \in \mathcal{C}$ whenever $h \in \mathcal{H}$ and $c \in \mathcal{C}$. Therefore, the composition of substitutions is a substitution too.

As usual, we write $vars(E)$ for the set of variables that occur in a syntactic expression E, and say that E is *ground* if $vars(E) = \emptyset$. A substitution θ is *ground* if $\theta(X)$ is ground for all $X \in dom(\theta)$. The substitution with empty domain is denoted by ϵ, and is called *empty substitution.* If A is set of hedges or contexts, then we define $A\theta := \{e\theta \mid e \in A\}$. Also, if $c \in \mathcal{C}$, H is a set of hedges, and C, C_1, C_2 are sets of contexts, then we consider the following operations:

$c.H := \{c[h] \mid h \in H\}$, $C.H := \{c[h] \mid c \in C, h \in H\}$,
$C_1.C_2 := \{c_1[c_2] \mid c_1 \in C_1, c_2 \in C_2\}$,
$H* := \bigcup_{n=0}^{\infty} H_n$ where $H_0 := \{\varepsilon\}$ and $H_{n+1} := H_n \cup \{h\ h' \mid h \in H, h' \in H_n\}$,
$C+ := \bigcup_{n=1}^{\infty} C_{(n)}$ where $C_{(1)} := C$ and $C_{(n+1)} := C_{(n)} \cup \{c\ c' \mid c \in C, c' \in C_{(n)}\}$.

2.1 Patterns, Pattern Definitions, and Membership Constraints

We distinguish three kinds of patterns: hedge patterns, strict hedge patterns, and context patterns. Patterns can be given names by *pattern definitions* and can contain variables subjected to *membership constraints.*

We assume two countably infinite sets of names: $\mathcal{N}_h$ for hedge patterns and $\mathcal{N}_c$ for context patterns, such that $\mathcal{N}_h \cap \mathcal{N}_c = \emptyset$, and assume that $\mathtt{H}$ ranges over $\mathcal{N}_h$ and $\mathtt{C}$ ranges over $\mathcal{N}_c$. We distinguish 3 classes of patterns: for hedges, for strict hedges, and for contexts. They are built from hedge pattern elements and from context pattern elements as shown below, where $\mathcal{HPE}$ is the set of hedge pattern elements, $\mathcal{HPE}_s$ is the set of strict hedge pattern elements, and $\mathcal{CPE}$ is the set of context pattern elements.

$$
\begin{array}{rcll}
\pi & ::= & \omega \mid \chi & \textit{patterns} \\
\omega & \in & \mathcal{HPE}^\star & \textit{hedge patterns} \\
\overline{\omega} & \in & \mathcal{HPE}^\star \mathcal{HPE}_s \mathcal{HPE}^\star & \textit{strict hedge patterns} \\
\chi & \in & \mathcal{HPE}^\star \mathcal{CPE}(\mathcal{CPE} \cup \mathcal{HPE})^\star & \textit{context patterns} \\
o & ::= & \_ \mid \_\_ \mid f(\omega) \mid \tilde{}(\omega) \mid \overline{C}(\overline{\omega}) \mid \mathtt{C}(\overline{\omega}) \mid \overline{\omega} + \overline{\omega} & \textit{strict hedge pattern elements} \\
\nu & ::= & o \mid x \mid \_\_\_ \mid \omega + \omega \mid \omega{*} \mid \mathtt{H} & \textit{hedge pattern elements} \\
\xi & ::= & \bullet \mid f(\chi) \mid \tilde{}(\chi) \mid \overline{C}(\chi) \mid \mathtt{C}(\chi) \mid \chi\texttt{+} \mid \chi + \chi & \textit{context pattern elements}
\end{array}
$$

We write $\mathcal{HP}$ for the set of hedge patterns, $\mathcal{HP}_s$ for the set of strict hedge patterns, and $\mathcal{CP}$ for the set of context patterns. Note that $\mathcal{H} \subset \mathcal{HP}$ and $\mathcal{C} \subset \mathcal{CP}$. If $\chi, \chi_1 \in \mathcal{CP}$ and $\omega \in \mathcal{HP}$ then $\chi[\chi_1]$ is the context pattern produced by replacing $\bullet$ with χ_1 in χ, and $\chi[\omega]$ is the hedge pattern produced by replacing $\bullet$ with ω in χ. Similarly, the notion of substitution can be extended to a mapping θ on $\mathcal{V} \cup \mathcal{HP} \cup \mathcal{CP}$ such that $\theta(x) \in \mathcal{HP}$ for all $x \in \mathcal{V}_{\mathrm{h}}$ and $\theta(\overline{C}) \in \mathcal{CP}$ for all $\overline{C} \in \mathcal{CP}$; and substitution instantiation can be extended to patterns as follows:

- $\pi\theta = \pi$ if $\pi \in \mathcal{N}_h \cup \{\varepsilon, \_, \_\_, \_\_\_, \bullet\}$, $x\theta = \theta(x)$,
- $(\pi_1\ \pi_2)\theta = (\pi_1\theta)\ (\pi_2\theta)$, $(\pi_1 + \pi_2)\theta = (\pi_1\theta) + (\pi_2\theta)$,
- $q(\pi)\theta = q(\pi\theta)$ if $q \in \{\tilde{}\} \cup \mathcal{F} \cup \mathcal{N}_c$, $\overline{C}(\pi)\theta = c[\pi\theta]$ if $\overline{C} \in \mathcal{V}_{\mathrm{c}}$ and $c = \theta(\overline{C})$,
- $\omega{*}\theta = \omega\theta{*}$, $\chi\texttt{+}\theta = \chi\theta\texttt{+}$.

Our kinds of patterns are preserved by substitution instantiation: If θ is a substitution, $\mathcal{A} \in \{\mathcal{HP}, \mathcal{HP}_s, \mathcal{CP}\}$ and $\pi \in \mathcal{A}$, then $\pi\theta \in \mathcal{A}$. If $\chi \in \mathcal{CP}$, $\mathcal{A} \in \{\mathcal{HP}_s, \mathcal{CP}\}$ and $\pi \in \mathcal{A}$, then $\chi[\pi]$ denotes the pattern obtained by replacing all holes of c with π. Note that $\chi[\pi] \in \mathcal{A}$.

Pattern definitions provide names for patterns. They are of the form $N = \pi$, subject to the following restrictions: (1) $\pi \in \mathcal{HP}$ if $N \in \mathcal{N}_h$, and $\pi \in \mathcal{CP}$ if $N \in \mathcal{N}_c$; (2) if $N' \in \mathcal{N}_h \cup \mathcal{N}_c$ occurs in π, then it is either below a symbol $f \in \mathcal{F} \cup \{\tilde{}\}$, or after a strict hedge pattern element or context pattern element. These restrictions ensure a tight correspondence with name-free tree patterns: they allow to read off a (possibly infinite) name-free tree pattern as the (possibly infinite) unfolding of a given name.

Membership constraints introduce constraints for the possible bindings of variables. They are of the form $(X \ \mathtt{in}\ \pi, \phi)$ with $\phi \in \{0, 1\}$ and X is a hedge (resp. context) variable if π is a hedge (resp. context) pattern.

We store pattern definitions and membership constraints in a set $\mathcal{E}$, which we call *environment.* We write $\mathit{vars}(\mathcal{E})$ for the set of variables with occurrences in $\mathcal{E}$,

$dvars(\mathcal{E})$ for the set of variables constrained in $\mathcal{E}$, $defs(\mathcal{E})$ for the set of pattern definitions in $\mathcal{E}$, and $ns(\mathcal{E})$ for the set of names defined in $\mathcal{E}$. The dependency graph of $\mathcal{E}$ is the graph $\mathcal{DG}(\mathcal{E})$ with set of nodes $vars(\mathcal{E}) \cup ns(\mathcal{E})$ and with an arc from Z_1 to Z_2 iff Z_2 occurs in the definition or membership constraint of Z_1.

Environments are required to satisfy the following conditions: (a) $\mathcal{DG}(\mathcal{E})$ has no cycles through a variable of $\mathcal{E}$, (b) every name of $ns(\mathcal{E})$ is defined exactly once in $\mathcal{E}$; and (c) every variable of $vars(\mathcal{E})$ is constrained at most once in $\mathcal{E}$. $\mathcal{E}$ is *total* if $vars(\mathcal{E}) = dvars(\mathcal{E})$. Restriction (a) is a form of variable occur check test on patterns, while restrictions (b) and (c) allow to regard $\mathcal{E}$ as a function on $dom(\mathcal{E}) := dvars(\mathcal{E}) \cup ns(\mathcal{E})$ that yields tuples defined as follows: $\mathcal{E}(X) := \langle \pi, \phi \rangle$ if $(X \,\mathtt{in}\, \pi, \phi) \in \mathcal{E}$; and $\mathcal{E}(N) := \langle \pi \rangle$ if $N = \pi \in \mathcal{E}$. We write $\mathcal{E}_i(Z)$ for the i-th component of $\mathcal{E}(Z)$, $rvars_{\mathcal{E}}(Z)$ for the variables reachable from Z in $\mathcal{DG}(\mathcal{E})$, and $vRng(\mathcal{E}) := \bigcup_{Z \in dom(\mathcal{E})} vars(\mathcal{E}_1(Z))$.

If $\mathcal{E}$ is an environment and θ is a substitution, then the instantiation of $\mathcal{E}$ with θ is the set $\mathcal{E}\theta := \{N = \pi\theta \mid (N = \pi) \in \mathcal{E}\} \cup \{(X \,\mathtt{in}\, \pi\theta, \phi) \mid X \notin dom(\theta)$ and $(X \,\mathtt{in}\, \pi, \phi) \in \mathcal{E}\}$. Note that environments are not invariant under substitution instantiation. However, if $\bigcup_{X \in dom(\theta)} vars(\theta(X)) \cap vars(\mathcal{E}) = \emptyset$, then $\mathcal{E}\theta$ is an environment. In particular, ground instantiations of environments are environments.

Matching relations. Hedge patterns can match hedge values, and context patterns can match context values. The matching relation $v \in_{\mathcal{E}} \pi \rightsquigarrow \sigma$ (resp. $c \in \pi \rightsquigarrow \sigma$) assumes that $\mathcal{E}$ is a total environment and $vars(\pi) \subseteq vars(\mathcal{E})$, and asserts that $v \in \mathcal{H}_0$ (resp. $c \in \mathcal{C}_0$) is matched by pattern π in environment $\mathcal{E}$ with matching substitution σ. The matching relation is the least relation closed under the inference rules of Fig. 1. The rules which make use of unions of substitutions are applicable only if those substitutions are pairwise compatible. We write $v \in_{\mathcal{E}} \omega$ (resp. $c \in_{\mathcal{E}} \chi$) if $v \in_{\mathcal{E}} \omega \rightsquigarrow \sigma$ (resp. $c \in_{\mathcal{E}} \chi \rightsquigarrow \sigma$) for some σ, and $v \notin_{\mathcal{E}} \omega$ (resp. $c \notin_{\mathcal{E}} \chi$) otherwise.

A ground substitution σ is *solution* of a total environment $\mathcal{E}$, notation $\sigma \in Sol(\mathcal{E})$, if $vars(\mathcal{E}) \subseteq dom(\sigma)$ and

1. $X\sigma \in_{\mathcal{E}} \mathcal{E}_1(X) \rightsquigarrow \theta$ and $\sigma \sim \theta$ holds for all $X \in vars(\mathcal{E})$; and
2. for all $\mathtt{H} \in ns(\mathcal{E}) \cap \mathcal{N}_h$ (resp. $\mathtt{C} \in ns(\mathcal{E}) \cap \mathcal{N}_c$) there exists $v \in \mathcal{H}_0$ (resp. $c \in \mathcal{C}_0$) such that $v \in_{\mathcal{E}\sigma} \mathtt{H}$ (resp. $c \in_{\mathcal{E}\sigma} \mathtt{C}(\bullet)$).

$\mathcal{E}$ is *consistent* if it has a solution. We have the following remarkable result:

Lemma 1. *Consistency of a total environment is decidable.*

The declarative characterization of matching given in Fig. 1 does not provide an algorithm for the computation of matchers. The main reason for this is the existence of infinite matching derivations when matching against a pattern $\omega*$ and ω matches ε. (See Example 1.)

$$\frac{}{\varepsilon \in_{\mathcal{E}} \varepsilon \rightsquigarrow \{\}} \quad \frac{[v \neq \varepsilon]}{v \in_{\mathcal{E}} \_\_ \rightsquigarrow \{\}} \quad \frac{}{v \in_{\mathcal{E}} \_\_\_ \rightsquigarrow \{\}} \quad \frac{v \in_{\mathcal{E}\{x \mapsto v\}} \mathcal{E}_1(x) \rightsquigarrow \sigma \quad [v \neq \varepsilon \text{ if } \mathcal{E}_2(x) = 1]}{v \in_{\mathcal{E}} x \rightsquigarrow \sigma \cup \{x \mapsto v\}}$$

$$\frac{v \in_{\mathcal{E}} \mathcal{E}_1(\mathtt{H}) \rightsquigarrow \sigma}{v \in_{\mathcal{E}} \mathtt{H} \rightsquigarrow \sigma} \quad \frac{v_1 \in_{\mathcal{E}} \nu \rightsquigarrow \sigma_1 \quad v_2 \in_{\mathcal{E}} \omega \rightsquigarrow \sigma_2}{v_1\ v_2 \in_{\mathcal{E}} \nu\ \omega \rightsquigarrow \sigma_1 \cup \sigma_2} \quad \frac{v \in_{\mathcal{E}} \omega \rightsquigarrow \sigma \quad [q \in \{f, \tilde{}\}]}{f(v) \in_{\mathcal{E}} q(\omega) \rightsquigarrow \sigma}$$

$$\frac{c \in_{\mathcal{E}} \mathcal{E}_1(\overline{C}) \rightsquigarrow \sigma_0 \quad v_1 \in_{\mathcal{E}} \overline{\omega} \rightsquigarrow \sigma_1 \ \ldots \ v_n \in_{\mathcal{E}} \overline{\omega} \rightsquigarrow \sigma_n \quad [(c \neq \bullet \text{ if } \mathcal{E}_2(\overline{C}) = 1) \wedge \forall i.(v_i \neq \varepsilon)]}{c\langle v_1, \ldots, v_n \rangle \in_{\mathcal{E}} \overline{C}(\overline{\omega}) \rightsquigarrow \bigcup_{i=0}^{n} \sigma_i \cup \{\overline{C} \mapsto c\}}$$

$$\frac{c \in_{\mathcal{E}} \mathcal{E}_1(\mathtt{C}) \rightsquigarrow \sigma_0 \quad v_1 \in_{\mathcal{E}} \overline{\omega} \rightsquigarrow \sigma_1 \ \ldots \ v_n \in_{\mathcal{E}} \overline{\omega} \rightsquigarrow \sigma_n \quad [\forall i.(v_i \neq \varepsilon)]}{c\langle v_1, \ldots, v_n \rangle \in_{\mathcal{E}} \mathtt{C}(\overline{\omega}) \rightsquigarrow \bigcup_{i=0}^{n} \sigma_i} \quad \frac{}{f(v) \in_{\mathcal{E}} \_ \rightsquigarrow \sigma}$$

$$\frac{v \in_{\mathcal{E}} \omega_i \rightsquigarrow \sigma \quad [1 \leq i \leq 2]}{v \in_{\mathcal{E}} \omega_1 + \omega_2 \rightsquigarrow \sigma} \quad \frac{}{\varepsilon \in_{\mathcal{E}} \omega* \rightsquigarrow \{\}} \quad \frac{v_1 \in_{\mathcal{E}} \omega \rightsquigarrow \sigma_1 \quad v_2 \in_{\mathcal{E}} \omega* \rightsquigarrow \sigma_2}{v_1\ v_2 \in_{\mathcal{E}} \omega* \rightsquigarrow \sigma_1 \cup \sigma_2}$$

$$\frac{}{\bullet \in_{\mathcal{E}} \bullet \rightsquigarrow \{\}} \quad \frac{v \in_{\mathcal{E}} \nu \rightsquigarrow \sigma_1 \quad c \in_{\mathcal{E}} \chi \rightsquigarrow \sigma_2}{v\ c \in_{\mathcal{E}} \nu\ \chi \rightsquigarrow \sigma_1 \cup \sigma_2} \quad \frac{c \in_{\mathcal{E}} \xi \rightsquigarrow \sigma_1 \quad v \in_{\mathcal{E}} \omega \rightsquigarrow \sigma_2}{c\ v \in_{\mathcal{E}} \xi\ \omega \rightsquigarrow \sigma_1 \cup \sigma_2}$$

$$\frac{c_1 \in_{\mathcal{E}} \xi \rightsquigarrow \sigma_1 \quad c_2 \in_{\mathcal{E}} \chi \rightsquigarrow \sigma_2}{c_1\ c_2 \in_{\mathcal{E}} \xi\ \chi \rightsquigarrow \sigma_1 \cup \sigma_2} \quad \frac{c \in_{\mathcal{E}} \chi \rightsquigarrow \sigma \quad [q \in \{f, \tilde{}\}]}{f(c) \in_{\mathcal{E}} q(\chi) \rightsquigarrow \sigma} \quad \frac{c \in_{\mathcal{E}} \chi_i \rightsquigarrow \sigma \ [1 \leq i \leq 2]}{c \in_{\mathcal{E}} \chi_1 + \chi_2 \rightsquigarrow \sigma}$$

$$\frac{c \in_{\mathcal{E}} \mathcal{E}_1(\overline{C}) \rightsquigarrow \sigma_0 \quad c_1 \in_{\mathcal{E}} \chi \rightsquigarrow \sigma_1 \ \ldots \ c_n \in_{\mathcal{E}} \chi \rightsquigarrow \sigma_n \quad [c \neq \bullet \text{ if } \mathcal{E}_2(\overline{C}) = 1]}{c\langle c_1, \ldots, c_n \rangle \in_{\mathcal{E}} \overline{C}(\chi) \rightsquigarrow \bigcup_{i=0}^{n} \sigma_i \cup \{\overline{C} \mapsto c\}}$$

$$\frac{c \in_{\mathcal{E}} \mathcal{E}_1(\mathtt{C}) \rightsquigarrow \sigma_0 \quad c_1 \in_{\mathcal{E}} \chi \rightsquigarrow \sigma_1 \ \ldots \ c_n \in_{\mathcal{E}} \chi \rightsquigarrow \sigma_n}{c\langle c_1, \ldots, c_n \rangle \in_{\mathcal{E}} \mathtt{C}(\chi) \rightsquigarrow \bigcup_{i=0}^{n} \sigma_i}$$

$$\frac{c \in_{\mathcal{E}} \chi}{c \in_{\mathcal{E}} \chi\mathsf{+}} \quad \frac{c_1 \in_{\mathcal{E}} \chi \rightsquigarrow \sigma_1 \quad c_2 \in_{\mathcal{E}} \chi\mathsf{+} \rightsquigarrow \sigma_2}{c_1\, c_2 \in_{\mathcal{E}} \chi\mathsf{+} \rightsquigarrow \sigma_1 \cup \sigma_2}$$

Fig. 1. Inference rules for matching

Example 1. The formula $\varepsilon \in_{\emptyset} \varepsilon*$ has an infinite matching derivation:

$$\frac{\varepsilon \in_{\emptyset} \varepsilon \rightsquigarrow \{\} \qquad \dfrac{\vdots}{\varepsilon \in_{\emptyset} \varepsilon* \rightsquigarrow \sigma}}{\varepsilon \in_{\emptyset} \varepsilon* \rightsquigarrow \sigma}$$

Note that this kind of infinite matching derivation can not be avoided by imposing restrictions on the structure of the environment. □

It is desirable to identify an algorithmic characterization of the matching relation. More precisely, we want to identify a terminating procedure that, for any given environment $\mathcal{E}$, pattern π and $u \in \mathcal{H}_0 \cup \mathcal{C}_0$ such that $\mathit{vars}(\pi) \cup \mathit{ns}(\pi) \subseteq \mathit{dom}(\mathcal{E})$, computes all substitutions σ such that the relation $u \in_{\mathcal{E}} \pi \rightsquigarrow \sigma$ holds. Also, the algorithm should abandon as soon as possible the computation of "blind" matchings that compute incompatible (partial) matching substitutions.

We address this problem by solving a more general class of problems: matching problems.

2.2 Matching Problems

A *matching equation* is a pair $\omega \ll v$ with ω a hedge pattern and v a value. A *matching problem* is a system $\langle \Phi \mid \mathcal{E} \rangle$ where Φ is a comma-separated sequence of matching equations, and $\mathcal{E}$ is a total environment such that $\mathit{vars}(\Phi) \cup \mathit{ns}(\Phi) \subseteq \mathit{dom}(\mathcal{E})$. A substitution σ is a *solution* (or *matcher*) of a matching problem $P = \langle \omega_1 \ll v_1, \ldots, \omega_n \ll v_n \mid \mathcal{E} \rangle$ if $\sigma \in \mathit{Sol}(\mathcal{E})$, $v_i \in_{\mathcal{E}} \omega_i \rightsquigarrow \sigma_i$ and $\sigma_i \sim \sigma$ for $1 \leq i \leq n$. We write $\mathit{Sol}(P)$ for the set of solutions of a matching problem P, $\mathit{vars}(P)$ for the set of variables with occurrences in P, and $\square_{\mathcal{E}}$ instead of $\langle \ \mid \mathcal{E} \rangle$.

In this paper we identify an algorithm to solve this kind of matching problems. We define the mapping $\mathit{sz}_{\mathcal{E}} : \mathcal{HP} \cup \mathcal{CP} \to \mathbb{N}$ by

- $\mathit{sz}_{\mathcal{E}}(\bullet) = \mathit{sz}_{\mathcal{E}}(\varepsilon) = \mathit{sz}_{\mathcal{E}}(\_) = \mathit{sz}_{\mathcal{E}}(\_\_) = \mathit{sz}_{\mathcal{E}}(\_\_\_) := 1$,
- $\mathit{sz}_{\mathcal{E}}(\pi_1 + \pi_2) := 1 + \mathit{sz}_{\mathcal{E}}(\pi_1) + \mathit{sz}_{\mathcal{E}}(\pi_2)$,
- $\mathit{sz}_{\mathcal{E}}(\chi\ \pi) := \mathit{sz}_{\mathcal{E}}(\chi) + 1$,
- $\mathit{sz}_{\mathcal{E}}(\omega\ \pi) := \begin{cases} \mathit{sz}_{\mathcal{E}}(\omega) + 1 & \text{if } \omega \in \mathcal{HP}_s, \text{ or } \omega = x \in \mathcal{V}_{\mathrm{h}} \text{ and } \mathcal{E}_2(x) = 1, \\ \mathit{sz}_{\mathcal{E}}(\omega) + \mathit{sz}_{\mathcal{E}}(\pi) + 1 & \text{otherwise} \end{cases}$
- $\mathit{sz}_{\mathcal{E}}(q(\pi)) := 1$ if $q \in \mathcal{F} \cup \{\tilde{}\}$,
- $\mathit{sz}_{\mathcal{E}}(x) := \mathit{sz}_{\mathcal{E}}(\mathcal{E}_1(x)) + 1$,
- $\mathit{sz}_{\mathcal{E}}(\overline{C}(\pi)) := \mathit{sz}_{\mathcal{E}}(\mathcal{E}_1(\overline{C})[\pi]) + 1$,
- $\mathit{sz}_{\mathcal{E}}(\mathtt{H}) := \mathit{sz}_{\mathcal{E}}(\mathcal{E}_1(\mathtt{H})) + 1$, $\mathit{sz}_{\mathcal{E}}(\mathtt{C}(\pi)) := \mathit{sz}_{\mathcal{E}}(\mathcal{E}_1(\mathtt{C})[\pi]) + 1$,
- $\mathit{sz}_{\mathcal{E}}(\omega\mathtt{*}) := \mathit{sz}_{\mathcal{E}}(\omega) + 1$, and $\mathit{sz}_{\mathcal{E}}(\chi\mathtt{+}) := \mathit{sz}_{\mathcal{E}}(\chi) + 1$.

This mapping is well defined under our restrictions on pattern definitions and environments. We call $\mathit{sz}_{\mathcal{E}}(\pi)$ the *size* of π in $\mathcal{E}$. This function will be instrumental in proving termination of our procedure for solving the matching problems.

Standard forms. A matching problem P is in *standard form* (Smp for short) if it is of the form $\langle \omega_1 \ll v_1, \ldots, \omega_m \ll v_m \mid \mathcal{E} \rangle$ such that

- Every ω_i is of the form $e_1\ \ldots\ e_{n_i}$ with $e_j \in \mathcal{V}_s \cup \{\_\_\} \cup \{\overline{C}(\_\_) \mid \overline{C} \in \mathcal{V}_c\}$ for all $1 \leq j < n_i$, and $e_{n_i} \in \mathcal{V}_s \cup \{\_\_\_\}$,
- $\omega_1\ \ldots\ \omega_m$ is a linear pattern with $\mathit{vars}(\omega_1\ \ldots\ \omega_m) \cap \mathit{vRng}(\mathcal{E}) = \emptyset$, and
- $\_\_$ and $\_\_\_$ do not occur in $\mathcal{E}$.

To every matching problem $P = \langle \omega_1 \ll v_1, \ldots, \omega_m \ll v_m \mid \mathcal{E} \rangle$ we associate an Smp $\mathit{tr}(P) = \langle x_1 \ll v_1, \ldots, x_m \ll v_m \mid \mathcal{E}' \cup \bigcup_{i=1}^{m} \{(x_i\ \mathtt{in}\ \omega_i, 0)\} \rangle$ where the hedge variables $x_1, \ldots, x_m$ are distinct and fresh, and $\mathcal{E}'$ is obtained from $\mathcal{E}$ by replacing $\_\_$ with $\_\ \_*$, and $\_\_\_$ with $\_*$. Note that $\mathit{Sol}(P) = \{\theta|_{-\{x_1,\ldots,x_m\}} \mid \theta \in \mathit{Sol}(\mathit{tr}(P))\}$. Therefore, for solving matching problems it is sufficient to be able to solve Smps.

3 The Matching Algorithm

In Subsection 3.1 we present a transformation system $\mathcal{M}$ consisting of rules of the form $\langle P \mid L \rangle \Rightarrow_{\sigma} \langle P' \mid L' \rangle$ where P and P' are Smps, σ is a substitution, and L, L' are lists of values. To solve an Smp P_0, we search for all derivations

$$\langle P_0 \mid [\,] \rangle \Rightarrow_{\sigma_1} \langle P_1 \mid L_1 \rangle \Rightarrow_{\sigma_2} \ldots \Rightarrow_{\sigma_n} \langle P_n = \square_{\mathcal{E}} \mid L_n \rangle,$$

which we abbreviate by $\langle P_0 \mid [\,]\rangle \Rightarrow^n_\sigma \langle P_n \mid L_n\rangle$, or simply $\langle P_0 \mid [\,]\rangle \Rightarrow^*_\sigma \langle P_n \mid L_n\rangle$, where $[\,]$ is the empty list and σ is the substitution $\sigma_1 \cdots \sigma_n$. If $n > 0$ then we may also write $\langle P_0 \mid [\,]\rangle \Rightarrow^+_\sigma \langle P_n \mid L_n\rangle$. Also, we write $\mathit{Ans}(P_0)$ for the set of substitutions $\{\sigma\theta|_{vars(P_0)} \mid \langle P_0 \mid [\,]\rangle \Rightarrow^*_\sigma \langle \Box_{\mathcal{E}} \mid L\rangle \text{ and } \theta \in \mathit{Sol}(\mathcal{E})\}$, and call the elements of $\mathit{Ans}(P_0)$ *computed answers.*

In Sect. 3.2 we indicate the main properties of $\mathcal{M}$. First, we prove that this matching procedure is terminating. We do this by indicating a terminating order $\succ$ on Smp-s such that $P \succ P'$ whenever $\langle P \mid L\rangle \Rightarrow^+_\sigma \langle P' \mid L'\rangle$. Next, we show that $\mathcal{M}$ is sound and complete, and thus $\mathit{Ans}(P) = \{\theta|_{vars(P)} \mid \theta \in \mathit{Sol}(P)\}$.

3.1 System $\mathcal{M}$

System $\mathcal{M}$ consists of 22 transformation rules:

(t) *Trivial.*
$\langle\langle \varepsilon \ll \varepsilon, \varPhi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_\epsilon \langle\langle \varPhi \mid \mathcal{E}\rangle \mid L\rangle$.

(hve1) *Hedge variable elimination 1.*
$\langle\langle x\,\omega \ll v,\, \varPhi \mid \{(x \;\texttt{in}\; \omega_1, 0)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{x\mapsto\varepsilon\}} \langle\langle \omega \ll v,\, \varPhi \mid \mathcal{E}\sigma\rangle \mid L\rangle$
where $\omega_1 = \varepsilon$ or $\omega_1 = \omega_2{*}$.

(hve2) *Hedge variable elimination 2.*
$\langle\langle x\,\omega \ll f(v_1)\,v_2, \varPhi \mid \{(x \;\texttt{in}\; \_, \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{x\mapsto f(v_1)\}} \langle\langle \omega \ll v_2, \varPhi \mid \mathcal{E}\sigma\rangle \mid L\rangle$.

(hve3) *Hedge variable elimination 3.*
$\langle\langle x \;\; \omega \ll v, \varPhi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_\epsilon$
$\langle\langle x\,\omega \ll v,\, \varPhi \mid \{(x \;\texttt{in}\; \mathcal{E}_1(y), \max\{\mathcal{E}_2(x), \mathcal{E}_2(y)\})\} \cup \mathcal{E}|_{-\{x\}}\rangle \mid L\rangle$
if $\mathcal{E}_1(x) = y \in \mathcal{V}_\mathrm{h}$.

(hvi1) *Hedge variable imitation 1.*
$\langle\langle x\;\omega \ll f(v_1)\,v_2,\, \varPhi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{x\mapsto f(y)\}}$
$\langle\langle y \ll v_1, \omega \ll v_2, \varPhi \mid \mathcal{E}|_{-\{x\}}\sigma \cup \{(y \;\texttt{in}\; \omega_1, 0)\}\rangle \mid L\rangle$
where $\mathcal{E}_1(x) = q(\omega_1)$ with $q \in \{f, \tilde{}\}$, and $y \in \mathcal{V}_\mathrm{h}$ is fresh.

(hvi2) *Hedge variable imitation 2.*
$\langle\langle x\,\omega \ll\; v,\, \varPhi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{x\mapsto x_1\;\,x_2\}}$
$\langle\langle x_1\;\,x_2\;\,\omega \ll v, \varPhi \mid \mathcal{E}|_{-\{x\}}\sigma \cup \{(x_1 \;\texttt{in}\; \nu, \phi_1), (x_2 \;\texttt{in}\; \omega_1, \phi_2)\}\rangle \mid L\rangle$
where $\mathcal{E}_1(x) = \nu\;\omega_1$, $x_1, x_2 \in \mathcal{V}_\mathrm{h}$ are fresh and $\phi_1, \phi_2 \in \{0, 1\}$ such that $\phi_1 + \phi_2 = \mathcal{E}_2(x)$.

(hne1) *Name elimination for hedge pattern name.*
$\langle\langle x\;\omega \ll v, \varPhi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_\epsilon \langle\langle x\;\omega \ll v, \varPhi \mid \{(x \;\texttt{in}\, \mathcal{E}_1(\mathtt{N}), \mathcal{E}_2(x))\} \cup \mathcal{E}|_{-\{x\}}\rangle \mid L\rangle$
if $\mathcal{E}_1(x) = \mathtt{N}$.

(hne2) *Name elimination for compositional constraint of hedge variable.*
$\langle\langle x\;\omega \ll v,\, \varPhi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_\epsilon \langle\langle x\;\omega \ll v, \varPhi \mid \{(x \;\texttt{in}\; \chi[\overline{\omega}], \mathcal{E}_2(x))\} \cup \mathcal{E}|_{-\{x\}}\rangle \mid L\rangle$
if $\mathcal{E}_1(x) = \mathtt{C}(\overline{\omega})$ and $\mathcal{E}_1(\mathtt{C}) = \chi$.

(hvf1) *Flex constraint 1 of hedge variable.*
$\langle\langle x\;\omega \ll v, \varPhi \mid \{(x \;\texttt{in}\; \overline{C}(\overline{\omega}), \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma_1\cup\sigma_2}$
$\langle\langle x_1 \ll v_1, \ldots, x_n \ll v_n, \omega \ll v_{n+1}, \varPhi \mid \mathcal{E}'\sigma_2 \cup \bigcup_{i=1}^n \{(x_i \;\texttt{in}\; \overline{\omega}\sigma_1, 0)\}\rangle \mid L\rangle$
if $v \neq \varepsilon$, $\langle\langle \overline{C}(\_\_)\;\_\_\_ \ll v \mid \mathcal{E}\rangle \mid [\,]\rangle \Rightarrow^+_{\sigma_1} \langle \Box_{\mathcal{E}'} \mid [v_1, \ldots, v_n, v_{n+1}]\rangle$, $\sigma_1(\overline{C}) \neq \bullet$, and $\sigma_2 = \{x \mapsto \sigma_1(\overline{C})\langle x_1\;\ldots\;x_n\rangle\}$ with $x_1, \ldots, x_n \in \mathcal{V}_\mathrm{h}$ fresh.

(hvf2) *Flex constraint 2 of hedge variable.*

$\langle\langle x\,\omega \ll v,\, \mathit{\Phi} \mid \{(x\,\texttt{in}\,\overline{C}(\overline{\omega}), \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma}$

$\langle\langle x\,\omega \ll v, \mathit{\Phi} \mid \mathcal{E}' \cup \{(x\,\texttt{in}\,\overline{\omega}\sigma, 0)\}\rangle \mid L\rangle$

if $v \neq \varepsilon$, $\langle\langle\overline{C}(\_\_)\,\_\_\_ \ll v \mid \mathcal{E}\rangle \mid [\,]\rangle \Rightarrow^{+}_{\sigma} \langle\Box_{\mathcal{E}'} \mid L\rangle$, and $\sigma(\overline{C}) = \bullet$.

(hvr) *Repetition constraint of hedge variable.*

$\langle\langle x\;\omega \ll v, \mathit{\Phi} \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{x\mapsto x_1\,x_2\}}$

$\langle\langle x_1\,x_2\,\omega \ll v,\, \mathit{\Phi} \mid \mathcal{E}|_{-\{x\}}\sigma \cup \{(x_1\,\texttt{in}\,\omega_1, 1),\, (x_2\,\texttt{in}\,\omega_1\texttt{*}, 0)\}\rangle \mid L\rangle$

where $\mathcal{E}_1(x) = \omega_1\texttt{*}$, $v \neq \varepsilon$, and $x_1, x_2 \in \mathcal{V}_{\mathrm{h}}$ are fresh.

(cve) *Context variable elimination.*

$\langle\langle\overline{C}(\_\_)\,\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}\,\texttt{in}\,\bullet, 0)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{\overline{C}\mapsto\bullet\}} \langle\langle\_\_\;\omega \ll v, \mathit{\Phi} \mid \mathcal{E}\sigma\rangle \mid L\rangle$.

(cvi1) *Context variable imitation 1.*

$\langle\langle\overline{C}(\_\_)\,\omega \ll f(v_1)\;v_2,\, \mathit{\Phi} \mid \{(\overline{C}\,\texttt{in}\,q(\chi), \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{\overline{C}\mapsto f(\overline{C}_1(\bullet))\}}$

$\langle\langle\overline{C}_1(\_\_) \ll v_1, \omega \ll v_2, \mathit{\Phi} \mid \{(\overline{C}_1\,\texttt{in}\,\chi, 0)\} \cup \mathcal{E}\sigma\rangle \mid L\rangle$

if $q \in \{f,\, \tilde{}\,\}$ and $\overline{C}_1 \in \mathcal{V}_{\mathrm{c}}$ is fresh.

(cvi2) *Context variable imitation 2.*

$\langle\langle\overline{C}(\_\_)\,\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}\,\texttt{in}\,\xi\;\chi, \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{\overline{C}\mapsto\overline{C}_1(\bullet)\,\overline{C}_2(\bullet)\}}$

$\langle\langle\overline{C}_1(\_\_)\,\overline{C}_2(\_\_)\,\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}_1\,\texttt{in}\,\xi, 0), (\overline{C}_2\,\texttt{in}\,\chi, 0)\} \cup \mathcal{E}\sigma\rangle \mid L\rangle$

where $v \neq \varepsilon$, and $\overline{C}_1, \overline{C}_2 \in \mathcal{V}_{\mathrm{c}}$ are fresh.

(cvi3) *Context variable imitation 3.*

$\langle\langle\overline{C}(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}\,\texttt{in}\,\xi\;\omega_1, \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{\overline{C}\mapsto\overline{C}_1(\bullet)\,x\}}$

$\langle\langle\overline{C}_1(\_\_)\;x\;\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}_1\,\texttt{in}\,\xi, \phi_1),\, (x\,\texttt{in}\,\omega_1, \phi_2)\} \cup \mathcal{E}\sigma\rangle \mid L\rangle$

$\langle\langle\overline{C}(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}\,\texttt{in}\,\nu\;\chi, \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{\overline{C}\mapsto x\,\overline{C}_1(\bullet)\}}$

$\langle x\;\overline{C}_1(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}_1\,\texttt{in}\,\chi, \phi_1),\, (x\,\texttt{in}\,\nu, \phi_2)\} \cup \mathcal{E}\sigma\rangle \mid L\rangle$

where $v \neq \varepsilon$, $\overline{C}_1$ and x are fresh, and $\phi_1, \phi_2 \in \{0, 1\}$ such that $\phi_1 + \phi_2 = \phi$.

(cne) *Name elimination for compositional constraint of context variable.*

$\langle\langle\overline{C}(\_\_)\;\omega \ll v,\, \mathit{\Phi} \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\epsilon}$

$\langle\langle\overline{C}(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}\,\texttt{in}\,\chi_1[\chi], \mathcal{E}_2(\overline{C}))\} \cup \mathcal{E}|_{-\{\overline{C}\}}\rangle \mid L\rangle$

if $\mathcal{E}_1(\overline{C}) = \texttt{C}(\chi)$ and $\mathcal{E}_1(\texttt{C}) = \chi_1$.

(cvf1) *Flex constraint 1 of context variable.*

$\langle\langle\overline{C}(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}\,\texttt{in}\,\overline{C}_0(\chi), \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma_1\cup\sigma_2}$

$\langle\langle\overline{C}_1(\_\_) \ll v_1, \ldots, \overline{C}_n(\_\_) \ll v_n, \omega \ll v_{n+1}, \mathit{\Phi} \mid \mathcal{E}'\sigma_2 \cup$

$\bigcup_{i=1}^{n}\{(\overline{C}_i\,\texttt{in}\,\chi\sigma_1, 0)\}\rangle \mid L\rangle$

if $v \neq \varepsilon$, $\langle\langle\overline{C}_0(\_\_)\;\_\_\_ \ll v \mid \mathcal{E}\rangle|[\,]\rangle \Rightarrow^{+}_{\sigma_1} \langle\Box_{\mathcal{E}'} \mid [v_1, \ldots, v_n, v_{n+1}]\rangle$, $\sigma_1(\overline{C}_0) \neq \bullet$,
and $\sigma_2 = \big\{\overline{C} \mapsto \sigma(\overline{C}_0)\langle\overline{C}_1(\bullet)\;\ldots\;\overline{C}_n(\bullet)\rangle\big\}$ with $\overline{C}_1, \ldots, \overline{C}_n$ fresh.

(cvf2) *Flex constraint 2 of context variable.*

$\langle\langle\overline{C}(\_\_)\,\omega \ll v,\, \mathit{\Phi} \mid \{\overline{C}\,\texttt{in}\,(\overline{C}_0(\chi), \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma}$

$\langle\langle\overline{C}(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \mathcal{E}' \cup \{(\overline{C}_0\,\texttt{in}\,\chi\sigma, \phi)\}\rangle \mid L\rangle$

if $v \neq \varepsilon$, $\langle\langle\overline{C}_0(\_\_)\;\_\_\_ \ll v \mid \mathcal{E}\rangle|[\,]\rangle \Rightarrow^{+}_{\sigma} \langle\Box_{\mathcal{E}'} \mid L\rangle$, and $\sigma(\overline{C}_0) = \bullet$.

(cvr1) *Repetition constraint 1 of context variable.*

$\langle\langle\overline{C}(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \;\{\overline{C}\,\texttt{in}\,(\chi\texttt{+}, \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\epsilon}$

$\langle\langle\overline{C}(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}\,\texttt{in}\,\chi, \phi)\} \cup \mathcal{E}\rangle \mid L\rangle$

(cvr2) *Repetition constraint 2 of context variable.*

$\langle\langle\overline{C}(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \{\overline{C}\,\texttt{in}\,(\chi\texttt{+}, \phi)\} \uplus \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{\overline{C}\mapsto\overline{C}_1(\bullet)\,\overline{C}_2(\bullet)\}}$

$\langle\langle\overline{C}_1(\_\_)\;\overline{C}_2(\_\_)\;\omega \ll v, \mathit{\Phi} \mid \{(\overline{C}_1\,\texttt{in}\,\chi, 0), (\overline{C}_2\,\texttt{in}\,\chi\texttt{+}, 0)\} \cup \mathcal{E}\sigma\rangle \mid L\rangle$

where $\overline{C}_1, \overline{C}_2 \in \mathcal{V}_{\mathrm{c}}$ are fresh.

(any0) *Elimination of hedge wildcard.*
$\langle\langle \_\_\_ \ll v, \Phi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_\epsilon \langle\langle \Phi \mid \mathcal{E}\rangle \mid L{:}v\rangle$
(any1) *Elimination of strict hedge wildcard.*
$\langle\langle \_\_\ \omega \ll v_1\ v_2, \Phi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_\epsilon \langle\langle \omega \ll v_2, \Phi \mid \mathcal{E}\rangle \mid L{:}v_1\rangle$
where $v_1 \neq \varepsilon$.

3.2 Main properties

An easy proof by case distinction on the inference step shows that:

- If P is an Smp and $\langle P \mid L\rangle \Rightarrow_\sigma \langle P' \mid L'\rangle$ then P' is an Smp.
- If $\langle\langle \Phi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow^*_\sigma \langle\langle \Phi' \mid \mathcal{E}'\rangle \mid L'\rangle$ then $\mathit{defs}(\mathcal{E}') = \mathit{defs}(\mathcal{E}\sigma)$.

In the sequel we show that $\mathcal{M}$ is terminating, i.e., there are no infinite derivations of Smps. We start with some useful auxiliary notions.

Let $P = \langle \omega_1 \ll v_1, \ldots, \omega_m \ll v_m \mid \mathcal{E}\rangle$ be an Smp. Then $\omega_1\ \ldots\ \omega_m$ is a linear pattern of the form $e_1\ \ldots\ e_n$ with $e_i \in \mathcal{V}_\mathrm{h} \cup \{\_\_, \_\_\_\} \cup \{\overline{C}(\_\_) \mid \overline{C} \in \mathcal{V}_c\}$. In this case, we define

- $\mathit{idx}(P) := \min(\{n\} \cup \{k \mid (e_k = \_\_) \vee (e_k = x \wedge (x\ \mathtt{in}\ \omega, 1) \in \mathcal{E}) \vee (e_k = \overline{C}(\_\_))\})$
- the *pattern size* of P is the multiset

$$\begin{aligned} \mathit{sz}(P) := &\ \{\mathit{sz}_\mathcal{E}(e_i) \mid 1 \leq i \leq \mathit{idx}(P) \wedge e_i \in \mathcal{V}_\mathrm{h} \cup \{\_\_\}\} \uplus \\ &\ \{\mathit{sz}_\mathcal{E}(\overline{C}) \mid 1 \leq i \leq \mathit{idx}(P) \wedge e_i = \overline{C}(\_\_)\} \end{aligned}$$

where $\uplus$ stands for multiset union, and $\mathit{sz}_\mathcal{E}(\pi)$ is as defined on page 8. We define

1. $m_1(P) :=$ the multiset $\{|v_i| \mid 1 \leq i \leq m \wedge v_i \neq \varepsilon\}$.
2. $m_2(P) := \mathit{sz}(P)$,
3. $m_3(P) :=$ number of occurrences of $\_\_\_$ in $\omega_1\ \ldots\ \omega_m$, and
4. $m_4(P) := m$.

We define a partial order $\succ$ on the states of the transformation system $\mathcal{M}$ as follows: $\langle P \mid L\rangle \succ \langle P' \mid L'\rangle$ iff $P \succ^{\mathrm{lex}}_{1,2,3,4} P'$ where $\succ^{\mathrm{lex}}_{1,2,3,4}$ is the lexicographic combination of the orderings induced by m_1, m_2, m_3, m_4 on Smps. Since $\succ$ is a lexicographic combination of terminating orders, $\succ$ is terminating too.

Theorem 1. *If $\langle P \mid L\rangle \Rightarrow^n_\theta \langle P' \mid L'\rangle$ with $n > 0$ then $P \succ P'$.*

Proof. By induction on the lexicographic ordering on $\langle m(P), n\rangle$.

Obviously, it is enough to prove that if $\langle P \mid L\rangle \Rightarrow_\sigma \langle P' \mid L'\rangle$ then $P \succ P'$. We proceed by case distinction on the rule used in the transformation step. The following table indicates what happens when the transformation rule is not from {hvf1, hvf2, cvf1, cvf2}.

transformation rule	m_1	m_2	m_3	m_4
hve2,hvi1,cvi1,any1	$P >_1 P'$			
cve,hve1,hve3,hvi2, hne1,hne2,hvr,cvi2,cvi3, cne,cvr1,cvr2	$P \geq_1 P'$	$P >_2 P'$		
any0	$P \geq_1 P'$	$P =_2 P'$	$P >_3 P'$	
t	$P =_1 P'$	$P =_2 P'$	$P =_3 P'$	$P >_4 P'$

If the transformation rule is hvf1 or hvf2 then P is of the form

$$\langle x\ \omega \ll v, \Phi \mid \{x\ \mathtt{in}\ (\overline{C}(\omega_1), \phi)\} \uplus \mathcal{E}\rangle.$$

Since $\langle \overline{C}(\_\_)\ \_\_\_ \ll v \mid \mathcal{E}\rangle \prec P$, we can apply the induction hypothesis to conclude the non-existence of infinite derivations starting from $\langle \overline{C}(\_\_)\ \_\_\_ \ll v \mid \mathcal{E}\rangle$.

If $\sigma(\overline{C}) \neq \bullet$ then the transformation rule is hvf1 and $n > 1$. Moreover, $\sum_{i=1}^{n+1} |v_i| \leq |v|$, and since $|v_j| > 0$ for all $1 \leq j \leq n$, we conclude that $m_1(P) > m_1(P')$ and thus $P \succ P'$. If $\sigma(\overline{C}) = \bullet$ then the transformation rule is hvf2 and $n = 1$. It is not hard to conclude that in this case $\sigma(X) \in \{\bullet, \varepsilon\}$ for all $X \in dom(\sigma)$, and then $m_1(P) = m_1(P')$ and $m_2(P) > m_2(P')$.

If the transformation rule is cvf1 or cvf2 then P is of the form

$$\langle \overline{C}(\_\_)\ \omega \ll v, \Phi \mid \{\overline{C}\ \mathtt{in}\ (\overline{C}_0(\chi), \phi)\} \uplus \mathcal{E}\rangle.$$

Since $\langle \overline{C}_0(\_\_)\ \_\_\_ \ll v \mid \mathcal{E}\rangle \prec P$, we can apply the induction hypothesis and conclude the non-existence of infinite derivations starting from $\langle \overline{C}_0(\_\_)\ \_\_\_ \ll v \mid \mathcal{E}\rangle$.

If $\sigma(\overline{C}) \neq \bullet$ then the transformation rule is cvf1 and $n > 1$. Moreover, $\sum_{i=1}^{n+1} |v_i| \leq |v|$, and since $|v_j| > 0$ for all $1 \leq j \leq n$, we learn that $m_1(P) > m_1(P')$ and thus $P \succ P'$. If $\sigma(\overline{C}) = \bullet$ then the transformation rule is cvf2 and $n = 1$. It is not hard to conclude that in this case $\sigma(X) \in \{\bullet, \varepsilon\}$ for all $X \in dom(\sigma)$, and then $m_1(P) = m_1(P')$ and $m_2(P) > m_2(P')$. $\Box$

Soundness, i.e, the fact that $Ans(P) \subseteq Sol(P)$ for any Smp P, is an immediate corollary of the following theorem, which can be proved by induction on the lexicographic ordering on $\langle m_1(P), m_2(P), m_3(P), m_4(P), n\rangle$.

Theorem 2. *If $\langle P \mid L\rangle \Rightarrow_\theta^n \langle P' \mid L'\rangle$ and $\theta' \in Sol(P')$ then $\theta\theta' \in Sol(P)$.* $\Box$

Finally, we note that $\mathcal{M}$ is complete too, i.e., that $\{\theta|_{vars(P)} \mid \theta \in Sol(P)\} \subseteq Ans(P)$ for any Smp P. This is a consequence of Theorem 2 and of the following lemma.

Lemma 2. *If L is a list of values, P is an* Smp *with non-empty equational part, and $\theta \in Sol(P)$ then there exists a reduction step $\langle P \mid L\rangle \Rightarrow_\sigma \langle P' \mid L'\rangle$ such that $\theta = \sigma\theta'\ [dom(\theta)]$ for some $\theta' \in Sol(P')$.* $\Box$

4 Conclusion and Future Work

We have proposed a class of matching problems with regular patterns with variables for hedges and contexts, and membership constraints to restrict the admissible bindings of variables by matching. We have identified a sound and complete calculus to solve this kind of matching problems. The calculus is presented as a collection of 22 transformation rules that are used to compute all maximal derivations $\langle P \mid [\,]\rangle \Rightarrow_{\sigma_1} \ldots \Rightarrow_{\sigma_n} \langle \Box_\mathcal{E} \mid [\,]\rangle$, abbreviated $\langle P \mid [\,]\rangle \Rightarrow_\sigma^* \langle \Box_\mathcal{E} \mid [\,]\rangle$, where $\sigma = \sigma_1 \ldots \sigma_n$ and $\Box_\mathcal{E}$ is a Smp without equations. We proved that $Sol(P) = \{\theta\theta'|_{vars(P)} \mid \langle P \mid [\,]\rangle \Rightarrow_\theta^* \langle \Box_\mathcal{E} \mid [\,]\rangle \wedge \theta' \in Sol(\mathcal{E})\}$, and that the satisfiability of $\Box_\mathcal{E}$ is decidable. Thus, we can regard our calculus as a kind of preunification algorithm, because it does not attempt to solve the residual set of membership constraints, whose satisfiability is decidable.

The matching calculus is inherently nondeterministic, since our matching problems can have more than one solution. It is also the case that our calculus may compute some solutions more than once. The culprits for this redundancy are the definitions of rules hvi2, and cvi3 when the flag of the outermost variable of the leftmost equation is $\phi = 1$. In this case, both hvi2 and cvi3 have the choice to proceed either with (a) $\phi_1 = 1$, $\phi_2 = 0$, or (b) $\phi_1 = 0$, $\phi_2 = 1$. Unfortunately, these choices are not disjoint, i.e., they may yield same solution. This redundancy can be avoided if we collapse these two subcases of hvi2 and cvi3 into one case where $\phi_1 = 0$ and the value of flag ϕ_2 is established only *after* we compute the binding v_1 of the variable corresponding to flag ϕ_1: If $v_1 \in \{\varepsilon, \bullet\}$ then ϕ_2 should be 1, otherwise it should be 0. We illustrate how this strategy works by replacing hvi2 with

(hvi'2) *hedge variable imitation 2.*

$$\langle\langle x\,\omega \ll v, \Phi \mid \mathcal{E}\rangle \mid L\rangle \Rightarrow_{\sigma=\{x\mapsto x_1\ x_2\}} \langle\langle x_1\ x_2\ \omega \ll v, \Phi \mid \{x_1 \,\mathtt{in}\,(\nu, 0), x_2 \,\mathtt{in}\,(\omega_1, 1 - \mathrm{value}_{\mathcal{E}_2(x)}(x_1))\} \cup \mathcal{E}|_{-\{x\}}\sigma\rangle \mid L\rangle$$

where $\mathcal{E}_1(x) = \nu\,\omega_1$, $x_1, x_2 \in \mathcal{V}_\mathrm{h}$ are fresh,

and defining $\mathrm{value}_k(v) = \begin{cases} 0 \text{ if } k = 1 \text{ and } v = \varepsilon, \\ 1 \text{ otherwise.} \end{cases}$

As future work, we intend to integrate the pattern matching constructs and the algorithm described here with the programming language ρLog [8, 9].

References

1. Veronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: An XML-centric general-purpose language. In *Proceedings of 2003 ACM SIGPLAN International Conference on Functional Programming*. ACM Press, 2003.
2. Hubert Comon. Completion of Rewrite Systems with Membership Constraints. Part I: Deduction Rules. *Journal of Symbolic Computation*, 25(4):397–419, April 1998.
3. Hubert Comon. Completion of Rewrite Systems with Membership Constraints. Part II: Constraint Solving. *Journal of Symbolic Computation*, 25(4):421–453, April 1998.
4. Haruo Hosoya. Regular Expression Pattern Matching—A Simpler Design. Technical Report 1397, RIMS, Kyoto University, 2003.
5. Haruo Hosoya and Benjamin Pierce. Regular expression pattern matching for XML. *Journal of Functional Programming*, 13(6):961–1004, 2002.
6. Temur Kutsia and Mircea Marin. Matching with regular constraints. In Geoff Sutcliffe and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning. Proceedings of the 12th International Conference, LPAR'05*, volume 3835 of *LNAI*, pages 215–229, Montego Bay, Jamaica, December 2–6 2005. Springer Verlag.
7. Temur Kutsia and Mircea Marin. Solving regular constraints for hedges and contexts. In Jordi Levy, editor, *Proceedings of the 20th International Workshop on Unification (UNIF'06)*, pages 89–107, Seattle, USA, August 11 2006.

8. Mircea Marin and Tetsuo Ida. Progress of ρLog, a rule-based programming system. *Mathematica in Education and Research*, 11(1):50–66, 2006.
9. Mircea Marin and Temur Kutsia. Foundations of the Rule-Based System ρLog. *Journal of Applied Non-Classical Logic*, 16(2):151–168, 2006.
10. W3C Recommendation. XQuery 1.0: An XML Query Language, January 2007. Available online at `http://www.w3.org/TR/xquery/`.
11. Randal L. Schwartz, Tom Phoenix, and brian d foy. *Learning Perl*. O'Reilly, fourth edition, 2005.
12. Stephen Wolfram. *The Mathematica Book*. Wolfram Media, fifth edition, 2003.

A tractable logic of types

Allan Ramsay

School of Computer Science
University of Manchester
Manchester M60, 1 QD

Abstract. Numerous approaches to managing sort hierarchies have been proposed. Some are tractable but have limited expressive power (e.g. encoding sorts as bit strings [Fall, 1990]), others have considerable expressive power but are potentially very complex (e.g. description logics [Baader and Sattler, 2001]). The current paper proposes a logic of sorts which allows reasoning over semi-partitioned type lattices, so is more expressive than simple type hierarchies; but whose worst case complexity is linear in the size of the query, and hence is in general more tractable than description logic.
The key step in implementing this logic is that we represent sorts as paths through a lattice: in the simplest case, checking that two sorts are compatible simply involves checking that the paths that make them up unify. Managing negation is rather trickier, and requires us to plant constraints that are invoked when, and only when, the relevant paths are checked. This makes the algorithm more awkward, but does not increase the (theoretical or practical) complexity of the process.

1 Introduction

Many tasks require you to assign types to entities, and then to see whether one type is consistent with another or is subsumed by another. The work described here was developed for use within a natural language system where it is useful to be able to assign types to individuals and then to see whether those individuals are suitable candidates for various semantic roles. This can be used for choosing between different senses of a word, as in (1), or for choosing between prepositional attachment sites, as in (2).

(1) a. John fired his secretary.
    b. John fired a gun at his secretary.
(2) a. I cut the bread with the bread knife.
    b. I cut the bread with the poppy seeds on the crust.

There are numerous other application areas where it is useful to be able to reason about types. The discussion below will generally be illustrated by linguistically motivated examples, because that is the context in which this work was originally carried out, but the approach can be applied anywhere that reasoning about types is useful.

There are two crucial operations on types: are two types compatible, and does one type subsume another? The complexity of computing these relations depends on the way that type descriptions are specified. If, for instance, you decide that types form a strictly partitioned hierarchy then you can encode an individual type as a bit string, and you can compute compatibility and subsumption by simple operations on bit strings. For large hierarchies you may have to be clever about the encoding, but there are ways of computing compatibility and subsumption in strictly partitioned hierarchies that are (time and space) linear in the depth of the hierarchy. If, on the other hand, you decide to allow arbitrary Boolean combinations of types then your logic of types becomes propositional logic, which is NP-complete and hence is in principle intractable. That is not to say that there are no efficient theorem provers for propositional logic, but the potential requirement for large amounts of search means that in at least some cases theorem provers for full propositional logic can take a long time. The current paper introduces an intermediate level of expressivity, allowing type descriptions to consist of conjunctions of positive and negative literals drawn from entries in a semi-partitioned type lattice. This has considerably more expressive power than a simple partitioned type hierarchy, but the time and space complexity is linear in the depth of the hierarchy and the number of conjuncts in the types being compared (note: not the number of types in the hierarchy). This is in principle more tractable than propositional logic or description logic, which are both NP-complete, and in practice is extremely fast. Furthermore operations on types can easily be embedded within ordinary logic programs, so that it is easy to combine efficient reasoning over types with the full power of Horn-clause logic where that is required.

2 Semi-partitioned type lattices

The logic of types proposed here comes in two parts. The first involves specifying a set of basic types as a semi-partitioned type lattice. Once you have this you can then construct complex type descriptions by constructing conjunctions of positive and negative literals over the set of basic types. The algorithms for computing consistency and subsumption are defined for complex types.

2.1 Syntax

Basic types and constants: we use $t, t_1, t_2, \dots$ as names of basic types, and $a, a_1, a_2, \dots.$ as constants.

Partitions: (i) if $t, t_1, \dots, t_n$ are types then $t >>> [t_1, \dots, t_n]$ says that the t_i are a partition of t, i.e. that they are a disjoint set of sets whose union is t. (ii) if $t, t_1, \dots, t_n$ are types then $\neg t >>> [t_1, \dots, t_n]$ says that the t_i are a partition of the complement of t.

Subsets: (i) if $t, t_1, \dots, t_n$ are types then $t >> [t_1, \dots, t_n]$ says that each t_i is a subtype of t. In this case there is no expectation that t_i and t_j will be incompatible if i and j are different, nor that they completely cover t. (ii) if

$t, t_1, \ldots, t_n$ are types then $\neg t >> [t_1, \ldots, t_n]$ says that each t_i is a subset of the complement of t, so that t and t_i are disjoint for all i.

A set of subset and partition specifications describes a semi-partitioned type lattice. We get a type lattice rather than a hierarchy because a type may be a child of more than one parent; and the lattice is semi-partitioned because for some nodes the children form a partition and for others they do not. The collection of subset and partition specifications defines an obvious partial order on terms. We will write $t_1 <_L t_2$ if t_1 is below t_2 in the order defined by L (if the lattice in question is obvious we will just write $t_1 < t_2$). We will use this partial order when talking about formulae: it is not part of language of types itself.

Once we have specified a set of types we can assign individuals to types and we can construct complex types.

1. If t is a basic type then t and $\neg t$ are types.
2. If t_1 and t_2 are types then so is $t_1 \& t_2$.
3. If a is an individual and t is a type, then $(a \in t)$ is a formula whose intended meaning is that t is of type t.

Note that we do not allow negations of arbitrary types: $(\neg t_1 \& \neg t_2)$ is a well-defined type, $\neg(t_1 \& t_2)$ is not. The reason is that allowing negations of arbitrary types introduces disjunction into the logic, and introducing disjunction introduces choice, and hence complexity, into the algorithms for computing consistency and subsumption. However, the use of partitions allows us to achieve very much the same end. If the specification of the type lattice contains the entry $t_3 >> [t_1, t_2]$ then $((a \in t_1) or (a \in t_2))$ entails $(a \in t_3)$. Thus careful construction of the type lattice allows us to specify disjunctive basic types. All that is banned is the construction of disjunctive types in type assignment statements.

2.2 Semantics

The interpretation of this language is very simple. An interpretation I of a type lattice consists of a set $\mathcal{U}$ of individuals and an assignment of members of $\mathcal{U}$ to constants and subsets of $\mathcal{U}$ to types, where we will write $I(x)$ to denote the member or subset of $\mathcal{U}$ assigned to the term x. I is a model of a type lattice if the following conditions hold:

1. $I \models (a \in t)$ if $I(a) \in I(t)$.
2. $I(\neg t) = U \backslash I(t)$ for every type t.
3. $I \models (t' >> [\ldots, t, \ldots])$ if $I(t) \subseteq I(t')$.
4. $I \models (t >>> [\ldots, t_i, \ldots, t_j, \ldots])$ if $I(t_i) \subseteq I(t)$, $I(t_j) \subseteq I(t)$ and $I(t_i) \cap I(t_j) = \emptyset$.

3 Consistency and subsumption

3.1 Simple positive partitioned trees

We will start by considering the simplest case, namely where the lattice is in fact a fully partitioned tree with no negations, e.g. Fig. 1.

$concrete >> [living]$
$living >> [animal, plant, bacterium]$
$plant >> [fruit, vegetable, grass]$
$animal >> [bird, reptile, fish, insect, mammal]$
$mammal >> [cat, dog, ape]$
$ape >> [monkey, orangutang, human]$
$human >> [man, woman, child]$

Fig. 1. Positive partitioned tree

For any type in this tree, we can construct a path up to the top node, for instance the path from *man* to the top is $man \rightarrow human \rightarrow ape \rightarrow mammal \rightarrow animal \rightarrow living$. We represent such paths as Prolog open lists[1], writing them in reverse order and adding a truth value (`yes` or `no`) at each point. Thus the Prolog term representing this path is `[living=yes, animal=yes, mammal=yes, human=yes, man=yes | _]`. We will refer to the Prolog term corresponding to the type t as its DESCRIPTOR, written $d(t)$, and for an element of a path such as `living=yes` we will refer to the left-hand side as the KEY and the right-hand side as the VALUE.

It is easy to see that two types t and t' in a simple tree of this kind are compatible (i.e. if $I(t) \cap I(t') \neq \emptyset$) iff the corresponding paths are unifiable: (i) Suppose $d(t)$ and $d(t')$ are unifiable. Then one of them must be an extension of the other. Suppose wlog that $d(t)$ is an extension of $d(t')$. Then there must be a sequence of partitions $t' >> [\ldots, t_1, \ldots], t_2 >> [\ldots, t_3, \ldots], \ldots, t_n >> [\ldots, t, \ldots]$, in which case $I(t) \subseteq I(t')$, so if $I(t) \neq \emptyset$ then $I(t) \cap I(t') \neq \emptyset$. (ii) Suppose that $d(t)$ and $d(t')$ are not unifiable. Then there must be some point at which $d(t)$ contains `...,t0=yes,t1=yes,...` and $d(t')$ contains `...,t0=yes,t2=yes,...`, where `t1` and `t2` are different. This will have arisen because the lattice contains a partition $t_0 >> [\ldots, t_1, \ldots, t_2, \ldots]$, where t is below t_1 and t' is below t_2. But $I(t_1) \cap I(t_2) = \emptyset$, since $t_0 >> [\ldots, t_1, \ldots, t_2, \ldots]$ is a partition, and thence $I(t) \cap I(t') = \emptyset$ because $I(t) \subseteq I(t_1)$ and $I(t') \subseteq I(t_2)$. We will write $t_1 \approx t_2$ to say that t_1 and t_2 are compatible.

3.2 Partitioned lattices

In the example in Fig. 1, there was only one type with no supertypes, namely *living*. We can allow for multiple maximal elements if we insist that the top item in a path has a non-variable key (though the value does not have to be instantiated). Suppose, for instance, we extend Fig. 1 as in Fig. 2.

Fig. 2 has two new maximal elements, *male* and *adult*. There are therefore three paths leading up from *man*, namely $man \rightarrow ape \rightarrow mammal \rightarrow animal \rightarrow living \rightarrow concrete$, $man \rightarrow male$ and $man \rightarrow adult$.

[1] From here on, we will use `fixed width expressions` to denote Prolog terms, following the usual Prolog conventions for lists and variables.

$concrete >> [living]$
$living >> [animal, plant, bacterium]$
$male >> [man]$
$animal >> [bird, reptile, fish, insect, mammal]$
$mammal >> [cat, dog, ape]$
$ape >> [monkey, orangutang, human]$
$adult >> [man, woman]$
$human >> [man, woman, child]$

Fig. 2. Positive partitioned lattice

We represent each of these by a list as before, and we collect them together into a single list orded by the key of the top item in the path, so that $d(man)$ is now as shown in Fig. 3.

```
[[adult=yes,man=yes|A],
 [concrete=yes,living=yes,animal=yes,mammal=yes,ape=yes,human=yes,man=yes|B],
 [male=yes,man=yes|C]]
```

Fig. 3. Descriptor from positive partitioned lattice

The maximal elements in such a lattice each define a tree, so a set of paths drawn from a lattice actually correspond to points in a set of trees. Two sets of paths are thus compatible if wherever there are two paths with the same starting point they are pairwise compatible. If the sets of paths are ordered by the type of the top item then looking for pairs with the same head can be done in time proportional the sum of the lengths of the two lists (see Fig. 7). We can thus check compatibility of two types t_1 and t_2 drawn from a lattice extremely fast by comparing elements of $d(t_1)$ and $d(t_2)$ with matching heads, as above: the fact that $d(t_1)$ and $d(t_2)$ makes the search for paths with matching heads linear.

3.3 Negation

We would like to be able to add negative types into our lattices–to add, for instance, $\neg male >> [woman]$ and $\neg adult >> [child]$ to the lattice in Fig. 4.

$concrete >> [living]$
$living >> [animal, plant, bacterium]$
$male >> [man]$
$\neg male >> [woman]$
$animal >> [bird, reptile, fish, insect, mammal]$
$mammal >> [cat, dog, ape]$
$ape >> [monkey, orangutang, human]$
$adult >> [man, woman]$
$\neg adult >> [child]$
$human >> [man, woman, child]$

Fig. 4. Lattice with partitions and negation

The obvious way to do this is to allow `no` instead of `yes` as a value where required, as in the descriptor for *woman* in Fig. 5.

```
[[adult=yes,woman=yes|A],
 [concrete=yes,living=yes,animal=yes,mammal=yes,ape=yes,human=yes,woman=yes|B],
 [male=no,woman=yes|C]]
```

Fig. 5. Descriptor from partitioned lattice with negation

Then clearly *woman* $\not\approx$ *male*, since $d(woman)$ contains `[male=no,woman=yes|_]` and $d(male)$ contains `[male=yes|_]`.

This simple extension to the basic approach is not quite sufficient. Consider the type $\neg man$. This type will be compatible with any type that does not have `man=yes` at position 5 in its path, i.e. it would be compatible with `[living=yes, animal=yes, mammal=yes, human=yes, cat=yes | _]`, or with `[living=yes, animal=yes, fish=yes | _]`, or with `[living=yes, animal=yes, mammal=yes, human=yes, man=no | _]`. We therefore set the path for $\neg man$ to be `[_, _, _, _, X=V | _]` and we add the dynamic constraint `when(nonvar(X), \+ (X=man, V=yes))`. Dynamic constraints are activated when the condition part is satisfied, so this constraint will be executed if we try to unify this list with a list containing at least five members. Thus the path for $\neg man$ will unify with the path for *cat*: unifying `cat=yes` with `X=V` will trigger the constraint, with `X=cat` and `V=yes`, which will satisfy `\+ (X=man, V=yes))`. Unifying it with the path for *man* will fail, because the constraint will be triggered with `X=man` and `V=yes`, which will fail. It is important to note that attaching dynamic constraints to variables in this way adds a small constant time to the cost of unification, but it does not increase its complexity.

Note that the ability to define negated types lets us check whether t_1 subsumes another t_2 by seeing whether t_2 and $\neg t_1$ are compatible so that there is no need to define a separate algorithm for checking subsumption.

3.4 Partitions and subsets

We finally allow specification of non-partitioned subsets as well as partitions. The key issue here is that if $t_1 >>> [\ldots, t_0, \ldots]$ specifies that t_0 is a subset of t_1 and t_1 is a member of some partition then t_0 will also be governed by the partition. When constructing descriptors for types that are subtypes of other types, we have to be careful to include any partitions that involve the supertypes. This can easily lead to having multiple paths with the same head. Suppose, for instance, we change Fig. 4 to Fig. 6, where *adult* and *male* are subsets of *living*.

$concrete >> [living]$
$living >> [animal, plant, bacterium]$
$living >>> [male, adult]$
$male >> [man]$
$\neg male >> [woman]$
$animal >> [bird, reptile, fish, insect, mammal]$
$mammal >> [cat, dog, ape]$
$ape >> [monkey, orangutang, human]$
$adult >> [man, woman]$
$\neg adult >> [child]$
$human >> [man, woman, child]$

Fig. 6. Lattice with partitions, subsets, negation

Then *d*(*male*)=`[[concrete=yes, living=yes|_], [male=yes|_]]` and *d*(*human*)=`[[concrete=yes, living=yes, animal=yes, mammal=yes, ape=yes, human=yes|_]]`. So since *man* is covered by both *male* and *human*, the description of *man* has to include two paths with the same head. Under these circumstances we simply unify the relevant paths. Consider, for instance the type *living*&¬*human*. *d*(*living*&¬*human*) is `[[concrete=yes, living=yes, _, _, _, _G=_H|_]]` with the constraint `\+ (_G=human,\+_H=no)` attached to `_G`.

The observation that we can deal with descriptors that contain multiple paths with the same head by unifying them leads to a straightforward way of handling conjunctive types. We simply weave them together, preserving the order on paths, and where two paths have the same head we unify them, so that *d*(*cat*&*male*), for instance, becomes `[[concrete=yes, living=yes, animal=yes, mammal=yes, cat=yes | _], [male=yes | _]]`. We can do this when we construct a type, but we can also do it on the fly as we discover new facts about an individual during the course of other processing.

4 Complexity and performance

There are two issues to be considered here. How long does it take to construct descriptors, and how long does it take to compare descriptors? Of these the second is more important, since type descriptors for basic positive and negative types can be constructed at compile time if necessary (as happens with the T-box element of a description logic). Nonetheless it is worth considering both cases.

4.1 Constructing descriptors

To construct a descriptor for a type, we have to find the partitions and subsets that it is part of, and the partitions and subsets that these are part of, and ... To do this we first split a partition $t >> [t_1, \ldots, t_n]$ into a set of Prolog facts of the form `strictParent(t1, yes, t, yes), ...strictParent(tn, yes, t, yes)` (where `yes` would be replaced by `no` if t or t_i were negated), and likewise for subsets. Then the use of first argument indexing on Prolog facts means that finding each parent of a type takes a roughly constant amount of time (around 10^{-5} seconds), so that finding all the paths leading up from a type takes a time proportional to the number of types involved. The time taken for sorting the paths leading up from a term is $log_2(N)$ where N is the number of paths, which will not generally be very large. There is also a small cost for merging paths with the same head, but this is essentially negligible (unification of two lists with N members takes around $N \times 10^{-6}$ seconds). Thus constructing a type descriptor can be done fairly quickly (around 10^{-4} seconds for $d(man)$ with the lattice in Fig. 6, and less for simpler types with the same lattice). The time increases roughly linearly with the size of the descriptor, but is unaffected by the overall size of the lattice. In particular, the overall size of the lattice is not a factor, so there is no problem with having very large lattices.

4.2 Comparing descriptors

Comparing the descriptors for two types t_1 and t_2 involves essentially the same algorithm as merging paths with the same head. The algorithm is as in Fig. 7.

At worst this involves $max(length(d(t_1)), length(d(t_2))$ path unifications, where each unification is, as noted above, linear in the length of the path. For the examples above this ranges between 2×10^{-5} seconds (for comparing $d(cat)$ and $d(dog)$) and 5×10^{-5} seconds (for $d(man)$ and $d(\neg male)$). The time for comparing descriptors depends solely on the size of the descriptors–the size of the lattice from which they were derived is irrelevant.

```
$D_1 = d(t_1), D_2 = d(t_2)$
until $D_1 == []$ or $D_2 == []$
        $P_1 = hd(D_1), P_2 = hd(D_2)$
        $k_1 = key(P_1), k_2 = key(P_2)$
        if $k_1 == k_2$
           if $unify(P_1, P_2)$
              $D_1 = tl(D_1), D_2 = tl(D_2)$
           else
              return false
           endif
        elseif $k_1 < k_2$ % (lexicographic order)
           $D_1 = tl(D_1)$
        else
           $D_2 = tl(D_2)$
        endif
enduntil
return true
```

Fig. 7. Comparing type descriptors

5 Embedding in logic programs

The mechanisms described above can easily be incorporated into standard logic programs. One obvious application is for adding semantic constraints to natural language grammar. The example below illustrates the idea with a simple DCG. We are currently exploiting it in a rather more substantial grammar (see [Ramsay and Mansour, 2007] for details), but for clarity in the current context we will use a very simple DCG for illustration.

```
s(V+[R=S | PRED]) ==> np(S0), vp(V+[R=S1| PRED]), S#S0:S1.
vp(V+[SUBJ, R=OBJ]) ==> vtrans(V+[SUBJ, R=ARG]), np(NP), OBJ#ARG:NP.
np(N) ==> det, nn(N).
nn(N) ==> noun(N).
nn(N) ==> adj(A), nn(X), N#A:X.
noun(X) ==> [man], X --- man.
noun(X) ==> [cat], X --- cat.
noun(X) ==> [book], X --- book.
adj(X) ==> [stupid], X --- stupid.
det ==> [the].
det ==> [a].
vtrans(write+[agent=A, object=O]) ==> [wrote], A --- clever, O --- ~living.
```

Fig. 8. Clever people write books

The `s`, `vp` and `np` rules in this grammar are entirely orthodox apart from the fact that the structures that get returned from the `s` and `vp` are built by combining the information associated with the NPs that are the `subject` and `object` of the verb with the constraints on what that verb will accept as arguments. So if we parse *'the man wrote a book'* we obtain the structure in Fig. 5.

```
write
+ [agent=[[age=yes,adult=yes],
          [concrete=yes,living=yes,animal=yes,mammal=yes,
                                                ape=yes,human=yes,man=yes],
          [male=yes],
          [sentient=yes,intelligent=yes,clever=yes]],
   object=[[book=yes],[concrete=yes,artefact=yes]]]
```

Fig. 9. *'The man wrote a book'*

The information associated with the object in this analysis comes directly from the fact that the thing being written is a book. The information associated with the agent, on the other hand, is obtained by combining the fact that the subject is a man and the constraint that only clever things can write books. If we had said *'the stupid man wrote a book'* then the description of *'the stupid man'* would include the path `[sentient=yes, intelligent=yes, stupid=yes]`, which would clash with the requirement that the agent of a writing event should be compatible with `[sentient=yes, intelligent=yes, clever=yes]`. *'the cat wrote a book'* and *'the man wrote a cat'* would similarly be rejected, because the description of cats says that they are not intelligent (so a cat cannot be the agent of a writing event) but they are living (and hence a cat cannot be the object of such an event).

This is, of course, a very simple example, but putting selection restrictions on the entities that can play specified roles with respect to a given verb is well-known application of this kind of logic. There are numerous well-documented problems with straightforward application of this idea (e.g. the fact that you do not want to block the analysis of *'Don't be silly, of course he didn't write a cat'*), but these can be overcome by imposing penalties on analyses that violate the constraints rather than blocking them outright.

We can make this more subtle by introducing a measure of just how badly a constraint is violated, by measuring the point at which two paths clash, so that chimpanzees and orang-utangs are more similar than chimpanzees and cats. We could thus take the reciprocal of the point at which two paths clashed as the degree of mismatch. This introduces two new costs. (i) There is a small extra constant factor involved in keeping a depth counter as we walk along a pair of paths. The cost of this is trivial, and can just about be discounted. (ii) If we are just trying to see whether two descriptors match, we can quit as soon as the first mismatch is found. If we want to calculate the degree of mismatch, we have to inspect every path, since there may be multiple clashes (suppose, for instance, we wanted to compare `chimpanzee+male` and `orang_utang+female`). The number of paths to be compared is thus potentially greater.

These costs are fairly small, so in cases where we want to see how badly some constraint is violated they are probably worth paying. More significantly, while it is reasonable to suppose that the depth at which the first clash appears on a single path is a reliable indication of the degree of difference, it is hard to be sure that different paths are equally weighted. Consider for instance the descriptors for `man`, `woman` and `event` (Fig. 5).

```
man=[[age=yes, adult=yes],
     [concrete=yes,living=yes,animal=yes,mammal=yes,ape=yes,
                                         human=yes,man=yes],
     [edible=no],
     [male=yes],
     [sentient=yes, intelligent=yes]]
woman=[[age=yes, adult=yes],
       [concrete=yes,living=yes,animal=yes,mammal=yes,ape=yes,
                                           human=yes,woman=yes],
       [edible=no],
       [male=no],
       [sentient=yes, intelligent=yes]]
event=[[concrete=no, temporal=yes, interval=yes, event=yes]]
```

Fig. 10. `man, woman, event`

Intuitively, `man` and `woman` ought to be more similar than `man` and `event`. By the measure proposed above, the the difference between the descriptors for `man` and `woman` would be 1.14 (0.14 for the clash at level 7 between the paths starting with `concrete` and ending with `man/woman`, and 1 for the paths starting with `male=yes` and `male=no` respectively), whereas the score for `man` and `event` is 1.0, for the single clash at the head of the paths headed by `concrete=yes/concrete=no`.

This is clearly wrong. Part of the problem lies with the fact that we are ignoring the paths on which the descriptors for `man` and `woman` agree. We could compensate for this by having a positive score for paths where entries agree as well as a negative one for paths where they clash. The problem here would be coming up with an appropriate weighting for agreements versus disagreements. This is compounded by the fact that some paths are inherently more significant than others. It seems reasonable to suppose that the difference between male and female entities is less than the difference between concrete and abstract ones, but it is very hard to see how you weigh this difference. Are abstract entities twice as different from concrete ones as males are from females? Three times? Eighty four times?

It seems, then, that the degree of similarity/dissimilarity between two descriptors can be calculated at very little extra cost once you have decided on how to assign values to different paths; and that using a penalty score based on the depth of mismatch on a single path will provide a coherent measure, in that two entities that clash at say level 5 on a given path (e.g. chimpanzee and cat) can be reasonably assumed to be more different than two entities that clash at level 6 on the same path (chimpanzee and orang-utang). Devising a measure which can compare mismatches on different paths, or which can counterbalance mismatches on one path with matches on another, is fraught with the problems that bedevil all attempts to introduce numerical truth values [Voorbraak, 1991, Epstein, 1990, Gabbay, 1996]. Incorporating such a scoring scheme can be done easily, and at very little cost.

Choosing one is beyond the scope of the current paper. The example discussed above was introduced to show we can take a fairly orthodox technique and make it substantially more effective by incorporating a powerful logic of types at very little cost. Of course the simple DCG-style grammar used for this example provides a very simplistic account of natural language grammar. Nonetheless, the way that we incorporated semantic constraints into this simple grammar can be carried over into more sophisticated formalisms, and we are currently exploiting it in the framework described in [Ramsay, 1999, Ramsay and Seville, 2000], particularly lexical disambiguation in Arabic where written surface forms are highly ambiguous [Ramsay and Mansour, 2007]. Any such use of semantic constraints in parsing should be regarded as a preliminary filter, to be used largely to prioritise potential analyses rather than to rule them in or out, and should always be followed by more detailed reasoning about potential interpretations [Hirst, 1987, Asher and Lascarides, 1995, Wedekind, 1996]. Treated in this way, constraints on the kind of entities that can enter into syntactic relations (arguments for verbs and figure-ground pairs for prepositions are typical candidates) can be extremely useful.

6 Conclusions

The logic of sorts described above has considerable expressive power. Allowing partitions as well as superset relations, and allowing the terms used for expressing relations to include negation, makes the logic substantially more expressive than type logics which can be easily encoded as trees or bit-strings [Aït-Kaci et al., 1989, Fall, 1990]). For most logics of greater expressive power, e.g. description logics and propositional logic, checking consistency and subsumption relations between type descriptors are NP-complete. The logic described here strikes a balance between expressivity and complexity. The ability to allow multiple parents, to allow partitions when required, and to construct descriptors out of conjunctions and negations of base level types, provides much more expressive power than is available in simpler logics of types. The fact that consistency and subsumption checking are linear in the size of the descriptors (not the size of the lattice) in theory, and very fast in practice, means that we can work with lattices with tens or even hundreds of thousands of nodes. As such, this logic provides a practical alternative to description logics, where the theoretical complexity implies that in at least some cases the cost of checking subsumption and consistency may become prohibitive.

Bibliography

H Aït-Kaci, R Boyer, P Lincoln, and R Nasr. Efficient implementation of lattice operations. *ACM Transations on Programming Languages*, 11(115):115–146, 1989.

N Asher and A Lascarides. Lexical disambiguation in a discourse context. *Journal of Semantics*, 1995.

F Baader and U Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1), 2001.

R L Epstein. *The Semantic Foundations of Logic: Vol 1, propositional logics*. Nijhoff International Philosophy Series, Dordrecht, 1990.

A Fall. *Reasoning with taxonomies*. PhD thesis, Simon Fraser University, 1990.

D M Gabbay. *Labelled Deductive Systems*. Oxford University Press, Oxford, 1996.

G Hirst. *Semantic interpretation and the resolution of ambiguity*. Studies in natural language processing. Cambridge University Press, Cambridge, 1987.

A M Ramsay. Weak lexical semantics and multiple views. In H C Bunt and E G C Thijsse, editors, *3rd International Workshop on Computational Semantics*, pages 205–218, University of Tilburg, 1999.

A M Ramsay and H Mansour. A system for Arabic text-to-speech. *Computer Speech and Language*, 22:84–103, 2007.

A M Ramsay and H L Seville. Unscrambling English word order. In M Kay, editor, *Proceedings of the 18th International Conference on Computational Linguistics (COLING-2000)*, pages 656–662, Universität des Saarlandes, July 2000.

F Voorbraak. On the justification of Dempster's rule of combination. *Artificial Intelligence*, 48: 171–197, 1991.

J Wedekind. On inference-based procedures for lexical disambiguation. In J-I Tsujii, editor, *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 980–985, Copenhagen, 1996.

String Unification is Essentially Infinitary

Michael Hoche
EADS Deutschland GmbH, HMSI e.V.
Normannenweg 48, D-88090 Immenstaad

Jörg Siekmann,
DFKI
Stuhlsatzenweg 3, D-66123 Saarbrücken

Peter Szabo
HMSI e.V.
Kurt-Schumacherstr. 13, D-75180 Pforzheim

Abstract

A *unifier* of two terms s and t is a substitution σ such that $s\sigma = t\sigma$ and for first-order terms there exists a *most general unifier* σ in the sense that any other unifier δ can be composed from σ with some substitution λ, i.e. $\delta = \sigma \circ \lambda$.

This notion can be generalised to E-unification, where E is an equational theory, $=_E$ is equality under E and σ is an E-unifier if $s\sigma =_E t\sigma$. Depending on the equational theory E, the set of most general unifiers is always a singleton (as above), or it may have more than one, either finitely or infinitely many unifiers and for some theories it may not even exist, in which case we call the theory of type nullary.

String unification (or Löb's problem, Markov's problem, unification of word equations or Makanin's problem as it is often called in the literature) is the E-unification problem, where $E = \{f(x, f(y, z)) = f(f(x, y), z)$, i.e. unification under associativity or string unification once we drop the fs and the brackets. It is well known that this problem is infinitary and decidable.

Essential unifiers, as introduced by Hoche and Szabo, generalise the notion of a most general unifier and have a dramatically pleasant effect on the set of most general unifiers: the set of essential unifiers is often much smaller than the set of most general unifiers. Essential unification may even reduce an infinitary theory to an essentially finitary theory. The most dramatic reduction known so far is obtained for idempotent semigroups or bands as they are called in computer science: bands are of type nullary, i.e. there exist two unifiable terms s and t, but the set of most general unifiers is not enumerable. This is in stark contrast to essential unification: the set of essential unifiers for bands always exists and is finite.

We show in this paper that the early hope for a similar reduction of unification under associativity is not justified: string unification is *essentially infinitary.* But we give an enumeration algorithm for essential unifiers. And beyond, this algorithm terminates when the considered problem is finitary.

Keywords: E-unification, equational theory, essential unifiers, string unification, unification of words, universal algebra, varieties.

1 Introduction

Unification is a well established concept in artificial intelligence, automated theorem proving, computational linguistics, universal algebra, in theoretical and applied computer science, and e.g. semantics of programming languages. Surveys of unification theory can be found in [18, 6, 7]. A survey of the related topic of rewriting systems is presented in [9] and more recently in [12]. A standard textbook is Franz Baader, Tobias Nipkow, *Term Rewriting and All That* [6].

Unification is a general concept to solve equational problems, which is especially embedded in a plurality of deduction and inference mechanisms. For practical applications it is often crucial to have a finite or at least minimal representation of all the solutions, i.e. a minimal complete set of unifiers from which all other solutions (unifiers) can be derived.

For equational problems in the free algebra of terms (also known as syntactic unification), there exists always a unique unifier for solvable unification problems from which all other unifieres can be derived by instantiation. This unique unifier is called the *most general unifier*, [14]. For equational algebras however the situation is completely different: a minimal complete set of unifiers is not always finite and it may not even exist, which was conjectured by Gordon Plotkin in his seminal paper in 1972, [13]. Since then unification problems and equational theories have been classified with respect to the cardinality of their minimal complete set of unifiers. These results led to the development of general approaches and algorithms, which can be applied to a whole class of theories. This is the topic of *universal unification*, see e.g. [18].

More specifically, a unification problem $s =^?_E t$ for two given terms s and t under an equational theory E is the problem to find a minimal and complete set of unifiers $\mu\mathcal{U}\Sigma_E$ for s and t such that for every unifier $\sigma \in \mu\mathcal{U}\Sigma_E$ we have $s\sigma =_E t\sigma$. We say a unification problem is *unitary* if $\mu\mathcal{U}\Sigma_E$ is always a singleton, it is *finitary* if $\mu\mathcal{U}\Sigma_E$ is finite for every s and t and it is *infinitary* if there are terms s and t such that $\mu\mathcal{U}\Sigma_E$ is infinite. Unfortunately there are theories such that two terms are unifiable,but the set $\mu\mathcal{U}\Sigma_E$ is not recursively enumerable. In this case we call the problem *nullary* or of type zero.

It turned out that this well established view of unification theory changes drastically if we redefine the notion of a most general unifier. Recall that a unifier σ is most general if for any other unifier τ there exists a substitution λ such that

$$\tau = \sigma \circ \lambda$$

We generalise this notion and define an *essential unifier* σ if for any other unifier τ there exist substitutions λ_1 and λ_2 such that

$$\tau = \lambda_1 \circ \sigma \circ \lambda_2$$

where λ_1 has to have certain properties to be defined below.

We say a unification problem is e-unitary (is e-finitary) if the set of essential unifiers is always a singleton (is always finite). A unification problem is e-infinitary (e-nullary) if there are two terms such that the set of essential unifiers is infinite (is not recursive enumerable).

These notions were first introduced by Hoche and Szabo in [5] and it was shown in their paper that the unification problem for idempotent semigroups (bands) is e-finitary. Bands are well known since it was one of the early examples to demonstrate Plotkin's conjecture, that there exist nullary equational theories, which was shown one and a half decades later by Manfred Schmidt-Schauss,[15]. Now the unification problem for bands is nullary in the traditional sense but it is e-finitary in our sense: this is so far the most drastic reduction of the cardinality of the set of most general unifiers to a set of essential unifiers.

The question is: can similar results be obtained for other theories as well and a natural candidate for this kind of investigation is string unification. Why is that?

In the 1950s A. A. Markov was interested in the solvability of word equations in free semigroups: he noted that every word equation over a two constant alphabet can be translated into a set of diophantine equations. Using this translation he hoped to find a proof for the unsolvability of Hilbert's tenth problem by showing that the solvability of word equations is undecidable. This put the problem firmly on the map and others joined in: see the volumes edited by M. Lothaire and others on *Combinatorics on Words* [2]. The problem was finally solved in the affirmative in the seminal work by G. S. Makanin. An excellent exposition of Makanin's algorithm (with several improvements) is presented by Klaus Schulz [3] and by Volker Diekert (Chapter 12 in [2]).

Apart from its theoretical interest, the problem became more widely known, because of its relevance in computer science, artificial intelligence and automated reasoning. As opposed to the above works on decidability which just enumerate all solutions and make the decidability of the existence of a solution their primary focus, we are interested in the latter works, inspired by automated theorem proving, where the set $\mu\mathcal{U}\Sigma$ of the *most general* solutions is the focus of attention.

The most common and simple example to show that string unification is infinitary is the following

$$(1) \qquad xa = ax$$

with the set of most general unifiers

$$\mu\mathcal{U}\Sigma = \{\{x \mapsto a\}, \{x \mapsto aa\}, \{x \mapsto aaa\}, \ldots\}.$$

It is easy to show that indeed this is a solution set and it is not as immediate,but still not too hard to show that there does not exist any other more general set of unifiers $\mu\mathcal{U}\Sigma$ for this problem. Finally $\mu\mathcal{U}\Sigma$ is minimal, which again is obvious, as there are no variables in the a^n and thus they do not yield to instantiation. Hence in general

string unification is infinitary.

As we have said, this is a well known fact since the mid seventies and it is probably the most often quoted example in any lecture or monograph on unification theory.

A similar example

$$(2) \qquad xa = bx$$

is usually chosen to demonstrate that the naive string unification algorithm is not a decision procedure: although it is obvious that the above example is not unifiable, the actual algorithm would run forever.

However, problem (1) has a finite set (in fact an even e-unitary set) of essential unifiers

$$e\mathcal{U}\Sigma = \{\{x \mapsto a\}\} = \{\sigma_1\}$$

and any other unifier can be obtained with $\lambda_1 = \{x \mapsto a^{n-1}x\}, n > 0$ and $\lambda_2 = \varepsilon$. In other words, for any unifier $\sigma_n = \{x \mapsto a^n\}$:

$$\begin{aligned}\sigma_n &= \lambda_1\sigma_1\lambda_2\\ &= \{x \mapsto a^{n-1}x\} \circ \sigma_1 \circ \varepsilon\\ &= \{x \mapsto a^{n-1}x\} \circ \{x \mapsto a\} \circ \varepsilon\\ &= \{x \mapsto a^n\}\end{aligned}$$

where λ_1 obeys a certain structural property, to be defined in the next section.

Once this observation had been made a few years ago, there was an intense struggle to generalise this observation to any string unification problem and to prove the conjecture

string unification is e-finitary.

As we shall show in this paper, this conjecture is false in general, albeit it holds for certain subclasses of strings.

2 Basic Notions and Notation

Notation and basic definitions in unification theory are well known and have found their way into many and diverse research areas, standard survey articles are [18, 6, 7] and the monographs and textbooks on automated theorem proving usually contain sections on unification. Most recent results are presented at the Unification Workshop.[1]

[1]First workshop in Val d'Ajol in 1987 and since then annually. Since 1997, there is a website UNIF'987, UNIF'98, UNIF'99 up to UNIF'05 in Japan and UNIF'06 at the FLOC conference in Seattle.

For the reader's convenience we present some of the standard notation below, followed by the definitions of our novel approach for essential unifiers.

2.1 Unification theory: common definitions

An alphabet $\mathcal{F} = (F_n)_{n \in \mathcal{N}}$ provides a vocabulary, where the function symbols F_i, $i \in \mathcal{N}$ have the *arity* i. Function symbols with arity 0 are called *constants.* A set X gives us a denumerable set of *variable* symbols, usually denoted as x, y, z etc. and $\mathcal{F}$ and X constitute Σ, the *signature* of a term algebra.

The set of (first-order) terms $\mathcal{T}_{F,X}$ over a signature Σ generated by the variables X, is the smallest set containing the variables $x \in X$, and the terms $f(t_1, \ldots, t_n)$, whenever $f \in F_n$ is a function symbol of arity n and $t_1, \ldots, t_n \in \mathcal{T}_{F,X}$ are terms. The set of terms is a *(free) term algebra.*

The set of variable-free terms are called *ground terms.* The set of *variables* occurring in a term t is denoted by $\mathbf{Var}(t)$ and the set of *symbols* of $\mathcal{F}$ occurring in t is denoted by $\mathbf{Sym}(t)$. For a term t the set of *sub-terms* $\mathbf{Sub}(t)$ contains $t \in \mathbf{Sub}(t)$ itself and is closed recursively by containing $t_1, \ldots, t_n \in \mathbf{Sub}(t)$, if $f(t_1, \ldots, t_n) \in \mathbf{Sub}(t)$. For a set of terms $T = \{t_1, t_2, \ldots, t_n\}$ the subterms are defined by $\mathbf{Sub}(T) = \mathbf{Sub}(t_1) \cup \ldots \cup \mathbf{Sub}(t_n)$.

A *substitution* is the (unique) homomorphism in the term algebra generated by a mapping $\sigma : X \longrightarrow \mathcal{T}_{F,X}$ from a finite set of variables to terms. Substitutions are generally denoted by small Greek letters $\alpha, \beta, \gamma, \sigma$ etc. A substitution σ is represented explicitly as a function by a set of *variable bindings* $\sigma = \{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\}$. The application of the substitution σ to a term t, denoted $t\sigma$, is defined by induction on the structure of terms

$$t\sigma = \begin{cases} s_i & \text{if } t = x_i \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

The substitution $\varepsilon = \{\}$ with $t\varepsilon = t$ for all terms t in $\mathcal{T}_{F,X}$ is called the *identity.* A substitution $\sigma = \{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\}$ has the *domain*

$$\mathbf{Dom}(\sigma) := \{x | x\sigma \neq x\} = \{x_1, \ldots, x_m\};$$

and the *range* is the set of terms

$$\mathbf{Ran}(\sigma) := \bigcup_{x \in \mathbf{Dom}(\sigma)} \{x\sigma\} = \{s_1, \ldots, s_m\};$$

The set of variables occurring in the range is $\mathbf{VRan}(\sigma) := \mathbf{Var}(\mathbf{Ran}(\sigma))$ and $\mathbf{Var}(\sigma) = \mathbf{Dom}(\sigma) \cup \mathbf{VRan}(\sigma)$; the *restriction* of a substitution σ to a set of variables $Y \subseteq X$, denoted by $\sigma_{|Y}$, is the substitution which is equal to the identity everywhere except over $Y \cap \mathbf{Dom}(\sigma)$, where it is equal to σ.

Relations such as $=, \geq, \ldots$ between substitutions sometimes hold only if restricted to a certain set of variables V. A relation R which is restricted to V is denoted as R^V, and defined as $\sigma\ R^V\ \tau \Longleftrightarrow \sigma_{|V}\ R\ \tau_{|V}$.

The *composition* of two substitutions σ and θ is written $\sigma \circ \theta$ (emphases the composition) or just $\sigma\theta$ and is defined by $t\sigma\theta = (t\sigma)\theta$.

Two substitutions σ and θ are *equal*, denoted $\sigma = \theta$ iff $x\sigma = x\theta$ for every variable x.

A term t is an *instance* of a term s denoted $s \leq t$, if $t = s\sigma$ for some substitution σ, i.e.

$$s \leq t \Leftrightarrow \exists\sigma : s\sigma = t.$$

We also say s is more general or less specific than t. The relation $\leq$ is a quasi-ordering on terms called the *subsumption ordering*, whose associated equivalence relation and strict ordering are called subsumption equivalence and strict subsumption, respectively.

The *encompassment ordering* or *containment ordering* [4] is defined as the subterm ordering composed with the subsumption ordering, i.e. a subterm of t is an instance of s

$$s \sqsubseteq t \iff \exists\sigma : s\sigma \in \mathrm{Sub}(t).$$

Encompassment conveys the notion that s "appears" in t with a context "above" and a substitution "below". We say t *encompasses* s or s *is part of* t.

A substitution θ is called *more general* than σ, denoted $\theta \leq \sigma$, if there exists a λ such that $\sigma = \theta\lambda$, i.e.

$$\theta \leq \sigma \iff \exists\lambda : \theta\lambda = \sigma.$$

The relation $\leq$ is a pre-order, called the *instantiation* ordering for substitutions.

An *equation* or *identity* $s = t$ in a term algebra $\mathcal{T}_{F,X}$ is a pair (s,t) of terms and an algebra A satisfies the equation $s = t$ if for every homomorphism

$$h : \mathcal{T}_{F,X} \longrightarrow A,$$

$h(s) = h(t)$ that is, only if (s,t) is in the kernel of every homomorphism from $\mathcal{T}_{F,X}$ to A.

An *equational theory* is defined by a set of identities $E \subseteq \mathcal{T}_{F,X} \times \mathcal{T}_{F,X}$. It is the least congruence on the term algebra which is closed under substitution and contains E, and will be denoted by $=_E$. If $s =_E t$ we say s and t are *equal modulo E*. The sets $[\mathbf{s}]_{\mathbf{E}} = \{t | t =_E s\}$ are called congruence classes or *equivalence classes* (modulo E).

Let E be an equational theory and Σ the signature of the underlying term algebra. An *E-unification problem* (over Σ) is a finite set of equations

$$\Gamma = \{s_1 =^?_E t_1, \ldots, s_n =^?_E t_n\}$$

between Σ-terms with variables in a (countably infinite) set of variables V.

An *E-unifier of* Γ is a substitution σ, such that

$$s_1\sigma =_E t_1\sigma, \ldots, s_n\sigma =_E t_n\sigma.$$

The set of all E-unifiers of Γ is denoted by $\mathcal{U}\Sigma_E(\Gamma)$ or if the signature Σ is known from the context, we just write $\mathcal{U}_E(\Gamma)$ or even $\mathcal{U}(\Gamma)$.

A *complete set of E-unifiers* of Γ is a set C of substitutions, such that

(1) $C \subseteq \mathcal{U}\Sigma_E(\Gamma)$, i.e. each element of C is an E-unifier of Γ relative to a signature Σ and

(2) for each $\theta \in \mathcal{U}\Sigma_E(\Gamma)$ there exists $\sigma \in C$ with $\sigma \leq_E \theta$.

The set $\mu\mathcal{U}\Sigma_E(\Gamma)$ is a *minimal complete set of E-unifiers* for Γ, if it is a complete set, i.e. $\mu\mathcal{U}\Sigma_E \subseteq C$, and every two distinct elements of $\mu\mathcal{U}\Sigma_E$ are incomparable, i.e., $\sigma \leq_E \sigma'$ implies $\sigma =_E \sigma'$ for all $\sigma, \sigma' \in \mu\mathcal{U}\Sigma_E$. When a minimal complete set of E-unifiers of a unification problem Γ exists, it is unique up to subsumption equivalence.

The empty or unit substitution ε is a unifier in case $s =_E t$ are already equal. Minimal complete sets of unifiers need not always exist, and if they do, they might be singular, finite, or infinite. Since minimal complete sets of E-unifiers are isomorphic whenever they exist, theories can be classified with respect to their corresponding unification problem.

This leads naturally to the concept of a *unification hierarchy* which was first introduced in Siekmann's Ph.D. Thesis in 1975 [16], and further refined and extended by himself and his students, see [18, 6, 7] for surveys.

A *unification problem* Γ is *nullary*, if Γ does not have a minimal complete set of E-unifiers. The unification problem Γ is *unitary*, if it is not nullary and the minimal complete set of E-unifiers is of cardinality less or equal to 1. The unification problem Γ is *finitary*, if it is not nullary and the minimal complete set of E-unifiers is of finite cardinality. The unification problem Γ is *infinitary*, if it is not nullary and the minimal complete set of E-unifiers is of infinite cardinality.

An *equational theory E* is *unitary*, if all unification problems are unitary. An equational theory E is *finitary*, if all unification problems are finitary. An equational theory E is *infinitary*, if there is at least an infinitary unification problem and all unification problems have minimal complete sets of E-unifiers. If there exists a unification problem Γ not having a minimal complete set of E-unifiers, then the equational theory is *nullary* of *type zero.*

2.2 Additional Definitions: Essential Unifiers

Substitutions form a semigroup with respect to their composition. This fact was used to define the instantiation order on unifiers from above, namely

$$\sigma \leq \tau \iff \exists \lambda : \sigma \circ \lambda = \tau,$$

which led to the notion of a most general unifier.
As indicated above this concept does not generalise well on equational theories: the equational theory of associativity $A = \{x(yz) = (xy)z)\}$, i.e. the free semigroup with the unification problem $\{ax =^?_A xa\}$ has the infinite set of most general unifiers $\{\{x \mapsto a^n\} | n \geq 1\}$, as discussed in the introduction.
However, the essentially unifier in this set intuitively seems to be $\{x \mapsto a\}$, because every most general unifier contains this unifier in a certain sense, namely:

$$\{x \mapsto a^n\} = \{x \mapsto a^{n-1}x\} \circ \{x \mapsto a\}.$$

Now having in mind that substitutions form a semigroup, the dual of the instantiation ordering, i.e. left-composition instead of right-composition seems to change the infinitary problem into a finitary one if we redefine the order $\leq$ into $\exists \lambda : \sigma = \lambda\tau$, where $\sigma = \{x \to a\}$. But this is not compatible with the original notion of generality and it would not quite work in general.

Our solution to this dilemma is based on a lifting of the encompassment order on terms to an encompassment order on substitutions. More specifically we define a tripartition of a substitution i.e. an ordering concept which involves both left composition and right composition:

$$\sigma \trianglelefteq \tau \iff \exists\alpha\exists\beta : \tau = \alpha\sigma\beta.$$

And we say σ *is part of* τ and τ encompasses σ. This ordering concept, called *part ordering* in the following, is the result of lifting the encompassment order on terms and on substitutions, and it can be used to generate all unifiers as well. A unifier like σ above will be called an *essential unifier* if there is no left and right composition for σ and we shall now summarise the formalism to define the concept of an *essential unifier* (see [5] for more details). We say a substitution σ is part of a substitution τ, if and only if the domain of σ is a subset of the domain of τ and there exist α and β that "*build up*" σ into τ by means of composition, i. e. $\tau = \alpha\sigma\beta$, and σ has actually "*contributed*" in this decomposition of τ. The actual "*contribution*" of σ is important, since otherwise we would just end up again with the classical notion of a most general unifier. Technically this can be captured by the requirements, that the domain of σ and α are subsets of the domain of τ and whenever a variable x is in the domains of σ and α, then it is a variable in the range of α.

As usual this part relationship is generalized to equational theories E by considering all relationships modulo E, and we say: σ is a *part-substitution* of τ *modulo* E. This lifts the encompassment relation on basic terms to substitutions.

Part ordering on substitutions $\trianglelefteq_E$ is technically defined as follows:

Definition 2.1 (Part ordering of substitutions) *For substitutions σ and τ with $V = \mathbf{Var}(\tau)$*

*(1) σ is **part of** τ modulo E denoted as $\sigma \trianglelefteq_E \tau$, if there are two substitutions α and β with $\tau =_E^V \alpha\sigma\beta$, where $\mathbf{Dom}(\sigma) \cup \mathbf{Dom}(\alpha) \subseteq \mathbf{Dom}(\tau)$ and $\mathbf{Dom}(\sigma) \cap \mathbf{Dom}(\alpha) \subseteq \mathbf{VRang}(\alpha)$. In other words a substitution σ is part of a substitution τ if there is an instance of σ, namely $(\sigma\beta)$ which is a contributing (right) factor of τ, i. e. $\tau = \alpha(\sigma\beta)$.*[2]

*(2) σ is **proper part** of τ modulo E: $\sigma \triangleleft_E \tau$, if $\sigma \trianglelefteq_E \tau$ where the above two substitutions α and β with $\tau =_E^V \alpha\sigma\beta$ imply that $\alpha\beta \neq_E \iota$, where $\iota = \emptyset$.*

*(3) σ and τ are **part equivalent** modulo E: $\sigma \equiv_E \tau$ by $\sigma \trianglelefteq_E \tau \wedge \tau \trianglelefteq_E \sigma$*

*(4) σ is **not part of** τ modulo E: $\sigma \not\trianglelefteq_E \tau$ by $\neg(\sigma \trianglelefteq_E \tau)$.*

[2]Note the analogy to the encompassment order: "a term s is part of term t if there is an instance of s, namely $(s\sigma)$, which is a subterm of t."

(5) σ and τ are **part extrinsic** *modulo E: $\sigma \bowtie_E \tau$ by $\sigma \not\trianglelefteq_E \tau$ and $\tau \not\trianglelefteq_E \sigma$.*

Note that the contribution constraint assures that each term in the range of σ actually contributes to τ. This makes sure that we do not have unnecessary components like $x \mapsto f(a)$ which are just absorbed, as illustrated by the following example

$$\tau = \{x \mapsto a, y \mapsto b\} = \{x \mapsto a\}\{x \mapsto f(a)\}\{y \mapsto b\},$$

where $\{x \mapsto f(a)\}$ is obviously not a part of τ and absorbed by $\{x \mapsto a\}$. A substitution is a proper part if each part decomposition implies that the framing factors actually contribute something. As an illustration of the above definition consider the following example:

$$\tau = \{x \mapsto f(g(z), h(z)), y \mapsto g(z), z \mapsto j(x), v \mapsto k(y, z)\}$$

has a part $\sigma = \{y \mapsto g(z), z \mapsto j(y)\}$, since there is a left factor $\alpha = \{x \mapsto f(y, u)\}$ and a right factor $\beta = \{u \mapsto h(z), y \mapsto x, v \mapsto k(y, z)\}$. Obviously σ is a part of τ, since it contributes with $y \mapsto g(z)$ to α and it even contributes directly with $z \mapsto j(y)$ to τ, since z is not in the domain of α. Finally β completes the decomposition.

Proposition 2.2 *The part substitution ordering is indeed a pre-order, i.e. reflexive and transitive.*

The concept of an essential unifier can now easily be defined as:

Definition 2.3 *A unifier is* **essential** *if and only if it is minimal with respect to the part ordering, i.e.* **An essential e-unifier has no E-unifying part-substitution**.

The set of essential unifiers is indeed a generating set for all unifiers just as the traditional set of most general unifiers. This can be shown with the existence of a corresponding closure operator. A set of unifiers $C(\Gamma)$ is *e-complete* if for each unifier σ there exists a unifier τ in C which is part of σ. A complete set of unifiers $C(\Gamma)$ is *e-minimal* if any two distinct elements are not part of each other. Such a set is denoted as $e\mathcal{U}\Sigma_E(\Gamma)$ This set exists and is unique, because if there exist two complete sets of essential unifiers $e\mathcal{U}\Sigma_E^1$ and $e\mathcal{U}\Sigma_E^2$ with τ in $e\mathcal{U}\Sigma_E^1 \backslash e\mathcal{U}\Sigma_E^2$ and σ in $e\mathcal{U}\Sigma_E^2 \backslash e\mathcal{U}\Sigma_E^1$ then since $e\mathcal{U}\Sigma_E^1$ is complete, there exist the substitutions α and β for τ such that $\sigma =_E^V \alpha\tau\beta$. Since $e\mathcal{U}\Sigma_E^2$ is a set of essentials it follows $\sigma =_E \tau$, contradicting the assumption.

Lemma 2.4 *Let E be an equational theory and Γ a unification problem. Then the set of essential unifiers $e\mathcal{U}\Sigma_E(\Gamma)$ is a generating set for the set of all unifiers $\mathcal{U}\Sigma_E(\Gamma)$.*

A proof can be found in [5].

Lemma 2.5 $e\mathcal{U}\Sigma_E(\Gamma) \subseteq \mu\mathcal{U}\Sigma_E(\Gamma)$

The interesting observation is that the above subset of essential unifiers can be extremely small in comparison to its superset, as we shall see in the following.

3 Essential String Unification

We are interested now in the A-unification problem, i.e. unification in the free semigroup, where

$$A = \{f(x, f(y, z)) = f(f(x, y), z)\}$$

and the set of terms are built up as usual over constants, variables, but only one function symbol f. In this case, we can just drop the fs and brackets and write strings (or words) over the alphabet of constants and variables. A set of string equations will be denoted as $\Gamma = \{u_1 = v_1, \ldots, u_n = v_n\}$ and $\mathbf{Var}(\Gamma)$ is the set of free variable symbols occurring in u_i and v_i. Let $V = \mathbf{Var}(\Gamma)$, then a *(string-) unifier* $\sigma : V \mapsto \Sigma^*$ is a solution for Γ if $u_i\sigma = v_i\sigma, 1 \le i \le n$. The set of all unifiers is denoted as $\mathcal{U}(\Gamma)$. A unifier σ is *ground* if its range contains only constants and no variables. Now let us look at a few motivating examples, which show that indeed an infinite set of most general unifiers $\mu\mathcal{U}\Sigma$ collapses to a finite set of essential unifiers $e\mathcal{U}\Sigma$, supporting the hypothesis that the infinitary string unification problem is essentially finitary (which is false in general, as we shall see below).

Our first example is the well known string unification problem mentioned in the introduction:

$$ax =^? xa \text{ with } \sigma_n = \{x \mapsto a^n\}, n > 0$$

has infinitely many most general unifiers, but there is just *one* e-unifier $\sigma_0 = \{x \mapsto a\}$ because of

$$\sigma_n = \{x \mapsto a^{n-1}x\} \circ \sigma_0.$$

The next example has two variables[3]

$$xy =^? yx$$

and has infinitely many most general unifiers

$$\sigma_{i,j} = \{x \mapsto z^i, y \mapsto z^j\}, i, j > 0, \text{ where } i \text{ and } j \text{ are relative prime,}$$

but it has only one e–unifier $\sigma_0 = \{x \mapsto z, y \mapsto z\}$ because of

$$\sigma_{i,j} = \{x \mapsto z^{i-1}x, y \mapsto z^{j-1}y\} \circ \sigma_0$$

Our next example is taken from J. Karhumäki *Combinatorics of Words.* The system

$$\left\{ \begin{array}{l} xaba =^? baby \\ abax =^? ybab \end{array} \right\}$$

[3]see `http://www.math.uwaterloo.ca/~snburris/htdocs/scav/e_unif/e_unif.html`, example 15

has infinitely many most general unifiers

$$\sigma_n = \{x \mapsto b(ab)^n, y \mapsto (ab)^n a\}, n \geq 0$$

But it has only one e-unifier, namely σ_0 because of

$$\sigma_n = \{x \mapsto x(ab)^n, y \mapsto (ab)^n y\} \circ \sigma_0.$$

Exploiting the analogy between the first and the second example above, we can easily construct the following example (and many more in this spirit): But the unification problem

$$xxyyxx =^? yyxyxyy$$

has only one most general unifiers

$$\sigma_n = \{x \mapsto z^3, y \mapsto z^2\},$$

and this is the only e–unifier.
The fifth example is taken from J. Karhumäki as well:

$$axxby =^? xaybx$$

has infinitely many most general unifiers

$$\sigma_{i,j} = \{x \mapsto a^i, y \mapsto (a^i b)^j a^i\}, i \geq 1, j \geq 0$$

but it has only one e-unifier $\sigma_{1,0} = \{x \mapsto a, y \mapsto a\}$ which is essential because of

$$\sigma_{i,j} = \{x \mapsto ya^{i-1}, y \mapsto (a^i b)^j xa^{i-1}\} \circ \sigma_{1,0}$$

The final example is a bit more elaborate but still in the same spirit.

$$zaxzbzy =^? yyzbzaz$$

has infinitely many most general unifiers

$$\sigma_n = \{x \mapsto b^{2n}a, y \mapsto b^n ab^n, z \mapsto b^n\}, n > 0$$

but it has only one e-unifier, namely $\sigma_1 = \{x \mapsto bba, y \mapsto bab, z \mapsto b\}$ because of

$$\sigma_n = \{x \mapsto b^{2n-2}x, y \mapsto b^{n-1}yb^{n-1}, z \mapsto b^{n-1}z\} \circ \sigma_1$$

3.1 String Unification with at most one variable is e-unitary

So let us assume our unification problem

$$\Gamma = \{u_1 =^? v_1, \ldots, u_n =^? v_n\}$$

over the signature Σ consists of at most one variable, but arbitrary many constants. Without loss of generality, each arbritrary set of string equations is equivalent to a single string equation preserving the solutions. For example Diekert used the following construction

$$\{u_1 a \dots u_n a u_1 b \dots u_n b =^? v_1 a \dots v_n a v_1 b \dots v_n b\}$$

where a and b are distinct constants. The two equational problems have the same solutions.

Let $\Gamma = \{u_0 x u_1 ... x u_n = v_0 x v_1 ... x v_m\}, u_i, v_i$ are ground strings, x in $\Sigma = X \cup F$ and $V = \mathbf{Var}(\Gamma) = \{x\}$. The following facts are well known.

1. The equation in Γ can be reduced to the form $u_0 x u_1 ... x u_n = x v_1 ... x v_m$, where u_0 is not the empty string and either u_n is nonempty and v_m is empty or vice versa. This form implies also that any unifier is a prefix of the string u_0^k.
2. if $m \neq n$ there is at most one unifier.
3. If $m = n = 1$, i.e. $\Gamma = \{u_0 x = x v_1\}$, and the unifiers are of the form: $x \mapsto (pq)^i p, i \geq 0$, where pq is *primitive*. Note: A word is primitive if it is not the power of some other word, i.e. it cannot be represented as $uv^n w$, for some words u, v, w and $n > 1$.
4. Considering $m = n > 1$ the unifiers are of the form: $x \mapsto (pq)^{i+1} p, i \geq 0$, where pq is *primitive.*
5. For a given Γ there exist at most one infinite solution of the form: $\sigma_i = \{x \mapsto (pq)^{i+1} p\}, i \geq 0$.
6. Unifiers of string equations with at most one variable are ground substitutions. We were not able to find a publication with a proof. We show this result below.

These results are now used to show that *string unification with only one variable* is *e-unitary.* The first step is to prove that all unifiers are ground substitutions. The second step is to prove that all unifiers share an essential unifier.

Proposition 3.1 *Let $\Gamma = \{u_0 x u_1 ... x u_n = x v_1 ... x v_n\}$ be a string equation with at most* one *variable x and $\mathcal{U}(\Gamma) = \{x \mapsto (pq)^{i+1} p\}, i \geq 0$. Then $\mathbf{Var}(pq)$ is empty, i.e. all unifiers are ground substitutions.*

Proof.

1. Suppose p contains a variable z, i.e. $p = p_1 z p_2$ where p_1 is ground. Applying the unifier $x \mapsto (pq)^{i+1} p$ yields

   $u_0 (pq)^{i+1} p u_1 \dots = (pq)^{i+1} p v_1 \dots = u_0 (p_1 z p_2 q)^{i+1} p u_1 \dots = (p_1 z p_2 q)^{i+1} p v_1 \dots$

   Consider the prefixes $u_0 p_1 \dots = p_1 z \dots$ Since $|u_0 p_1| \geq |p_1 z|$ and u_0 is nonempty, z must be a symbol in $u_0 p_1$, which is impossible.

2. Suppose q contains a variable z, i.e. $q = q_1 z q_2$ where q_1 is ground. Applying a unifier $x \mapsto (pq)^{i+1}p$ yields

   $u_0(pq)^{i+1}pu_1 \ldots = (pq)^{i+1}pv_1 \ldots = u_0(pq_1zq_2)^{i+1}pu_1 \ldots = (pq_1zq_2)^{i+1}pv_1$

   Consider the prefixes $u_0pq_1 \ldots = pq_1z \ldots$. Since $|u_0pq_1| \geq |pq_1z|$ and u_0 is nonempty, z must be a symbol in q_1 which is impossible.

Hence $\mathbf{Var}(pq)$ is empty. ■

Theorem 3.2 *String unification with one variable is e-unitary.*

Proof. Without loss of generality, let $\Gamma = \{u_0xu_1...xu_n = xv_1...xv_n\}$ be a string unification problem in *one* variable x and

$$\mathcal{U}(\Gamma) = \{\{x \mapsto (pq)^{i+1}p\} : i \geq 0\}.$$

Then there are the following decompositions, where V ={x}

1. In case of $n = 1$ and p is empty then

   $\{x \mapsto (pq)^i p\} =^V \{x \mapsto (pq)^i x\} \circ \{x \mapsto p\} \circ \varepsilon$
   if p is nonempty then
   $\{x \mapsto q^i\} =^V \{x \mapsto q^{i-1}x\} \circ \{x \mapsto q\}$

   either $\{x \mapsto p\}$ or $\{x \mapsto q\}$ are essential unifiers.

2. In case of $n > 1$ and p is empty then

   $\{x \mapsto (pq)^{i+1}p\} =^V \{x \mapsto (pq)^i x\} \circ \{x \mapsto pqp\}$
   or p is nonempty then
   $\{x \mapsto q^{i+1}\} =^V \{x \mapsto q^i x\} \circ \{x \mapsto q\}$

   either $\{x \mapsto pqp\}$ or $\{x \mapsto q\}$ are essential unifiers.

Hence the unification problem is e-unitary. ■

3.2 String unification is e-infinitary

String unification with at most one variable in the signature Σ is e-finitary as we have seen above and surely there are many more special cases of signature restrictions, where the set of e-unifiers is always finite. Special cases of this nature have been investigated extensively for the solvability problem of words[4].

Theorem 3.3 *String unification with more than one variable is e-infinitary*

[4]Google scholar finds 70,300 entries in 0.13 sec for "word equation" (not all of which is relevant) and several 100,000 more entries if you are patient enough to continue the search and to filter gold from garbage.

Proof. For $\Gamma = \{xby = ayayb\}$ the set of essential unifiers is

$$e\mathcal{U}(\Gamma) = \{\{x \mapsto ab^n a, y \mapsto b^n\} : n > 0\}$$

Correctness
Any substitution $\sigma_n = \{x \mapsto ab^n a, y \mapsto b^n\}$ is a unifier since $(xby)\sigma_n = ab^n abb^n = ab^n ab^{n+1} = (ayayb)\sigma_n$.

Completeness
We show that any unifier is of the form $\{x \mapsto ab^n a, y \mapsto b^n\}$. Now considering some unifier $\{x \mapsto u, y \mapsto v\}$. Since $\Gamma = \{xby = ayayb\}, u = au'$ and $v = v'b^k$, $k > 1$. Applying the unifier in $xby = ayayb$ yields $au'bv'b^k = av'b^k av'b^k b$. Since v' can not contain any a $v' = b^i$. Hence the unifier is now $\sigma = \{x \mapsto au', y \mapsto b^j\}$, where $j = i + k$. Thus $\Gamma\sigma = \{au'bb^j = ab^j ab^j b\}$, $xbb^j = ab^i ab^i b$, and $x = ab^j a$.

Essential
We show that the set $\{\{x \mapsto ab^n a, y \mapsto b^n\} : n > 0\}$ is e-minimal. So take any pair of different unifiers $\{x \mapsto ab^m a, y \mapsto b^m\}$ and $\{x \mapsto ab^n a, y \mapsto b^n\}$ and we show that they are incomparable with respect to the part ordering. Suppose $m < n$, then $|ab^n a| > |ab^m a| > |b^m|$, therefore $\{x \mapsto ab^m a, y \mapsto b^m\} \neq \alpha\{x \mapsto ab^n\, y \mapsto b^n\}\beta$. Now the other way round; the longer unifier could contain the shorter, but then there exists a decomposition $\{x \mapsto ab^n a, y \mapsto b^n\} = \alpha\{x \mapsto ab^m a, y \mapsto b^m\}\beta$ where w.l.o.g. $\alpha = \{x \mapsto u, y \mapsto v\}$. Since $y \mapsto b^n$ it follows $v \in \{b, y\}^*$, because the longer unifier maps y to b^n. Now let's look at $x \mapsto ab^n a$ contains only two times the letter a, $u = u_1 x u_2$ with $u_1, u_2 \in \{a, b\}^*$ or $u \in \{a, b\}^*$. In the latter case x occurs not in the range of α, and x is in the domain of $\{x \mapsto ab^n a, y \mapsto b^n\}$. Thus, in this case $\{x \mapsto ab^m a, y \mapsto b^m\}$ is not a part of $\{x \mapsto ab^n a, y \mapsto b^n\}$. In the case of $u = u_1 x u_2 = ab^n a$ with $x \mapsto ab^m a$, it follows that u_1 and u_2 are empty, contradicting $ab^n a \neq ab^m a$. ■

3.3 A General A-Theorem

Let E be a set of equational axioms containing the associativity axiom of a binary operator $*$, i.e. $A = \{x * (y * z) = (x * y) * z\}$ and $E = A \cup R$, where R is some set of equations. We call the theory modulo E A-separate, if any equation in R can not be applied to a pure string $x_1 * x_2 * \cdots * x_n$, where the brackets are suppressed.

For instance consider distributivity (which is an infinitary unification theory, see [19]

$$D = \{x * (y + z) = (x * y) + (x * z), (x + y) * z = (x * z) + (y * z)\},$$

then the theory of $E = A \cup D$ is A-separate. To see this, note that no equation of D can be applied to a string of $x_1 * x_2 * \cdots * x_n$, simply because there are no sums involving the plus sign $+$, but each equation in D has the sum symbol $+$ on its left and on its right side.

Formally, $E = A \cup R$ is A-separate, if for all elements u of the A-theory $u =_R v$ implies $u = v$.

Theorem 3.4 *All A-separate E-theories are e-infinitary*

Proof. Reconsider the unification problem of section 3.2 above. It has in the associative sub-algebra infinitely many e-unifiers. Each of the elements of the range of the essential unifiers is not affected by the remaining equational axioms in $R = E \backslash A$, since E is A-separate. Hence each A-separate theory is e-infinitary. As noted above the theory $A \cup D$ is A-separate. ■

Corollary 3.5 *The theory $A \cup D$ is e-infinitary.*

4 Idempotent semigroups are e-finitary

The following theory of *idempotent Semigroups or Bands* defined by

$$AI = \{f(x, f(y, z)) = f(f(x, y), z), f(x, x) = x\}$$

demonstrates another interesting applicability of essential unifiers. This theory is not A-separate. This theory is nullary with respect to the instantiation order, since there are solvable AI-unification problems which do not posses a minimal complete set of AI-unifiers with respect to the instantiation ordering [1, 15].

However, with respect to the part ordering $\trianglelefteq_E$ this well-known situation changes completely as this theory is essentially finitary. Associativity and idempotency constitute the algebra of idempotent strings and it was shown in [5] that:

Theorem 4.1 *The theory AI is not nullary with respect to essential unifiers*

Proposition 4.2 *AI is not unitary with respect to essential unifiers.*

And finally the most striking result:

Theorem 4.3 *The theory AI is finitary with respect to essential unifiers.*

5 A derivation system for A–Unification

Let Σ be the set of symbols (alphabet) and let X be the set of variables. Let u, v, w be strings, i.e. elements of the free monoid $(X \cup \Sigma)^*$. Let $\Gamma = \{u =^? v\}$ be a A-unification problem. A solution σ is a substitution, such that the equality $u\sigma = v\sigma$ is valid, denoted by $\sigma \models u = v$.

Let Λ be the homomorphism between the strings of the free monoid $(X \cup \Sigma)^*$ into $\mathcal{P}^{\Sigma}(X)$, where $\mathcal{P}(X)$ are the polynomials in X, that is defined by

$$\Lambda : x \mapsto \begin{cases} x \text{ for } x \in Var(\Gamma) \\ 1 \text{ otherwise} \end{cases} \quad \text{and} \quad \Lambda(uv) = \Lambda(u) + \Lambda(v).$$

Extend the notation for unification problems $\Gamma = \{u =^? v\}$ by

$$\Lambda(\Gamma) = \{\Lambda(u) =^?_P \Lambda(v)\}\}$$

mapping a string unification problem to a system of linear equations, where P is the set of Peano axioms; and for substitutions $\sigma = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$ to

$$\Lambda(\sigma) = \sigma\Lambda = \{x_1 \mapsto \Lambda(u_1), \ldots, x_n \mapsto \Lambda(u_n)\}.$$

Since substitutions are in the following assumed to consists only of free variables, the image is a vector of lengths $\sigma\Lambda = \{x_1 \mapsto |u_1|, \ldots, x_n \mapsto |u_n|\}$.

For instance let Σ be the alphabet $\{a, b\}$ and $\Gamma = \{xby =^? ayayb\}$. Then

$$\Lambda(\Gamma) = \{x + 1 + y =^? 1 + y + 1 + y + 1\} = \{x =^? y + 2\}.$$

For the unifier $\sigma = \{x \mapsto abba, y \mapsto bb\}$,

$$\Lambda(\sigma) = \{x \mapsto 4, y \mapsto 2\},$$

which is obviously a solution of $\Lambda(\Gamma)$.

Lemma 5.1 *If σ is an A-unifier for Γ, then the linear diophantine equation $\Lambda(\Gamma)$ has an integer solution $\Lambda(\sigma) : x \mapsto \Lambda(\sigma(x))$.*

Proof. Follows from the homomorphism definition. ■

Let $\sigma = \sigma_\top\sigma_\bot$ be a leaf decomposition for a substitution σ, $\sigma_\bot(x) \in \Sigma \cup X$. Define for a solution $\alpha : X \mapsto \mathcal{N}$ of a linear diophantine equation Δ, i.e. $\alpha \models \Delta$, a substitution δ_α with $\delta_\alpha(x) = x_1 \ldots x_n$, where $n = \alpha(x)$ and all x_i are free variables, i.e. not in $Var(\sigma)$.

Let Ψ be the reduction system of the following reduction rules

$$\begin{array}{ll} \text{Truncation} & \dfrac{[u_l u_r =^? u_l v_r, S]}{[u_r =^? v_r, S]}, \quad \dfrac{[u_l u_r =^? v_l u_r, S]}{[u_l =^? v_l, S]} \\ \text{Generation} & \dfrac{[\Gamma, S], \alpha \models \Lambda(\Gamma), \exists\lambda : X \to \Sigma \cup X : \delta_\alpha\lambda \in e\mathcal{U}(\Gamma)}{[\Gamma, S \cup \{\delta_\alpha\lambda\}]} \end{array}$$

Define $A \Longrightarrow_\Psi B$ if $\frac{A}{B}$ in Ψ, and let $\Longrightarrow_\Psi^*$ be the transitive closure of $\Longrightarrow_\Psi$.

Lemma 5.2 $[\Gamma, \emptyset] \Longrightarrow_\Psi^* [\Gamma', S \cup \{\sigma\}]$ *iff* $\sigma \models \Gamma$.

Proof. "$\Rightarrow$" by definition of the Generation Rule. "$\Leftarrow$" $\sigma \models \Gamma$ implies $\Lambda(\sigma) \models \Lambda(\Gamma)$. Let $\Lambda(\sigma) =: \alpha$. Thus the Generation Rule is applicable with $\sigma = \delta_\alpha\lambda$ for a λ. ■

Define the ordering $\alpha \leq \beta$ by $\sum_{x \in Dom(\alpha)} \alpha(x) \leq \sum_{x \in Dom(\beta)} \beta(x)$.

Lemma 5.3 $\lambda \trianglelefteq \sigma$ *implies* $\Lambda(\lambda) \leq \Lambda(\sigma)$.

Proof. $\lambda \triangleleft \sigma$ implies that there exists α and β such that $\sigma = \alpha\lambda\beta$. Thus $\Lambda(\sigma) = \Lambda(\alpha) + \Lambda(\lambda) + \Lambda(\beta)$. Hence

$$\sum_{x \in Dom(\sigma)} \Lambda(\sigma(x)) = \sum_{x \in Dom(\alpha)} \Lambda(\alpha(x)) + \sum_{x \in Dom(\lambda)} \Lambda(\lambda(x)) + \sum_{x \in Dom(\beta)} \Lambda(\beta(x)).$$

Hence $\Lambda(\lambda) \leq \Lambda(\sigma)$. ■

Lemma 5.4 *Let $\Gamma = \{u =^? v\}$ be an A-unification problem with $\alpha \models \Lambda(\Gamma)$, such that there exists $\lambda : X \to \Sigma \cup X$ with $\delta_\alpha \lambda \models \Gamma$, then λ is unique.*

Proof. Syntactic unification is unitary. Hence there exists a function *uni* that maps a solution α of $\Lambda(\Gamma)$ to a unifier $uni(\alpha) = \delta_\alpha \lambda$. ■

Lemma 5.5 *Let Γ be an A-unification problem and $\alpha < \beta$ be two solutions of $\Lambda(\Gamma)$. If there exist the two unifiers $uni(\alpha)$ and $uni(\beta)$, then $uni(\beta) \not\trianglelefteq uni(\alpha)$.*

Proof. Suppose the contrary $uni(\beta) \trianglelefteq uni(\alpha)$. Note $\Lambda(uni(\beta)) = \beta$ and $\Lambda(uni(\alpha)) = \alpha$. That implies $\beta \leq \alpha$, a contradiction. ■

A controlled algorithm for enumerating the essentials could look like

```
FOR ALL $i \geq 0$ DO COMPUTE
    $S(i) = \{\alpha \mid \alpha \models \Lambda(\Gamma), \sum_{x \in Dom(\alpha)} \alpha(x) = i\}$ (* diophantine equation *)
    $U(i) = \{uni(\alpha) \mid \alpha \in S(i)\}$                (* real unifiers *)
    $E(i) = E \cup \{\lambda \in U(i) \mid \neg\exists \sigma \in E : \sigma \trianglelefteq \lambda\}$     (* essential unifiers *)
    EXIT WHEN $P(E(i), \Gamma) = \emptyset$
    WHERE $P(E, \Gamma) = \{\sigma : X \to \Sigma^* \mid \sigma \models \Gamma \wedge \forall \beta \in E : \beta \not\trianglelefteq_A \sigma\}$
END FOR
```

Lemma 5.6 *For a finite set E of substitutions and a finite equational problem Γ modulo A the above predicate $P(E, \Gamma)$ is decidable.*

Proof.

$$\begin{aligned} P(E, \Gamma) &= \{\sigma : X \to \Sigma^* \mid \sigma \models \Gamma \wedge \forall \beta \in E \neg\exists \alpha, \gamma : \sigma = \alpha\beta\gamma\} \\ &= \{\sigma : X \to \Sigma^* \mid \sigma \models \Gamma \wedge \forall \beta \in E : \sigma \neq \alpha\beta\gamma\} \end{aligned}$$

To simplify the notation we assume without loss of generality that the substitutions might even erase variables, i.e., we consider homomorphisms in the free monoid instead of homomorphisms in the free semi group. The case considerations of the proof for semi groups is rather cumbersome but straight forward by introducing equation variations where the variables that might be erased are eliminated from the beginning.

Proposition 5.7 *For a pair of substitutions β and σ there exists a set $\mathbf{\Gamma}$ of equational systems with*

$\sigma = \alpha\beta\gamma$ if and only if at least one equational system of $\mathbf{\Gamma}$ is solvable.

Proof. Consider the following simplifications: If there is a factorzation $\sigma = \alpha\beta\gamma$ then without loss of generality there exists also a factorization $\sigma = \alpha'\beta\gamma'$ where the domain and the set of variables in the range of α' are made disjoint by renamings introducing free variables, i.e.

$$(Dom(\alpha') \cap VRan(\alpha')) \setminus Dom(\beta) = \emptyset$$

and for γ

$$Dom(\gamma') \cap VRan(\gamma') = \emptyset \text{ and } VRan(\gamma') \cap Dom(\alpha') = \emptyset$$

Let further without loss of generality

$$VRan(\alpha') \cap Dom(\gamma') = \emptyset \text{ and } Dom(\gamma') \subseteq VRan(\beta) = \emptyset$$

which can be reached by applying γ on α. Thus for $\sigma = \alpha\beta\gamma = \alpha'\beta\gamma'$ for each x in the domain of σ the following equation is valid

$$\sigma(x) = \begin{cases} \gamma'(\alpha'(x) = \alpha'(x) & x \in Dom(\alpha'), R_\alpha(x) = \emptyset \\ \gamma'(\beta(\alpha'(x))) & x \in Dom(\alpha'), R_\alpha(x) \neq \emptyset \\ \gamma'(\beta(x)) & x \notin Dom(\alpha'), x \in Dom(\beta) \\ \gamma'(x) & x \notin Dom(\alpha'), x \notin Dom(\beta) \end{cases}$$

where $R_\alpha(x) = Var(\alpha'(x)) \cap Dom(\beta)$.

Define a family of equation systems under the hypothesis of a family of $R_\alpha(x) \subseteq Dom(\beta)$, where x varies in $Dom(\sigma)$, and a domain $D_\alpha = Dom(\alpha) \subseteq Dom(\sigma)$. Thus all variations of hypotheses are defined by σ and β. Let the equation system be

$$\begin{array}{lcllr} x_\beta & =^? & \beta(x_\beta) & x_\beta \in Dom(\beta) & (1) \\ x_\sigma & =^? & u_{x_\sigma x_\beta} x_\beta v_{x_\sigma x_\beta} & x_\sigma \in D_\alpha, x_\beta \in R(x_\sigma) & (2) \\ x_\sigma & =^? & x_\beta & x_\sigma \notin D_\alpha & (3) \\ x_\sigma & =^? & \sigma(x_\sigma) & x_\sigma \in Dom(\sigma) & (4) \end{array}$$

$\Rightarrow$: $\sigma = \alpha\beta\gamma$ implies that there is a $\Gamma \in \mathbf{\Gamma}$ with a solution $\delta \models \Gamma$: Consider Γ with $D_\alpha = Dom(\alpha)$ and for $x \in Dom(\sigma)$ let $R_\alpha(x) = Var(\alpha(x)) \cap Dom(\beta)$. The equation sub-system (1) and (4) are obviously solved by β and σ. Equational sub-system (2) is solvable, since for every $x \in Dom(\sigma)$ that is mapped by α onto a string $\alpha(x)$ that contains a variable out of the domain of β, there are bindings for a prefix and a suffix, that solve each equation in (2). (3) is also solvable, since there exist a γ such that for each $x \in Dom(\sigma) \cap Dom(\beta)$ that does not occur in $Dom(\alpha)$ it is shown by $\sigma(x) = \gamma(\beta(x))$ that the equation is solvable.

$\Leftarrow$: If there is a $\delta \models \Gamma \in \mathbf{\Gamma}$, then there are α and γ with $\sigma = \alpha\beta\gamma$: Suppose Γ with D_α and $R_\alpha(x)$, $x \in Dom(\sigma)$. Define $\alpha(x) = \delta(u_{x_\sigma x_\beta}) x_\beta \delta(v_{x_\sigma x_\beta})$ for all $x \in D_\alpha$. Define $\gamma(x) = \delta(x)$ for all $x \notin D_\alpha$. This corresponds to the normalization considerations in the beginning of the proof of the proposition and shows $\sigma = \alpha\beta\gamma$. ■

Proposition 5.8 *For a finite set E of substitutions β there exists an equational system $\Gamma(E)$ with the property*

$$\sigma \models \Gamma(E) \textit{ if and only if there exists } \beta \in E \textit{ with } \sigma = \alpha\beta\gamma$$

Proof. This follows from the fact that the equational theory of semi groups is boolean closed, i.e. a boolean combination of equation systems can be expressed by a single equation system. ■

From the proposition follows that

$$\begin{aligned} P(E,\Gamma) &= \{\sigma : X \to \Sigma^* \mid \sigma \models \Gamma \wedge \forall \beta \in E : \neg\sigma = \alpha\beta\gamma\} \\ &= \{\sigma : X \to \Sigma^* \mid \sigma \models \Gamma \wedge \bigwedge_{\beta \in E} \neg\sigma \models \Gamma(\beta)\} \end{aligned}$$

For each equational problem Γ there exists a complementary equational problem $\overline{\Gamma}$ with $\neg(\sigma \models \Gamma)$ if and only if $\sigma \models \overline{\Gamma}$. Hence it follows

$$\begin{aligned} P(E,\Gamma) &= \{\sigma : X \to \Sigma^* \mid \sigma \models \Gamma \wedge \bigvee_{\beta \in E} \sigma \models \overline{\Gamma(\beta)}\} \\ &= \bigcup_{\beta \in E} \{\sigma : X \to \Sigma^* \mid \sigma \models (\Gamma \cup \overline{\Gamma(\beta)})\} \end{aligned}$$

Thus the predicate $P(E,\Gamma) = \emptyset$ is reformulated as a finite set of string unification problem which is known as being decidable. ■

As a corollary from the above considerations we state

Theorem 5.9 *The algorithm enumerates all essential unifiers for an A-unification problem Γ and terminates if the set of essential unifiers is completed.*

6 Conclusion

The results reported above come as a disappointment to some extent: while the set of e-unifiers is considerably 'smaller' — albeit still infinite in general — than the set of most general unifiers for a string unification problem, the anticipated collapse of the infinitary theory to an e-finitary theory did not hold up to scrutiny.

This may not surprise those familiar with this problem, in spite of the simplicity and immediate intuitiveness of the problem formulation (using strings) the solvability as well as the unification problem turned out to be of exceptional difficulty and complexity.

For practical purposes, e.g. as a unification component within an automated theorem proving system, based on resolution or rewriting,there are two problem left over

1. To find a unification algorithm which generates — as efficiently as possible — the set of e-unifiers
2. To show how the reasoning machinery can be built upon e-unifiers instead of most general unifiers.

We have a solution to both problems, however far from anything practically useful: the unification algorithm is to resolution based theorem proving what the addition-and-multiplication unit is to a general purpose computer — and hence deserves the utmost effort in engineering, measured not in MiPs but in LiPs (logical inferences per sec, i.e. in fact the number of unifications p.sec) which was the hallmark of the fifth generation computer race in the 1980s.

Acknowledgements

We like to thank our first reviewer and the anonymous reviewers of this workshop.

The theoretically inclined reader, familiar with unification theory, may forgive us our lengthy motivational introduction and notational sections but unfortunately, as our first reviewer pointed out rightly, the concept of an essential unifier is still not as widely known as we would like, let alone fully appreciated as a fundamentally new concept. The results and counterexamples in this paper, however, make sense only in this particular context — hence the introduction.

References

[1] F. Baader. *Unification in idempotent semigroups is of type zero.* Journal of Automated Reasoning, 2(3), 1986.

[2] M. Lothaire (ed) *Algebraic Combinatorics on Words* Cambridge University Press, 2001.

[3] K. Schulz *Word Unification and Transformation of Generalized Equations* Journal of Automated Reasoning, vol. 11, pp 149-184, 1993

[4] F. Baader, T. Nipkow *Term Rewriting and all That* Cambridge University Press, 1998.

[5] M. Hoche, P. Szabó. *Essential Unifiers* Journal of Applied Logic, vol. 4, no. 1, 2006 University Press.

[6] F. Baader, J. Siekmann. *Unification theory.* in D. Gabbay, C. Hogger, and J. Robinson, eds, "Handbook of Logic in Artificial Intelligence and Logic Programming", 1994, Oxford University Press.

[7] F. Baader, W. Snyder. *Unification Theory.* In A. Robinson, A. Voronkov, editors, Handbook of Automated Reasoning Volume 1. Elsevier Science Publishers B. V. (North-Holland), 2001.

[8] H.-J. Bürckert, A. Herold, and M. Schmidt-Schauß. *On equational theories, unification and (un)decidability.* J. of Symbolic Computation 8, 3-49, 1989

[9] N. Dershowitz, J.-P. Jouannaud. *Rewrite Systems.* In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, chapter 6, pages 244-320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[10] E. Eder. *Properties of substitutions and unifications.* Journal of Symbolic Computation, 1(1):31-46, 1985.

[11] G. Huet. *A complete proof of correctness of the Knuth and Bendix completion algorithm.* Journal of Computer and System Sciences, 23:11-21, 1981.

[12] C. Kirchner, H. Kirchner. *Rewriting Solving Proving.* http://www.loria.fr/∼ckirchne or http://www.loria.fr/∼hkirchne

[13] G. Plotkin. *Building-in equational theories.* Machine Intelligence, 7:73-90, 1972.

[14] J.A.Robinson. *A machine-oriented logic based on the resolution principle.* Journal of the ACM, 12(1):23-41, 1965.

[15] M. Schmidt-Schauß. *Unification under Associativity and Idempotence is of type nullary.* Journal of Automated Reasoning, 2(3), 1986.

[16] J. Siekmann. *Unification and matching problems.* Ph.D. thesis, 1975, Essex University.

[17] J. Siekmann, P. Szabó. *A noetherian and confluent rewrite system for idempotent Semigroups.* Semigroup Forum, 25:83-110, 1982.

[18] J. Siekmann. *Unification theory.* Journal of Symbolic Computation, 7(3 & 4): 207-274, 1989. Special issue on unification. Part one.

[19] P. Szabó. *Unifikationstheorie erster Ordnung.* PhD thesis, University Karlsruhe, 1982.

[20] A. C. Varzi, *Parts, wholes, and part-whole relations.* The prospects of mereotopology, Data and Knowledge Engineering (DKE) Journal 20, North-Holland, Elsevier, 1996.

Author Index