Regular Expression Order-Sorted Unification and Matching

Temur Kutsia[a], Mircea Marin[b]

[a]*RISC, Johannes Kepler University Linz, Austria*
[b]*West University of Timişoara, Romania*

Abstract

We extend order-sorted unification by permitting regular expression sorts for variables and in the domains of function symbols. The obtained signature corresponds to a finite bottom-up unranked tree automaton. We prove that regular expression order-sorted (REOS) unification is of type infinitary and decidable. The unification problem presented by us generalizes some known problems, such as, e.g., order-sorted unification for ranked terms, sequence unification, and word unification with regular constraints. Decidability of REOS unification implies that sequence unification with regular hedge language constraints is decidable, generalizing the decidability result of word unification with regular constraints to terms. A sort weakening algorithm helps to construct a minimal complete set of REOS unifiers from the solutions of sequence unification problems. Moreover, we design a complete algorithm for REOS matching, and show that this problem is NP-complete and the corresponding counting problem is #P-complete.

Keywords: Unification, matching, ordered-sorts, regular expressions.

1. Introduction

Order-sorted algebra has been introduced in (Goguen, 1978), motivated by searching a better way to treat errors in abstract data types and to speed up certain theorem proving methods. In order-sorted algebras, variables and arguments of function symbols range over certain subsets of the universe of terms, specified by the sorts. Walther (1988) gave a syntactic unification algorithm for order-sorted terms, and characterized the relationship between sort hierarchies and the cardinality of minimal complete sets of unifiers.

Schmidt-Schauß (1989) extended Walther's work, permitting term declarations in sorted signatures. He studied syntactic unification algorithms and their complexities in various kinds of signatures. He also gave a complete procedure for sorted equational unification. Frisch and Cohn (1992) gave an abstract version of the sorted unification algorithm, independent of the sorted language being used, and reformulated Schmidt-Schauß's results in this setting. Uribe (1992) proved decidability of sorted unification in the so called semi-linear sort theories: A problem which Schmidt-Schauß left open. Weidenbach (1996) further generalized Schmidt-Schauß's and Uribe's results for syntactic sorted unification to more complex sort theories.

Since the original work by Goguen, several variants of the order-sorted algebra have been proposed, see (Goguen and Diaconescu, 1994) for a survey. Some of these variants permit overloaded function symbols. A desirable property of overloaded order-sorted algebras is the existence of a least sort for terms. Goguen and Meseguer (1992) gave conditions on the signature to guarantee the existence of such a sort. Equational unification algorithms for overloaded order-sorted algebras have been proposed in (Kirchner, 1988; Meseguer et al., 1989; Schmidt-Schauß, 1989; Boudet, 1992; Hendrix and Meseguer, 2012).

All the above mentioned work was done for order-sorted algebras over *ranked signatures,* where function symbols have a fixed arity. Comon (1989) observed an interesting relation between such signatures and tree automata: A finite ranked order-sorted signature is a finite bottom-up ranked tree automaton. Based on this observation, Comon and Delor (1994) used some strong properties of regular languages (decidability of emptiness and finiteness, stability under intersection, union and complement) to bring together the order-sorted framework and simplification of first-order equational formulas.

In this paper, we move from ranked to unranked signatures. Unranked terms/trees are commonly used as an abstract model of XML documents, program schemata, multithreaded recursive program configurations with an unbounded number of parallel processes, variadic functions in programming languages, etc. Rewriting, programming, model checking, knowledge representation techniques over unranked expressions have also been explored. Solving equations in one form or another is a fundamental problem in these applications. This is the problem we address in this paper.

More precisely, we generalize unification from ranked order-sorted terms without overloading to unranked order-sorted terms with overloading. Our sorts for variables and for function domains are described by regular expressions over basic sorts. Table 1 shows the detailed comparison of our language with the one in (Walther, 1988). The basic sorts in both papers are partially ordered. We consider the set $\mathcal{R}_{\mathcal{B}}$ of regular expressions over a poset $(\mathcal{B}, \preceq)$ of basic sorts, extend the partial order $\preceq$ to $\mathcal{R}_{\mathcal{B}}$, and, like Walther, restrict ourselves to syntactic unification.

The language in (Walther, 1988)	The language in this paper
The set of basic sorts $\mathcal{B}$, partially ordered with $\preceq$.	The finite set of basic sorts $\mathcal{B}$, partially ordered with $\preceq$.
Sets of variables $\mathcal{V}_{\mathsf{s}}$ for each $\mathsf{s} \in \mathcal{B}$.	Sets of variables $\mathcal{V}_{\mathsf{R}}$ for each $\mathsf{R} \in \mathcal{R}_{\mathcal{B}}$.
Sets of function symbols $\mathcal{F}_{\mathsf{w} \to \mathsf{s}}$ for $\mathsf{w} \in \mathcal{B}^*$, $\mathsf{s} \in \mathcal{B}$.	Sets of function symbols $\mathcal{F}_{\mathsf{R} \to \mathsf{s}}$ for $\mathsf{R} \in \mathcal{R}_{\mathcal{B}}$, $\mathsf{s} \in \mathcal{B}$.
The sets of function symbols and variables are pairwise disjoint.	The sets of function symbols are not required to be disjoint.

Table 1: Comparison with the order-sorted language from (Walther, 1988).

We abbreviate the regular expression order sorts used in the current paper as REOS. To guarantee the existence of a least sort, we extend the condition of *preregularity* defined for ranked order-sorted signatures in (Goguen and Meseguer, 1992) to REOS signatures. The *finite overloading property* of the REOS signature (the same function symbol can belong only to finitely many different sets of function symbols) guarantees that a least sort is effectively computable.

Table 1 reveals that our variables have regular expression sorts, thus they may be instantiated with term sequences by sort-preserving substitutions. The problem of unification in an unsorted language where variables stand for term sequences (sequence unification, SEQU) has been studied earlier, see, e.g. (Kutsia, 2007) and the discussion on related work thereof. Our work can be seen as a generalization of those to the sorted setting. It is well-known that a generalization of unsorted unification algorithms to the sorted ones is not trivial: Depending on the sort theory, it can happen that unification problems in unsorted and sorted versions of the same language belong to different unification types (e.g., unitary vs finitary, unitary vs infinitary, etc.) Putting it to an extreme, a sort theory may make a sorted version of the standard syntactic unification problem undecidable. See, e.g., (Weidenbach,

1996) for a more detailed discussion on sort theories and their effect on unification.

Like SEQU (Kutsia, 2007), REOS unification (REOSU, in short) problems may also have infinitely many incomparable unifiers. We prove that REOSU, in fact, is infinitary. It amounts to proving that REOSU is not of type zero, i.e., that a minimal complete set of unifiers always exists. Moreover, we prove that REOSU is decidable and describe sort weakening techniques which can be used to obtain a minimal complete set of sorted unifiers from the unsorted ones. A direct procedure to compute this set without transforming/filtering the unsorted unifiers can be found in the technical report (Kutsia and Marin, 2012).

The decidability result of REOSU has an interesting consequence: Decidability of sequence unification with regular hedge constraints. (Hedges are finite sequences of unranked terms.) This result generalizes decidability of word unification with regular constraints (Schulz, 1990) to term sequences.

Talking about related work, there are other known unification problems which can be seen as specializations of REOSU. The diagram in Fig. 1 illustrates how REOSU generalizes the syntactic unification SYNU (Robinson, 1965), word unification WU (Makanin, 1977; Schulz, 1990), order-sorted unification OSU (Walther, 1988), sequence unification SEQU (Kutsia, 2007), and word unification with regular constraints WRCU (Schulz, 1990):

$$\begin{tikzcd}
 & & \text{REOSU} \arrow[dll, no head] \arrow[d, no head] \arrow[drr, no head] & & \\
\text{OSU} \arrow[dr, no head] & & \text{SEQU} \arrow[dl, no head] \arrow[dr, no head] & & \text{WRCU} \arrow[dl, no head] \\
 & \text{SYNU} & & \text{WU} &
\end{tikzcd}$$

Figure 1: Relationship between REOSU and other unification problems.

The precise relationships between these problems can be described as follows:

- From OSU one can obtain SYNU by considering only one basic sort.
- 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 unranked function symbol for concatenation.

- WU is also a special case of WRCU where none of the variables is constrained.

- From REOSU we can get OSU (with finitely many basic sort symbols only, because this is what REOSU considers), if instead of arbitrary regular expression 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 to only one basic sort, say s, the variables that correspond to sequence variables in SEQU to have the sort s^*, individual variables to be of the sort s, and function symbols to have the sort $\mathsf{s}^* \to \mathsf{s}$.

- WRCU can be obtained from REOSU by the same restriction that gives WU from SEQU and, in addition, identifying the constants in REOSU to the sorts they belong to.

The order-sorted unification problems considered in (Schmidt-Schauß, 1989; Weidenbach, 1996) extend OSU from (Walther, 1988) by introducing term declarations. REOSU does not consider such declarations.

When it comes to applications of unification, finitary fragments and variants are of special interest. A particularly useful such restriction is matching, where one side of the unification problem is variable-free (ground). We study REOS matching in this paper, give a complete matching algorithm, and prove that it terminates and never computes the same matcher more than once. We also prove its NP-completeness and the #P-completeness of the corresponding counting problem. The REOS matching can be seen as an abstract model of the basic pattern matching algorithm on which the programming language of the Mathematica system (Wolfram, 2003) is based.

Yet another interesting feature of our language is that we can relate regular expression order-sorted signatures and unranked tree automata (Comon et al., 2007) similarly to the relationship between the ranked order-sorted signatures and automata mentioned above. Namely, we show that a REOS signature is exactly a finite bottom-up unranked tree automaton. Taking into account the closure properties of unranked tree automata, this result can help, for instance, in developing simplification techniques for arbitrary equational formulas in the REOS framework. We do not go into a more detailed discussion here, as this topic requires a thorough investigation which is beyond the scope of the current paper.

Regular expression typed pattern matching is presented in the programming languages XDuce (Hosoya and Pierce, 2003b), designed for manipulating XML, and in XHaskell (Sulzmann and Lu, 2007), an extension of Haskell. 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 (Hosoya and Pierce, 2003a). Unlike XDuce types, our sorts are regular expressions over words and we perform regular word language manipulations rather than working with tree languages. Moreover, we deal not only with matching, but also with full-scale unification. Other work related to REOS matching is described in (Kutsia and Marin, 2005a,b), where some variables in matching are constrained by regular hedge languages.

In this paper we study REOSU in the empty theory (i.e., the syntactic case). It would be interesting to see how one can extend equational OSU (Kirchner, 1988; Meseguer et al., 1989; Boudet, 1992; Hendrix and Meseguer, 2012) with regular expression sorts, but this problem is beyond the scope of this paper.

2. Preliminaries

In this paper, for unification and matching we use the notation and terminology of Baader and Snyder (2001). For the notions related to sorted theories, we follow Goguen and Meseguer (1992).

2.1. Sorts

We consider a finite poset $(\mathcal{B}, \preceq)$ of basic sorts ranged over by $\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}, \mathsf{t}$. We write $\mathsf{s} \prec \mathsf{r}$ if $\mathsf{s} \preceq \mathsf{r}$ and $\mathsf{s} \neq \mathsf{r}$. Also, we write $\mathcal{R}_{\mathcal{B}}$ for the set of regular expressions over $\mathcal{B}$, built by the grammar $\mathsf{R} ::= \mathsf{s} \mid \mathsf{1} \mid \mathsf{R}_1.\mathsf{R}_2 \mid \mathsf{R}_1{+}\mathsf{R}_2 \mid \mathsf{R}^*$. We use capital SANS SERIF font letters for them. Usually, we omit the subscript and write $\mathcal{R}$ for $\mathcal{R}_{\mathcal{B}}$, and call the elements of $\mathcal{R}$ *regular expression sorts*.

The regular language $[\![\mathsf{R}]\!]$ denoted by a regular expression R is defined in the standard way: $[\![\mathsf{s}]\!] = \{\mathsf{s}\}$, $[\![\mathsf{1}]\!] = \{\lambda\}$, $[\![\mathsf{R}_1.\mathsf{R}_2]\!] = [\![\mathsf{R}_1]\!].[\![\mathsf{R}_2]\!]$, $[\![\mathsf{R}_1{+}\mathsf{R}_2]\!] = [\![\mathsf{R}_1]\!] \cup [\![\mathsf{R}_2]\!]$, $[\![\mathsf{R}^*]\!] = [\![\mathsf{R}]\!]^*$, where λ 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 $[\![\mathsf{R}]\!]$.

Besides regular expression sorts, we also consider *functional expression sorts*, which are pairs made of $\mathsf{R} \in \mathcal{R}$ and $\mathsf{s} \in \mathcal{B}$, written as $\mathsf{R} \to \mathsf{s}$. The relation $\preceq$ on $\mathcal{B}$ is extended to words of basic sorts, sets of words, and regular expression sorts as follows:

1. if $w_1 \in \mathcal{B}^*$ and $w_2 \in \mathcal{B}^*$, 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 $W_1 \subseteq \mathcal{B}^*$ and $W_2 \subseteq \mathcal{B}^*$, then $W_1 \preceq W_2$ iff for each $w_1 \in W_1$ there is $w_2 \in W_2$ such that $w_1 \preceq w_2$;
3. if $\mathsf{R}_1 \in \mathcal{R}$ and $\mathsf{R}_2 \in \mathcal{R}$, 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 2.1. *Let* $\mathsf{S} \in \mathcal{R}$ *and* $\mathsf{R} \in \mathcal{R}$. *Then* $\mathsf{S} \preceq \mathsf{R}$ *iff* $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$.

Proof. An easy proof by induction on the structure of $\mathsf{R} \in \mathcal{R}$ reveals that

(1) $[\![\overline{\mathsf{R}}]\!] \preceq [\![\mathsf{R}]\!] \preceq [\![\overline{\mathsf{R}}]\!]$, therefore $\mathsf{R} \simeq \overline{\mathsf{R}}$, and
(2) for all $w \in \mathcal{B}^*$ we have $\{w\} \preceq [\![\overline{\mathsf{R}}]\!]$ iff $w \in [\![\overline{\mathsf{R}}]\!]$.

(2) implies $W \preceq [\![\overline{\mathsf{R}}]\!]$ iff $W \subseteq [\![\overline{\mathsf{R}}]\!]$ for all $W \subseteq \mathcal{B}^*$. In particular, for $W = [\![\overline{\mathsf{S}}]\!]$ we obtain $[\![\overline{\mathsf{S}}]\!] \preceq [\![\overline{\mathsf{R}}]\!]$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$.

If $\mathsf{S} \preceq \mathsf{R}$ then $\overline{\mathsf{S}} \simeq \mathsf{S} \preceq \mathsf{R} \simeq \overline{\mathsf{R}}$. Since $\preceq$ is transitive, we learn $\overline{\mathsf{S}} \preceq \overline{\mathsf{R}}$, that is, $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$. Conversely, if $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$ then obviously $\overline{\mathsf{S}} \preceq \overline{\mathsf{R}}$. Since $\mathsf{S} \preceq \overline{\mathsf{S}}$ and $\overline{\mathsf{R}} \preceq \mathsf{R}$, by transitivity of $\preceq$ we conclude that $\mathsf{S} \preceq \mathsf{R}$. □

Thus, we can decide $\mathsf{S} \preceq \mathsf{R}$ by deciding $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$. This can be achieved with the rewriting-based algorithm of Antimirov (1995). The problem is PSPACE-complete, but this rewriting approach has an advantage over the standard technique of translating regular expressions into automata: In some cases, it provides derivations of polynomial size, while any algorithm based on the translation of regular expressions into DFAs causes an exponential blow-up.

Corollary 1. *Let* $\mathsf{S} \in \mathcal{R}$ *and* $\mathsf{R} \in \mathcal{R}$. *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 by $\max(S)$. 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 $\simeq$.

Example 2.2. Let $\mathsf{s}_1, \mathsf{s}_2, \mathsf{r}, \mathsf{q}$ be basic sorts ordered as follows: $\mathsf{s}_1 \prec \mathsf{r}$, $\mathsf{s}_2 \prec \mathsf{r}$, $\mathsf{s}_1 \prec \mathsf{q}$, $\mathsf{s}_2 \prec \mathsf{q}$. Let $S_1 = \{\mathsf{s}_1, \mathsf{s}_2, \mathsf{r}, \mathsf{q}\}$, $S_2 = \{\mathsf{s}_2, \mathsf{r}, \mathsf{q}\}$, and $S_3 = \{\mathsf{r}, \mathsf{q}\}$. Then

- $\max(S_1) = \max(S_2) = \max(S_3) = \{\mathsf{r}, \mathsf{q}\}$.
- S_1 has no lower bounds.
- s_2 is the only lower bound of S_2. Obviously, $\mathsf{s}_2 = \mathrm{glb}(S_2)$.
- s_1, s_2, and $\mathsf{s}_1{+}\mathsf{s}_2$ are lower bounds of S_3 and $\mathsf{s}_1{+}\mathsf{s}_2 = \mathrm{glb}(S_3)$.

To avoid excessive use of parentheses in regular expressions, we give the Kleene star $*$ the highest priority, followed by concatenation . and then by choice $+$. For instance, $\mathsf{s.r}^*{+}\mathsf{q}$ stands for $(\mathsf{s}.(\mathsf{r}^*)){+}\mathsf{q}$.

The following subsection recalls results from the factorization theory of regular languages. We anticipate that these results will be useful in the study of unification problems that will show up in Sect. 2.4.

2.2. Linear Form and Split of a Regular Expression

We recall the notion of linear form for regular expressions from (Antimirov, 1996) by adapting the notation to our setting and using the set of basic sorts $\mathcal{B}$ as alphabet. This notion, together with the split of a regular expression, will be needed later, in sort-related algorithms. Linear forms help 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 $(\mathsf{s}, \mathsf{R}) \in \mathcal{B} \times \mathcal{R}$ is called a *monomial.* A *linear form* of a regular expression R, denoted $\mathit{lf}(\mathsf{R})$, is a finite set of monomials, representing all possible ways of splitting away the first symbol of regular expressions. Linear forms are defined recursively as follows:

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

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 $l \odot \mathsf{1} = l$ and $l \odot \mathsf{Q} = \{(\mathsf{s}, \mathsf{S.Q}) \mid (\mathsf{s}, \mathsf{S}) \in l, \mathsf{S} \neq \mathsf{1}\} \cup \{(\mathsf{s}, \mathsf{Q}) \mid (\mathsf{s}, \mathsf{1}) \in l\}$ if $\mathsf{Q} \neq \mathsf{1}$. The set $\hat{\mathit{lf}}(\mathsf{R})$ is defined as $\{\mathsf{s.Q} \mid (\mathsf{s}, \mathsf{Q}) \in \mathit{lf}(\mathsf{R})\}$.

Example 2.3. If $\mathsf{R} = \mathsf{s}^*.(\mathsf{s.s}{+}\mathsf{r})^*$ then $\hat{\mathit{lf}}(\mathsf{R}) = \{\mathsf{s.R}, \mathsf{s.s}.(\mathsf{s.s}{+}\mathsf{r})^*, \mathsf{r}.(\mathsf{s.s}{+}\mathsf{r})^*\}$.

Definition 2.4 (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) $\mathsf{Q}.\mathsf{R} \preceq \mathsf{S}$ 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 (Conway, 1971): A pair $(\mathsf{Q},\mathsf{R}) \in \mathcal{R}^2$ is a *2-factorization* of $\mathsf{S} \in \mathcal{R}$ if (1) $[\![\mathsf{Q}.\mathsf{R}]\!] \subseteq [\![\mathsf{S}]\!]$ 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 2.5. (Q,R) *is a split of* S *iff* $(\overline{\mathsf{Q}},\overline{\mathsf{R}})$ *is a 2-factorization of* $\overline{\mathsf{S}}$.

Proof. (Q,R) is a split of S iff (1) $\mathsf{Q}.\mathsf{R} \preceq \mathsf{S}$ 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 $\mathsf{R} \simeq \mathsf{R}'$. By Lemma 2.1, these conditions are equivalent to (1') $[\![\overline{\mathsf{Q}.\mathsf{R}}]\!] \subseteq [\![\overline{\mathsf{S}}]\!]$ and (2') if $(\mathsf{Q}',\mathsf{R}') \in \mathcal{R}^2$, $[\![\overline{\mathsf{Q}}]\!] \subseteq [\![\overline{\mathsf{Q}'}]\!]$, $[\![\overline{\mathsf{R}}]\!] \subseteq [\![\overline{\mathsf{R}'}]\!]$, and $[\![\overline{\mathsf{Q}'.\mathsf{R}'}]\!] \subseteq [\![\overline{\mathsf{S}}]\!]$, then $[\![\overline{\mathsf{Q}}]\!] = [\![\overline{\mathsf{Q}'}]\!]$ and $[\![\overline{\mathsf{R}}]\!] = [\![\overline{\mathsf{R}'}]\!]$. It is not hard to see that (1') and (2') are the same as saying that $(\overline{\mathsf{Q}},\overline{\mathsf{R}})$ is a 2-factorization of $\overline{\mathsf{S}}$. □

In (Conway, 1971) 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 Lemma 2.5 a regular expression has finitely many splits modulo $\simeq$ that can be effectively computed. For instance, the regular expression $\mathsf{s}^*.\mathsf{r}.\mathsf{r}^*$ has three splits modulo $\simeq$: $(\mathsf{s}^*, \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*)$, $(\mathsf{s}^*\mathsf{r}^*, \mathsf{r}.\mathsf{r}^*)$, and $(\mathsf{s}^*.\mathsf{r}.\mathsf{r}^*, \mathsf{r}^*)$.

The following lemma is an easy consequence of Lemma 2.5 above and Conway's Theorem 1 (Conway, 1971, Ch. 6):

Lemma 2.6. $\mathsf{R}_1'.\mathsf{R}_2' \preceq \mathsf{R}$ *iff there exists a split* $(\mathsf{R}_1,\mathsf{R}_2)$ *of* R *such that* $\mathsf{R}_1' \preceq \mathsf{R}_1$ *and* $\mathsf{R}_2' \preceq \mathsf{R}_2$.

2.3. Terms and Term Sequences

These notions are defined with respect to a regular expression order-sorted (REOS) signature and a countable set of sorted variables. A *REOS signature* is a triple $\Sigma = (\mathcal{B}, \preceq, \mathcal{F})$ made of a finite set $\mathcal{B}$ of basic sorts, a partial ordering $\preceq$ on $\mathcal{B}$ which is extended to the set $\mathcal{R}$ of regular expressions over $\mathcal{B}$, and a set $\mathcal{F} = \bigcup_{\mathsf{R}\in\mathcal{R},\mathsf{s}\in\mathcal{B}} \mathcal{F}_{\mathsf{R}\to\mathsf{s}}$ corresponding to a family $\{\mathcal{F}_{\mathsf{R}\to\mathsf{s}} \mid \mathsf{R} \in \mathcal{R}, \mathsf{s} \in \mathcal{B}\}$ of sets of function symbols which satisfy the following conditions:

Functional equivalence: If $\mathsf{R}_1 \to \mathsf{s}_1 \simeq \mathsf{R}_2 \to \mathsf{s}_2$ then $\mathcal{F}_{\mathsf{R}_1\to\mathsf{s}_1} = \mathcal{F}_{\mathsf{R}_2\to\mathsf{s}_2}$.

Monotonicity: If $f \in \mathcal{F}_{\mathsf{R}_1\to\mathsf{s}_1} \cap \mathcal{F}_{\mathsf{R}_2\to\mathsf{s}_2}$ and $\mathsf{R}_1 \preceq \mathsf{R}_2$, then $\mathsf{s}_1 \preceq \mathsf{s}_2$.

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.

The corresponding set of variables is $\mathcal{V} = \bigcup_{\mathsf{R}\in\mathcal{R}} \mathcal{V}_{\mathsf{R}}$, where every $\mathcal{V}_{\mathsf{R}}$ is a countably infinite set of variables such that $\mathcal{V}_{\mathsf{R}_1} = \mathcal{V}_{\mathsf{R}_2}$ iff $\mathsf{R}_1 \simeq \mathsf{R}_2$ and $\mathcal{V}_{\mathsf{R}_1} \cap \mathcal{V}_{\mathsf{R}_2} = \emptyset$ iff $\mathsf{R}_1 \not\simeq \mathsf{R}_2$.

As usual, we assume that $\mathcal{F} \cap \mathcal{V} = \emptyset$.

Definition 2.7. The set of *terms of sort* $\mathsf{R} \in \mathcal{R}$ *over* Σ *and* $\mathcal{V}$, denoted by $\mathcal{T}_{\mathsf{R}}(\Sigma, \mathcal{V})$, and the set of *term sequences of sort* $\mathsf{R} \in \mathcal{R}$ *over* Σ *and* $\mathcal{V}$, denoted by $\mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$, are the least sets satisfying the properties:

- $\mathcal{V}_{\mathsf{R}} \subseteq \mathcal{T}_{\mathsf{R}}(\Sigma, \mathcal{V})$.
- $\mathcal{T}_{\mathsf{R}'}(\Sigma, \mathcal{V}) \subseteq \mathcal{T}_{\mathsf{R}}(\Sigma, \mathcal{V})$ and $\mathcal{S}_{\mathsf{R}'}(\Sigma, \mathcal{V}) \subseteq \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$ if $\mathsf{R}' \preceq \mathsf{R}$.
- $\epsilon \in \mathcal{S}_{1}(\Sigma, \mathcal{V})$.
- The term sequence $t_1, \dots, t_n \in \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$,[1] $n \geq 1$, if there exist $\mathsf{R}_1 \in \mathcal{R}$, $\dots$, $\mathsf{R}_n \in \mathcal{R}$ such that $t_i \in \mathcal{T}_{\mathsf{R}_i}(\Sigma, \mathcal{V})$ and $\mathsf{R}_1 . \cdots . \mathsf{R}_n = \mathsf{R}$.
- $f(t_1, \dots, t_n) \in \mathcal{T}_{\mathsf{R}}(\Sigma, \mathcal{V})$, if $\mathsf{R} = \mathsf{s}$, $f : \mathsf{R}' \to \mathsf{s}$, and $t_1, \dots, t_n \in \mathcal{S}_{\mathsf{R}'}(\Sigma, \mathcal{V})$.

Thus, the set of sorted terms is $\bigcup_{\mathsf{R}\in\mathcal{R}} \mathcal{T}_{\mathsf{R}}(\Sigma, \mathcal{V})$, which we denote by $\mathcal{T}(\Sigma, \mathcal{V})$. The set of sorted term sequences $\mathcal{S}(\Sigma, \mathcal{V})$ is defined similarly. Note that $\mathcal{T}_{\mathsf{R}}(\Sigma, \mathcal{V}) \subseteq \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$ holds for all $\mathsf{R} \in \mathcal{R}$. In other words, we do not distinguish between a term and a singleton term sequence. Sorted terms of the form $a(\epsilon)$ are abbreviated with a. For readability, we may write term sequences within parentheses, usually when there is more than one element in the sequence.

From now on we assume implicitly that all terms and term sequences under consideration are sorted, therefore we will stop mentioning them to be sorted. We denote terms by symbols t, s, and r, and term sequences by $\tilde{t}, \tilde{s}$, and $\tilde{r}$. For variables, we use x, y, z, u, v, and w.

If $\tilde{t} = (t_1, \dots, t_n)$ and $\tilde{s} = (s_1, \dots, s_m)$, $n, m \geq 0$, we slightly overload the comma, writing $(\tilde{t}, \tilde{s})$ for the term sequence $(t_1, \dots, t_n, s_1, \dots, s_m)$. Obviously, when $n = 0$, i.e., when $\tilde{t} = \epsilon$, then $(\tilde{t}, \tilde{s}) = \tilde{s}$. Similarly, for $\tilde{s} = \epsilon$ we have $(\tilde{t}, \tilde{s}) = \tilde{t}$.

[1]Note that $t_1, \dots, t_n \in \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$ means that the sequence $t_1, \dots, t_n$ belongs to $\mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$. It should *not* be read as $t_1 \in \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V}), \dots, t_n \in \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$.

A desirable property of our sorted term algebra is the existence of a least sort for each term. To guarantee this property, we have identified the following extra condition on the REOS signature:

Preregularity: If $f \in \mathcal{F}_{\mathsf{R}_1 \to \mathsf{s}_1}$ and $\mathsf{R}_0 \preceq \mathsf{R}_1$, then the set $\{\mathsf{s} \mid f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$ and $\mathsf{R}_0 \preceq \mathsf{R}\}$ has a $\preceq$-least element.

This condition is the natural generalization of the notion of preregular order-sorted signature (Goguen and Meseguer, 1992) for REOS signatures.

Lemma 2.8. *If Σ is a preregular signature, then every term sequence $\tilde{t}$ has a $\preceq$-least sort that is unique modulo $\simeq$.*

Proof. Suppose $\tilde{t} \in \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$. We prove the existence of a $\preceq$-least sort of $\tilde{t}$ by induction on length of the proof that $\tilde{t} \in \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$. If $\tilde{t}$ is a variable then $\tilde{t} \in \mathcal{T}_{\mathsf{R}}(\Sigma, \mathcal{V})$ follows from the existence of $\mathsf{Q}_1, \ldots, \mathsf{Q}_n$ such that $\tilde{t} \in \mathcal{V}_{\mathsf{Q}_1} \subseteq \mathcal{T}_{\mathsf{Q}_1}(\Sigma, \mathcal{V}) \subseteq \cdots \subseteq \mathcal{T}_{\mathsf{Q}_n}(\Sigma, \mathcal{V})$, where $\mathsf{Q}_n = \mathsf{R}$ and $\mathsf{Q}_1 \preceq \cdots \preceq \mathsf{Q}_n$. It follows that the set of sorts $M_{\tilde{t}} := \{\mathsf{Q} \mid \tilde{t} \in \mathcal{V}_{\mathsf{Q}}\}$ is a complete set of $\preceq$-minimal sorts of $\tilde{t} \in \mathcal{V}$. Since $\mathsf{Q} \simeq \mathsf{Q}'$ for all $\mathsf{Q} \in M_{\tilde{t}}$ and $\mathsf{Q}' \in M_{\tilde{t}}$, it follows that any $\tilde{t} \in \mathcal{V}$ has a $\preceq$-least sort modulo $\simeq$, which is any Q such that $\tilde{t} \in \mathcal{V}_{\mathsf{Q}}$.

If $\tilde{t} = \epsilon$ then $\tilde{t} \in \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$ follows from $\tilde{t} \in \mathcal{S}_{\mathsf{Q}_1}(\Sigma, \mathcal{V}) \subseteq \ldots \subseteq \mathcal{S}_{\mathsf{Q}_n}(\Sigma, \mathcal{V})$ with $1 = \mathsf{Q}_1 \preceq \ldots \preceq \mathsf{Q}_n = \mathsf{R}$. Thus 1 is the $\preceq$-least sort of ϵ modulo $\simeq$.

Now, suppose $\tilde{t} = f(\tilde{s})$. Because $\tilde{t}$ is sorted, there exist $\mathsf{Q} \in \mathcal{R}$ and $\mathsf{s} \in \mathcal{B}$ such that $f \in \mathcal{F}_{\mathsf{Q} \to \mathsf{s}}$ and $\tilde{s} \in \mathcal{S}_{\mathsf{Q}}(\Sigma, \mathcal{V})$. By induction hypothesis, there exists a $\preceq$-least sort Q' such that $\tilde{s} \in \mathcal{S}_{\mathsf{Q}'}(\Sigma, \mathcal{V})$. Since Σ is preregular, there exists a $\preceq$-least sort s_0 of the set $M_{\mathsf{Q}'} := \{\mathsf{s}' \mid f \in \mathcal{F}_{\mathsf{R}' \to \mathsf{s}'}$ and $\mathsf{Q}' \preceq \mathsf{R}'\}$. Thus s_0 is the $\preceq$-least sort of $\tilde{t}$ modulo $\simeq$. In fact, s_0 can be computed effectively because the set $M_{\mathsf{Q}'}$ is finite due to the finite overloading property.

The only other possibility is $\tilde{t} = (t_1, \ldots, t_m) \in \mathcal{S}_{\mathsf{R}}(\Sigma, \mathcal{V})$, because $t_i \in \mathcal{T}_{\mathsf{R}_i}(\Sigma, \mathcal{V})$ for $1 \leq i \leq m$ and $\mathsf{R}_1. \cdots .\mathsf{R}_m \preceq \mathsf{R}$. By induction hypothesis, there exist $\mathsf{R}'_1 \in \mathcal{R}, \ldots, \mathsf{R}'_m \in \mathcal{R}$ such that R'_i is the $\preceq$-least sort of t'_i and $\mathsf{R}'_i \preceq \mathsf{R}_i$ for $1 \leq i \leq m$. Then $\mathsf{R}'_1. \cdots .\mathsf{R}'_m \preceq \mathsf{R}_1. \cdots .\mathsf{R}_m \preceq \mathsf{R}$, and thus $\mathsf{R}'_1. \cdots .\mathsf{R}'_m$ is the $\preceq$-least sort of $\tilde{t}$ modulo $\simeq$. $\square$

From now on we assume that our signature is preregular, and write either $\mathsf{R} \simeq \mathit{lsort}(\tilde{t})$ or $\tilde{t} : \mathsf{R}$ to express the fact that R is a $\preceq$-least sort modulo $\simeq$ of some term sequence $\tilde{t}$. Also, we write $f : \mathsf{R} \to \mathsf{s}$ instead of $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$. Note that, if $x \in \mathcal{V}_{\mathsf{R}}$ then $\mathit{lsort}(x) \simeq \mathsf{R}$. With this notation, we can formulate the following corollary, which is an immediate consequence of the last paragraph of the proof of Lemma 2.8:

Corollary 2. *If $\tilde{t}$ is a term sequence $(t_1, \dots, t_n)$ with $n \geq 1$, then $\mathit{lsort}(\tilde{t}) \simeq \mathit{lsort}(t_1). \cdots .\mathit{lsort}(t_n)$.*

The set of variables of a term sequence $\tilde{t}$ is denoted by $\mathit{var}(\tilde{t})$. $\tilde{t}$ is ground if $\mathit{var}(\tilde{t}) = \emptyset$. These notions extend to sets of term sequences, etc. We denote the set of ground term sequences (resp. ground terms) over a signature Σ by $\mathcal{S}(\Sigma)$ (resp. $\mathcal{T}(\Sigma)$). For a basic sort s, its semantics $\mathit{sem}(\mathsf{s})$ is the set $\mathcal{T}_{\mathsf{s}}(\Sigma)$ of ground terms of sort s. The semantics of a regular expression sort is given by the set of ground term sequences of the corresponding sort: $\mathit{sem}(\mathsf{1}) = \{\epsilon\}$, $\mathit{sem}(\mathsf{R}_1.\mathsf{R}_2) = \{(\tilde{s}_1, \tilde{s}_2) \mid \tilde{s}_1 \in \mathit{sem}(\mathsf{R}_1), \tilde{s}_2 \in \mathit{sem}(\mathsf{R}_2)\}$, $\mathit{sem}(\mathsf{R}_1{+}\mathsf{R}_2) = \mathit{sem}(\mathsf{R}_1) \cup \mathit{sem}(\mathsf{R}_2)$, $\mathit{sem}(\mathsf{R}^*) = \mathit{sem}(\mathsf{R})^*$. This definition, together with the definition of $\preceq$ and $\mathcal{S}(\Sigma, \mathcal{V})$, implies that if $\mathsf{R} \preceq \mathsf{Q}$, then $\mathit{sem}(\mathsf{R}) \subseteq \mathit{sem}(\mathsf{Q})$.

2.4. Substitutions and Unification Problems

A mapping $\varphi : \mathcal{V} \to \mathcal{S}(\Sigma, \mathcal{V})$ is *well-sorted* if $\mathit{lsort}(\varphi(x)) \preceq \mathit{lsort}(x)$. A *substitution* is a well-sorted mapping from variables to term sequences, which is the identity almost everywhere. This means that the set $\mathit{dom}(\varphi) := \{x \in \mathcal{V} \mid \varphi(x) \neq x\}$, called the *domain of substitution* φ, is a finite set for all substitutions φ. A substitution is a *variable renaming* if it maps the variables from its domain to distinct variables.

Substitutions are denoted by lowercase Greek letters φ, ϑ, ψ, μ, ω, and ε, where ε stands for the identity substitution. A substitution φ can apply to a term t or a term sequence $\tilde{t}$ and result in the *instances* (under φ): $t\varphi$ of t and $\tilde{t}\varphi$ of $\tilde{t}$. They are defined as $x\varphi = \varphi(x)$, $f(\tilde{t})\varphi = f(\tilde{t}\varphi)$, and $(t_1, \dots, t_n)\varphi = (t_1\varphi, \dots, t_n\varphi)$. For instance, if $\varphi = \{x \mapsto (g(a), y), y \mapsto \epsilon, z \mapsto a\}$, then $(x, f(x, z), b, y, z)\varphi = (g(a), y, f(g(a), y, a), b, a)$.

The notion of substitution composition is defined in the standard way. (See, e.g., Baader and Snyder (2001).) We use juxtaposition $\varphi\vartheta$ for composition of φ with ϑ, and write $\tilde{t} \leq \tilde{s}$ to indicate that $\tilde{t}$ *subsumes* $\tilde{s}$, that is, there exists a substitution φ such that $\tilde{t}\varphi = \tilde{s}$. In this case we also say that $\tilde{t}$ is *more general* than $\tilde{s}$. The notation $\varphi \leq_{\mathcal{X}} \vartheta$ is for *subsumption (more generality) with respect to the set of variables* $\mathcal{X}$, that is, when there exists a substitution ψ such that $x\varphi\psi = x\vartheta$ for all $x \in \mathcal{X}$. The notation $\varphi_{\mathcal{X}}$ stands for the *restriction* of φ to the set of variables $\mathcal{X}$. It means that $\varphi|_{\mathcal{X}}$ is a substitution with the property $x\varphi|_{\mathcal{X}} = x\varphi$ for all $x \in \mathcal{X}$.

Lemma 2.9. *$\mathit{lsort}(\tilde{t}\varphi) \preceq \mathit{lsort}(\tilde{t})$ holds for any term sequence $\tilde{t}$ and substitution φ.*

Proof. By induction on the structure of $\tilde{t}$. If $\tilde{t} = \epsilon$ then $\tilde{t}\varphi = \epsilon = \tilde{t}$, thus $\mathit{lsort}(\tilde{t}\varphi) = \mathit{lsort}(\tilde{t})$. Otherwise $\tilde{t} = (t_1, \dots, t_n)$ where $n \geq 1$ and $t_i \in \mathcal{T}(\Sigma, \mathcal{V})$ for $1 \leq i \leq n$. Note that, if $\mathit{lsort}(t_i\varphi) \preceq \mathit{lsort}(t_i)$ for $1 \leq i \leq n$ then $\mathit{lsort}(\tilde{t}\varphi) \simeq (\mathit{lsort}(t_1\varphi). \cdots .\mathit{lsort}(t_n\varphi)) \preceq (\mathit{lsort}(t_1). \cdots .\mathit{lsort}(t_n)) \simeq \mathit{lsort}(\tilde{t})$.

We still have to prove that $\mathit{lsort}(t\varphi) \preceq \mathit{lsort}(t)$ for any term t and substitution φ. If t is a variable, then the lemma follows from the definition of substitution. If $t = f(\tilde{t})$ with $\mathit{lsort}(t) \simeq \mathsf{s}$ then there exists $f : \mathsf{S} \to \mathsf{s}$ with $\mathit{lsort}(\tilde{t}) \preceq \mathsf{S}$. Also, $\mathit{lsort}(\tilde{t}\varphi) \preceq \mathit{lsort}(\tilde{t})$ by the induction hypothesis. Let $M := \{\mathsf{r} \mid f \in \mathcal{F}_{\mathsf{R}\to\mathsf{r}}$ and $\mathit{lsort}(\tilde{t}\varphi) \preceq \mathsf{R}\}$. Then $\mathsf{s} \in M$ because $\mathit{lsort}(\tilde{t}\varphi) \preceq \mathit{lsort}(\tilde{t}) \preceq \mathsf{S}$ and $f \in \mathcal{F}_{\mathsf{S}\to\mathsf{s}}$. Σ is preregular, therefore M has a $\preceq$-least element s_0. This means $\mathsf{s}_0 \preceq \mathsf{s}$ and the existence of $\mathsf{S}_0 \in \mathcal{R}$ with $f : \mathsf{S}_0 \to \mathsf{s}_0$ and $\mathit{lsort}(\tilde{t}\varphi) \preceq \mathsf{S}_0$. Thus $t\varphi = f(\tilde{t}\varphi) \in \mathcal{T}_{\mathsf{s}_0}(\Sigma, \mathcal{V})$. Therefore $\mathit{lsort}(t\varphi) \preceq \mathsf{s}_0 \preceq \mathsf{s} \simeq \mathit{lsort}(t)$. □

An *equation* is a pair of term sequences, written as $\tilde{s} \doteq \tilde{t}$.

Definition 2.10. A *regular expression order sorted unification problem* 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 φ is a *unifier* of Γ if $\tilde{s}_i\varphi = \tilde{t}_i\varphi$ for all $1 \leq i \leq n$. 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 $\varphi \in U$ such that $\varphi \leq_{var(\Gamma)} \vartheta$.

Minimality: If there are $\varphi_1 \in U$ and $\varphi_2 \in U$ such that $\varphi_1 \leq_{var(\Gamma)} \varphi_2$, then $\varphi_1 = \varphi_2$.

3. Relating REOS Signatures and Unranked Tree Automata

Regular expression ordered sorts over finite signatures are related to finite automata for unranked trees in the same way as ordered sorts are related to finite automata for ranked trees. In order to understand the correspondence, we recall the notion of *finite bottom-up unranked tree automaton* (a.k.a. hedge automaton, see, e.g., (Comon et al., 2007; Jacquemard and Rusinowitch, 2008)). This is a tuple $\mathcal{A} = (Q, F, Q_f, \delta)$ where

- Q is a finite set of states (nonterminals),
- F is a finite unranked alphabet (terminals),

- δ is a finite set of rules of the form $q_1 \to q_2$ or $f(R) \to q$ where $f \in F$, R is a regular expression over Q and q_1, q_2, and q are from Q, and
- Q_f (final states) is a subset of Q.

The *move relation* of $\mathcal{A}$ over ground trees $\mathcal{T}(F \cup Q)$ is defined as follows: For all $t_1 \in \mathcal{T}(F \cup Q)$ and $t_2 \in \mathcal{T}(F \cup Q)$, the relation $t_1 \longrightarrow_{\mathcal{A}} t_2$ holds if

- there exists a context $C[]$ and a rule $f(R) \to q \in \delta$ such that $t_1 = C[f(q_1, \dots, q_n)]$, the word $q_1 \cdots q_n \in [\![R]\!]$ and $t_2 = C[q]$, or
- there exists a context $C[]$ and a rule $q_1 \to q_2 \in \delta$ such that $t_1 = C[q_1]$ and $t_2 = C[q_2]$.

A tree $t \in \mathcal{T}(F)$ is *recognized* by $\mathcal{A}$ at state q if $t \longrightarrow^*_{\mathcal{A}} q$ holds. The *language* $\mathcal{L}(\mathcal{A})$ *accepted by* $\mathcal{A}$ is defined as the set of ground unranked trees $\mathcal{L}(\mathcal{A}) = \{t \in \mathcal{T}(F) \mid \text{there exists } q \in Q_f \text{ such that } t \longrightarrow^*_{\mathcal{A}} q\}$.

The finite bottom-up unranked tree automaton that corresponds to a REOS signature $\Sigma = (\mathcal{B}, \preceq, \mathcal{F})$ with $\mathcal{F}$ finite is $\mathcal{A}_\Sigma := (\mathcal{B}, \mathcal{F}, \mathcal{B}, \delta)$ where the roles of states and final states are played by $\mathcal{B}$, the role of terminals is played by $\mathcal{F}$, and δ contains rules of two kinds:

1. For each $\mathsf{r} \preceq \mathsf{s}$, the ϵ-transition rule $\mathsf{r} \to \mathsf{s}$.
2. For each $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$, the transition rule $f(\mathsf{R}) \to \mathsf{s}$.

It is easy to see that $t \in \mathcal{T}_{\mathsf{s}}(\Sigma)$ iff $t \longrightarrow^*_{\mathcal{A}_\Sigma} \mathsf{s}$.

Conversely, if $\mathcal{A} = (Q, F, Q_f, \delta)$, then we can define the REOS signature $\Sigma_{\mathcal{A}} := (Q, \preceq, \mathcal{F})$ where

- $q_1 \preceq q_2$ iff $q_1 \to q_2 \in \delta$, and
- $\mathcal{F}_{\mathsf{R} \to q} := \{f \in F \mid f(\mathsf{R}) \to q \in \delta\}$,

and note that $t \longrightarrow^*_{\mathcal{A}} q$ iff $t \in \mathcal{T}_q(\Sigma_{\mathcal{A}})$.

4. Sort-Related Algorithms

In this section we single out some useful algorithms that operate on sorts. These algorithms will be useful later.

4.1. Computing Least Sorts

We can extract from the constructive proof of Lemma 2.8 the following set of inference rules for the judgment $\tilde{t} : \mathsf{R}$ which expresses the fact that the least sort of the term sequence $\tilde{t}$ is R.

$$\frac{}{\epsilon : 1} \qquad \frac{x \in \mathcal{V}_{\mathsf{R}}}{x : \mathsf{R}} \qquad \frac{t_1 : \mathsf{R}_1 \quad \ldots \quad t_m : \mathsf{R}_m}{(t_1, \ldots, t_m) : \mathsf{R}_1 . \cdots . \mathsf{R}_m}$$

$$\frac{f : \mathsf{Q} \to \mathsf{q} \quad \tilde{t} : \mathsf{R} \quad \mathsf{R} \preceq \mathsf{Q} \quad \mathsf{s} = \mathit{least\_elem}_{\preceq}\{\mathsf{s}' \mid f \in \mathcal{F}_{\mathsf{R}' \to \mathsf{s}'} \text{ and } \mathsf{R} \preceq \mathsf{R}'\}}{f(\tilde{t}) : \mathsf{s}}$$

Hence, computation of least sorts involves deciding $\preceq$ between two regular expressions. As we have already mentioned in Sect. 2, this problem is PSPACE-complete, but Antimirov's approach in some cases provides derivations of polynomial size. In Sect. 8.2 below we will show that the computation of least sorts needed in matching problems can be done in polynomial time.

4.2. Computing Greatest Lower Bounds

Assume that $\mathsf{R}_1 \in \mathcal{R}, \ldots, \mathsf{R}_n \in \mathcal{R}$. If $\bigcap_{i=1}^{n} [\![\overline{\mathsf{R}_i}]\!] = \emptyset$ then $\mathsf{R}_1, \ldots, \mathsf{R}_n$ have no lower bound with respect to $\preceq$, because if Q were such a lower bound then, by Lemma 2.1, $[\![\overline{\mathsf{Q}}]\!] \subseteq [\![\overline{\mathsf{R}_i}]\!]$ for all $i \in \{1, \ldots, n\}$. This implies $\emptyset \neq [\![\overline{\mathsf{Q}}]\!] \subseteq \bigcap_{i=1}^{n} [\![\overline{\mathsf{R}_i}]\!] = \emptyset$, which is a contradiction. From now on, we write $\mathrm{glb}(\{\mathsf{R}_1, \ldots, \mathsf{R}_n\}) = \bot$ in the situation when $\mathsf{R}_1 \in \mathcal{R}, \ldots, \mathsf{R}_n \in \mathcal{R}$ and $\bigcap_{i=1}^{n} [\![\overline{\mathsf{R}_i}]\!] = \emptyset$ (that is, when $\mathsf{R}_1, \ldots, \mathsf{R}_n$ have no lower bound).

Otherwise, we can use standard techniques from the theory of regular languages to compute $\mathsf{Q} \in \mathcal{R}$ such that $[\![\mathsf{Q}]\!] = \bigcap_{i=1}^{n} [\![\overline{\mathsf{R}_i}]\!]$, and note that such a Q is a greatest lower bound of $\mathsf{R}_1, \ldots, \mathsf{R}_n$. Thus, in this case we can write $\mathrm{glb}(\{\mathsf{R}_1, \ldots, \mathsf{R}_n\}) = \mathsf{Q}$, where Q is a regular expression sort computed to fulfill the condition $[\![\mathsf{Q}]\!] = \bigcap_{i=1}^{n} [\![\overline{\mathsf{R}_i}]\!]$.

Gelade and Neven (2012) showed that computing the intersection of two regular expressions (and, hence, computing their glb) takes time exponential in the size of the input. They also proved that in constructing a regular expression for the intersection of two expressions, an exponential blow-up can not be avoided.

4.3. Computing Weakening Substitutions

A *weakening substitution* of a term sequence $\tilde{t}$ towards a sort $\mathsf{Q} \in \mathcal{R}$ is a variable renaming θ such that $\tilde{t}\theta \in \mathcal{S}_{\mathsf{Q}}(\Sigma, \mathcal{V})$. Alternatively, we call θ a *solution* of the *weakening pair* $\tilde{t} \rightsquigarrow \mathsf{Q}$. We generalize this notion to finite sets of weakening pairs, which we call *weakening problems*, and consider θ a solution of such a set W iff θ is a solution for every weakening pair $\tilde{t} \rightsquigarrow \mathsf{Q} \in W$.

Note that weakening substitutions may not exist. Such a situation happens, for instance, for weakening pairs $\tilde{t} \rightsquigarrow \mathsf{Q}$ with $\tilde{t}$ a ground term sequence and $\mathit{lsort}(\tilde{t}) \not\preceq \mathsf{Q}$.

The notion of weakening substitution has a very simple intuitive meaning: Given a pair $\tilde{t} \rightsquigarrow \mathsf{Q}$, we wish to relax the sorts of the variables in $\tilde{t}$ by replacing them with variables of smaller sorts, such that the renamed version of $\tilde{t}$ is in $\mathcal{S}_{\mathsf{Q}}(\Sigma, \mathcal{V})$. 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 map x to $f(y)$ directly, because $\mathit{lsort}(f(y)) \simeq \mathsf{s}_2 \not\preceq \mathsf{s} \simeq \mathit{lsort}(x)$. However, if we weaken the least sort of $f(y)$ to s_1, then the mapping becomes possible. To weaken the least sort of $f(y)$, we take its instance under substitution $\{y \mapsto z\}$, where $z \in \mathcal{V}_{\mathsf{R}_1}$, which gives $\mathit{lsort}(f(z)) \simeq \mathsf{s}_1$. Hence, the substitution $\{y \mapsto z, x \mapsto f(z)\}$ is a unifier of x and $f(y)$, leading to the common instance $f(z)$.

Now we describe an algorithm that computes weakening substitutions for weakening problems. Our weakening algorithm is called $\mathfrak{W}$, and works by applying exhaustively the following rules to pairs of the form $W; \varphi$ where W is a weakening problem and φ is a substitution. In the rules here and elsewhere $\uplus$ stands for disjoint union:

E-w: **Elimination in Weakening**

$\{\tilde{s} \rightsquigarrow \mathsf{Q}\} \uplus W; \varphi \Longrightarrow W; \varphi \qquad$ if $\mathit{lsort}(\tilde{s}) \preceq \mathsf{Q}$.

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

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

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 $\mathsf{s}.\mathsf{S} \in \max(\hat{\mathit{lf}}(\mathsf{Q}))$.

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

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

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**

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

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

$\mathsf{V\text{-}w}$: **Variable Weakening**

$$\{x \rightsquigarrow \mathsf{Q}\} \uplus W; \varphi \Longrightarrow W\varphi; \varphi\{x \mapsto w\}$$

where $lsort(x) \not\preceq \mathsf{Q}$ and $\mathrm{glb}(\{lsort(x), \mathsf{Q}\}) \neq \bot$ and w is a fresh variable from $\mathcal{V}_{\mathrm{glb}(\{lsort(x),\mathsf{Q}\})}$.

If none of the rules are applicable to $W; \varphi$, then it is transformed into $\bot$, indicating failure. By exhaustive search, transforming each $W; \varphi$ 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 $W; \varepsilon$ is denoted by $weak(W)$. It is easy to see that the elements of $weak(W)$ 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 4.1. *$\mathfrak{W}$ is terminating.*

Proof. The measure of a weakening pair $\tilde{t} \rightsquigarrow \mathsf{Q}$ is 1 + the size of $\tilde{t}$, and the measure of a weakening problem W is the multiset of the measures of its constituent weakening pairs. The multiset extension of the standard ordering on nonnegative integers is well-founded. The rules in $\mathfrak{W}$ strictly decrease the measure for the sets on which they operate and, hence, $\mathfrak{W}$ is terminating. $\square$

Theorem 4.2 (Soundness of the Weakening Algorithm)**.** *If W is a weakening problem then each $\omega \in weak(W)$ is a weakening substitution of W.*

Proof. It is enough to show that if a rule in $\mathfrak{W}$ transforms $W_1; \varphi$ into $W_2; \varphi\vartheta$ and ψ is a weakening substitution for W_2, then $\vartheta\psi$ is a weakening substitution for W_1. For E-w, it is trivial. For D1-w it follows from two facts: First, if $\mathsf{s.S} \in \max(\hat{lf}(\mathsf{Q}))$ then $\mathsf{s.S} \preceq \mathsf{Q}$, and second, $\preceq$-monotonicity of concatenation: If $\mathsf{R}_1 \preceq \mathsf{Q}_1$ and $\mathsf{R}_2 \preceq \mathsf{Q}_2$ then $\mathsf{R}_1.\mathsf{R}_2 \preceq \mathsf{Q}_1.\mathsf{Q}_2$. For D2-w it follows from $\preceq$-monotonicity of concatenation and from the definition of split. For AS-w, it is implied by the selection of R and r, whereas for V-w it is implied by the definition of glb and Lemma 2.9. $\square$

Theorem 4.3 (Completeness of the Weakening Algorithm)**.** *Let W be a weakening problem. For every weakening substitution ω of W there exists $\omega' \in \mathit{weak}(W)$ such that $\omega' \leq_{\mathit{var}(W)} \omega$.*

Proof. The proof is by induction on the measure of W defined in the proof of Theorem 4.1. The lemma holds trivially when $W = \emptyset$. If W contains a weakening pair $\tilde{s} \rightsquigarrow \mathsf{Q}$ such that $\mathit{lsort}(\tilde{s}) \preceq \mathsf{Q}$, then W is of the form $\{\tilde{s} \rightsquigarrow \mathsf{Q}\} \uplus W'$ and W' has smaller measure than W. Since ω is a weakening substitution for W' as well, by induction hypothesis, there exists an $\mathfrak{W}$-derivation $W'; \varepsilon \Longrightarrow^* \emptyset; \omega'$ such that $\omega' \leq_{\mathit{var}(W')} \omega$, and we can assume without loss of generality that $\omega' \leq_{\mathit{var}(W)} \omega$. Since we can prepend the $\mathsf{E\text{-}w}$ step $\{\tilde{s} \rightsquigarrow \mathsf{Q}\} \uplus W'; \varepsilon \Longrightarrow W'; \varepsilon$ to the former $\mathfrak{W}$-derivation, we conclude that $\omega' \in \mathit{weak}(W)$ and $\omega' \leq_{\mathit{var}(W)} \omega$.

The remaining case to be considered is when $\mathit{lsort}(\tilde{r}) \not\preceq \mathsf{Q}$ for all weakening pairs $(\tilde{r} \rightsquigarrow \mathsf{Q}) \in W$. Assume $(\tilde{r} \rightsquigarrow \mathsf{Q}) \in W$ is such a weakening pair. Let $W = \{\tilde{r} \rightsquigarrow \mathsf{Q}\} \uplus W'$. The proof proceeds by case distinction on the syntactic structure of $\tilde{r}$.

- $\tilde{r} = (f(\tilde{t}), \tilde{s})$, $\tilde{s} \neq \epsilon$. Since $\mathit{lsort}(\tilde{r}\omega) \preceq \mathsf{Q}$, there exists $\mathsf{s.S} \in \max(\hat{\mathit{lf}}(\mathsf{Q}))$ such that $\mathit{lsort}(f(\tilde{t})\omega) \preceq \mathsf{s}$ and $\mathit{lsort}(\tilde{s}\omega) \preceq \mathsf{S}$. In this case we can perform the $\mathsf{D1\text{-}w}$ step $\pi = (W; \varepsilon \Longrightarrow W''; \varepsilon)$ where $W'' = \{f(\tilde{t}) \rightsquigarrow \mathsf{s}, \tilde{s} \rightsquigarrow \mathsf{S}\} \uplus W'$. Since W'' has the smaller measure than W, and since ω is a weakening substitution for W'', we can apply the induction hypothesis to infer the existence of a $\mathfrak{W}$-derivation $\Pi = (W''; \varepsilon \Longrightarrow^* \emptyset; \omega')$ such that $\omega' \leq_{\mathit{var}(W'')} \omega$. Note that $\mathit{var}(W'') = \mathit{var}(W)$. By prepending the $\mathsf{D2\text{-}w}$ step π to the $\mathfrak{W}$-derivation Π we conclude that $\omega' \in \mathit{weak}(W)$.

- $\tilde{r} = (x, \tilde{s})$, $\tilde{s} \neq \epsilon$. Since $\mathit{lsort}(\tilde{r}\omega) \preceq \mathsf{Q}$, by Lemma 2.6 there exists a split $(\mathsf{Q}_1, \mathsf{Q}_2)$ of Q such that $\mathit{lsort}(x\omega) \preceq \mathsf{Q}_1$ and $\mathit{lsort}(\tilde{s}\omega) \preceq \mathsf{Q}_2$. In this case we can perform the $\mathsf{D2\text{-}w}$ step $\pi = (W; \varepsilon \Longrightarrow W''; \varepsilon)$, where $W'' = \{x \rightsquigarrow \mathsf{Q}_1, \tilde{s} \rightsquigarrow \mathsf{Q}_2\} \uplus W'$. Since W'' has the smaller measure than W, and ω is a weakening substitution for W'', we can use the arguments similar to the previous case to conclude that $\omega' \in \mathit{weak}(W)$.

- $\tilde{r} = f(\tilde{t})$. Since $\mathit{lsort}(\tilde{r}\omega) \preceq \mathsf{Q}$, there exist R and s such that $\mathsf{R.r}$ is a maximal sort with $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{r}}$, $\mathsf{r} \preceq \mathsf{Q}$, and $\mathit{lsort}(\tilde{t}\omega) \preceq \mathsf{R}$. In this case we can perform the $\mathsf{AS\text{-}w}$ step $\pi = (W; \varepsilon \Longrightarrow W''; \varepsilon)$, where $W'' = \{\tilde{t} \rightsquigarrow \mathsf{R}\} \uplus W'$. Since W'' has the smaller measure than W, and ω is a weakening substitution for W'', we can use the arguments similar to the cases above to conclude that $\omega' \in \mathit{weak}(W)$.

- $\tilde{r} = x$. Since $lsort(x\omega) \preceq \mathsf{Q}$, there exists $\mathsf{R}' := \mathrm{glb}(lsort(x), \mathsf{Q}) \in \mathcal{R}$ and $lsort(x\omega) \preceq \mathsf{R}'$. In this case we can perform the V-w step $\pi = (W; \varepsilon \Longrightarrow W'\varphi; \varphi)$, where $\varphi = \{x \mapsto w\}$, w a fresh variable from $\mathcal{V}_{\mathsf{R}'}$. Then $\omega \cup \{w \mapsto x\omega\}$ is a weakening substitution of $W'\varphi$. Since $W'\varphi$ has the smaller measure that W, we can apply the induction hypothesis to infer the existence of a $\mathfrak{W}$-derivation $\Pi = (W'\varphi; \varepsilon \Longrightarrow^* \emptyset; \omega'')$ such that $\omega'' \leq_{var(W'\varphi)} \omega \cup \{w \mapsto x\omega\}$. Let $\omega' = \varphi\omega''$. Then we have $\omega' \leq_{var(W'\varphi)\cup\{x\}} \omega \cup \{w \mapsto x\omega\}$ and $\omega' \leq_{var(W'\varphi)\cup\{x\}\setminus\{w\}} \omega$. But $var(W'\varphi) \cup \{x\} \setminus \{w\} = var(W)$. From Π, we can construct a $\mathfrak{W}$-derivation $\Pi' = (W'\varphi; \varphi \Longrightarrow^* \emptyset; \omega')$. Prepending the step π to Π' we get that $\omega' \in weak(W)$ and $\omega' \leq_{var(W)} \omega$.

□

Example 4.4. 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 AS-w, leading to $weak(W) = \{\{x \mapsto z, w \mapsto z\}\}$.

Example 4.5. 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 $weak(W) = \{\{x \mapsto u_1, y \mapsto v_1\}, \{x \mapsto u_2, y \mapsto v_2\}, \{x \mapsto u_3, y \mapsto v_3\}\}$ where

$$\begin{array}{lll} u_1 : \mathsf{s}^*, & u_2 : \mathsf{s}^*.\mathsf{r}^* & u_3 : \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*, \\ v_1 : \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*, & v_2 : \mathsf{r}.\mathsf{r}^*, & v_3 : \mathsf{r}^*. \end{array}$$

Example 4.6. 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 $weak(W) = \{\{x \mapsto w\}\}$ where $w : (\mathsf{s}_1{+}\mathsf{s}_2)^*$.

5. Unification Type

The sequence unification problems (SEQU problems in short) have been studied in (Kutsia, 2007). They can be seen as REOSU problems built over one basic sort s, all function symbols having the sort $\mathsf{s}^* \to \mathsf{s}$, and each variable having either the sort s (individual variable) or s^* (sequence variable). We can also ignore the sort information, keeping just the explicit distinction between individual and sequence variables.

Unification problems are characterized by the existence and cardinality of their minimal complete sets of unifiers. It is called the type of unification, whose definition we give here following Baader and Snyder (2001). For simplicity, the word "theory" in the definition means REOSU or SEQU, i.e., syntactic theories over $\mathcal{F}$ or its unsorted version. Similarly, the phrase "unification problem" refers to a REOSU problem over $\mathcal{F}$ or a SEQU problem over the unsorted version of $\mathcal{F}$.

Definition 5.1. Let Γ be a unification problem over $\mathcal{F}$. It has type *unitary* (*finitary, infinitary*) iff it has a minimal complete set of unifiers of cardinality 1 (finite cardinality, infinite cardinality). If Γ has no minimal complete set of unifiers, then it has type *zero*. We abbreviate type unitary with 1, type finitary by ω, type infinitary by ∞, and type zero by 0, and order them as $1 < \omega < \infty < 0$. Then the unification type of a theory is the maximal type of a unification problem in the theory.

The SEQU problems in this section will be assumed to contain only sequence variables and no individual variables. Let Γ_{re} be a REOSU problem and Γ_{seq} be the corresponding SEQU problem. It means, Γ_{seq} is obtained from Γ_{re} by forgetting the sort information and replacing every variable with a sequence variable. Each unifier of Γ_{re} is, obviously, a unifier of Γ_{seq}. On the other hand, not all unifiers of Γ_{seq} solve Γ_{re}: They might not preserve sorts.

In (Kutsia, 2007), it was shown that SEQU is infinitary. It is obvious that REOSU is at least infinitary. We would like to show that it is indeed infinitary and not of type zero.

Let S_{seq} be a minimal complete set of unifiers of Γ_{seq} and ϑ be a unifier of Γ_{re}. Although ϑ solves Γ_{seq}, it is not necessary that $\vartheta \in S_{\text{seq}}$, because it might not be a minimal unifier for Γ_{seq}. However, since S_{seq} is complete, there should be a substitution $\varphi \in S_{\text{seq}}$ such that $\varphi \leq_{var(\Gamma_{\text{seq}})} \vartheta$. Hence, any unifier of Γ_{re} is an instance of an element of S_{seq}.

For each substitution $\varphi = \{x_1 \mapsto \tilde{t}_1, \ldots, x_n \mapsto \tilde{t}_n\}$, we define the set of weakening substitutions for φ as $\Omega(\varphi) = \mathit{weak}(\{\tilde{t}_1 \rightsquigarrow \mathit{lsort}(x_1), \ldots, \tilde{t}_n \rightsquigarrow \mathit{lsort}(x_n)\})$.

Let $S(\varphi)$ be the set of substitutions $S(\varphi) = \{\varphi\omega_\varphi \mid \omega_\varphi \in \Omega(\varphi)\}$. This set is finite, because $\Omega(\varphi)$ is finite. Let $S^{\min}_{\mathcal{X}}(\varphi)$ denote the set obtained from $S(\varphi)$ by minimizing it with respect to the subsumption ordering $\leq_{\mathcal{X}}$ on a set of variables $\mathcal{X}$. Without loss of generality, we can assume $dom(\vartheta) \subseteq \mathcal{X}$ for each $\vartheta \in S^{\min}_{\mathcal{X}}(\varphi)$.

Let V be the set of variables $V = var(\Gamma_{\text{re}}) = var(\Gamma_{\text{seq}})$. By S_{re} we denote a set of substitutions defined as $S_{\text{re}} = \cup_{\varphi \in S_{\text{seq}}} S^{\min}_V(\varphi)$. Then we have the following lemma:

Lemma 5.2. *S_{re} is a complete set of unifiers for Γ_{re}.*

Proof. Every element of S_{re} is a unifier of Γ_{re}. This easily follows from the fact that these substitutions are well-sorted instances of elements of S_{seq}. To prove completeness, we take a unifier ϑ of Γ_{re} and show that there exists $\psi \in S_{\text{re}}$ such that $\psi \leq_V \vartheta$.

Since S_{seq} is a complete set of unifiers of Γ_{seq} and ϑ is a unifier of Γ_{seq}, there exists $\varphi \in S_{\text{seq}}$ such that for each $x \in V$, $x\varphi \leq x\vartheta$. ϑ is well-sorted. Therefore, $lsort(x) \preceq lsort(x\vartheta)$ for all $x \in V$. If $lsort(x) \preceq lsort(x\varphi)$ holds for all $x \in V$, then, by the construction of Γ_{re}, we have $\varphi \in \Gamma_{\text{re}}$ and we can take $\psi = \varphi$. Otherwise, let x be a variable for which $lsort(x) \not\preceq lsort(\varphi)$. Since $x\varphi \leq x\vartheta$ and $lsort(x) \preceq lsort(x\vartheta)$, we can weaken x towards $lsort(\varphi)$ with a weakening substitution ω such that $lsort(x) \preceq lsort(\varphi\omega)$ and $x\varphi\omega \leq x\vartheta$. But then $\varphi\omega \in \Gamma_{\text{re}}$ by the construction of Γ_{re}, and we can take $\psi = \varphi\omega$. Hence, for any unifier ϑ of Γ_{re} there is a substitution $\psi \in \Gamma_{\text{re}}$ such that $\psi \leq_V \vartheta$. Therefore, S_{re} is a complete set of unifiers for Γ_{re}. □

To prove that REOSU is not of type zero, we should show that any unification problem has a minimal complete set of unifiers.

Lemma 5.3. *The set S_{re} is minimal.*

Proof. Assume by contradiction that S_{re} is not minimal. Then it contains two elements φ' and ϑ' such that $\varphi' \leq_V \vartheta'$, i.e., there exists $\psi' \neq \varepsilon$ such that $\varphi'\psi' =_V \vartheta'$. We consider the following four possible cases:

1. $\varphi' \in S_{\text{seq}}$ and $\vartheta' \notin S_{\text{seq}}$. Then $\varphi'\psi' = \varphi\omega_\varphi\psi' =_V \vartheta'$ for $\varphi \in S_{\text{seq}}$ and $\omega_\varphi \in \Omega(\varphi)$. If $\varphi \neq \vartheta'$, then the previous equality contradicts minimality of S_{seq}. If $\varphi = \vartheta'$, then Γ_{re} contains two substitutions φ' and ϑ', comparable with respect to $\leq_V$, both obtained by weakening the same substitution $\varphi \in \Gamma_{\text{seq}}$. However, this contradicts the way how Γ_{re} was constructed: $S^{\min}_V(\varphi)$ is supposed to be minimal.

2. $\varphi' \notin S_{\mathrm{seq}}$ and $\vartheta' \in S_{\mathrm{seq}}$. Then $\varphi'\psi' =_V \vartheta' = \vartheta\omega_\vartheta$ where $\vartheta \in S_{\mathrm{seq}}$ and $\omega_\vartheta \in \Omega(\vartheta)$. Since ω_ϑ is a variable renaming, $\varphi'\psi'\omega_\vartheta^{-1} =_V \vartheta$. If $\varphi' \neq \vartheta$, the latter equality contradicts minimality of S_{seq}. If $\varphi' = \vartheta$, then Γ_{re} contains two substitutions φ' and ϑ', comparable with respect to $\leq_V$, both obtained by weakening the same substitution $\vartheta \in \Gamma_{\mathrm{seq}}$. However, this contradicts the way how Γ_{re} was constructed: $S_V^{\min}(\vartheta)$ is supposed to be minimal.
3. $\varphi' \notin S_{\mathrm{seq}}$ and $\vartheta' \notin S_{\mathrm{seq}}$. Then $\varphi\omega_\varphi\psi' = \varphi'\psi' =_V \vartheta' = \vartheta\omega_\vartheta$ for $\varphi \in S_{\mathrm{seq}}$ and $\vartheta \in S_{\mathrm{seq}}$. Since ω_ϑ is a variable renaming, we have $\varphi\omega_\varphi\psi'\omega_\vartheta^{-1} =_V \vartheta$. Then we reason in the same way as above to obtain a contradiction.
4. $\varphi' \in S_{\mathrm{seq}}$ and $\vartheta' \in S_{\mathrm{seq}}$. It immediately contradicts minimality of S_{seq}.

Hence, S_{re} is minimal. □

Lemma 5.2 and Lemma 5.3 imply that Γ_{re} has a minimal complete set of unifiers. Hence, REOSU is not of type zero and the following theorem holds:

Theorem 5.4. *REOSU has the infinitary unification type.*

6. Decidability of REOSU

To show decidability, we define a translation from REOSU problems into word equations with regular constraints. The idea is similar to the one of Levy and Villaret (2001), used to translate context equations into traversal equations, or of Kutsia et al. (2007, 2010), used to translate left-hole context equations into word equations with regular constraints.

In the proof we need the notion of depth for various syntactic constructs. The *depth* of a term and a term sequence is defined in the standard way: $\mathit{depth}(x) = 1$, $\mathit{depth}(f(\tilde{t})) = 1 + \mathit{depth}(\tilde{t})$, $\mathit{depth}(\epsilon) = 0$, $\mathit{depth}(t_1, \dots, t_n) = \max\{\mathit{depth}(t_i) \mid 1 \leq i \leq n\}$, $n > 0$. The *depth* of an equation $\tilde{s} \doteq \tilde{t}$ is the maximum between $\mathit{depth}(\tilde{s})$ and $\mathit{depth}(\tilde{t})$. The *depth* of a substitution is defined as $\mathit{depth}(\varphi) = \max\{\mathit{depth}(x\varphi) \mid x \in \mathcal{V}\}$. The *depth* of a REOSU problem Γ is the maximum depth of the equations it contains.

For each basic sort s we assume to have at least one function symbol $a : \mathsf{1} \to \mathsf{s}$.[2] We call them constants of the sort s (slightly abusing the terminology) and proceed as follows:

[2]In fact, it is enough to require to have at least one function symbol $a : \mathsf{R} \to \mathsf{s}$ for each $\preceq$-minimal basic sort s such that $\mathsf{1} \preceq \mathsf{R}$.

- First, we show that each solvable REOSU problem Γ has a unifier φ with the property $\mathit{depth}(\varphi) \leq \mathit{size}(\Gamma)$, where $\mathit{size}(\Gamma)$ is the number of function symbols and variables 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 WRCU is decidable, we get decidability of REOSU.

We now elaborate on these items. First, observe the following: Given a REOSU problem Γ, let φ be a substitution with $\mathit{dom}(\varphi) \subseteq \mathit{var}(\Gamma)$ such that $\varphi(x) = \epsilon$ for all $x \in \mathit{dom}(\varphi)$. There can be only finitely many such φ's for a given Γ. Let ϑ be a substitution such that $\vartheta(x) \neq \epsilon$ for all $x \in \mathit{dom}(\vartheta)$. (We call such substitutions *nonerasing* substitutions.) Then $\varphi\vartheta$ is a unifier of Γ iff ϑ is a unifier of $\Gamma\varphi$. Hence, we can assume without loss of generality that we are looking for nonerasing unifiers only, because if we can find such a unifier ϑ of $\Gamma\varphi$, we can construct a unifier $\varphi\vartheta$ of Γ.

Unifier depth bound. We show that the depth of unifiers is bounded by the size of unification problems. For this, we need to relate terms and their instances to tree representations. We adopt a representation of terms by labeled trees (adapting the representation described in (Krajiček and Pudlák, 1988) to SEQU), where the labels are terms that occur in the unification problem. Here, a *labeled tree* is a tree whose nodes are labeled with terms such that the following conditions are satisfied:

- If a node N is labeled by a term $f(s_1, \ldots, s_n)$, then either
  - $n = 0$ and N is a leaf node, or
  - $n > 0$ and N has n successor nodes $N_1, \ldots, N_n$ such that each N_i is labeled by s_i, $1 \leq i \leq n$.
- A node N labeled with a variable x is called a *substitution node.* If N is a leaf node, it is a substitution node for ε. Otherwise, it is a substitution node for $\{x \mapsto (s_1, \ldots, s_n)\}$ where $s_1, \ldots, s_n$ are the labels of the children $N_1, \ldots, N_n$ of N, enumerated from left to right. We require that all nodes with same label x must be for the same substitution.

Trees labeled in this way have three important properties:

L1: Nodes with identical labels are roots of identical labeled subtrees. An immediate consequence of this fact is that the labels of nodes along a branch from root to a leaf are distinct terms.

L2: There exists an enumeration $\vartheta_1, \dots, \vartheta_m$ of the substitutions of the substitution nodes which is consistent with a top-down traversal of the nodes of the tree. This means that, whenever node M is above N, the substitution of M is enumerated before that of N.

L3: The term $t\vartheta_1 \dots \vartheta_m$ is the same for any consistent enumeration $\vartheta_1, \dots,$ ϑ_m of the substitutions in a labeled tree $\overline{T}$ with root label t. For this reason, we say that $\overline{T}$ *represents* the term $t\vartheta_1 \dots \vartheta_m$.

A corollary of these properties is that, if $\overline{T}$ represents a term t, then $\mathit{depth}(t)$ coincides with the maximum number of edges along a branch in $\overline{T}$, from which we remove each edge connecting a substitution node to its child.

We will need to consider the unifier depth for unsorted unifiers, because, as we saw in Section 5, a sorted unifier can be obtained from an unsorted one by a weakening substitution. The latter does not affect the depth of terms it applies to. Hence, for a REOSU problem Γ, it is enough to consider its corresponding SEQU problem Γ_{seq}, as we did in Section 5, and reason about the depth if its depth-minimal non-erasing unifiers.

Suppose $\Gamma = \{\tilde{s}_1 \doteq \tilde{t}_1, \dots, \tilde{s}_n \doteq \tilde{t}_n\}$ and let $V = \mathit{var}(\Gamma)$. A corresponding SEQU problem is $\{\mathsf{f}(\tilde{s}_1) \doteq \mathsf{f}(\tilde{t}_1), \dots, \mathsf{f}(\tilde{s}_n) \doteq \mathsf{f}(\tilde{t}_n)\}$ over the signature $\mathcal{F}_u$ where $\mathcal{F}_u$ is the unsorted version of the signature $\mathcal{F}$ of Γ, and $\mathsf{f} \in \mathcal{F}_u$ appears only as the root symbol of the equation sides. We denote this problem by Γ_{seq}. We recall the fact that the system of inference rules $\mathfrak{I}$ described in (Kutsia, 2007) can be used to compute a complete set of depth-minimal non-erasing unifiers of Γ_{seq}. Thus we can assume that $\varphi = \varphi_1 \dots \varphi_m$ is such a unifier of Γ_{seq}, where the substitutions $\varphi_1, \dots, \varphi_m$ are produced by an $\mathfrak{I}$-derivation

$$\langle \Gamma_0;\ \varphi_0 \rangle \Longrightarrow \langle \Gamma_1;\ \varphi_1 \rangle \Longrightarrow \cdots \Longrightarrow \langle \Gamma_m;\ \varphi_1\varphi_2 \dots \varphi_m \rangle \tag{1}$$

in which $\Gamma_0 = \Gamma_{\text{seq}}$, $\varphi_0 = \varepsilon$, $\Gamma_m = \emptyset$, and every step is produced by applying one of the inference rules described in (Kutsia, 2007). From now on we will say that t is a *strict subterm* of Γ_i, $0 \le i \le m$, if t is a strict subterm of a left- or right-hand side of an equation in Γ_i. There are some interesting properties of the derivation (1) we will make use of:

P1: In each Γ_i, $0 \le i \le m$, variables always appear under a function symbol, and the symbol f may appear only as the root of equation sides.

P2: Each φ_i, $1 \leq i \leq m$, has one of the following three forms: ε, $\{x \mapsto s\}$, or $\{x \mapsto (s, x)\}$, for some x and s with $x \notin \mathit{var}(s)$, where s is a strict subterm of Γ_{i-1}.

P3: Every strict subterm of Γ_i is a subterm of some $t\varphi_1 \ldots \varphi_i$ where t is a strict subterm of Γ_0. This follows from the observation that the every strict subterm of some Γ_i, $1 \leq i \leq m$, is either $s\varphi_i$ or a strict subterm of $s\varphi_i$, where s is a strict subterm of Γ_{i-1}.

Let $\mathit{subterms}(\Gamma')$ be the set of strict subterms of a unification problem Γ'. Obviously, $\mathit{subterms}(\Gamma_0)$ coincides with the set of terms and subterms which appear in Γ, therefore $|\mathit{subterms}(\Gamma_0)| \leq \mathit{size}(\Gamma)$. Some inference rules of $\mathfrak{I}$ compute substitutions $\varphi_i = \{x \mapsto (s, x)\}$, thus reintroducing the variable x in Γ_{i+1}. To keep track of which version of variable x we talk about, we add an index j to every version of x, and if the index of x in Γ_i is j, we perform $\varphi_i = \{x_j \mapsto (s, x_{j+1})\}$ which assigns version $j+1$ to the reintroduced variable x. Initially, we assume that all variables in Γ_0 have version 0, that is, we identify every variable x with x_0.

With respect to derivation (1), we associate to every term $t \in \mathit{subterms}(\Gamma_0)$ and $0 \leq i \leq m$ a labeled tree $\overline{T}_i(t)$ which represents $t\varphi_1 \ldots \varphi_i$, such that:

H1. the root node of $\overline{T}_i(t)$ is labeled with t,
H2. all its nodes are labeled with terms from $\mathit{subterms}(\Gamma_0) \cup \bigcup_{j=0}^{i} \mathit{var}(\Gamma_j)$, and
H3. all its substitution nodes are for some substitution from $\{\varphi_1, \ldots, \varphi_i\}$.

Let $\{t_1, \ldots, t_k\}$ be an enumeration of all terms in $\mathit{subterms}(\Gamma_0)$. We will define the sets of labeled trees $\mathcal{T}_i := \{\overline{T}_i(t) \mid t \in \mathit{subterms}(\Gamma_0)\}$ by recursion on i ($0 \leq i \leq m$), and prove simultaneously (by induction on i) that the properties H1–H3 hold for the inductively defined labeled trees.

For each $t \in \mathit{subterms}(\Gamma_0)$, $\overline{T}_0(t)$ is the labeled tree obtained from the tree representation of t by labelling the node at position p with the subterm of t at position p. (Positions in terms and trees are defined in the standard way, see, e.g., (Baader and Nipkow, 1998).) Thus, the labeled trees in $\mathcal{T}_0$ have no substitution nodes.

If $i > 0$, we distinguish two cases:

1. $\varphi_i = \epsilon$. In this case, $\overline{T}_i(t) = \overline{T}_{i-1}(t)$ for all $t \in \mathit{subterms}(\Gamma_0)$.
2. $\varphi_i = \{x_j \mapsto s\}$ or $\varphi_i = \{x_j \mapsto (s, x_{j+1})\}$ where $s \in \mathit{subterms}(\Gamma_{i-1})$. By property P3, there exists a first subterm t_j in the enumeration of

$\mathit{subterms}(\Gamma_0)$ such that s is a subterm of $t_j\varphi_1 \dots \varphi_{i-1}$. By construction, $\overline{T}_{i-1}(t_j)$ represents the term $t_j\varphi_1 \dots \varphi_{i-1}$. Therefore, there exists the leftmost innermost labeled subtree of $\overline{T}_{i-1}(t_j)$ which represents s, which we denote by $\overline{T}(s)$. If $\varphi_i = \{x_j \mapsto (s, x_{j+1})\}$ we also consider the labeled tree $\overline{T}(x_{j+1})$ made of only one node with label x_{j+1}, and define

$$\overline{T}_i(t) := \begin{cases} \overline{T}_{i-1}(t)[x_j \mapsto (\overline{T}(s), \overline{T}(x_{j+1}))] & \text{if } \varphi_i = \{x_j \mapsto (s, x_{j+1})\}, \\ \overline{T}_{i-1}(t)[x_j \mapsto \overline{T}(s)] & \text{if } \varphi_i = \{x_j \mapsto s\}. \end{cases}$$

where $\overline{T}_i(t)[x_j \mapsto (\overline{T}_1, \dots, \overline{T}_n)]$ denotes the labeled tree produced by connecting the labeled trees $\overline{T}_1, \dots, \overline{T}_n$ to every leaf node with label x_j in $\overline{T}_i(t)$.

This construction combined with the induction hypothesis implies that $\overline{T}_i(t)$ represents $t\varphi_1 \dots \varphi_i$, and that assumptions H1–H3 hold for it.

Let $t \in \mathit{subterms}(\Gamma_0)$. The mere existence of $\overline{T}_m(t)$, which is due to the incremental construction described before, implies that $\mathit{depth}(t\varphi)$ is the maximum number of edges along a branch in $\overline{T}_m(t)$, from which we remove the number of edges going down from a substitution node. But this is the same as saying that the depth of $t\varphi$ coincides with the maximum number of nodes with non-variable labels along a branch in $\overline{T}_m(t)$. By property L1 of labeled trees and property H2 of the construction, the labels of nodes with non-variable labels along a branch are distinct elements from $\mathit{subterms}(\Gamma_0)$. Therefore, $\mathit{depth}(t\varphi) \leq |\mathit{subterms}(\Gamma_0)|$. Since $|\mathit{subterms}(\Gamma_0)| \leq \mathit{size}(\Gamma)$, we proved the following lemma:

Lemma 6.1. *$\mathit{depth}(t\varphi) \leq \mathit{size}(\Gamma)$ for all terms and subterms t in Γ.*

An immediate consequence of this result is the following corollary:

Corollary 3. *$\mathit{depth}(\vartheta) \leq \mathit{size}(\Gamma)$.*

Example 6.2. Let $\Gamma = \{(x, g(z), y) \doteq (g(y), y, g(z))\}$ with $\mathsf{s} \preceq \mathsf{r}$, $g : \mathsf{s}^* \to \mathsf{s}$, $g : \mathsf{r}^* \to \mathsf{r}$, $x : \mathsf{s}$, $y : \mathsf{s}^*$, $z : \mathsf{r}$. Let also v be a variable with $v : \mathsf{s}$. Among the depth-minimal nonerasing unifiers of Γ are the substitutions $\vartheta_1 = \{x \mapsto g(g(v)), y \mapsto g(v), z \mapsto v\}$, $\vartheta_2 = \{x \mapsto g(g(v), g(v)), y \mapsto (g(v), g(v)), z \mapsto v\}$, $\vartheta_3 = \{x \mapsto g(g(v), g(v), g(v)), y \mapsto (g(v), g(v), g(v)), z \mapsto v\}, \dots$ Inspecting these unifiers, one can notice that, for instance, $x\vartheta_2$ is an instance of $g(y)$, obtained by replacing y with a sequence of instances of $g(z)$; $y\vartheta_2$ is a sequence of instances of $g(z)$; $z\vartheta_2$ is an instance of z. The depth of all these unifiers is 3.

Let ρ be a grounding substitution for $\Gamma\vartheta$, mapping each variable in $\Gamma\vartheta$ to a sequence of terms of appropriate sort and depth 1. Such terms exist: For each basic sort s, we can take $a(\epsilon)$, where a is a constant of sort s. We assumed there is at least one constant of each basic sort in the signature. Then $\mathit{depth}(\vartheta\rho) = \mathit{depth}(\vartheta) \leq \mathit{size}(\Gamma)$ by Corollary 3. Hence, $\vartheta\rho$ is a depth-minimal nonerasing ground unifier of Γ.

Translation into a WRCU 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:

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

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

Example 6.3. Let $\Gamma = \{f(x,y) \doteq f(f(y,a),b,c)\}$ with $\mathsf{s} \preceq \mathsf{r}$, $x : \mathsf{s}$, $y : \mathsf{r}^*$, $f : \mathsf{r}^* \to \mathsf{s}$, $a : \mathsf{1} \to \mathsf{s}$ and $b, c : \mathsf{1} \to \mathsf{r}$. Then Γ has a solution $\varphi = \{x \mapsto f(b,c,a), y \mapsto (b,c)\}$. On the other hand, $\mathit{Tr}(\Gamma) = \{fx\#yf \doteq ffy\#aaf\#bb\#ccf\}$ is a word unification problem, which has three nonerasing solutions: $\psi_1 = \{x \mapsto fbb\#cc\#aaf, y \mapsto bb\#cc\}$, $\psi_2 = \{x \to fcc\#aaf\#bb, y \mapsto cc\}$, $\psi_3 = \{x \mapsto faaf\#bbf\#cc, y \mapsto aaf\#bbf\#cc\}$. It is easy to see that $\psi_1 = \mathit{Tr}(\varphi)$, but ψ_2 and ψ_3 are extra substitutions introduced by the transformation. However, they are of different nature: $\mathit{Tr}^{-1}(\psi_2)$ 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. $\mathit{Tr}^{-1}(\psi_3)$ does not exist (which indicates that Tr is not surjective).

Lemma 6.4. *If φ is a substitution and $\tilde{t}$ is a sequence of REOS terms, then $\mathit{Tr}(\tilde{t})\,\mathit{Tr}(\varphi) = \mathit{Tr}(\tilde{t}\varphi)$.*

Proof. By structural induction on $\tilde{t}$. $\square$

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 prevent 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

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

That is, we translate only those $\tilde{t}$'s whose minimal sort is bounded by R and the 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: $L_1 ._{\#} L_2 = \{w_1 \# w_2 \mid w_1 \in L_1, w_2 \in L_2\}$, $L^{0\#} = \{\lambda\}$, $L^{1\#} = L$, $L^{n\#} = L ._{\#} L^{(n-1)\#}$ and $L^{*\#} = \cup_{n=0}^{\infty} L^{n\#}$.

For each 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 $depth(\Gamma)$ is reached:

- Depth 1:

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

- Depth $n > 1$:

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

Note 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 6.5. Consider again Γ and the sort information from Example 6.3. Now it gets translated into the WRCU problem

$$\Delta = \{fx\#yf \doteq ffy\#aaf\#bb\#ccf, x \in Tr(\mathsf{s},\Gamma), y \in Tr(\mathsf{r}^*,\Gamma)\}.$$

$Tr(\mathsf{s},\Gamma)$ contains (among others) $fbb\#cc\#aaf$, but neither $fcc\#aaf\#bb$ nor $faaf\#bbf\#cc$ are in it. $Tr(\mathsf{r}^*,\Gamma)$ contains (among others) $bb\#cc$. Hence, ψ_1 from Example 6.3 is a solution of Δ, but ψ_2 and ψ_3 are not.

Finally, we have the theorem:

Theorem 6.6. *Let $\Gamma = \{\tilde{s}_1 \doteq \tilde{t}_1, \dots, \tilde{s}_n \doteq \tilde{t}_n\}$ be a REOSU problem with $var(\Gamma) = \{x_1,\dots,x_m\}$ such that $x_i : \mathsf{R}_i$ for each $1 \leq i \leq m$. Let $\Delta = \{Tr(\tilde{s}_1) \doteq Tr(\tilde{t}_1), \dots, Tr(\tilde{s}_n) \doteq Tr(\tilde{t}_n), x_1 \in Tr(\mathsf{R}_1,\Gamma), \dots, x_m \in Tr(\mathsf{R}_m,\Gamma)\}$ be a word unification problem with regular constraints, obtained by translating Γ. Then Γ is solvable iff Δ is solvable.*

Proof. ($\Rightarrow$) Let φ be a depth-minimal unifier of Γ. Then, by Lemma 6.4, $Tr(\tilde{s}_i)Tr(\varphi) = Tr(\tilde{s}_i\varphi) = Tr(\tilde{t}_i\varphi) = Tr(\tilde{t}_i)Tr(\varphi)$ for each $1 \leq i \leq n$. On the other hand, for each $1 \leq j \leq m$, all terms in $x_j\varphi$ are from $\mathcal{T}(\mathcal{F}_\Gamma)$, $x_j Tr(\varphi) = Tr(x_j\varphi)$, $depth(x_j\varphi) \leq depth(\varphi) \leq size(\Gamma)$, and $lsort(x_j\varphi) \preceq \mathsf{R}_j$. It implies that $x_j Tr(\varphi) \in Tr(\mathsf{R}_j,\Gamma)$. Hence, $Tr(\varphi)$ solves Δ.

($\Leftarrow$) Let ψ be a solution of Δ. For each $1 \leq j \leq m$, since $x_j\psi \in Tr(\mathsf{R}_j,\Gamma)$, by definition of $Tr(\mathsf{R}_j,\Gamma)$, there exists a sequence $\tilde{r}_j$ such that all terms in $\tilde{r}$ are from $\mathcal{T}(\mathcal{F}_\Gamma)$, $depth(\tilde{r}) \leq size(\Gamma)$, $lsort(\tilde{r}) \preceq \mathsf{R}_j$, and $Tr(\tilde{r}) = x_j\psi$. Hence, $Tr^{-1}(\psi)$ exists. Obviously, $x_j Tr^{-1}(\psi) = \tilde{r}_j$ for each $1 \leq j \leq m$. By Lemma 6.4, $Tr(\tilde{t})\psi = Tr(\tilde{t})Tr(Tr^{-1}(\psi)) = Tr(\tilde{t}Tr^{-1}(\psi))$ for each $\tilde{t}$. In particular, for each $Tr(\tilde{s}_i) \doteq Tr(\tilde{t}_i) \in \Delta$, we have $Tr(\tilde{s}_i Tr^{-1}(\psi)) = Tr(\tilde{t}_i Tr^{-1}(\psi))$. Since Tr is injective, it implies $\tilde{s}_i Tr^{-1}(\psi) = \tilde{t}_i Tr^{-1}(\psi)$ for each $1 \leq i \leq n$. Hence, $Tr^{-1}(\psi)$ is a unifier of Γ. $\square$

Hence, the problem of deciding solvability of REOSU has been reduced (by a solvability-preserving transformation) to the problem of deciding solvability of WRCU. Since the latter is decidable (Schulz, 1990), we conclude with the following result:

Theorem 6.7 (Decidability)**.** *Solvability of REOSU is decidable.*

7. Decidability of Sequence Unification with Regular Hedge Constraints

Decidability of REOSU has an interesting consequence: Decidability of sequence unification with regular hedge constraints. It generalizes decidability of word unification with regular constraints (Schulz, 1990) to sequences. To prove it, we first need to introduce some definitions.

In Sect. 5, we mentioned that SEQU problems can be seen as REOSU problems built over one basic sort s, all function symbols have the sort $\mathsf{s}^* \to \mathsf{s}$, and each variable has either the sort s (individual variable) or s^* (sequence variable). We do not mention sorts explicitly, when we talk about SEQU problems.

A *finite hedge automaton* $\mathcal{A}$ is a tuple $(Q, F, \mathsf{R}_f, \delta)$ where Q, F, and δ are defined exactly as in the case of unranked tree automata in Sect. 3, while R_f is a regular expression over Q. The automaton is *deterministic* if for all rules $f(\mathsf{R}_1) \to q_1, f(\mathsf{R}_2) \to q_2 \in \delta$, $q_1 \neq q_2$ implies $[\![\mathsf{R}_1]\!] \cap [\![\mathsf{R}_2]\!] = \emptyset$. (We also assume that there are no two rules $f(\mathsf{R}_1) \to q, f(\mathsf{R}_2) \to q \in \delta$: They are replaced by $f(\mathsf{R}_1{+}\mathsf{R}_2) \to q$.)

For hedge automata, the move relation is defined similarly as for the unranked tree case, with the difference that it can act on hedges (sequences) of unranked trees instead of unranked trees. The language $\mathcal{L}(\mathcal{A})$ recognized by a finite hedge automaton $\mathcal{A}$ is the set of hedges $\mathcal{L}(\mathcal{A}) = \{(t_1, \dots, t_n) \in \mathcal{T}(F)^n \mid \text{there exist } q_1, \dots, q_n \text{ such that } t_i \longrightarrow^*_{\mathcal{A}} q_i \text{ holds for each } 1 \leq i \leq n \text{ and } q_1 \cdots q_n \in [\![\mathsf{R}_f]\!]\}$.

A *sequence unification problem with regular constraints* (SEQURC) is a triple

$$\Pi = \Delta; \{X_1 \,\mathsf{in}\, \mathsf{R}_1, \dots, X_m \,\mathsf{in}\, \mathsf{R}_m\}; (Q, F, \delta),$$

where $\Delta = \{s_1 \doteq t_1, \dots, s_n \doteq t_n\}$ is a SEQU problem built over F and individual and sequence variables. For all $1 \leq j \leq m$, the variables X_j are some of the sequence variables occurring in Δ, and the regular expressions R_i are built over Q such that $(Q, F, \mathsf{R}_i, \delta)$ is a deterministic unranked hedge automaton. A solution of such a SEQURC problem is a substitution φ that solves Γ and satisfies the constraints: $X_j\varphi \in \mathcal{L}(Q, F, \mathsf{R}_j, \delta)$ for all $1 \leq j \leq m$.

Now, we encode the SEQURC problem Π above as a REOSU problem Γ_Π over the signature $\varphi = (\mathcal{B}, \preceq, \mathcal{F})$ defined as follows:

- The equations in Γ_Π are those in Δ.
- The set of basic sorts $\mathcal{B}$ is defined as $Q \cup \{\mathsf{t}\}$ where t is a new sort.

- The partial ordering on $\mathcal{B}$ is assumed to be $\preceq = \{(q, \mathsf{t}) \mid q \in Q\}$, that is, t is assumed to be the $\preceq$-maximal basic sort of $\mathcal{B}$.
- $\mathcal{F}$ is the set of all symbols that occur in F and in Δ, $f \in \mathcal{F}_{\mathsf{t}^* \to \mathsf{t}}$ for all $f \in \mathcal{F}$ and, in addition, $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$ whenever $f(\mathsf{R}) \to \mathsf{s} \in \delta$.

As for the variables in Γ_Π, we assume that $X_i \in \mathcal{V}_{\mathsf{R}_i}$ for $1 \leq i \leq m$, $X \in \mathcal{V}_{\mathsf{t}^*}$ for any other sequence variable X in Δ, and $x \in \mathcal{V}_{\mathsf{t}}$ for any individual variable x in Δ.

Lemma 7.1. $\Sigma = (\mathcal{B}, \preceq, \mathcal{F})$ *is a preregular REOS signature.*

Proof. $\mathcal{B}$ is obviously finite. We extend the $\preceq$ ordering on $\mathcal{B}$ to the set of regular expressions over $\mathcal{B}^*$ in the usual way. $\mathcal{F}$ is also finite (since it consists only of function symbols occurring in F and in Γ) and, therefore, finitely overloading. Also, it is easy to see that $\mathcal{F}$ is monotonic and preregular.

- *Monotonicity:* We may have only one kind of overloading: The same f may belong to $\mathcal{F}_{\mathsf{R} \to \mathsf{s}}$ (that comes from the automaton in SEQURC) and to $\mathcal{F}_{\mathsf{t}^* \to \mathsf{t}}$. Since $\mathsf{R} \preceq \mathsf{t}^*$ and $\mathsf{s} \preceq \mathsf{t}$, the monotonicity property holds.
- *Preregularity:* Let $f \in \mathcal{F}_{\mathsf{t}^* \to \mathsf{t}}$. Then for all $\mathsf{R}_0 \preceq \mathsf{t}^*$, the set of sorts $\{\mathsf{s} \mid f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}} \text{ and } \mathsf{R}_0 \preceq \mathsf{R}\}$ is either $\{\mathsf{t}\}$ or $\{\mathsf{t}, q\}$ for some q. Both sets have a $\preceq$-least element. If $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$, then for all $\mathsf{R}_0 \preceq \mathsf{R}$, the set $\{\mathsf{s}' \mid f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}'} \text{ and } \mathsf{R}_0 \preceq \mathsf{R}\}$ is $\{\mathsf{s}\}$. Hence, preregularity also holds.

□

Lemma 7.2. Π *is solvable iff the corresponding REOSU* Γ_Π *is solvable.*

Proof. If φ is a solution of Π, then it can solve each equation in Δ, i.e., in a sort-free version of Γ_Π. To show that φ respects the sorts for Γ_Π, it is enough to notice that for the constrained sequence variables X_j in Π, we have $X_j\varphi \in \mathcal{L}(Q, F, \mathsf{R}_j, \delta)$, and, hence, the least sort of the encoding of $X_j\varphi$ is $\preceq \mathsf{R}_j$. Hence, each solution of Π is a solution of Γ_Π. On the other hand, with a similar argument we can see that all unifiers of Γ_Π are solutions of Π. □

The lemmas 7.1 and 7.2 imply decidability of SEQURC:

Theorem 7.3. *Solvability of SEQURC is decidable.*

8. Computing Unifiers and Matchers

8.1. Unification Procedure

To compute unifiers for a REOSU problem, one can ignore the sort information, treat each variable as a sequence variable, employ the SEQU procedure (Kutsia, 2002, 2007) 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. (Schmidt-Schauß, 1986, 1989; Meseguer et al., 1989; Smolka et al., 1989; Hendrix and Meseguer, 2012). 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 $\varphi = \{x_1 \mapsto \tilde{t}_1, \dots, x_n \mapsto \tilde{t}_n\}$ of the unsorted version of an REOSU problem Γ. We can assume without loss of generality that φ is idempotent. Then we form a weakening problem $W = \{\tilde{t}_1 \rightsquigarrow \mathit{lsort}(x_1), \dots, \tilde{t}_n \rightsquigarrow \mathit{lsort}(x_n)\}$, and find the set of weakening substitutions $\mathit{weak}(W)$. If $\mathit{weak}(W) = \emptyset$, then φ can not be weakened further to a solution of Γ. Otherwise, $\varphi\vartheta$ is a solution of Γ for each $\vartheta \in \mathit{weak}(W)$. Completeness and minimality of the obtained set of solutions is proved in Lemma 5.2 and Lemma 5.3.

A drawback of this approach is that it is a so called generate-and-test method. It is not able to detect derivations that fail because of sort incompatibility, until the weakening algorithm is run on the generated SEQU unifiers. Early failure detection requires weakening to be incorporated into the unification rules. There is a pretty straightforward (although technically a bit involved) way of doing this. We do not go into detail here. The interested reader can find the corresponding algorithm in Kutsia and Marin (2012).[3]

By restricting sorts or occurrences of variables, various terminating fragments of REOSU can be obtained. Some of such fragments are listed below:

- Sorts of all variables in a REOSU problem 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

[3]This approach is similar to the one for ranked terms described in (Meseguer et al., 1989), where an order-sorted version of the algorithm of Martelli and Montanari (1982) is presented.

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 another 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 (Walther, 1988). 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.

- Variables whose sort contains the star occur in the last argument position. This is a pretty useful terminating (unitary) fragment for which more optimized algorithm can be designed, based on the ideas of a similar fragment in sequence unification (Kutsia, 2007).

- One side of each equation in Γ is ground. In this case Γ is finitary. These are REOS matching problems. For them there is no need to invoke the weakening algorithm. Because of its practical importance, we consider the matching fragment in more details.

8.2. Matching Algorithm

A matching equation is a pair of term sequences $\tilde{s} \ll \tilde{t}$, where $\tilde{t}$ is ground. A *regular expression order sorted matching* problem or, shortly, a REOSM problem is a finite set of matching equations. A substitution φ is a *matcher* of a REOSM problem $\{\tilde{s}_1 \ll \tilde{t}_1, \ldots, \tilde{s}_n \ll \tilde{t}_n\}$ iff $\tilde{s}_i\varphi = \tilde{t}_i$ for all $1 \leq i \leq n$.

REOSM is a special case of REOSU. Unlike REOSU, in REOSM there is no need to compute weakening substitutions: Solving regular language membership problem suffices. The rules of the REOSM procedure can be formulated as follows:

T-M: **Trivial**

$\{\epsilon \ll \epsilon\} \uplus \Gamma; \varphi \Longrightarrow \Gamma; \varphi.$

D-M: **Decomposition**

$\{(f(\tilde{t}), \tilde{t}') \ll (f(\tilde{s}), \tilde{s}')\} \uplus \Gamma; \varphi \Longrightarrow \{\tilde{t} \ll \tilde{s}, \tilde{t}' \ll \tilde{s}'\} \cup \Gamma; \varphi.$

E-M: **Elimination**

$$\{(x, \tilde{t}) \ll (\tilde{s}, \tilde{s}')\} \uplus \Gamma; \varphi \Longrightarrow \{\tilde{t}\vartheta \ll \tilde{s}'\} \cup \Gamma\vartheta; \varphi\vartheta,$$

if $\mathit{lsort}(\tilde{s}) \preceq \mathit{lsort}(x)$ and $\vartheta = \{x \mapsto \tilde{s}\}$.

To match a term sequence $\tilde{s}$ to a ground term sequence $\tilde{t}$, we create the initial system $\{\tilde{s} \ll \tilde{t}\}; \varepsilon$ and apply the rules exhaustively as long as it is possible. Problems to which no rule applies are transformed into $\bot$. The REOSM algorithm defined in this way is denoted by $\mathfrak{M}$.

The E-M rule is the only one which makes a choice: There can be various ways to split the sequence in the right hand side of the selected equation into $\tilde{s}$ and $\tilde{s}'$ such that the rule condition is satisfied.

Derivations are sequences of rule applications. A derivation of the form $\Gamma; \varepsilon \Longrightarrow^* \emptyset; \varphi$ is called a *successful derivation* and φ is called a *computed substitution* of Γ. We denote the set of substitutions computed by $\mathfrak{M}$ for Γ with $\mathit{comp}(\mathfrak{M}(\Gamma))$.

It is easy to check that the matching rules above are sound. It implies that every computed substitution of Γ is a matcher of Γ.

The fact that we do not need to use weakening suggests that $\mathit{comp}(\mathfrak{M}(\Gamma))$ is a subset of the complete set of matchers of the unsorted version of Γ.

Example 8.1. Let $\Gamma = \{f(x, y) \ll f(f(a, c), b, c)\}$ with $\mathsf{s} \preceq \mathsf{r}$, $x : \mathsf{s}.(\mathsf{s}{+}\mathsf{1})$, $y : \mathsf{r}^*$, $f : \mathsf{r}^* \to \mathsf{s}$, $a, b : \mathsf{1} \to \mathsf{s}$ and $c : \mathsf{1} \to \mathsf{r}$. Then $\mathit{comp}(\mathfrak{M}(\Gamma)) = \{\varphi_1, \varphi_2\}$, where $\varphi_1 = \{x \mapsto f(a, c), y \mapsto (b, c)\}$ and $\varphi_2 = \{x \mapsto (f(a, c), b), y \mapsto c\}$.

If we forget the sort information, then there are two more matchers for Γ: $\{x \mapsto \epsilon, y \mapsto (f(a, c), b, c)\}$ and $\{x \mapsto (f(a, c), b, c), y \mapsto \epsilon\}$.

To prove termination, we first define inductively the *norm* $\|\tilde{t}\|$ of term sequence $\tilde{t}$:

- $\|x\| = 2$,
- $\|f(\tilde{t})\| = \|\tilde{t}\| + 2$,
- $\|(t_1, \ldots, t_n)\| = \|t_1\| + \cdots + \|t_n\| + 1$.

The norm of a matching equation $\tilde{t} \ll \tilde{s}$ is $\|\tilde{s}\|$. We associate to each REOSM problem Γ its *measure,* which is a pair $\langle n, M\rangle$, where n is the number of distinct variables in Γ and M is the multiset of norms of matching equations in Γ. Measures are compared lexicographically. This ordering is well-founded. Each matching rule strictly reduces the measure: T-M and D-M do not increase n and decrease M, whereas E-M decreases n. Hence, we have

Theorem 8.2 (Termination of $\mathfrak{M}$). *The algorithm* $\mathfrak{M}$ *terminates on any matching problem.*

Moreover, for a REOSM problem Γ, the algorithm $\mathfrak{M}$ is able to compute any matcher whose domain is $\mathit{var}(\Gamma)$ and computes any matcher exactly once:

Theorem 8.3 (Completeness and Minimality of $\mathfrak{M}$). $\mathit{comp}(\mathfrak{M}(\Gamma))$ *is a minimal complete set of matchers of a REOSM problem* Γ. *Moreover, no matcher is computed more than once.*

Proof. Let μ be an arbitrary matcher of Γ. We can construct a derivation in $\mathfrak{M}$ that computes a matcher that coincides with μ on $\mathit{var}(\Gamma)$ as follows: Starting from Γ, we apply to each selected equation the $\mathsf{T\text{-}M}$ or $\mathsf{D\text{-}M}$ rule whenever applicable. If the selected equation is such that the $\mathsf{E\text{-}M}$ rule should apply, we take $x\mu$ in the role of $\tilde{s}$ in this rule. This process terminates, computing a matcher whose domain is $\mathit{var}(\Gamma)$ and which coincides to μ on the domain. Hence, for each matcher μ of Γ, the set $\mathit{comp}(\mathfrak{M}(\Gamma))$ contains an element that coincides with μ on $\mathit{var}(\Gamma)$. Thus, completeness holds.

The claim that no matcher is computed more than once follows from the fact that from the matching rules, only $\mathsf{E\text{-}M}$ causes branching in the search space. If at the branching point a variable x is instantiated in two different ways, with $\tilde{s}_1$ on one branch and with $\tilde{s}_2$ on another, that there is no chance the instantiations of x further on those branches to become the same, because $\tilde{s}_1$ and $\tilde{s}_2$ are distinct ground hedges. It follows that no matcher is computed more than once.

Minimality follows from the fact that given two matchers φ_1 and φ_2 of Γ, neither $\varphi_1 \leq_{\mathit{var}(\Gamma)} \varphi_2$ nor $\varphi_2 \leq_{\mathit{var}(\Gamma)} \varphi_1$ holds, since φ_1 and φ_2 are syntactic matchers, which map each $x \in \mathit{var}(\Gamma)$ to a ground term or ground term sequence. □

Now we show that REOSM is NP-complete. The input consists of the matching problem Γ and the sort information for each variable and function symbol appearing in Γ. The sort information contains declarations of the form $f \in \mathcal{F}_{\mathsf{R}\to\mathsf{s}}$ for each f occurring in the matching problem (such declarations are finitely many for each f because of the finite overloading property), $x \in \mathcal{V}_{\mathsf{R}}$ for each x occurring in Γ, and the finite set of basic sorts together with the subsort relation on it.

Membership in NP depends on whether the condition in the rule $\mathsf{E\text{-}M}$ (i.e., $\mathit{lsort}(\tilde{s}) \preceq \mathit{lsort}(x)$) can be checked in polynomial time. Computing

$\mathit{lsort}(x)$ is easy: $\mathit{lsort}(x) \simeq \mathsf{R}$, where $x \in \mathcal{V}_{\mathsf{R}}$. So, it is just a lookup. As for $\mathit{lsort}(\tilde{s})$, note that $\tilde{s}$ is a ground sequence. Therefore, $\mathit{lsort}(\tilde{s})$ is (modulo $\simeq$) either a concatenation of basic sorts, or $\mathsf{1}$. Then $[\![\mathit{lsort}(\tilde{s})]\!] \simeq \{w\}$, i.e., it consists of a single word w over basic sorts. If $\mathit{lsort}(\tilde{s}) \simeq \mathsf{s}_1.\cdots.\mathsf{s}_n$ for some basic sorts $\mathsf{s}_1, \ldots, \mathsf{s}_n$, then $w = \mathsf{s}_1 \cdots \mathsf{s}_n$. If $\mathit{lsort}(\tilde{s}) \simeq \mathsf{1}$, then $w = \lambda$. In any case, for checking $\mathit{lsort}(\tilde{s}) \preceq \mathit{lsort}(x)$, we just need to check $w \in [\![\overline{\mathsf{R}}]\!]$. For this, the only thing which is not straightforward is the computation of w (i.e., of $\mathit{lsort}(\tilde{s})$) in polynomial time. The other operations involved in the check are polynomial: R, as we said above, is just looked up; computing $\overline{\mathsf{R}}$ amounts replacing each basic sort appearing in R with the sum of all its basic subsorts; the membership test for regular languages is polynomial, see, e.g. (Thompson, 1968; Ponty, 2000).

To see that $\mathit{lsort}(\tilde{s})$ can be computed in polynomial time, we reason as follows: We compute the least sort of terms bottom-up. Given the least sort r_i of the ground terms r_i, $1 \leq i \leq n$, for computing the least sort of $f(r_1, \ldots, r_n)$, due to preregularity and groundness of r_i's, we need to find the $\preceq$-least element of the finite set of basic sorts $\{\mathsf{s} \mid f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}} \text{ and } \mathsf{r}_1 \cdots \mathsf{r}_n \in [\![\overline{\mathsf{R}}]\!]\}$. If $n = 0$, then instead of the word $\mathsf{r}_1 \cdots \mathsf{r}_n$ we have the empty word λ. Checking $\mathsf{r}_1 \cdots \mathsf{r}_n \in [\![\overline{\mathsf{R}}]\!]$ (resp. $\lambda \in [\![\overline{\mathsf{R}}]\!]$) is polynomial and has to be done as many times as there is a declaration $f \in \mathcal{F}_{\mathsf{R} \to \mathsf{s}}$ in the input. It is straightforward to come up with a linear algorithm for selecting the least element from a partially ordered set which is known to contain such an element. Hence, the least sort of each subterm in $\tilde{s}$ can be computed in polynomial time with respect to the size of input, which implies that $\mathit{lsort}(\tilde{s})$ can also be computed in polynomial time. (cf. discussion on polynomial-time computation of least sorts in the ranked order-sorted case in (Eker, 2011).)

Next, we concentrate on NP-hardness. It can be proved by reduction from positive 1-IN-3-SAT problem (Schaefer, 1978). A positive 1-IN-3-SAT problem is given by a set of clauses $\{C_1, \ldots, C_n\}$ where each clause C_i contains exactly three positive literals $p_{i1} \vee p_{i2} \vee p_{i3}$ from a set of literals $p_1, \ldots, p_m$. A truth assignment solves the problem if it maps exactly one literal from each clause to true. To encode this problem as a REOSM problem, we introduce three basic sorts: true, false, and value, ordering them as $\mathsf{true} \preceq \mathsf{value}$ and

$\mathsf{false} \preceq \mathsf{value}$. We also have the following function symbols:

$$\begin{array}{rlrl}
\mathit{and} & : \mathsf{value}^* \to \mathsf{value} & \mathit{assign} & : \mathsf{value}^* \to \mathsf{value} \\
 & : \mathsf{value}^*.\mathsf{false}.\mathsf{value}^* \to \mathsf{false} & & : \mathsf{value}^*.\mathsf{true}.\mathsf{value}^* \to \mathsf{true} \\
 & : \mathsf{true}^* \to \mathsf{true} & & : \mathsf{false}^* \to \mathsf{false} \\
\mathit{or} & : \mathsf{value}^* \to \mathsf{value} & t & : 1 \to \mathsf{true} \\
 & : \mathsf{value}^*.\mathsf{true}.\mathsf{value}^* \to \mathsf{true} & f & : 1 \to \mathsf{false} \\
 & : \mathsf{false}^* \to \mathsf{false} & &
\end{array}$$

For each p_i, we introduce a variable $x_i : \mathsf{value}$ and for each clause C_j, a pair of variables $y_1^j : \mathsf{value}^*$ and $y_2^j : \mathsf{value}^*$. Obviously, we obtain a REOS signature. Then the given positive 1-IN-3-SAT problem is encoded as the following REOSM problem:

$$\begin{aligned}
\{\mathit{and}(\mathit{assign}(y_1^1, \mathit{or}(x_{11}, x_{12}, x_{13}), y_2^1), \ldots,\\
\mathit{assign}(y_1^n, \mathit{or}(x_{n1}, x_{n2}, x_{n3}), y_2^n)) \ll\\
\mathit{and}(\mathit{assign}(\mathit{or}(t, f, f), \mathit{or}(f, t, f), \mathit{or}(f, f, t)), \ldots,\\
\mathit{assign}(\mathit{or}(t, f, f), \mathit{or}(f, t, f), \mathit{or}(f, f, t)))\}
\end{aligned}$$

This encoding is polynomial and preserves solvability in both directions. It implies that REOSM is NP-hard. Hence, we proved the following theorem:

Theorem 8.4. *REOSM is NP-complete.*

Now we turn to complexity of the counting problem for REOS matching. First, we introduce some definitions, following (Hermann and Kolaitis, 1995).

Assume Σ_1 and Σ_2 are nonempty alphabets and let $w : \Sigma_1^* \to \mathcal{P}(\Sigma_2^*)$ be a function from the set Σ_1^* of words over Σ_1 to the power set $\mathcal{P}(\Sigma_2^*)$ of Σ_2^*. If x is a word in Σ^*, then $w(x)$ is called the *witness set* for x. Its elements are called *witnesses* for x. Every such witness function w can be identified with the following counting problem w: Given a word $x \in \Sigma^*$, find the number of witnesses for x in the set $w(x)$. Below $|x|$ stands for the length of a word x and $|S|$ for the cardinality of the set S.

Valiant (1979a,b) defined the class #P as the class of functions counting the number of accepting paths of a nondeterministic polynomial-time Turing machine. Here we work with a different but equivalent description of this class that appears in (Kozen, 1991). With this definition, #P is the class of witness functions w such that

(#P.1) there is a polynomial-time algorithm to determine, for a given x and y, whether $y \in w(x)$;

(#P.2) there exists a natural number k such that for all $y \in w(x)$, $|y| \leq |x|^k$ (note that k can depend on w).

Counting problems relate to each other via counting reductions. They are defined as follows: Let $w : \Sigma_1^* \to \mathcal{P}(\Sigma_2^*)$ and $v : \Pi_1^* \to \mathcal{P}(\Pi_2^*)$ be two counting problems. A *counting reduction* from w to v is a pair of polynomial-time computable functions $\sigma : \Sigma_1^* \to \Pi_1^*$ and $\tau : \mathbb{N} \to \mathbb{N}$, such that $|w(x)| = \tau(|v(\sigma(x))|)$ for all $x \in \Sigma_1^*$.

A counting problem v is *#P-hard* if for each counting problem w in #P there is a counting reduction from w to v. If in addition v is a member of #P, then v is *#P-complete.*

Now we associate to REOSM the following problem, which we call #REOSM:

Input: A REOS term sequence $\tilde{s}$ and a ground REOS term sequence $\tilde{t}$.

Output: Cardinality of the minimal complete set of matchers of $\{\tilde{s} \ll \tilde{t}\}$.

The main result about counting complexity of REOS matching is #P-completeness of #REOSM:

Theorem 8.5. *#REOSM is #P-complete.*

Proof. First, we show that #REOSM is in #P and then prove its #P-hardness.

Membership in #P: We should find a function w which satisfies the conditions (#P.1) and (#P.2) above. This is pretty straightforward: In the role of w we can take a function which for (a string representation of) any $\tilde{s}$ and ground $\tilde{t}$ returns the set consisting of string representations of the substitutions from the minimal complete set of matchers $\{\tilde{s} \ll \tilde{t}\}$. (Note that the minimal complete set of REOS matchers of Γ is unique, if we restrict substitution domain to $var(\Gamma)$.) Now, for such a w, the condition (#P.1) is satisfied because for any substitution φ we can check in polynomial time whether $\tilde{s}\varphi = \tilde{t}$ holds (and, hence, whether for a string representation y of φ and for a string representation x of $\tilde{s} \ll \tilde{t}$, the inclusion $y \in w(x)$ holds). The fact that w fulfills the condition (#P.2) follows from the observation that the size of φ does not exceed the size of $\tilde{t}$, since $\tilde{s}\varphi = \tilde{t}$.

#P-Hardness: Examining the reduction from positive 1-IN-3-SAT problem to REOSM above, we can see that it is a counting reduction: To each solution of the 1-IN-3-SAT problem corresponds exactly one matcher. Hence, the function τ in the definition of counting reduction is the identity function. (Such counting reductions are called parsimonious reductions.) Now #P-hardness follows from the fact that #-positive 1-IN-3-SAT problem is #P-complete (Creignou and Hermann, 1996). $\square$

9. Conclusion

We studied unification in order-sorted theories with regular expression sorts. A regular expression order-sorted signature can be viewed as a bottom-up finite unranked tree automaton. We proved that REOSU is infinitary and decidable. Based on the latter result, we generalized decidability of word unification with regular constraints to terms, proving decidability of sequence unification with regular hedge language constraints. We designed a sort weakening algorithm which helps to construct solutions of a REOSU problem from the solutions of the unsorted problem of sequence unification. Besides, we studied REOS matching, developed its solving algorithm, proved that the problem is NP-complete and the corresponding counting problem is #P-complete.

There are some interesting research questions we did not consider in this paper. An instance of such a problem is simplification of arbitrary equational formulas in the regular expression order-sorted framework. One can think about generalizing the procedure of Comon and Delor (1994) from the ranked order-sorted setting to a REOS language, exploring relationships between REOS signatures and unranked tree automata. Another interesting direction of future work would be to study REOS unification modulo equational theories.

Acknowledgments

We would like to thank José Meseguer for pertinent hints to the literature.

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 the Austrian Science Fund (FWF) under the project SToUT (P 24087-N18).

References

Antimirov, V. M., 1995. Rewriting regular inequalities (extended abstract). In: Reichel, H. (Ed.), FCT. Vol. 965 of Lecture Notes in Computer Science. Springer, pp. 116–125.

Antimirov, V. M., 1996. Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. 155 (2), 291–319.

Baader, F., Nipkow, T., 1998. Term Rewriting and All That. Cambridge University Press.

Baader, F., Snyder, W., 2001. Unification theory. In: Robinson, J. A., Voronkov, A. (Eds.), Handbook of Automated Reasoning. Elsevier and MIT Press, pp. 445–532.

Boudet, A., 1992. Unification in order-sorted algebras with overloading. In: Kapur (1992), pp. 193–207.

Comon, H., 1989. Inductive proofs by specification transformation. In: Dershowitz, N. (Ed.), RTA. Vol. 355 of Lecture Notes in Computer Science. Springer, pp. 76–91.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., Tommasi, M., 2007. Tree automata techniques and applications. http://tata.gforge.inria.fr.

Comon, H., Delor, C., 1994. Equational formulae with membership constraints. Inf. Comput. 112 (2), 167–216.

Conway, J. H., 1971. Regular Algebra and Finite Machines. Chapman and Hall, London.

Creignou, N., Hermann, M., 1996. Complexity of generalized satisfiability counting problems. Inf. Comput. 125 (1), 1–12.

Eker, S., 2011. Fast sort computations for order-sorted matching and unification. In: Agha, G., Danvy, O., Meseguer, J. (Eds.), Formal Modeling: Actors, Open Systems, Biological Systems. Vol. 7000 of Lecture Notes in Computer Science. Springer, pp. 299–314.

Frisch, A. M., Cohn, A. G., 1992. An abstract view of sorted unification. In: Kapur (1992), pp. 178–192.

Gelade, W., Neven, F., 2012. Succinctness of the complement and intersection of regular expressions. ACM Trans. Comput. Log. 13 (1), 4.

Goguen, J. A., 1978. Order sorted algebra. Tech. Rep. Tech. Report 14, UCLA Computer Science Department.

Goguen, J. A., Diaconescu, R., 1994. An oxford survey of order sorted algebra. Mathematical Structures in Computer Science 4 (3), 363–392.

Goguen, J. A., Meseguer, J., 1992. Order-sorted algebra i: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theor. Comput. Sci. 105 (2), 217–273.

Hendrix, J., Meseguer, J., 2012. Order-sorted equational unification revisited. Electr. Notes Theor. Comput. Sci. 290, 37–50.

Hermann, M., Kolaitis, P. G., 1995. The complexity of counting problems in equational matching. J. Symb. Comput. 20 (3), 343–362.

Hosoya, H., Pierce, B. C., 2003a. Regular expression pattern matching for XML. J. Funct. Program. 13 (6), 961–1004.

Hosoya, H., Pierce, B. C., 2003b. XDuce: a statically typed XML processing language. ACM Trans. Internet Techn. 3 (2), 117–148.

Jacquemard, F., Rusinowitch, M., 2008. Closure of hedge-automata languages by hedge rewriting. In: Voronkov, A. (Ed.), RTA. Vol. 5117 of Lecture Notes in Computer Science. Springer, pp. 157–171.

Kapur, D. (Ed.), 1992. Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings. Vol. 607 of Lecture Notes in Computer Science. Springer.

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

Kozen, D., 1991. The Design and Analysis of Algorithms. Springer-Verlag, New York.

Krajiček, J., Pudlák, P., 1988. The number of proof lines and the size of proofs in first order logic. Archive for Mathematical Logic 27, 69–84.

Kutsia, T., 2002. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In: Calmet, J., Benhamou, B., Caprotti, O., Henocque, L., Sorge, V. (Eds.), AISC. Vol. 2385 of Lecture Notes in Computer Science. Springer, pp. 290–304.

Kutsia, T., 2007. Solving equations with sequence variables and sequence functions. J. Symb. Comput. 42 (3), 352–388.

Kutsia, T., Levy, J., Villaret, M., 2007. Sequence unification through currying. In: Baader, F. (Ed.), RTA. Vol. 4533 of Lecture Notes in Computer Science. Springer, pp. 288–302.

Kutsia, T., Levy, J., Villaret, M., 2010. On the relation between context and sequence unification. J. Symb. Comput. 45 (1), 74–95.

Kutsia, T., Marin, M., 2005a. Can context sequence matching be used for querying XML? In: Vigneron, L. (Ed.), 19th International Workshop on Unification, UNIF 2005. Nara, Japan, pp. 77–92.

Kutsia, T., Marin, M., 2005b. Matching with regular constraints. In: Sutcliffe, G., Voronkov, A. (Eds.), LPAR. Vol. 3835 of Lecture Notes in Computer Science. Springer, pp. 215–229.

Kutsia, T., Marin, M., 2012. Regular expression order-sorted unification and matching. Tech. Rep. 12-14, RISC, Johannes Kepler University Linz, http://www.risc.jku.at/publications/download/risc_4685/TR-12-14.pdf.

Levy, J., Villaret, M., 2001. Context unification and traversal equations. In: Middeldorp, A. (Ed.), RTA. Vol. 2051 of Lecture Notes in Computer Science. Springer, pp. 169–184.

Makanin, G. S., 1977. The problem of solvability of equations in a free semigroup. Math. USSR Sbornik 32 (2), 129–198.

Martelli, A., Montanari, U., 1982. An efficient unification algorithm. ACM Trans. Program. Lang. Syst. 4 (2), 258–282.

Meseguer, J., Goguen, J. A., Smolka, G., 1989. Order-sorted unification. J. Symb. Comput. 8 (4), 383–413.

Ponty, J.-L., 2000. An efficient null-free procedure for deciding regular language membership. Theor. Comput. Sci. 231 (1), 89–101.

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

Schaefer, T. J., 1978. The complexity of satisfiability problems. In: Lipton, R. J., Burkhard, W. A., Savitch, W. J., Friedman, E. P., Aho, A. V. (Eds.), STOC. ACM, pp. 216–226.

Schmidt-Schauß, M., 1986. Unification in many-sorted eqational theories. In: Siekmann, J. H. (Ed.), CADE. Vol. 230 of Lecture Notes in Computer Science. Springer, pp. 538–552.

Schmidt-Schauß, M., 1989. Computational Aspects of an Order-Sorted Logic with Term Declarations. Vol. 395 of Lecture Notes in Computer Science. Springer.

Schulz, K. U., 1990. Makanin's algorithm for word equations - two improvements and a generalization. In: Schulz, K. U. (Ed.), IWWERT. Vol. 572 of Lecture Notes in Computer Science. Springer, pp. 85–150.

Smolka, G., Nutt, W., Goguen, J. A., Meseguer, J., 1989. Order-sorted equational computation. In: Nivat, M., Aït-Kaci, H. (Eds.), Resolution of Equations in Algebraic Structures. Vol. 2. Academic Press, pp. 297–367.

Sulzmann, M., Lu, K. Z. M., 2007. Xhaskell - adding regular expression types to haskell. In: Chitil, O., Horváth, Z., Zsók, V. (Eds.), IFL. Vol. 5083 of Lecture Notes in Computer Science. Springer, pp. 75–92.

Thompson, K., 1968. Regular expression search algorithm. Commun. ACM 11 (6), 419–422.

Uribe, T. E., 1992. Sorted unification using set constraints. In: Kapur (1992), pp. 163–177.

Valiant, L. G., 1979a. The complexity of computing the permanent. Theor. Comput. Sci. 8, 189–201.

Valiant, L. G., 1979b. The complexity of enumeration and reliability problems. SIAM J. Comput. 8 (3), 410–421.

Walther, C., 1988. Many-sorted unification. J. ACM 35 (1), 1–17.

Weidenbach, C., 1996. Unification in sort theories and its applications. Ann. Math. Artif. Intell. 18 (2-4), 261–293.

Wolfram, S., 2003. The Mathematica Book, 5th Edition. Wolfram Media.