International Conference on Rewriting Techniques and Applications 2010 (Edinburgh), pp. 193-208
http://rewriting.loria.fr/rta/

ORDER-SORTED UNIFICATION WITH REGULAR EXPRESSION SORTS

TEMUR KUTSIA [1] AND MIRCEA MARIN [2]

[1] Research Institute for Symbolic Computation
Johannes Kepler University Linz, Austria
E-mail address: `kutsia@risc.uni-linz.ac.at`
URL: `http://www.risc.uni-linz.ac.at/people/tkutsia/`

[2] Graduate School of Systems and Information Engineering
University of Tsukuba, Japan
E-mail address: `mmarin@cs.tsukuba.ac.jp`
URL: `http://www.score.is.tsukuba.ac.jp/~mmarin/`

Abstract. We extend first-order order-sorted unification by permitting regular expression sorts for variables and in the domains of function symbols. The set of basic sorts is finite. The obtained signature corresponds to a finite bottom-up hedge automaton. The unification problem in such a theory generalizes some known unification problems. Its unification type is infinitary. We give a complete unification procedure and prove decidability.

Introduction

In first-order order-sorted unification [Wal88], the set of basic sorts $\mathcal{B}$ is assumed to be partially ordered, variables are of basic sorts $\mathsf{s} \in \mathcal{B}$ and function symbols have sorts of the form $\mathsf{w} \to \mathsf{s}$, where w is a finite word over $\mathcal{B}$ and $\mathsf{s} \in \mathcal{B}$. We extend this framework by introducing regular expression sorts R over $\mathcal{B}$, allowing variables to be of sorts R and function symbols to have sorts $\mathsf{R} \to \mathsf{s}$. Another extension is that overloading function symbols is allowed. Under some reasonable conditions imposed over the signature [GM92], terms have the least sort.

Our signature has an interesting relation with automata. It is well-known that an order-sorted signature is a finite bottom-up tree automaton [Com89]. In our case, an order-sorted signature with regular expression sorts is exactly a finite bottom-up hedge automaton.

In this paper we study the unification problem for terms over an order-sorted signature with regular expression sorts. We call this problem *regular expression order-sorted unification* (REOSU) and show that it is infinitary, prove that it is decidable, and give a complete unification procedure.

1998 ACM Subject Classification: F.4.1 [Theory of Computation]: Mathematical Logic and Formal Languages—Mathematical Logic, F.2.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity—Nonnumerical Algorithms and Problems, F.4.3 [Theory of Computation]: Mathematical Logic and Formal Languages—Formal Languages.

Key words and phrases: Unification, sorts, regular expressions.

Proceedings of the 21st International Conference on Rewriting Techniques and Applications, Edinburgh, July, 2010
Editors: Christopher Lynch
LIPIcs - Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/LIPIcs.RTA.2010.193

REOSU extends some known problems as shown on the diagram below, illustrating its relations with syntactic unification (SYNU [Rob65]), word unification (WU [Sch90]), order-sorted unification (OSU [Wal88]), sequence unification (SEQU [Kut07]), and word unification with regular constraints (WURC [Sch90]):

$$\begin{tikzcd}
 & & \text{REOSU} \arrow[lld] \arrow[d] \arrow[rrd] & & \\
\text{OSU} \arrow[rd] & & \text{SEQU} \arrow[ld] \arrow[rd] & & \text{WURC} \arrow[ld] \\
 & \text{SYNU} & & \text{WU} &
\end{tikzcd}$$

Following the arrows, the problems are related as follows:

- From OSU one can obtain SYNU by restricting the sort hierarchy to be empty.
- SEQU problems without sequence variables (i.e., with individual variables only) constitute SYNU problems.
- WU is a special case of SEQU with constants, sequence variables, and only one flexible arity function symbol for concatenation.
- WU is also a special case of WURC where none of the variables are constrained.
- From REOSU we can get OSU (but with finitely many basic sort symbols only, because this is what REOSU considers) if instead of arbitrary regular sorts in function domains we allow only words over basic sorts, restrict variables to be of only basic sorts, and forbid function symbol overloading.
- SEQU can be obtained if we restrict REOSU with only one basic sort, say s, the variables that correspond to sequence variables in SEQU have the sort s^*, individual variables are of the sort s, and function symbols have the sort $\mathsf{s}^* \to \mathsf{s}$.
- WURC can be obtained from REOSU by the same restriction that gives WU from SEQU and, in addition, identifying the constants there to the corresponding sorts.

Order-sorted unification described in [SS89, Wei96] extends OSU from [Wal88] in a way that is not compatible with REOSU.

Regular expressions are presented in types in the programming language XDuce, designed for manipulating XML. These types are regular expressions over trees. They are ordered by a subtyping relation. Pattern matching for such regular expression types has been studied in [HP03]. Unlike XDuce types, our sorts are regular expressions over words and we perform word regular language manipulations rather than working with tree languages. Moreover, we are dealing with full-scale unification instead of matching.

In this paper we are dealing with REOSU in the empty theory (i.e., the syntactic case). It would also be interesting to see how one can extend equational OSU [Kir88, MGS89, Bou92, HM08] with regular expression sorts, but this problem is beyond the scope of this paper.

The paper is organized as follows. In Section 1 we give basic definitions and recall some known results. In Section 2 algorithms operating on sorts are given. Section 3 describes a complete unification procedure and discusses decidability. Section 4 concludes. Proofs can be found in the appendix.

For unification, we use the notation and terminology of [BS01]. For the notions related to sorted theories, we follow [GM92].

1. Preliminaries

Sorts

We consider a finite set $\mathcal{B}$ of basic sorts, partially ordered with the relation $\preceq$. Its elements are denoted with lowercase letters in $\mathsf{sans\ serif}$ font. $\mathsf{s} \prec \mathsf{r}$ means $\mathsf{s} \preceq \mathsf{r}$ and $\mathsf{s} \neq \mathsf{r}$. We write $\mathcal{R}$ for the set of regular expressions over $\mathcal{B}$, which is built in the standard way: $\mathsf{R} ::= \mathsf{s} \mid 1 \mid \mathsf{R}_1.\mathsf{R}_2 \mid \mathsf{R}_1{+}\mathsf{R}_2 \mid \mathsf{R}^*$. We use capital SANS SERIF font letters for them. The regular language denoted by a regular expression is: $[\![\mathsf{s}]\!] = \{\mathsf{s}\}$, $[\![1]\!] = \{\epsilon\}$, $[\![\mathsf{R}_1.\mathsf{R}_2]\!] = [\![\mathsf{R}_1]\!].[\![\mathsf{R}_2]\!]$, $[\![\mathsf{R}_1{+}\mathsf{R}_2]\!] = [\![\mathsf{R}_1]\!] \cup [\![\mathsf{R}_2]\!]$, $[\![\mathsf{R}^*]\!] = [\![\mathsf{R}]\!]^*$, where ϵ stands for the empty word, $[\![\mathsf{R}_1]\!].[\![\mathsf{R}_2]\!]$ is the concatenation of the regular languages $[\![\mathsf{R}_1]\!]$ and $[\![\mathsf{R}_2]\!]$, and $[\![\mathsf{R}]\!]^*$ is the Kleene star of $[\![\mathsf{R}]\!]$.

A *regular expression sort* is an element of $\mathcal{R}$, and a *functional expression sort* is an expression of the form $\mathsf{R} \to \mathsf{s}$ with $\mathsf{R} \in \mathcal{R}$ and $\mathsf{s} \in \mathcal{B}$. The relation $\preceq$ on $\mathcal{B}$ is extended to words of basic sorts, sets of words, and regular expression sorts as follows: (1) if $w_1, w_2 \in \mathcal{B}^*$ then $w_1 \preceq w_2$ iff $w_1 = \mathsf{s}_1 \cdots \mathsf{s}_n$, $w_2 = \mathsf{r}_1 \cdots \mathsf{r}_n$ and $\mathsf{s}_i \preceq \mathsf{r}_i$ for all $1 \leq i \leq n$; (2) if $W_1, W_2 \subseteq \mathcal{B}^*$ then $W_1 \preceq W_2$ iff for each $w_1 \in W_1$ there is $w_2 \in W_2$ such that $w_1 \preceq w_2$; and (3) if $\mathsf{R}_1, \mathsf{R}_2 \in \mathcal{R}$ then $\mathsf{R}_1 \preceq \mathsf{R}_2$ iff $[\![\mathsf{R}_1]\!] \preceq [\![\mathsf{R}_2]\!]$. Note that $\preceq$ is a quasi-order on the sets $\mathcal{B}$, $2^{\mathcal{B}^*}$, and $\mathcal{R}$. In particular, we can define the equivalence relation $\preceq$ on $\mathcal{R}$ by: $\mathsf{R}_1 \simeq \mathsf{R}_2$ iff $\mathsf{R}_1 \preceq \mathsf{R}_2$ and $\mathsf{R}_2 \preceq \mathsf{R}_1$. We extend this equivalence relation to functional sorts: $\mathsf{R}_1 \to \mathsf{s}_1 \simeq \mathsf{R}_2 \to \mathsf{s}_2$ iff $\mathsf{R}_1 \simeq \mathsf{R}_2$ and $\mathsf{s}_1 = \mathsf{s}_2$.

The *closure* $\overline{\mathsf{R}}$ of $\mathsf{R} \in \mathcal{R}$ is the regular expression defined as follows: $\overline{\mathsf{s}} = \sum_{\mathsf{r} \preceq \mathsf{s}} \mathsf{r}$, $\overline{1} = 1$, $\overline{\mathsf{R}_1.\mathsf{R}_2} = \overline{\mathsf{R}}_1.\overline{\mathsf{R}}_2$, $\overline{\mathsf{R}_1{+}\mathsf{R}_2} = \overline{\mathsf{R}}_1{+}\overline{\mathsf{R}}_2$, $\overline{\mathsf{R}^*} = \overline{\mathsf{R}}^*$. Closures of regular expressions enable the decidability of relations $\preceq$ and $\simeq$ on $\mathcal{R}$:

Lemma 1. *Let* $\mathsf{S}, \mathsf{R} \in \mathcal{R}$. *Then* $\mathsf{S} \preceq \mathsf{R}$ *iff* $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$.

Corollary 1. *Let* $\mathsf{S}, \mathsf{R} \in \mathcal{R}$. *Then* $\mathsf{S} \simeq \mathsf{R}$ *iff* $[\![\overline{\mathsf{S}}]\!] = [\![\overline{\mathsf{R}}]\!]$.

The set of all $\preceq$-maximal elements of a set of sorts $S \subseteq \mathcal{R}$ is denoted $\max(S)$. R is a lower bound of S if $\mathsf{R} \preceq \mathsf{Q}$ for all $\mathsf{Q} \in S$. A lower bound G of S is a greatest lower bound, denoted $\mathrm{glb}(S)$, if $\mathsf{R} \preceq \mathsf{G}$ for all lower bounds R of S. Note that if $\mathrm{glb}(S)$ exists, then it is unique modulo $\simeq$.

Terms

For each R we assume a countable set of variables $\mathcal{V}_{\mathsf{R}}$ such that $\mathcal{V}_{\mathsf{R}_1} = \mathcal{V}_{\mathsf{R}_2}$ iff $\mathsf{R}_1 \simeq \mathsf{R}_2$ and $\mathcal{V}_{\mathsf{R}_1} \cap \mathcal{V}_{\mathsf{R}_2} = \emptyset$ if $\mathsf{R}_1 \not\simeq \mathsf{R}_2$. Also, for each $\mathsf{R} \in \mathcal{R}, \mathsf{s} \in \mathcal{B}$ we assume a set of function symbols $\mathcal{F}_{\mathsf{R} \to \mathsf{s}}$ such that $\mathcal{F}_{\mathsf{R}_1 \to \mathsf{s}_1} = \mathcal{F}_{\mathsf{R}_2 \to \mathsf{s}_2}$ iff $\mathsf{R}_1 \to \mathsf{s}_1 \simeq \mathsf{R}_2 \to \mathsf{s}_2$. Moreover, the following conditions should be satisfied:

Preregularity: If $f \in \mathcal{F}_{\mathsf{R}_1 \to \mathsf{s}_1}$ and $\mathsf{R}_2 \preceq \mathsf{R}_1$, then there is a $\preceq$-least element in the set $\{\mathsf{s} \mid f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}} \text{ and } \mathsf{R}_2 \preceq \mathsf{R}\}$.

Finite overloading: For each f, the set $\{\mathcal{F}_{\mathsf{R} \to \mathsf{s}} \mid \mathsf{R} \in \mathcal{R}, \mathsf{s} \in \mathcal{B}, f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}\}$ is finite.

We say that R is a sort of x if $x \in \mathcal{V}_{\mathsf{R}}$. Similarly, $\mathsf{R} \to \mathsf{s}$ is a sort of f if $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$. Function symbols from $\mathcal{F}_{1 \to \mathsf{s}}$ are called constants. We use the letters a, b, c to denote them. We will write $f : \mathsf{R} \to \mathsf{s}$ for $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$, $a : \mathsf{s}$ for $a \in \mathcal{F}_{1 \to \mathsf{s}}$, and $x : \mathsf{R}$ for $x \in \mathcal{V}_{\mathsf{R}}$. Setting $\mathcal{V} = \cup_{\mathsf{R} \in \mathcal{R}} \mathcal{V}_{\mathsf{R}}$ and $\mathcal{F} = \cup_{\mathsf{R} \in \mathcal{R}, \mathsf{s} \in \mathcal{B}} \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$, we define the sets $\mathcal{T}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$ of *terms of sort* $\mathsf{R} \in \mathcal{R}$ *over* $\mathcal{V}$ *and* $\mathcal{F}$, and $\mathcal{TS}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$ of *term sequences of sort* $\mathsf{R} \in \mathcal{R}$ *over* $\mathcal{V}$ *and* $\mathcal{F}$, as the least sets satisfying the properties:

- $\mathcal{V}_{\mathsf{R}} \subseteq \mathcal{T}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$.
- $\mathcal{T}_{\mathsf{R}'}(\mathcal{F}, \mathcal{V}) \subseteq \mathcal{T}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$ if $\mathsf{R}' \preceq \mathsf{R}$.
- $\epsilon \in \mathcal{TS}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$ if $1 \preceq \mathsf{R}$.
- $(t_1, \dots, t_n) \in \mathcal{TS}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$ if there exist $\mathsf{R}_1, \dots, \mathsf{R}_n \in \mathcal{R}$ such that $t_i \in \mathcal{T}_{\mathsf{R}_i}(\mathcal{F}, \mathcal{V})$ and $\mathsf{R}_1. \cdots .\mathsf{R}_n \preceq \mathsf{R}$.
- $f(\tilde{t}) \in \mathcal{T}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$, if $\mathsf{R} = \mathsf{s}$, $f : \mathsf{R}' \to \mathsf{s}$, and $\tilde{t} \in \mathcal{TS}_{\mathsf{R}'}(\mathcal{F}, \mathcal{V})$.

The set of terms over $\mathcal{V}$ and $\mathcal{F}$ is defined as $\mathcal{T}(\mathcal{F}, \mathcal{V}) = \cup_{\mathsf{R} \in \mathcal{R}} \mathcal{T}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$. We abbreviate terms $a(\epsilon)$ with a. The *depth* of a term and a term sequence is defined in the standard way: $depth(x) = 1$, $depth(f(\tilde{t})) = 1 + depth(\tilde{t})$, $depth(\epsilon) = 0$, $depth(t_1, \dots, t_n) = \max\{depth(t_i) \mid 1 \le i \le n\}$, $n > 0$.

Lemma 2. *Every term has a $\preceq$-least sort* R *that is unique modulo $\simeq$.*

The $\preceq$-least sort of a term t modulo $\simeq$ is called the *least sort* of t, and is denoted by $lsort(t)$. In the same way, the $\preceq$-least sort of a term sequence $(t_1, \dots, t_n)$, $n \ge 1$, is defined uniquely modulo $\simeq$ as $lsort(t_1). \cdots .lsort(t_n)$ and is denoted by $lsort(t_1, \dots, t_n)$. When $n = 0$, i.e., for the empty sequence, $lsort(\epsilon) = 1$.

The set of variables of a term t is denoted by $var(t)$. A term t is ground if $var(t) = \emptyset$. These notions extend to term sequences, sets of term sequences, etc.

For a basic sort s, its semantics $sem(\mathsf{s})$ is the set $\mathcal{T}_{\mathsf{s}}(\mathcal{F})$ of ground terms of sort s. The semantics of a regular sort is given by the set of ground term sequences of the corresponding sort: $sem(1) = \{\epsilon\}$, $sem(\mathsf{R}_1.\mathsf{R}_2) = \{(\tilde{s}_1, \tilde{s}_2) \mid \tilde{s}_1 \in sem(\mathsf{R}_1), \tilde{s}_2 \in sem(\mathsf{R}_2)\}$, $sem(\mathsf{R}_1{+}\mathsf{R}_2) = sem(\mathsf{R}_1) \cup sem(\mathsf{R}_2)$, $sem(\mathsf{R}^*) = sem(\mathsf{R})^*$. This definition, together with the definition of $\preceq$ and $\mathcal{T}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$, implies that if $\mathsf{R} \preceq \mathsf{Q}$, then $sem(\mathsf{R}) \subseteq sem(\mathsf{Q})$.

Substitutions and Unification Problems

A *substitution* is a well-sorted mapping from variables to term sequences, which is identity almost everywhere. Substitutions are denoted with lowercase Greek letters, where ε stands for the identity substitution. Well-sortedness of σ means that $lsort(\sigma(x)) \preceq lsort(x)$ for all x. The notions of substitution application, term and term sequence instances, substitution composition, restriction, and subsumption are defined in the standard way. We use postfix notation for instances, juxtaposition for composition, and write $\sigma \le_{\mathcal{X}} \vartheta$ for subsumption meaning that σ is more general than ϑ on the set of variables $\mathcal{X}$. The *depth* of a substitution is defined as $depth(\sigma) = \max\{depth(x\sigma) \mid x \in \mathcal{V}\}$.

Lemma 3. *$lsort(t\sigma) \preceq lsort(t)$ and $lsort(\tilde{t}\sigma) \preceq lsort(\tilde{t})$ hold for any term t, term sequence $\tilde{t}$ and substitution σ.*

An *equation* is a pair of term sequences, written as $\tilde{s} \doteq \tilde{t}$. Its *depth* is the maximum between $depth(\tilde{s})$ and $depth(\tilde{t})$. A *regular expression order sorted unification* or, shortly, REOSU problem Γ is a finite set of equations between sorted term sequences $\{\tilde{s}_1 \doteq \tilde{t}_1, \dots, \tilde{s}_n \doteq \tilde{t}_n\}$. A substitution σ is a *unifier* of Γ if $\tilde{s}_i\sigma = \tilde{t}_i\sigma$ for all $1 \le i \le n$. A *minimal complete set* of unifiers of Γ is a set U of unifiers of Γ satisfying the following conditions:

Completeness: For any unifier ϑ of Γ there is $\sigma \in U$ such that $\sigma \le_{var(\Gamma)} \vartheta$.
Minimality: If there are $\sigma_1, \sigma_2 \in U$ such that $\sigma_1 \le_{var(\Gamma)} \sigma_2$, then $\sigma_1 = \sigma_2$.

The *depth* of a REOSU problem Γ is the maximum depth of the equations it contains.

Linear Form and Split of a Regular Expression

We recall the notion of linear form for regular expressions from [Ant96] by adapting the notation to our setting and using the set of basic sorts $\mathcal{B}$ for alphabet. This notion, together with the split of a regular expression, will be needed later, in sort-related algorithms: When we decompose a hedge in the weakening process, we have to split the corresponding sort as well. Linear form helps to split a sort into a basic sort and another sort, while the split operation decomposes it into two (not necessarily basic) sorts.

A pair (s, R) is called a *monomial*. A *linear form* of a regular expression R, denoted $\mathit{lf}(\mathsf{R})$, is a finite set of monomials defined recursively as follows:

$$\begin{array}{rcl@{\qquad}rcl}
\mathit{lf}(1) &=& \emptyset & \mathit{lf}(\mathsf{R}^*) &=& \mathit{lf}(\mathsf{R}) \odot \mathsf{R}^* \\
\mathit{lf}(\mathsf{s}) &=& \{(\mathsf{s}, 1)\} & \mathit{lf}(\mathsf{R.Q}) &=& \mathit{lf}(\mathsf{R}) \odot \mathsf{Q} \quad \text{if } \epsilon \notin [\![\mathsf{R}]\!] \\
\mathit{lf}(\mathsf{s{+}r}) &=& \mathit{lf}(\mathsf{s}) \cup \mathit{lf}(\mathsf{r}) & \mathit{lf}(\mathsf{R.Q}) &=& \mathit{lf}(\mathsf{R}) \odot \mathsf{Q} \cup \mathit{lf}(\mathsf{Q}) \quad \text{if } \epsilon \in [\![\mathsf{R}]\!]
\end{array}$$

These equations involve an extension of concatenation $\odot$ that acts on a linear form and a regular expression and returns a linear form. It is defined as $l \odot 1 = l$ and $l \odot \mathsf{Q} = \{(\mathsf{s}, \mathsf{S.Q}) \mid (\mathsf{s}, \mathsf{S}) \in l, \mathsf{S} \neq 1\} \cup \{(\mathsf{s}, \mathsf{Q}) \mid (\mathsf{s}, 1) \in l\}$ if $\mathsf{Q} \neq 1$.

As an example, $\mathit{lf}(\mathsf{R}) = \{(\mathsf{s}, \mathsf{R}), (\mathsf{s}, \mathsf{s.(s.s{+}r)^*}), (\mathsf{r}, \mathsf{(s.s{+}r)^*})\}$ for $\mathsf{R} = \mathsf{s^*.(s.s{+}r)^*}$. The set $\hat{\mathit{lf}}(\mathsf{R})$ is defined as $\{\mathsf{s.Q} \mid (\mathsf{s}, \mathsf{Q}) \in \mathit{lf}(\mathsf{R})\}$.

Definition 1 (Split)**.** Let $\mathsf{S} \in \mathcal{R}$. A *split* of S is a pair $(\mathsf{Q}, \mathsf{R}) \in \mathcal{R}^2$ such that (1) $\mathsf{Q.R} \preceq \mathsf{S}$ and (2) if $(\mathsf{Q}', \mathsf{R}') \in \mathcal{R}^2$, $\mathsf{Q} \preceq \mathsf{Q}'$, $\mathsf{R} \preceq \mathsf{R}'$, and $\mathsf{Q}'.\mathsf{R}' \preceq \mathsf{S}$, then $\mathsf{Q} \simeq \mathsf{Q}'$ and $\mathsf{R} \simeq \mathsf{R}'$.

We recall the definition of 2-factorization from [Con71]: A pair $(\mathsf{Q}, \mathsf{R}) \in \mathcal{R}^2$ is a *2-factorization* of $\mathsf{S} \in \mathcal{R}$ if (1) $[\![\mathsf{Q.R}]\!] \subseteq [\![\mathsf{S}]\!]$ and (2) if $(\mathsf{Q}', \mathsf{R}') \in \mathcal{R}^2$, $[\![\mathsf{Q}]\!] \subseteq [\![\mathsf{Q}']\!]$, $[\![\mathsf{R}]\!] \subseteq [\![\mathsf{R}']\!]$, and $[\![\mathsf{Q}'.\mathsf{R}']\!] \subseteq [\![\mathsf{S}]\!]$, then $[\![\mathsf{Q}]\!] = [\![\mathsf{Q}']\!]$ and $[\![\mathsf{R}]\!] = [\![\mathsf{R}']\!]$.

Lemma 4. (Q, R) *is a split of* S *iff* $(\overline{\mathsf{Q}}, \overline{\mathsf{R}})$ *is a 2-factorization of* $\overline{\mathsf{S}}$.

In [Con71] it has been shown that the 2-factorizations of a regular expression are finitely many modulo $\simeq$, and that they can be effectively computed. By the lemma above a regular expression has finitely many splits modulo $\simeq$ that can be effectively computed. For instance, the regular expression $\mathsf{s^*.r.r^*}$ has two splits modulo $\simeq$: $(\mathsf{s}^*, \mathsf{s^*.r.r^*})$ and $(\mathsf{s^*.r.r^*}, \mathsf{r}^*)$.

Relating REOS Signatures and Hedge Automata

Regular expression ordered sorts are related to regular hedge automata in the same way as ordered sorts are related to tree automata. Namely, a REOS signature is a finite bottom-up hedge automaton.

To illustrate this relation, we first recall the definition of nondeterministic finite hedge automaton (NFHA) from [CDG+]: An NFHA over Σ is a tuple (Q, Σ, Q_f, Δ) where Q is a finite set of states, $Q_f \subseteq Q$ is a set of final states, and Δ is a finite set of transition rules of the following types:

- $a(R) \to q$ where $a \in \Sigma$, $q \in Q$, and $R \subseteq Q^*$ is a regular language over Q, or
- $q' \to q$ (called ϵ-transitions), where $q', q \in Q$.

Now, we can take our set of basic sorts $\mathcal{B}$ in the role of Q, the set $\mathcal{F}$ in the role of Σ, assume $Q_f = Q$, and define Δ as follows: For each $\mathsf{r} \prec \mathsf{s}$, the ϵ-transition rule $\mathsf{r} \to \mathsf{s}$ is in Δ. For each $f : \mathsf{R} \to \mathsf{s}$, the rule $f(\mathsf{R}) \to \mathsf{s}$ is also in Δ. It is easy to see that our ground terms are exactly the unranked trees recognized by this automaton. A ground term of sort s is an unranked tree recognized by the automaton at state s.

2. Sort-Related Algorithms

In this section we identify algorithms to decide $\preceq$ on $\mathcal{R}$, to compute the greatest lower bounds for regular expression sorts, and to compute sort-weakening substitutions.

Deciding $\preceq$

Without an ordering on basic sorts, $\preceq$ would be the standard inequality for regular word expressions which can be decided, for instance, by Antimirov's algorithm [Ant95] that employs partial derivatives. The problem is PSPACE-complete, but this rewriting approach has an advantage over the standard technique of translating regular expressions into automata: With it, in some cases solving derivations can have polynomial size, while any algorithm based on translation of regular expressions into DFA's causes an exponential blow-up.

In our case, we can rely on the property that $\mathsf{S} \preceq \mathsf{R}$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$, proved in Lemma 1. The property $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$ can be decided by Antimirov's original algorithm on $\overline{\mathsf{S}}$ and $\overline{\mathsf{R}}$.

Computing Greatest Lower Bounds

A greatest lower bound of regular expressions would be their intersection, if we did not have ordering on the basic sorts. Intersection can be computed either in the standard way, by translating them into automata, or by Antimirov & Mosses's rewriting algorithm [AM95] for regular expressions extended with the intersection operator. Computation requires double exponential time.

Here we can employ the regular expression intersection algorithm [AM95] to compute a greatest lower bound, with one modification: To compute the intersection between two alphabet letters (i.e. between two basic sorts), instead of standard check whether they are the same, we compute the maximal elements in the set of their lower bounds. There can be several such maximal elements. This can be easily computed based on the ordering on basic sorts. Then we can take the sum of these elements and it will be a greatest lower bound. This construction allows to compute a greatest lower bound of two regular expressions, which is unique modulo $\simeq$.

An implementation of Antimirov-Mosses algorithm [Sul09] requires only minor modifications to deal with the ordering on alphabet letters (basic sorts). Hence, for S and R we compute here $\mathrm{glb}(\mathsf{S}, \mathsf{R})$ and we know that if Q is a regular expression with $[\![\mathsf{Q}]\!] = [\![\overline{\mathsf{S}}]\!] \cap [\![\overline{\mathsf{R}}]\!]$, then $\mathrm{glb}(\mathsf{S}, \mathsf{R}) \simeq \mathsf{Q}$.[1]

Computing Weakening Substitutions

Now we describe an algorithm that computes a substitution to weaken the sort of a term sequence towards a given sort. The necessity of such an algorithm can be demonstrated on a simple example: Assume we want to unify x and $f(y)$ for $x : \mathsf{s}$, $f : \mathsf{R}_1 \to \mathsf{s}_1$, $f : \mathsf{R}_2 \to \mathsf{s}_2$, $y : \mathsf{R}_2$, where $\mathsf{s}_1 \prec \mathsf{s} \prec \mathsf{s}_2$ and $\mathsf{R}_1 \prec \mathsf{R}_2$. We can not unify x with $f(y)$ directly, because $\mathit{lsort}(f(y)) = \mathsf{s}_2 \not\preceq \mathsf{s} = \mathit{lsort}(x)$. However, if we weaken the least sort of $f(y)$ to s_1, then unification is possible. To weaken the least sort of $f(y)$, we take its instance under

[1]We say that the computation of glb fails, if the (modification of) Antimirov-Mosses algorithm returns 0, and express it as $\mathrm{glb}(\mathsf{S}, \mathsf{R}) = \bot$.

substitution $\{y \mapsto z\}$, where $z \in \mathcal{V}_{\mathsf{R}_1}$, which gives $\mathit{lsort}(f(z)) = \mathsf{s}_1$. Hence, the substitution $\{y \mapsto z, x \mapsto f(z)\}$ is a unifier of x and $f(y)$, leading to the common instance $f(z)$.

A *weakening pair* is a pair of a term sequence $\tilde{t}$ and a sort Q, written $\tilde{t} \rightsquigarrow \mathsf{Q}$. A substitution ω is called a *weakening substitution* of a set W of weakening pairs iff $\mathit{lsort}(\tilde{t}\omega) \preceq \mathsf{Q}$ for each $\tilde{t} \rightsquigarrow \mathsf{Q} \in W$.

Our weakening algorithm is called $\mathfrak{W}$, and works by applying exhaustively the following rules to pairs of the form $W; \sigma$ where W is a set of weakening pairs and σ is a substitution:

R-w: **Remove a Weakening Pair**

$\{\tilde{t} \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow W; \sigma \qquad$ if $\mathit{lsort}(\tilde{t}) \preceq \mathsf{Q}$.

D1-w: **Decomposition 1 in Weakening**

$\{(f(\tilde{t}), \tilde{s}) \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \{f(\tilde{t}) \rightsquigarrow \mathsf{s}, \tilde{s} \rightsquigarrow \mathsf{S}\} \cup W; \sigma$

if $\mathit{lsort}(f(\tilde{t}), \tilde{s}) \not\preceq \mathsf{Q}$, $\mathit{var}(f(\tilde{t}), \tilde{s}) \neq \emptyset$, $\tilde{s} \neq \epsilon$ and $\mathsf{s.S} \in \max(\hat{\mathit{lf}}(\mathsf{Q}))$.

D2-w: **Decomposition 2 in Weakening**

$\{(x, \tilde{s}) \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \{x \rightsquigarrow \mathsf{Q}_1, \tilde{s} \rightsquigarrow \mathsf{Q}_2\} \cup W; \sigma$

if $\mathit{lsort}(x, \tilde{s}) \not\preceq \mathsf{Q}$, $\tilde{s} \neq \epsilon$ and $(\mathsf{Q}_1, \mathsf{Q}_2)$ is a split of Q.

AS-w: **Argument Sequence Weakening**

$\{f(\tilde{t}) \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \{\tilde{t} \rightsquigarrow \mathsf{R}\} \cup W; \sigma$

where $\mathit{lsort}(f(\tilde{t})) \not\preceq \mathsf{Q}$, $\mathit{var}(f(\tilde{t})) \neq \emptyset$, $\mathsf{R.r}$ is a maximal sort such that $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{r}}$ and $\mathsf{r} \preceq \mathsf{Q}$.

V-w: **Variable Weakening**

$\{x \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow W\sigma; \sigma\{x \mapsto w\}$

where $\mathrm{glb}(\{\mathit{lsort}(x), \mathsf{Q})\}) \neq \bot$ and w is a fresh variable from $\mathcal{V}_{\mathrm{glb}(\{\mathit{lsort}(x), \mathsf{Q})\})}$.

If none of the rules are applicable to $W; \sigma$, then it is transformed into $\bot$, indicating failure. By exhaustive search, transforming each $W; \sigma$ in all possible ways, we generate a complete search tree whose branches form *derivations*. The branches that end with $\bot$ are called failing branches. The branches that end with $\emptyset; \omega$ are called successful branches and ω is a substitution computed by $\mathfrak{W}$ along this branch. The set of all substitutions computed by $\mathfrak{W}$ starting from $W; \epsilon$ is denoted by $\mathit{weak}(W)$. It is easy to see that the elements of $\mathit{weak}(W)$ are variable renaming substitutions.

It is essential that the signature has the finite overloading property, which guarantees that the rule AS-w does not introduce infinite branching. Since the linear form and split of a regular expression are both finite, the other rules do not cause infinite branching either. $\mathfrak{W}$ is terminating, sound, and complete, as the following theorems show.

Theorem 1. *$\mathfrak{W}$ is terminating.*

Theorem 2 (Soundness of the Weakening Algorithm)**.** *Each $\omega \in \mathit{weak}(W)$ is a weakening substitution of W.*

Theorem 3 (Completeness of the Weakening Algorithm)**.** *For every weakening substitution ω of W there exists $\omega' \in \mathit{weak}(W)$ such that $\omega' \leq_{\mathit{var}(W)} \omega$.*

Example 1. Let $W = \{x \rightsquigarrow \mathsf{q}, f(x) \rightsquigarrow \mathsf{s}\}$ be a weakening problem with $x : \mathsf{r}$, $f : \mathsf{s} \to \mathsf{s}$, $f : \mathsf{r} \to \mathsf{r}$ and the sorts $\mathsf{r}_1 \prec \mathsf{r}$, $\mathsf{r}_2 \prec \mathsf{r}$, $\mathsf{r}_1 \prec \mathsf{q}$, $\mathsf{r}_2 \prec \mathsf{q}$, $\mathsf{s} \prec \mathsf{r}_1$, $\mathsf{s} \prec \mathsf{r}_2$. Then the weakening algorithm first transforms $W; \varepsilon$ into $\{f(w) \rightsquigarrow \mathsf{s}\}; \{x \mapsto w\}$ with $w : \mathsf{r}_1 + \mathsf{r}_2$ by the rule V-w.

The obtained weakening pair is then transformed into $\emptyset; \{\{x \mapsto z, w \mapsto z\}\}$ with $z : \mathsf{s}$ by $\mathsf{AS\text{-}w}$, leading to $\mathit{weak}(W) = \{\{x \mapsto z\}\}$.

Example 2. Let $W = \{(x, y) \rightsquigarrow \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*\}$ be a weakening problem with $x : \mathsf{q}_1^*.\mathsf{p}_1^*$, $y : \mathsf{q}_2^*.\mathsf{p}_2^*$, and the sorts $\mathsf{s} \prec \mathsf{q}_1$, $\mathsf{s} \prec \mathsf{q}_2$, $\mathsf{r} \prec \mathsf{p}_1$, $\mathsf{r} \prec \mathsf{p}_2$. Then the weakening algorithm computes $\mathit{weak}(W) = \{\{x \mapsto u_1, y \mapsto v_1\}, \{x \mapsto u_2, y \mapsto v_2\}\}$ where $u_1 : \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*$, $v_1 : \mathsf{r}^*$, $u_2 : \mathsf{s}^*$ and $v_2 : \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*$.

Example 3. Let $W = \{x \rightsquigarrow \mathsf{q}^*\}$ be a weakening problem with $x : \mathsf{r}^*$ and the sorts $\mathsf{s}_1 \prec \mathsf{r}$, $\mathsf{s}_2 \prec \mathsf{r}$, $\mathsf{s}_1 \prec \mathsf{q}$, $\mathsf{s}_2 \prec \mathsf{q}$, $\mathsf{p}_1 \prec \mathsf{s}_1$, $\mathsf{p}_2 \prec \mathsf{s}_2$. Then the weakening algorithm computes $\mathit{weak}(W) = \{\{x \mapsto w\}\}$ where $w : (\mathsf{s}_1 + \mathsf{s}_2)^*$.

3. Unification Type, Unification Procedure, Decidability

Unification Type

Let Γ_{re} be a REOSU problem and Γ_{seq} its version without sorts, i.e. a SEQU problem. Each unifier of Γ_{re} is either a unifier of Γ_{seq} or is obtained from a unifier of Γ_{seq} by composing it with a weakening substitution as follows: If $\sigma = \{x_1 \mapsto \tilde{t}_1, \ldots, x_n \mapsto \tilde{t}_n\}$ is a unifier of Γ_{seq}, then the set of weakening substitutions for σ is $\Omega(\sigma) = \mathit{weak}(\{\tilde{t}_1 \rightsquigarrow \mathit{lsort}(x_1), \ldots, \tilde{t}_n \rightsquigarrow \mathit{lsort}(x_n)\})$. For each $\omega_\sigma \in \Omega(\sigma)$, $\sigma\omega_\sigma$ is a unifier of Γ_{re}. Since SEQU is infinitary, the type of REOSU can be either infinitary or nullary, and we show now that it is not nullary.

Let S_{seq} be a minimal complete set of unifiers of Γ_{seq} and S_{re} be the set containing the unifiers of Γ_{re} that are either in S_{seq} or are obtained by weakening unifiers in S_{re}. Since $\{\sigma\omega_\sigma \mid \omega_\sigma \in \Omega(\sigma)\}$ is finite for each σ, we can assume that S_{re} contains only a minimal subset of it for each σ. The set S_{re} is complete. Assume by contradiction that it is not minimal. Then it contains σ' and ϑ' such that $\sigma' \leq_{\mathit{var}(\Gamma_{\mathrm{re}})} \vartheta'$, i.e., there exists φ' such that $\sigma'\varphi' =_{\mathit{var}(\Gamma_{\mathrm{re}})} \vartheta'$. If $\vartheta' \in S_{\mathrm{seq}}$, then we have $\sigma'\varphi' = \sigma\omega_\sigma\varphi' =_{\mathit{var}(\Gamma)} \vartheta'$ for an $\omega_\sigma \in \Omega(\sigma)$, which contradicts minimality of S_{seq}. If $\sigma' \in S_{\mathrm{seq}}$, then $\sigma'\varphi' =_{\mathit{var}(\Gamma_{\mathrm{re}})} \vartheta' = \vartheta\omega_\vartheta$ where $\omega_\vartheta \in \Omega(\vartheta)$. Since ω_ϑ is variable renaming, $\sigma'\varphi'\omega_\vartheta^{-1} =_{\mathit{var}(\Gamma_{\mathrm{seq}})} \vartheta$, which again contradicts minimality of S_{seq}. Both σ' and ϑ' can not be from S_{seq} because S_{seq} is minimal. If neither σ' nor ϑ' is in S_{seq}, then we have $\sigma\omega_\sigma\varphi' = \sigma'\varphi' =_{\mathit{var}(\Gamma_{\mathrm{re}})} \vartheta' = \vartheta\omega_\vartheta$ and again a contradiction: $\sigma\omega_\sigma\varphi'\omega_\vartheta^{-1} =_{\mathit{var}(\Gamma_{\mathrm{seq}})} \vartheta$.

Hence, for any Γ_{re} there is a complete set of unifiers with no two elements comparable with respect to $\leq_{\mathit{var}(\Gamma_{\mathrm{re}})}$, which implies that Γ_{re} has a minimal complete set of unifiers and REOSU is not nullary.

Unification Procedure

To compute unifiers for a REOSU problem, one way is, first, to ignore the sort information, employ the SEQU procedure [Kut02, Kut07] on the unsorted problem, and then weaken each computed substitution to obtain their order-sorted instances. In fact, such an approach is not uncommon in order-sorted unification, see, e.g. [SS89, MGS89, SNGM89, HM08]. It has an advantage of being a modular method that reuses an existing solving procedure.

In our case, this approach can be realized as follows: Assume a SEQU procedure computes a unifier $\sigma = \{x_1 \mapsto \tilde{t}_1, \ldots, x_n \mapsto \tilde{t}_n\}$[2] of the unsorted version of an REOSU problem Γ.

[2]We assume without loss of generality that σ is idempotent.

Then we form a set of weakening pairs $W = \{\tilde{t}_1 \rightsquigarrow \mathsf{Q}_1, \ldots, \tilde{t}_n \rightsquigarrow \mathsf{Q}_n\}$, where the Q's are the sorts of the corresponding x's, and find the set of weakening substitutions $\mathit{weak}(W)$. If $\mathit{weak}(W) = \emptyset$, then σ can not be weakened further to a solution of Γ. Otherwise, $\sigma\vartheta$ is a solution of Γ for each $\vartheta \in \mathit{weak}(W)$.

A drawback of this approach is that it is so called generate-and-test method. It is not able to detect early enough derivations that fail because of sort incompatibility. Early failure detection requires weakening to be tailored in the unification rules. This is what we consider in more details now.

The following transformation rules act on pairs of the form $\Gamma; \sigma$ with Γ a unification problem and σ a substitution, and are designed to define a sound and complete rule-based procedure for REOSU problems.

P: **Projection**

$\Gamma; \sigma \Longrightarrow \Gamma\vartheta; \sigma\vartheta$,

for $\vartheta = \{x_1 \mapsto \epsilon, \ldots, x_n \mapsto \epsilon\}$ with $x_i \in \mathit{var}(\Gamma)$ and $1 \preceq \mathit{lsort}(x_i)$ for $1 \leq i \leq n$.

T: **Trivial**

$\{\tilde{t} \doteq \tilde{t}\} \cup \Gamma; \sigma \Longrightarrow \Gamma; \sigma.$

TP: **Trivial Prefix**

$\{(\tilde{r}, \tilde{t}) \doteq (\tilde{r}, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq \tilde{s}\} \cup \Gamma; \sigma, \qquad$ if $\tilde{r} \neq \epsilon$ and $\tilde{t} \neq \tilde{s}$.

D: **Decomposition**

$\{(f(\tilde{t}), \tilde{t}') \doteq (f(\tilde{s}), \tilde{s}')\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq \tilde{s}, \tilde{t}' \doteq \tilde{s}'\} \cup \Gamma; \sigma,$

if $\mathrm{glb}(\{\mathit{lsort}(f(\tilde{t})), \mathit{lsort}(f(\tilde{s}))\}) \neq \bot$ and $\tilde{t} \neq \tilde{s}$.

O: **Orient**

$\{(t, \tilde{t}) \doteq (x, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{(x, \tilde{s}) \doteq (t, \tilde{t})\} \cup \Gamma; \sigma, \qquad$ where $t \notin \mathcal{V}$.

WkE1: **Weakening and Elimination 1**

$\{(x, \tilde{t}) \doteq (s, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

where $s \notin \mathcal{V}$, $x \notin \mathit{var}(s)$, $\omega \in \mathit{weak}(\{s \rightsquigarrow \mathit{lsort}(x)\})$, and $\vartheta = \omega \cup \{x \mapsto s\omega\}$.

WkE2: **Weakening and Elimination 2**

$\{(x, \tilde{t}) \doteq (y, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

where $\mathsf{R} = \mathrm{glb}(\mathit{lsort}(x), \mathit{lsort}(y)) \not\preceq 1$ and $\vartheta = \{x \mapsto w, y \mapsto w\}$ for a fresh variable $w \in \mathcal{V}_{\mathsf{R}}$.

WkWd1: **Weakening and Widening 1**

$\{(x, \tilde{t}) \doteq (s, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{(z, \tilde{t}) \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

if $s \notin \mathcal{V}$, $x \notin \mathit{var}(s)$, there is $(\mathsf{r}, \mathsf{R}) \in \mathit{lf}(\mathit{lsort}(x))$ with $\mathsf{R} \not\preceq 1$, $\omega \in \mathit{weak}(\{s \rightsquigarrow \mathsf{r}\})$, $z \in \mathcal{V}_{\mathsf{R}}$ is a fresh variable and $\vartheta = \omega \cup \{x \mapsto (s\omega, z)\}$.

WkWd2: **Weakening and Widening 2**

$\{(x, \tilde{t}) \doteq (y, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{(z, \tilde{t}) \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

where (S, R) is a split of $\mathit{lsort}(x)$ such that $\mathsf{R} \not\preceq 1$, $w \in \mathcal{V}_{\mathsf{R}'}$ is a fresh variable with $\mathsf{R}' = \mathrm{glb}(\{\mathsf{S}, \mathit{lsort}(y)\}) \not\preceq 1$, z is a fresh variable with $\mathit{lsort}(z) = \mathsf{R}$, and $\vartheta = \{x \mapsto (w, z), y \mapsto w\}$.

WkWd3: Weakening and Widening 3

$\{(x, \tilde{t}) \doteq (y, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq (z, \tilde{s})\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta$,

where (S, R) is a split of $\mathit{lsort}(y)$ such that $\mathsf{R} \not\simeq 1$, $w \in \mathcal{V}_{\mathsf{R}'}$ is a fresh variable with $\mathsf{R}' = \mathrm{glb}(\{\mathsf{S}, \mathit{lsort}(x)\}) \not\simeq 1$, z is a fresh variable with $\mathit{lsort}(z) = \mathsf{R}$, and $\vartheta = \{x \mapsto w, y \mapsto (w, z)\}$.

Note that $\mathsf{R}' \not\simeq 1$ in WkWd2 and WkWd3 implies that in those rules $\mathsf{S} \not\simeq 1$. We denote this set of transformation rules with $\mathfrak{T}$.

Theorem 4 (Soundness of Unification Rules)**.** *The rules of $\mathfrak{T}$ are sound.*

To solve a unification problem Γ, we create the initial pair $\Gamma; \varepsilon$ and first apply the projection rule to it in all possible ways. From each obtained problem we select an equation and apply the other rules exhaustively to that selected equation, developing the search tree in a breadth-first way. If no rule applies, the problem is transformed to $\bot$. The obtained procedure is denoted by $\mathfrak{P}(\Gamma)$. Branches in the search tree form *derivations*. The derivations that end with $\bot$ are *failing derivations*. The derivations that end with $\emptyset; \varphi$ are *successful derivations*. The set of all φ's at the end of successful derivations of $\mathfrak{P}(\Gamma)$ is called the *computed substitution set* of $\mathfrak{P}(\Gamma)$ and is denoted by $\mathit{comp}(\mathfrak{P}(\Gamma))$. From Theorem 4 by induction on the length of derivations one can prove that every $\varphi \in \mathit{comp}(\mathfrak{P}(\Gamma))$ is a unifier of Γ.

One can observe that under this control, variables are replaced with ϵ only at the projection phase. In particular, no variable introduced in intermediate stages gets eliminated with ϵ or replaced by a variable whose sort is 1.

Theorem 5 (Completeness of the Unification Procedure)**.** *Let Γ be a REOSU problem with a unifier ϑ. Then there exists $\sigma \in \mathit{comp}(\mathfrak{P}(\Gamma))$ such that $\sigma \leq_{\mathit{var}(\Gamma)} \vartheta$.*

Note that the set $\mathit{comp}(\mathfrak{P}(\Gamma))$, in general, is not minimal.[3]

By restricting sorts or occurrences of variables, various terminating fragments of REOSU can be obtained. We mention only four of them here:

- If sorts of all variables in Γ are star-free, then Γ is finitary. To show this, we first transform Γ into Γ', replacing each occurrence of a variable $x : \mathsf{R}_1.\mathsf{R}_2$ in Γ by a sequence of two fresh variables $x_1 : \mathsf{R}_1$ and $x_2 : \mathsf{R}_2$. Then, for each $y : \mathsf{R}_1 + \mathsf{R}_2$ in Γ', we obtain a new problem Γ'_1 by replacing each occurrence of y by a fresh variable $y_1 : \mathsf{R}_1$, and a a new problem Γ'_2 replacing each occurrence of y by a fresh variable $y_2 : \mathsf{R}_1$. Applying these transformations on each of the obtained problems iteratively, we reach a finite set of order-sorted unification problems, where each variable is of a basic sort. Since the set of basic sorts is finite, such problems are finitary [Wal88]. Γ is solvable if and only if at least one of the obtained problems is solvable. The transformation establishes a one-to-one correspondence between the unifiers of obtained problems and the unifiers of Γ, which implies that Γ is finitary.
- If variables whose sort contains the star occur in the last argument position. This is a pretty useful terminating fragment. One can formulate more optimized transformation rules for it and show termination based on the ideas of a similar fragment in sequence unification [Kut07].

[3]However, if in the rules WkE1 and WkE2 the substitution ω is selected from a minimal subset of the corresponding weakening set, one can show that $\mathit{comp}(\mathfrak{P}(\Gamma))$ is almost minimal. (Almost minimality is defined in [Kut07]).

- The previous fragment can be extended to another terminating fragment, called postfix-closed, where each occurrence of the same star-sorted variable is followed by the same sequence everywhere, like, e.g., in the problem $\{f(a, f(y, b), x, y, b) \doteq f(z, x, y, b)\}$, where the sorts of the variables x and y contain the star.
- If one side of each equation in Γ is ground, then Γ is finitary. These are REOS matching problems. For them, termination of $\mathfrak{P}(\Gamma)$ can be proved based on the ideas of termination proof for sequence matching in [Kut07]. Note that for REOS matching there is no need to invoke the weakening algorithm.

Now we demonstrate on an example how the unification procedure $\mathfrak{P}$ works:

Example 4. Let $\{f(x, y, z) \doteq f(f(x), g(u), a, b)\}$ be a REOSU problem, where the basic sorts are s, r, and q, ordered as $\mathsf{s} \prec \mathsf{q}$, $\mathsf{r} \prec \mathsf{q}$, and the symbols have the following sorts:

$$\begin{array}{ll} x, z : \mathsf{s}^* & f : \mathsf{q}^* \to \mathsf{r} \\ y, u : \mathsf{q} & g : \mathsf{q} \to \mathsf{q} \\ a, b : \mathsf{s} & g : \mathsf{s} + \mathsf{r} \to \mathsf{s}. \end{array}$$

That means, g is overloaded. We show a successful derivation for this problem. The first two steps are decomposition and projection:

$$\begin{aligned} &\{f(x, y, z) \doteq f(f(x), g(u), a, b)\}; \varepsilon \Longrightarrow_{\mathsf{D}} \\ &\{(x, y, z) \doteq (f(x), g(u), a, b)\}; \varepsilon \Longrightarrow_{\mathsf{P}} \\ &\{(y, z) \doteq (f(\epsilon), g(u), a, b)\}; \{x \mapsto \epsilon\} \end{aligned}$$

The weakening pair $f(\epsilon) \rightsquigarrow \mathsf{q}$ has ε as a weakening substitution. Hence, we can make the next step with the $\mathsf{WkE1}$ rule:

$$\begin{aligned} &\{(y, z) \doteq (f(\epsilon), g(u), a, b)\}; \{x \mapsto \epsilon\} \Longrightarrow_{\mathsf{WkE1}} \\ &\{z \doteq (g(u), a, b)\}; \{x \mapsto \epsilon, y \mapsto f(\epsilon)\} \end{aligned}$$

Now, $(\mathsf{s}, \mathsf{s}^*) \in \mathit{lf}(\mathit{lsort}(z))$. The least sort of $g(u)$ is $\mathsf{q} \not\preceq \mathsf{s}$. However, we can weaken $g(u)$ towards s: The weakening pair $g(u) \rightsquigarrow \mathsf{s}$ has a solution $\{u \mapsto v\}$, where $v \in \mathcal{V}_{\mathsf{s}+\mathsf{r}}$ is a fresh variable. We perform the $\mathsf{WkWd1}$ step, introducing a fresh variable $z_1 \in \mathcal{V}_{\mathsf{s}^*}$:

$$\begin{aligned} &\{z \doteq (g(u), a, b)\}; \{x \mapsto \epsilon, y \mapsto f(\epsilon)\} \Longrightarrow_{\mathsf{WkWd1}} \\ &\{z_1 \doteq (a, b)\}; \{x \mapsto \epsilon, y \mapsto f(\epsilon), u \mapsto v, z \mapsto (g(v), z_1)\} \end{aligned}$$

The next step is again $\mathsf{WkWd1}$. To make it, we take a weakening substitution ε for $a \rightsquigarrow \mathsf{s}^*$, a fresh variable $z_2 = \mathcal{V}_{\mathsf{s}^*}$ and proceed:

$$\begin{aligned} &\{z_1 \doteq (a, b)\}; \{x \mapsto \epsilon, y \mapsto f(\epsilon), u \mapsto v, z \mapsto (g(v), z_1)\} \Longrightarrow_{\mathsf{WkWd1}} \\ &\{z_2 \doteq b\}; \{x \mapsto \epsilon, y \mapsto f(\epsilon), u \mapsto v, z \mapsto (g(v), a, z_2), z_1 \mapsto (a, z_2)\} \end{aligned}$$

The last two steps in the derivation are $\mathsf{WkE1}$ and T. $\mathsf{WkE1}$ uses the weakening substitution ε for $b \rightsquigarrow \mathsf{s}^*$:

$$\begin{aligned} &\{z_2 \doteq b\}; \{x \mapsto \epsilon, y \mapsto f(\epsilon), u \mapsto v, z \mapsto (g(v), a, z_2), z_1 \mapsto (a, z_2)\} \Longrightarrow_{\mathsf{WkE1}} \\ &\{\epsilon \doteq \epsilon\}; \{x \mapsto \epsilon, y \mapsto f(\epsilon), u \mapsto v, z \mapsto (g(v), a, b), z_1 \mapsto (a, b), z_2 \mapsto b\} \Longrightarrow_{\mathsf{T}} \\ &\emptyset; \{x \mapsto \epsilon, y \mapsto f(\epsilon), u \mapsto v, z \mapsto (g(v), a, b), z_1 \mapsto (a, b), z_2 \mapsto b\}. \end{aligned}$$

Finally, restricting the computed substitution to the variables of the original problem $\{f(x,y,z) \doteq f(f(x),g(u),a,b)\}$, we obtain its unifier $\{x \mapsto \epsilon,\ y \mapsto f(\epsilon),\ u \mapsto v,\ z \mapsto (g(v),a,b)\}$.

Decidability

To show decidability, we define a translation from REOSU problems into word equations with regular constraints. The idea is similar to the one of [LV01], used to translate context equations into traversal equations, or of [KLV09], used to translate left-hole context equations into word equations with regular constraints.

For each basic sort we assume at least one constant of that sort and proceed as follows:

- First, we show that each solvable REOSU problem Γ has a unifier σ with the property $\mathit{depth}(\sigma) \leq \mathit{size}(\Gamma)$, where $\mathit{size}(\Gamma)$ is the number of alphabet symbols in Γ.
- Next, we transform a REOSU problem Γ into a WU problem with regular constraints by a transformation that preserves solvability in both directions. The transformation uses the minimal unifier depth bound when translating sort information. Since WURC is decidable, we get decidability of REOSU.

We now elaborate on these items. We can assume without loss of generality that we are looking for the unifiers that do not map any variable to ϵ (nonerasing unifiers).

Unifier depth bound. Let ϑ be a depth-minimal nonerasing unifier of Γ with the domain $\mathit{dom}(\vartheta) \subseteq \mathit{var}(\Gamma)$ and let ρ be a grounding substitution for $\Gamma\vartheta$, mapping each variable in $\Gamma\vartheta$ to a sequence of constants of appropriate sort. We denote $\vartheta\rho$ by σ. Then for each $x \in \mathit{var}(\Gamma)$, $x\sigma$ consists of terms of the form $t\sigma$, where t is either a subterm of Γ, or a constant, or is obtained from a subterm of Γ by replacing variables with sequences of constants. Since there are $\mathit{size}(\Gamma)$ subterms in Γ and we can not repeat application of a subterm on itself, $\mathit{depth}(t\sigma) \leq \mathit{size}(\Gamma)$. Therefore, $\mathit{depth}(x\sigma) \leq \mathit{size}(\Gamma)$ for all $x \in \mathit{dom}(\sigma)$ which implies $\mathit{depth}(\sigma) \leq \mathit{size}(\Gamma)$.

Translation into a WURC problem. Let Γ be a REOSU problem. For the translation, we restrict ourselves to the function symbols occurring in Γ and, additionally, one constant for each basic sort, if Γ does not contain a constant of that sort. This alphabet is finite. We denote it by $\mathcal{F}_\Gamma$.

First, we ignore the sort information and define a transformation Tr from term sequences into words as follows:

$$\begin{aligned}
\mathit{Tr}(x) &= x \\
\mathit{Tr}(f(\tilde{t})) &= f\,\mathit{Tr}(\tilde{t})f \\
\mathit{Tr}(\epsilon) &= \epsilon \\
\mathit{Tr}(t_1,\ldots,t_n) &= \mathit{Tr}(t_1)\#\cdots\#\mathit{Tr}(t_n),\ n > 1
\end{aligned}$$

where # is just a letter that does not occur in $\mathcal{F}_\Gamma$. A mapping σ from variables to term sequences is translated into a substitution for words $\mathit{Tr}(\sigma)$ defined as $x\,\mathit{Tr}(\sigma) = \mathit{Tr}(x\sigma)$ for each x. Tr is an injective function. Its inverse is denoted by Tr^{-1}.

Example 5. Let $\Gamma = \{f(x,y) \doteq f(f(y,a),b,c)\}$ with $\mathsf{s} \preceq \mathsf{r}$, $x : \mathsf{s}$, $y : \mathsf{r}^*$, $f : \mathsf{r}^* \to \mathsf{s}$, $a : \mathsf{s}$ and $b, c : \mathsf{r}$. Then Γ has a solution $\sigma = \{x \mapsto f(b,c,a), y \mapsto (b,c)\}$. On the other hand, $\mathit{Tr}(\Gamma) = \{fx\#yf \doteq ffy\#aaf\#bb\#ccf\}$ is a word unification problem with the nonerasing solutions $\varphi_1 = \{x \mapsto fbb\#cc\#aaf, y \mapsto bb\#cc\}$, $\varphi_2 = \{x \to fcc\#aaf\#bb, y \mapsto cc\}$, $\varphi_3 = \{x \mapsto faaf\#bbf\#cc, y \mapsto aaf\#bbf\#cc\}$. It is easy to see that $\varphi_1 = Tr(\sigma)$, but φ_2 and φ_3 are extra substitutions introduced by the transformation. However, they are of different nature: $Tr^{-1}(\varphi_2)$ exists and it is a mapping $\{x \mapsto (f(c,a),b), y \mapsto c\}$, but it is not a substitution because it is not well-sorted. $Tr^{-1}(\varphi_3)$ does not exist (which indicates that Tr is not surjective).

Lemma 5. *If σ is a substitution and $\tilde{t}$ is a sequence of REOS terms, then $Tr(\tilde{t})Tr(\sigma) = Tr(\tilde{t}\sigma)$.*

This lemma implies that if a REOSU Γ is solvable, then $Tr(\Gamma)$ is solvable. The converse, in general, is not true, because the transformation introduces extra solutions. However, translating sort information and considering word equations with regular constraints prevents extra solutions to appear and we get solvability preservation in both directions, as we will see below.

We start with translating sort information: For each $x \in var(\Gamma)$, we transform $x : \mathsf{R}$ into a membership constraint $x \in Tr(\mathsf{R},\Gamma)$, where $Tr(\mathsf{R},\Gamma)$ is defined as the set

$$\begin{aligned} Tr(\mathsf{R},\Gamma) = \{\, Tr(\tilde{t}) \mid & \text{ the terms in } \tilde{t} \text{ are from } \mathcal{T}(\mathcal{F}_\Gamma), \\ & lsort(\tilde{t}) \preceq \mathsf{R} \text{ and } depth(\tilde{t}) \le size(\Gamma)\}. \end{aligned}$$

That is, we translate only those $\tilde{t}$'s whose minimal sort does not exceed R and whose depth is bounded by $size(\Gamma)$.

We show now that $Tr(\mathsf{R},\Gamma)$ is a regular word language. First, we introduce a notation for regular word languages: We write $L_1 ._\# L_2$ for the language $\{w_1\#w_2 \mid w_1 \in L_1, w_2 \in L_2\}$. $L^{0\#} = \{\epsilon\}$, $L^{1\#} = L$, $L^{n\#} = L._\# L^{(n-1)\#}$ and $L^{*\#} = \cup_{n=0}^{\infty} L^{n\#}$.

For each $\mathcal{R}$, the language $Tr(\mathsf{R},\Gamma)$ is constructed level by level, first for the term sequences of depth 1, then for depth 2, and so on, until the depth bound $depth(\Gamma)$:

- Depth 1:

$$\begin{aligned} Tr_1(\mathsf{s},\Gamma) &= \{aa \mid a \in \mathcal{F}_\Gamma, a : \mathsf{s}', \mathsf{s}' \preceq \mathsf{s}\} \text{ (This set is finite.)} \\ Tr_1(1,\Gamma) &= \{\epsilon\} \\ Tr_1(\mathsf{R}_1 + \mathsf{R}_2,\Gamma) &= Tr_1(\mathsf{R}_1,\Gamma) \cup Tr_1(\mathsf{R}_2,\Gamma) \\ Tr_1(\mathsf{R}_1.\mathsf{R}_2,\Gamma) &= Tr_1(\mathsf{R}_1,\Gamma)._\# Tr_1(\mathsf{R}_2,\Gamma) \\ Tr_1(\mathsf{R}^*,\Gamma) &= Tr_1(\mathsf{R},\Gamma)^{*\#} \end{aligned}$$

- Depth $n > 1$:

$$\begin{aligned} Tr_n(\mathsf{s},\Gamma) &= Tr_{n-1}(\mathsf{s},\Gamma) \cup \{fwf \mid f \in \mathcal{F}_\Gamma, f : \mathsf{R} \to \mathsf{s}', \\ &\qquad w \in Tr_{n-1}(\mathsf{R}',\Gamma), \mathsf{R}' \preceq \mathsf{R}, \mathsf{s}' \preceq \mathsf{s}\} \\ Tr_n(1,\Gamma) &= \{\epsilon\} \\ Tr_n(\mathsf{R}_1 + \mathsf{R}_2,\Gamma) &= Tr_n(\mathsf{R}_1,\Gamma) \cup Tr_n(\mathsf{R}_2,\Gamma) \\ Tr_n(\mathsf{R}_1.\mathsf{R}_2,\Gamma) &= Tr_n(\mathsf{R}_1,\Gamma)._\# Tr_n(\mathsf{R}_2,\Gamma) \\ Tr_n(\mathsf{R}^*,\Gamma) &= Tr_n(\mathsf{R},\Gamma)^{*\#} \end{aligned}$$

It shows that $Tr_n(\mathsf{R},\Gamma)$ is regular for each n. From this construction it follows that $Tr(\mathsf{R},\Gamma) = Tr_{size(\Gamma)}(\mathsf{R},\Gamma)$ and, hence, $Tr(\mathsf{R},\Gamma)$ is regular.

Example 6. Consider again Γ and the sort information from Example 5. Now it gets translated into a WURC problem $\Delta = \{fx\#yf \doteq ffy\#aaf\#bb\#ccf,\ x \in \mathit{Tr}(\mathsf{s},\Gamma), y \in \mathit{Tr}(\mathsf{r}^*,\Gamma)\}$. $\mathit{Tr}(\mathsf{s},\Gamma)$ contains (among others) $fbb\#cc\#aaf$, but neither $fcc\#aaf\#bb$ nor $faaf\#bbf\#cc$. $\mathit{Tr}(\mathsf{r}^*,\Gamma)$ contains (among others) $bb\#cc$. Hence, φ_1 from Example 5 is a solution of Δ, but φ_2 and φ_3 are not.

Finally, we have the theorem:

Theorem 6. *Let* $\Gamma = \{\tilde{s}_1 \doteq \tilde{t}_1, \dots, \tilde{s}_n \doteq \tilde{t}_n\}$ *be a REOSU problem with* $\mathit{var}(\Gamma) = \{x_1, \dots, x_m\}$ *such that* $x_i : \mathsf{R}_i$ *for each* $1 \le i \le m$. *Let* $\Delta = \{\mathit{Tr}(\tilde{s}_1) \doteq \mathit{Tr}(\tilde{t}_1), \dots, \mathit{Tr}(\tilde{s}_n) \doteq \mathit{Tr}(\tilde{t}_n), x_1 \in \mathit{Tr}(\mathsf{R}_1,\Gamma), \dots, x_m \in \mathit{Tr}(\mathsf{R}_m,\Gamma)\}$ *be a word unification problem with regular constraints, obtained by translating* Γ. *Then* Γ *is solvable iff* Δ *is solvable.*

Hence, the problem of deciding solvability of REOSU has been (polynomially) reduced to the problem of deciding solvability of WURC. Since the latter is decidable, we conclude with the following result:

Theorem 7 (Decidability)**.** *Solvability of REOSU is decidable.*

4. Conclusion

We studied unification in order-sorted theories with regular expression sorts. We showed how it generalizes some known unification problems, proved its decidability and gave a complete unification procedure. A regular expression order-sorted signature can be viewed as a bottom-up finite hedge automaton. Such automata are considered to be a suitable framework for manipulating XML data. Since our language can model, to some extent, DTD and XML Schema, one can see a possible application (perhaps of its fragments) in the area related to XML processing.

Acknowledgments

This research has been partially supported by the EC FP6 Programme for Integrated Infrastructures Initiatives under the project SCIEnce—Symbolic Computation Infrastructure for Europe (Contract No. 026133) and by JSPS Grant-in-Aid no. 20500025 for Scientific Research (C).

References

[AM95] V. Antimirov and P. D. Mosses. Rewriting extended regular expressions. *Theoretical Computer Science*, 143(1):51–72, 1995.

[Ant95] V. Antimirov. Rewriting regular inequalities (extended abstract). In H. Reichel, editor, *Fundamentals of Computation Theory, 10th International Symposium FCT'95*, volume 965 of *LNCS*, pages 116–125. Springer, 1995.

[Ant96] V. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.

[Bou92] A. Boudet. Unification in order-sorted algebras with overloading. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction, CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1992.

[BS01] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.

[CDG$^+$] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available from: `http://www.grappa.univ-lille3.fr/tata`, version from October 12, 2007.

[Com89] H. Comon. Inductive proofs by specification transformation. In N. Dershowitz, editor, *Proc. 3rd International Conference on Rewriting Techniques and Applications, RTA'89*, volume 355 of *LNCS*, pages 76–91. Springer, 1989.

[Con71] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

[GM92] J. A. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

[HM08] J. Hendrix and J. Meseguer. Order-sorted unification revisited. In G. Kniesel and J. Sousa Pinto, editors, *Pre-proceedings of the 9th International Workshop on Rule-Based Programming, RULE'08*, pages 16–29, 2008.

[HP03] H. Hosoya and B. Pierce. Regular expression pattern matching for XML. *J. Functional Programming*, 13(6):961–1004, 2003.

[Kir88] C. Kirchner. Order-sorted equational unification. Presented at the fifth International Conference on Logic Programming (Seattle, USA), 1988. Also as rapport de recherche INRIA 954, December 1988.

[KLV09] T. Kutsia, J. Levy, and M. Villaret. On the relation between context and sequence unification. *J. Symbolic Computation*, 45(1):74–95, 2009.

[Kut02] T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning and Symbolic Computation. Proc. of Joint AISC'2002 – Calculemus'2002 Conference*, volume 2385 of *LNAI*, pages 290–304. Springer, 2002.

[Kut07] T. Kutsia. Solving equations with sequence variables and sequence functions. *J. Symbolic Computation*, 42(3):352–388, 2007.

[LV01] J. Levy and M. Villaret. Context unification and traversal equations. In A. Middeldorp, editor, *Proc. of the 12th International Conference on Rewriting Techniques and Applications, RTA'01*, volume 2041 of *LNCS*, pages 169–184. Springer, 2001.

[MGS89] J. Meseguer, J. A. Goguen, and G. Smolka. Order-sorted unification. *J. Symbolic Computation*, 8(4):383–413, 1989.

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

[Sch90] K. U. Schulz. Makanin's algorithm for word equations – two improvements and a generalization. In K. Schulz, editor, *Word Equations and Related Topics*, number 572 in LNCS, pages 85–150. Springer, 1990.

[SNGM89] G. Smolka, W. Nutt, J. A. Goguen, and J. Meseguer. Order-sorted equational computation. In M. Nivat and H. Aït-Kaci, editors, *Resolution of Equations in Algebraic Structures*, volume 2, pages 297–367. Academic Press, 1989.

[SS89] M. Schmidt-Schauß. *Computational Aspects of an Order-sorted Logic with Term Declarations.* Number 395 in Lecture Notes in Computer Science. Springer, 1989.

[Sul09] M. Sulzmann. regexpr-symbolic: Regular expressions via symbolic manipulation. `http://hackage.haskell.org/package/regexpr-symbolic`, 2009.

[Wal88] Ch. Walther. Many-sorted unification. *J. ACM*, 35(1):1–17, 1988.

[Wei96] Ch. Weidenbach. Unification in sort theories and its applications. *Annals of Mathematics and Artificial Intelligence*, 18(2):261–293, 1996.