Matching of Order-Sorted Terms with Regular Expression Sorts and Second-Order Variables

Temur Kutsia*
Research Institute for Symbolic Computation
Johannes Kepler University Linz, Austria

Mircea Marin†
Graduate School of Systems and Inf. Eng.
University of Tsukuba, Japan

Abstract

We construct a sound and complete matching procedure for order-sorted terms with regular expression sorts and second-order variables. Since the theory is infinitary, there are matching problems on which the procedure does not terminate. Restricting the rules of the procedure, we obtain an incomplete terminating algorithm and give some examples of its possible applications in source code searching.

1 Introduction

Order-sorted matching has been successfully used in programming and specification languages, like, e.g. in ML [3] and dialects and languages in its family, or in the languages from the OBJ family [8], just to name a few. Matching of terms in order-sorted theories with regular expression sorts (REOSM in short, for regular expression order-sorted matching) and second-order variables, discussed in this extended abstract, generalizes the first-order order-sorted matching in various ways. First, it allows regular expressions in sorts, permitting, for instance, sorted variables like $x : (\mathsf{int}.\mathsf{char})^*$ (alternating sequence of integers and characters) or sorted function symbols as $f : (\mathsf{int} + \mathsf{char})^* \to \mathsf{int}$. Regular operators can occur in the domain sort but not in the image sort. On the other hand, it also allows a special kind of sorted second-order variables that can be instantiated only by terms with a single lambda abstraction, like $\lambda x : \mathsf{s}^*.f(x,a,g(x))$. Such terms generalize the standard lambda-terms of the corresponding chape in the sense that they can be applied to an arbitrary sequence of terms (instead of a single term only): Applying $\lambda x : \mathsf{s}^*.f(x,a,g(x))$ to the sequence (y,b) gives $f(y,b,a,g(y,b))$, provided that the sort of this sequence is less or equal than s^* in the given ordering on sorts.

The set of basic sorts we consider in this paper is finite and partially ordered. We then extend this ordering to regular expressions over basic sorts, introduce an alphabet of sorted function symbols and variables, and define terms. To ensure uniqueness of sorts for terms, some reasonable syntactic restrictions are imposed on the alphabet, like monotonicity and preregularity in the sense of [2], or the property that allows only finitely many overloading sorts for function symbols.

Matching problems in such theories reveal a quite surprising behavior: Some problems may have infinitely many incomparable matchers. An example of such an equation is $X(x) \ll f(a)$ where sorts allow the domain of the second-order variable X to be the empty sequence of terms and the variable x can be replaced with the empty sequence. Assuming, for instance, that these sorts are $X : \mathsf{s}^* \to \mathsf{r}$ and $x : \mathsf{s}^*$ for some basic sorts s and r, and having $f : \mathsf{s}^* \to \mathsf{r}$, we obtain (among others) infinitely many matchers $\{X \mapsto \lambda y : \mathsf{s}^*.f(a,y), x \mapsto \varepsilon\}$, $\{X \mapsto \lambda y : \mathsf{s}^*.f(a,y,y), x \mapsto \varepsilon\}$, $\{X \mapsto \lambda y : \mathsf{s}^*.f(a,y,y), x \mapsto \varepsilon\}$, where ε stands for the empty sequence and disappears whenever it occurs in some other sequence. Each of this substitutions maps $X(x)$ to $f(a)$, but none is more general than the other. In this sense REOSM resembles matching in theories with sequence variables and flat function symbols [7]: Another example

*Supported by the European Commission Framework 6 Programme for Integrated Infrastructures Initiatives under the project SCIEnce—Symbolic Computation Infrastructure for Europe (Contract No. 026133).

†Supported by JSPS Grant-in-Aid no. 20500025 for Scientific Research (C).

of a theory with infinitary matching, which has not been artificially constructed just for showing that infinitary matching exists, but, rather, came from problems with practical applications.

Designing REOSM, we had some potential applications in mind, like, e.g., source code searching, XML transformation, or linguistic querying. It requires, of course, a finitary matching algorithm (that can not be complete, taking into account the example above). This is what we develop in this paper, but to get to that, we first construct a sound and complete solving procedure for arbitrary REOSM problems, identify fragments on which it terminates, and then modify it to obtain a generally incomplete algorithm (that is still complete for a quite large fragment).

To compare to some known problems, REOSM extends order-sorted matching given in the form of a special case of order-sorted unification [11], sequence matching [6], context matching [10], context sequence matching [5], and some special cases of full second-order matching [4].

The paper is organized as follows. In Section 2 we introduce regular expression sorts and define terms and substitutions. In Section 3 we construct a matching procedure, show its soundness, completeness, identify reasons for nontermination, terminating fragments, and identify some restrictions that render the procedure into a terminating but incomplete algorithm that is still complete for a large class of interesting matching problems. We also give the main idea behind the decidability proof. Section 4 gives some examples of source code searching queries expressed as REOSM problems. Section 5 concludes.

2 Preliminaries

Sorts. We consider a finite set $\mathscr{B}$ of basic sorts, partially ordered with the relation $\preceq$. Its elements are denoted with lower case letters in $\mathsf{sans\ serif}$ font. $\mathsf{s} \prec \mathsf{r}$ means $\mathsf{s} \preceq \mathsf{r}$ and $\mathsf{s} \neq \mathsf{r}$. Regular expression sorts (shortly, sorts) are regular expressions over $\mathscr{B}$, built in the usual way: $\mathsf{R} ::= \mathsf{s} \mid 1 \mid \mathsf{R}_1.\mathsf{R}_2 \mid \mathsf{R}_1{+}\mathsf{R}_2 \mid \mathsf{R}^*$.

The set of all regular expression sorts is denoted by $\mathscr{R}$. We use capital $\mathsf{SANS\ SERIF}$ font letters R, Q, S for them. They define the corresponding regular language in the standard way: $[\![\mathsf{s}]\!] = \{\mathsf{s}\}$, $[\![1]\!] = \{\varepsilon\}$, $[\![\mathsf{R}_1.\mathsf{R}_2]\!] = \{(\tilde{s}_1, \tilde{s}_2) \mid \tilde{s}_1 \in [\![\mathsf{R}_1]\!], \tilde{s}_2 \in [\![\mathsf{R}_2]\!]\}$, $[\![\mathsf{R}_1{+}\mathsf{R}_2]\!] = [\![\mathsf{R}_1]\!] \cup [\![\mathsf{R}_2]\!]$, $[\![\mathsf{R}^*]\!] = [\![\mathsf{R}]\!]^*$, where ε stands for the empty word.

We extend the ordering $\preceq$ to words of basic sorts of equal lengths by $\mathsf{s}_1 \cdots \mathsf{s}_n \preceq \mathsf{r}_1 \cdots \mathsf{r}_n$ iff $\mathsf{s}_i \preceq \mathsf{r}_i$ for all $1 \leq i \leq n$. This ordering is extended to sets of words of basic sorts, defining $S_1 \preceq S_2$ iff for each $w_1 \in S_1$ there is $w_2 \in S_2$ such that $w_1 \preceq w_2$. Finally, we order the regular expression sorts with $\preceq$, defining $\mathsf{R}_1 \preceq \mathsf{R}_2$ iff $[\![\mathsf{R}_1]\!] \preceq [\![\mathsf{R}_2]\!]$. If $\mathsf{R}_1 \preceq \mathsf{R}_2$ and $\mathsf{R}_2 \preceq \mathsf{R}_1$ then we write $\mathsf{R}_1 \simeq \mathsf{R}_2$. If $\mathsf{R}_1 \preceq \mathsf{R}_2$ and not $\mathsf{R}_2 \preceq \mathsf{R}_1$, then $\mathsf{R}_1 \prec \mathsf{R}_2$.

Functional sorts over regular expressions are obtained from regular expression sorts by applying the sort constructor $\to$ to a regular sort R and basic sort s. As usual, we write $\mathsf{R} \to \mathsf{s}$ instead of $\to(\mathsf{R}, \mathsf{s})$, and extend the ordering $\preceq$ to functional sorts as follows $\mathsf{R}_1 \to \mathsf{s}_1 \preceq \mathsf{R}_2 \to \mathsf{s}_2$ iff $\mathsf{R}_2 \preceq \mathsf{R}_1$ and $\mathsf{s}_1 \preceq \mathsf{s}_2$. The relations $\simeq$ and $\prec$ also extend to functional sorts. The set of functional sorts over regular expressions is denoted by $\mathscr{R}^{\to}$.

The set of all $\preceq$-maximal elements in a set of regular sorts $S \subseteq \mathscr{R}$ is denoted $\max(S)$. R is a lower bound of S if $\mathsf{R} \preceq \mathsf{Q}$ for all $\mathsf{Q} \in S$. A lower bound G of S is the greatest lower bound, denoted $\mathrm{glb}(S)$, if $\mathsf{R} \preceq \mathsf{G}$ for all lower bounds R of S. These similar notions can be formulated for functional sorts.

The sort $(\sum_{\mathsf{s} \in \mathscr{B}} \mathsf{s})^*$ is the *top regular expression sort* and is denoted by $\top$. Obviously, $\mathsf{R} \preceq \top$ for any R.

Terms. For each R we assume a countable set of first-order variables $\mathscr{V}_{\mathsf{R}}$ such that $\mathscr{V}_{\mathsf{R}_1} = \mathscr{V}_{\mathsf{R}_2}$ iff $\mathsf{R}_1 \simeq \mathsf{R}_2$ and $\mathscr{V}_{\mathsf{R}_1} \cap \mathscr{V}_{\mathsf{R}_2} = \emptyset$ if $\mathsf{R}_1 \not\simeq \mathsf{R}_2$. The set of second-order variables $\mathscr{V}_{\mathsf{R} \to \mathsf{s}}$ for each R and s is defined analogously. Also, we assume as a signature a family of sets of function symbols $\{\mathscr{F}_{\mathsf{R} \to \mathsf{s}} \mid \mathsf{R} \in \mathscr{R}, \mathsf{s} \in \mathscr{B}\}$ such that $\mathscr{F}_{\mathsf{R}_1 \to \mathsf{s}_1} = \mathscr{F}_{\mathsf{R}_2 \to \mathsf{s}_2}$ iff $\mathsf{R}_1.\mathsf{s}_1 \simeq \mathsf{R}_2.\mathsf{s}_2$. Moreover, the following conditions should be satisfied:

- Monotonicity: If $f \in \mathscr{F}_{\mathsf{R}_1\to\mathsf{s}_1} \cap \mathscr{F}_{\mathsf{R}_2\to\mathsf{s}_2}$ and $\mathsf{R}_1 \preceq \mathsf{R}_2$, then $\mathsf{s}_1 \preceq \mathsf{s}_2$.
- Preregularity: If $f \in \mathscr{F}_{\mathsf{R}_1\to\mathsf{s}_1}$ and $\mathsf{R}_2 \preceq \mathsf{R}_1$, then there is a $\preceq$-least element in the set $\{\mathsf{s} \mid f \in \mathscr{F}_{\mathsf{R}\to\mathsf{s}} \text{ and } \mathsf{R}_2 \preceq \mathsf{R}\}$.
- Finite overloading: For each f, the set $\{\mathscr{F}_{\mathsf{R}\to\mathsf{s}} \mid f \in \mathscr{F}_{\mathsf{R}\to\mathsf{s}}\}$ is finite.

We say that R is the sort of x if $x \in \mathscr{V}_{\mathsf{R}}$. Similarly, $\mathsf{R}\to\mathsf{s}$ is a sort of X (resp. of f) if $X \in \mathscr{V}_{\mathsf{R}\to\mathsf{s}}$ (resp. if $f \in \mathscr{F}_{\mathsf{R}\to\mathsf{s}}$). Function symbols from $\mathscr{F}_{1\to\mathsf{s}}$ are called constants. We use the letters a,b,c to denote them. $\mathit{maxsort}(f)$ denotes the set $\max(\{\mathsf{R}.\mathsf{s} \mid f \in \mathscr{F}_{\mathsf{R}\to\mathsf{s}}\})$. We will write $f:\mathsf{R}\to\mathsf{s}$ for $f \in \mathscr{F}_{\mathsf{R}\to\mathsf{s}}$, $a:\mathsf{s}$ for $a \in \mathscr{F}_{1\to\mathsf{s}}$, $x:\mathsf{R}$ for $x \in \mathscr{V}_{\mathsf{R}}$, and $X:\mathsf{R}\to\mathsf{s}$ for $X \in \mathscr{V}_{\mathsf{R}\to\mathsf{s}}$. Setting $\mathscr{V} \in \cup_{\mathsf{R}\in\mathscr{R}}\mathscr{V}_{\mathsf{R}} \cup \cup_{\mathsf{R}\in\mathscr{R},\mathsf{s}\in\mathscr{B}}\mathscr{V}_{\mathsf{R}\to\mathsf{s}}$ and $\mathscr{F} = \cup_{\mathsf{R}\in\mathscr{R},\mathsf{s}\in\mathscr{B}}\mathscr{F}_{\mathsf{R}\to\mathsf{s}}$, we define the set of *sorted first-order terms* (or, just *first-order terms*) $\mathscr{T}(\mathscr{F},\mathscr{V})$ over $\mathscr{F}$ and $\mathscr{V}$ as the least family of sets of first order terms $\{\mathscr{T}_{\mathsf{R}}(\mathscr{F},\mathscr{V}) \mid \mathsf{R}\in\mathscr{R}\}$ satisfying the following conditions:

- $\mathscr{V}_{\mathsf{R}} \subseteq \mathscr{T}_{\mathsf{R}}(\mathscr{F},\mathscr{V})$.
- $\mathscr{T}_{\mathsf{R}_1}(\mathscr{F},\mathscr{V}) \subseteq \mathscr{T}_{\mathsf{R}_2}(\mathscr{F},\mathscr{V})$ if $\mathsf{R}_1 \preceq \mathsf{R}_2$.
- If $f:\mathsf{R}\to\mathsf{s}$ and $1 \preceq \mathsf{R}$, then $f(\varepsilon) \in \mathscr{T}_{\mathsf{s}}(\mathscr{F},\mathscr{V})$.
- If $f:\mathsf{R}\to\mathsf{s}$, $t_i \in \mathscr{T}_{\mathsf{R}_i}(\mathscr{F},\mathscr{V})$ for $1 \le i \le n$, $n \ge 1$, such that $\mathsf{R}_1.\cdots.\mathsf{R}_n \preceq \mathsf{R}$, then $f(t_1,\ldots,t_n) \in \mathscr{T}_{\mathsf{s}}(\mathscr{F},\mathscr{V})$.
- If $X:\mathsf{R}\to\mathsf{s}$ and $1 \preceq \mathsf{R}$, then $X(\varepsilon) \in \mathscr{T}_{\mathsf{s}}(\mathscr{F},\mathscr{V})$.
- If $X:\mathsf{R}\to\mathsf{s}$, $t_i \in \mathscr{T}_{\mathsf{R}_i}(\mathscr{F},\mathscr{V})$ for $1 \le i \le n$, $n \ge 1$, such that $\mathsf{R}_1.\cdots.\mathsf{R}_n \preceq \mathsf{R}$, then $X(t_1,\ldots,t_n) \in \mathscr{T}_{\mathsf{s}}(\mathscr{F},\mathscr{V})$.

We abbreviate terms $a(\varepsilon)$ with a. The letters s,t,r will be used to denote first-order terms.

Lemma 1. *For each first-order term t there exists a $\preceq$-minimal sort R that is unique modulo $\simeq$ such that $t \in \mathscr{T}_{\mathsf{R}}(\mathscr{F},\mathscr{V})$.*

Proof. By structural induction. Let t be a variable $x \in \mathscr{V}_{\mathsf{R}}$. Assume $x \in \mathscr{T}_{\mathsf{Q}}(\mathscr{F},\mathscr{V})$ for some Q. It implies that $x \in \mathscr{V}_{\mathsf{Q}'}$ with $\mathsf{Q}' \preceq \mathsf{Q}$. Then $x \in \mathscr{V}_{\mathsf{R}} \cap \mathscr{V}_{\mathsf{Q}'}$, which implies $\mathsf{R} \simeq \mathsf{Q}'$. Hence, R is the unique $\preceq$-minimal sort modulo $\simeq$ with $x \in \mathscr{T}_{\mathsf{R}}(\mathscr{F},\mathscr{V})$.

If t is a constant, the lemma follows from the monotonicity condition.

If $t = f(t_1,\ldots,t_n)$, then, by the induction hypothesis, each t_i has the $\preceq$-minimal sort R_i, $1 \le i \le n$, that is unique modulo $\simeq$. Let $f:\mathsf{Q}\to\mathsf{q}$ such that $\mathsf{R}_1.\cdots.\mathsf{R}_n \preceq \mathsf{Q}$. By preregularity, there exists a $\preceq$-least s such that $f:\mathsf{S}\to\mathsf{s}$ and $\mathsf{R}_1.\cdots.\mathsf{R}_n \preceq \mathsf{S}$. Hence, s is the $\preceq$-minimal sort such that $t \in \mathscr{T}_{\mathsf{s}}(\mathscr{F},\mathscr{V})$.

If $t = X(t_1,\ldots,t_n)$, then, by the induction hypothesis, each t_i has the $\preceq$-minimal sort R_i, $1 \le i \le n$, that is unique modulo $\simeq$. Let $X:\mathsf{R}\to\mathsf{s}$ such that $\mathsf{R}_1.\cdots.\mathsf{R}_n \preceq \mathsf{R}$. Since $\mathscr{V}_{\mathsf{R}\to\mathsf{s}}$ is the only set of variables to which X belongs to, such an s is unique modulo $\preceq$.

□

This $\preceq$-minimal sort R is called *the sort* of t and is denoted by $\mathit{sort}(t)$.

In our matching problems only first-order terms appear, but we need second-order terms to express solutions. Also, sequences of first-order terms can appear in the solutions. The sequences also make the problem expression more succinct, therefore we will allow them in the problems. A second order term of sort $\mathsf{R}\to\mathsf{s}$ is either a variable $X:\mathsf{R}\to\mathsf{s}$ or a lambda-term $\lambda x:\mathsf{R}.t$, where $x \in \mathscr{V}_{\mathsf{R}}$ and t is a first-order term with $\mathit{sort}(t) = \mathsf{s}$. The occurrences of x in t are bound and any other variable occurrence is free. It is

easy to see that the sort of each second-order term is unique modulo $\simeq$. Like in the first-order case, we denote it with *sort*. The letter T will be used to denote second-order terms.

The sort of a term sequence $\tilde{t} = (t_1, \ldots, t_n)$, $n \geq 1$, is defined uniquely modulo $\simeq$ as $sort(\tilde{t}) := sort(t_1).\cdots.sort(t_n)$. The length of $\tilde{t}$ is $\mathrm{len}(\tilde{t}) = n$. When $n = 0$ then $\tilde{t} = \varepsilon$, $sort(\varepsilon) = 1$, and $\mathrm{len}(\tilde{t}) = 0$.

The set of variables of a first-order term t is denoted by $var(t)$. For a second-order term T, $var(T)$ denotes the set of free variables. A (first- or second-order) term t is ground if $var(t) = \emptyset$. These notions extend to term sequences, sets of term sequences, etc.

Later we will need the application operation $T[\tilde{t}]$ that applies a second-order term $T := \lambda x : \mathsf{R}.t$ or $T := X \in \mathscr{V}_{\mathsf{R}\to\mathsf{s}}$ to a term sequence $\tilde{t}$ with $sort(\tilde{t}) \preceq \mathsf{R}$. This operation is defined as expected: i) If T is a variable $X \in \mathscr{V}_{\mathsf{R}\to\mathsf{s}}$, then $T[\tilde{t}]$ is the first-order term $X(t_1, \ldots, t_n)$ of sort s; and (ii) If $T = \lambda x : R.t$, then $T[\tilde{t}]$ is the first-order term obtained from t by replacing simultaneously all occurrences of x in t with the sequence of terms $t_1, \ldots, t_n$.

Substitutions. A substitution is a well-sorted mapping from first-order variables to sequences of first-order terms and from second-order variables to second-order terms, which is identity almost everywhere, i.e., all but finitely many first-order variables are mapped to themselves and all but finitely many second-order variables X are mapped to $\lambda x.X(x)$. Substitutions are denoted with lower case Greek letters, where ε stands for the identity substitution. Well-sortedness of σ means that $sort(\sigma(v)) \preceq sort(v)$ for all first- or second-order variable v. Application of a substitution σ on a term and on a term sequence is defined as follows: $v\sigma = \sigma(v)$, $f(\tilde{t})\sigma = f(\tilde{t}\sigma)$, $X(\tilde{t})\sigma = \sigma(X)[\tilde{t}\sigma]$, $(\lambda x : \mathsf{R}.t)\sigma = \lambda x' : \mathsf{R}.((t\{x \mapsto x'\})\sigma)$ where $x' \in \mathscr{V}_{\mathsf{R}}$ is fresh, $(t_1, \ldots, t_n)\sigma = (t_1\sigma, \ldots, t_n\sigma)$.

The notions of term and term sequence instances, substitution composition, restriction, and subsumption are defined in the standard way. We use juxtaposition for composition, and write $\sigma \leq_{\mathscr{X}} \vartheta$ for subsumption meaning that σ is more general than ϑ on the set of variables $\mathscr{X}$.

Lemma 2. *For a first-order term t, a first-order term sequence $\tilde{t}$, and a substitution σ we have $sort(t\sigma) \preceq sort(t)$ and $sort(\tilde{t}\sigma) \preceq sort(\tilde{t})$.*

Proof. We prove $sort(t\sigma) \preceq sort(t)$ by induction on term structure. If t is a variable, then the lemma follows from the definition of substitution. If $t = f(\varepsilon)$ then it is obvious. If $t = f(t_1, \ldots, t_n)$, $n \geq 1$, then by the induction hypothesis $sort(t_i\sigma) \preceq sort(t_i)$ for each $1 \leq i \leq n$. Therefore, $sort((t_1, \ldots, t_n)\sigma) \preceq sort(t_1, \ldots, t_n)$ which, by the definition of sorted terms, implies that $sort(t\sigma) = sort(f(t_1, \ldots, t_n)\sigma) \preceq sort(f(t_1, \ldots, t_n)) = sort(t)$. For $t = X(t_1, \ldots, t_n)$ we can use a similar reasoning.

To prove $sort(\tilde{t}\sigma) \preceq sort(\tilde{t})$, we use induction on the length of $\tilde{t}$. If the length is 0, then the lemma is obvious. Otherwise, let $\tilde{t}$ be $(t, \tilde{t}')$. By the case above we have $sort(t\sigma) \preceq sort(t)$. By the induction hypothesis we get $sort(\tilde{t}'\sigma) \preceq sort(\tilde{t}')$. Therefore, $sort(\tilde{t}\sigma) = sort((t, \tilde{t}')\sigma) \preceq sort((t, \tilde{t}')) = sort(\tilde{t})$. □

Lemma 3. *For a second-order term T and a substitution σ we have $sort(T\sigma) \preceq sort(T)$.*

Proof. If $T = X$, then the lemma follows from the definition of substitution. If $T = \lambda x.t$, then $T = \lambda x'.(t\{x \mapsto x'\})\sigma$ where x' is fresh with $sort(x') = sort(x)$. By Lemma 2, sort $sort((t\{x \mapsto x'\})\sigma) \preceq sort(t\{x \mapsto x'\}) = sort(t)$. Hence, $sort(T\sigma) \preceq sort(T)$. □

We also consider a *closure* $\overline{\mathsf{R}}$ of a sort R, defined as follows: $\overline{\mathsf{s}} = \sum_{\mathsf{r} \preceq \mathsf{s}} \mathsf{r}$, $\overline{1} = 1$, $\overline{\mathsf{R}_1.\mathsf{R}_2} = \overline{\mathsf{R}}_1.\overline{\mathsf{R}}_2$, $\overline{\mathsf{R}_1 + \mathsf{R}_2} = \overline{\mathsf{R}}_1 + \overline{\mathsf{R}}_2$, $\overline{\mathsf{R}^*} = \overline{\mathsf{R}}^*$. Closure makes operations on sorts considered later easier.

Lemma 4. *Let $\mathsf{S}, \mathsf{R} \in \mathscr{R}$. Then $\mathsf{S} \preceq \mathsf{R}$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$.*

Proof. For any $\mathsf{Q} \in \mathscr{R}$ and $v \in [\![\mathsf{Q}]\!]$, let $v{\downarrow}\mathsf{Q} := \{w \in [\![\mathsf{Q}]\!] \mid w \preceq v\}$. By the definition of closure, we have that $\{\max(v{\downarrow}\overline{\mathsf{Q}}) \mid v \in [\![\overline{\mathsf{Q}}]\!]\} = [\![\mathsf{Q}]\!]$. Now, we can reason as follows: $\mathsf{S} \preceq \mathsf{R}$ iff $[\![\mathsf{S}]\!] \preceq [\![\mathsf{R}]\!]$ iff $\{\max(v{\downarrow}\overline{\mathsf{S}}) \mid v \in [\![\overline{\mathsf{S}}]\!]\} \preceq \{\max(v{\downarrow}\overline{\mathsf{R}}) \mid v \in [\![\overline{\mathsf{R}}]\!]\}$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$. □

Corollary 5. *Let* $\mathsf{S},\mathsf{R} \in \mathscr{R}$. *Then* $\mathsf{S} \simeq \mathsf{R}$ *iff* $[\![\overline{\mathsf{S}}]\!] = [\![\overline{\mathsf{R}}]\!]$.

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

Deciding $\preceq$. If we did not have ordering on basic sorts, $\preceq$ would be the standard inequality for regular word expressions which can be decided, for instance, by Antimirov's algorithm [1] that employs partial derivatives. The problem is PSPACE-complete, but this rewriting approach has an advantage over the standard technique of translating regular expressions into automata: With it, in some cases solving derivations can have polynomial size, while any algorithm based on translation of regular expressions into DFA's causes an exponential blow-up.

In our case, we can rely on the property $\mathsf{S} \preceq \mathsf{R}$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$, proved in Lemma 4. To decide the later inclusion, we do not need to take into account ordering on basic sorts. Hence, it can be decided by the original Antimirov's algorithm on $\overline{\mathsf{S}}$ and $\overline{\mathsf{R}}$.

3 Matching Procedure

Matching equations may have infinitely many solutions. An example of such a problem is $\{X(x) \ll f(a)\}$ with $X : \mathsf{s}^* \to \mathsf{r}$, $x : \mathsf{s}^*$, $f : \mathsf{s}^* \to \mathsf{r}$, $a : \mathsf{s}$. Then among the solutions there are the substitutions $\{X \mapsto \lambda y : \mathsf{s}^*.\, f(a,y), x \mapsto \varepsilon\}$, $\{X \mapsto \lambda y : \mathsf{s}^*.\, f(a,y,y), x \mapsto \varepsilon\}$, $\{X \mapsto \lambda y : \mathsf{s}^*.\, f(a,y,y), x \mapsto \varepsilon\}$, etc. Hence, any complete method to solve matching equations would be nonterminating. In this paper we present an incomplete terminating algorithm that does not generate the solutions of the type we showed above. Its rules are given below. They transform a pair of a matching problem and a substitution into such a pair again.

T: **Trivial**

$\{\tilde{t} \ll \tilde{t}\} \cup \Gamma; \sigma \Longrightarrow_\varepsilon \Gamma; \sigma$.

TP: **Trivial Prefix**

$\{(\tilde{r},\tilde{t}) \ll (\tilde{r},\tilde{s})\} \cup \Gamma; \sigma \Longrightarrow_\varepsilon \{\tilde{t} \ll \tilde{s}\} \cup \Gamma; \sigma$, if $\tilde{r} \neq \varepsilon$ and $\tilde{t} \neq \tilde{s}$.

D: **Decomposition**

$\{(f(\tilde{t}),\tilde{t}') \ll (f(\tilde{s}),\tilde{s}')\} \cup \Gamma; \sigma \Longrightarrow_\varepsilon \{\tilde{t} \ll \tilde{s}, \tilde{t}' \ll \tilde{s}'\} \cup \Gamma; \sigma$, if $\tilde{t} \neq \tilde{s}$.

VE: **Variable Elimination**

$\{(x,\tilde{t}) \ll (\tilde{s},\tilde{r})\} \cup \Gamma; \sigma \Longrightarrow_\vartheta \{\tilde{t} \ll \tilde{r}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta$, where $sort(\tilde{s}) \preceq sort(x)$, and $\vartheta = \{x \mapsto \tilde{s}\}$.

P: **Projection**

$\{X(t) \ll s\} \cup \Gamma; \sigma \Longrightarrow_\vartheta \{t \ll s\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta$,

where $X : \mathsf{R} \to \mathsf{s}$ with $sort(t) \preceq \mathsf{R} \preceq \mathsf{s}$ and $\vartheta = \{X \mapsto \lambda x : \mathsf{s}.x\}$.

I: **Imitation**

$\{X(\tilde{t}) \ll f(\tilde{s}_1, \ldots, \tilde{s}_m)\} \cup \Gamma; \sigma \Longrightarrow_\theta \{\tilde{u}_1 \ll \tilde{s}_1, \ldots, \tilde{u}_m \ll \tilde{s}_m\} \cup \Gamma\vartheta; \sigma\vartheta$,

where $X \in \mathscr{V}_{\mathsf{R}\to\mathsf{s}}$ and there exist $J \subseteq \{1,\ldots,m\}$ and $y \in \mathscr{V}_{\mathsf{R}}$ fresh, such that (1) all $\tilde{s}_j$ with $j \in J$ are the same, which is of sort $\preceq sort(\tilde{t})$, (2) all $\tilde{s}_i$ with $i \in \{1,\ldots,m\} \setminus J$ are single terms, (3) $r_j = y$ for all $j \in J$ and $r_i = Y_i(y)$ with $Y_i \in \mathscr{V}_{\mathsf{R}\to sort(\tilde{s}_i)}$ fresh for all $i \in \{1,\ldots,m\} \setminus J$, (4) $\vartheta = \{X \mapsto \lambda y : \mathsf{R}.\, f(r_1,\ldots,r_m)\}$, and (5) $\tilde{u}_i = r_i\{y \mapsto \tilde{t}\vartheta\}$ for $1 \le i \le m$.

We denote this set of transformation rules with $\mathfrak{T}$. To solve a given matching problem Γ, we create an initial pair $\Gamma;\varepsilon$ and apply the rules on it exhaustively. A pair is transformed into $\bot$ (indicating failure) if no rule can be applied to it.

3.1 Properties of Transformation System $\mathfrak{T}$

We define the size $|\vartheta|$ of a substitution ϑ as the multiset $\{|x\vartheta| \mid x \in dom(\vartheta)\}$, where $|x\vartheta|$ denotes the number of symbols that occur in $x\vartheta$. Also, we define the size $|\Gamma|$ of a matching problem Γ as the multiset $\{|\tilde{t}_i| \mid \tilde{s}_i \ll \tilde{t}_i \in \Gamma\}$. By $sum(\vartheta)$ and $sum(\Gamma)$ we denote the sums $\sum_{i\in|\vartheta|} i$ and $\sum_{i\in|\Gamma|} i$, respectively. The system of transformation rules $\mathfrak{T}$ has the following important properties:

Local soundness: If $\Gamma;\sigma \Longrightarrow_\vartheta \Gamma';\sigma\vartheta$ and μ' is a matcher of Γ' then $\vartheta\mu'$ is a matcher of Γ.

Local completeness: If μ is a matcher of $\Gamma \neq \emptyset$ then for any substitution σ there exist $\Gamma;\sigma \Longrightarrow_\vartheta \Gamma';\sigma'$ and a matcher μ' of Γ' such that (1) $\mu =_{var(\Gamma)} \vartheta\mu'$ and (2) $(|\mu'|,|\Gamma'|) <_l (|\mu|,|\Gamma|)$ where $<_l$ is the lexicographic combination of two natural orderings on multisets of natural numbers.

Note that the substitutions μ and μ' are ground.

Soundness is easy to prove by case analysis on the transformation rules of $\mathfrak{T}$, whereas completeness can be proved by case analysis on the structure of μ and the syntactic structure of a matching equation of Γ.

An immediate consequence of local soundness is the following.

Theorem 6 (Soundness)**.** *If $\Gamma;\varepsilon \Longrightarrow^* \emptyset;\mu$ is an $\mathfrak{T}$-derivation then μ is a matcher of Γ.*

The following lemma follows from local completeness and the obvious observation that $>_l$ is a well-founded order.

Theorem 7 (Completeness)**.** *For any matcher μ of a matching problem Γ there exists an $\mathfrak{T}$-derivation $\Gamma;\varepsilon \Longrightarrow^* \emptyset;\mu'$ with $\mu' =_{dom(\mu')} \mu$.*

An interesting property of $\mathfrak{T}$ is that each derivation it generates has a finite length. It follows from the lemma below, if we define a complexity measure of a matching problem Γ as a pair ($|\Gamma|$, the number of distinct variables in Γ) and compare the measures lexicographically. The obtained ordering is well-founded.

Lemma 8. *Each rule in $\mathfrak{T}$ strictly reduces the complexity measure of a matching problem.*

Proof. The rules T, TP, D, and I reduce the first component. The rules VE and P reduce the second component without increasing the first one. □

This result implies that if we build the derivation tree in a depth-first way, we obtain a sound and complete matching procedure. The procedure may not terminate because, as we noticed at the beginning of this section, matching problems may have infinitely many matchers. Non-termination comes from the fact that the imitation rule is infinitely branching. The other rules are terminating, which implies the theorem:

Theorem 9. *For first-order REOSM problems $\mathfrak{T}$ is sound, complete, and terminating.*

It can be shown that if a matching problem Γ is solvable, then is has a matcher ϑ with the property $sum(\vartheta|_{var(\Gamma)}) \leq 2^{sum(\Gamma)}$, where $\vartheta|_{var(\Gamma)}$ stands for the restriction of ϑ on $var(\Gamma)$. Hence, if the application of the imitation rule on an equation $X(\tilde{t}) \ll f(\tilde{s}_1,\ldots,\tilde{s}_m)$ is restricted so that m never exceeds $2^{sum(\Gamma)}$ for the Γ that is being solved, then the procedure will terminate and return at least one matcher, if Γ is solvable. It implies the theorem:

Theorem 10. *REOSM with second-order variables is decidable.*

In the following section we identify a sufficient condition on matching problems that turns our matching procedure into an algorithm, which is incomplete in general, but remains complete on a certain fragment of matching problems with second-order variables and for all first-order problems.

3.2 A Sufficient Condition for Termination

It is not hard to see that the existence of infinitely many matchers for the problem $\Gamma = \{X(x) \ll f(a)\}$ mentioned before can be overcome if we disallow the usage in Γ of variables $x \in \mathcal{V}_{\mathsf{R}}$ or $X \in \mathcal{V}_{\mathsf{R}\to\mathsf{s}}$ with $1 \preceq \mathsf{R}$.

Based on this observation, we say that a matching problem Γ is *non-erasing* if it satisfies the conditions just mentioned above.

It is easy to see that if Γ is non-erasing and $\Gamma;\sigma \Longrightarrow_\vartheta \Gamma';\sigma\vartheta$ is a transformation step with T, TP, D, VE, or P, then Γ' is non-erasing too.

Now, suppose we are in the position to apply rule I to a non-erasing matching problem $\{X(\tilde{t}) \ll f(\tilde{s})\} \cup \Gamma$ where $X \in \mathcal{V}_{\mathsf{R}\to\mathsf{s}}$. Suppose μ is an arbitrary matcher of $\{X(\tilde{t}) \ll f(\tilde{s})\} \cup \Gamma$. Then $\mu(X) = \lambda y : \mathsf{R}.\, f(a_1,\ldots,a_m)$ where every a_i is either y or a term in $\mathcal{T}(\mathcal{F},\{y\})$. Note that $m \geq 1$ because otherwise $1 \preceq R$. Since $f(\tilde{s}) = X(\tilde{t})\mu = f(a_1\{y \mapsto \tilde{t}\}\mu,\ldots,a_m\{y \mapsto \tilde{t}\}\mu) = f(\tilde{s})$, we conclude that $(a_1\{y \mapsto \tilde{t}\}\mu,\ldots,a_m\{y \mapsto \tilde{t}\}\mu) = \tilde{s}$. But $a_i\{y \mapsto \tilde{t}\}\mu \neq \varepsilon$ because they are either terms (if $a_i = Y_i(y)$) or nonempty ground sequences of terms (if $a_i = y \in \mathcal{V}_{\mathsf{R}}$ with $1 \not\preceq \mathsf{R}$). This shows that $m \leq n$ where $n = \mathrm{len}(\tilde{s})$.

We conclude that the system $\mathfrak{T}'$ obtained from $\mathfrak{T}$ by replacing rule I with the rule BI below is sound, complete, and terminating for non-erasing problems.

BI: **Bounded Imitation**

$$\{X(\tilde{t}) \ll f(\tilde{s}_1,\ldots,\tilde{s}_m)\} \cup \Gamma;\sigma \Longrightarrow_\vartheta \{\tilde{u}_1 \ll \tilde{s}_1,\ldots,\tilde{u}_m \ll \tilde{s}_m\} \cup \Gamma\vartheta;\sigma\vartheta,$$

where $\tilde{s}_i \neq \varepsilon$ for all $1 \leq i \leq m$, $X \in \mathcal{V}_{\mathsf{R}\to\mathsf{s}}$, and there exist $J \subseteq \{1,\ldots,m\}$ and $y \in \mathcal{V}_{\mathsf{R}}$ fresh, such that (1) all $\tilde{s}_j$ with $j \in J$ are the same, which is of sort $\preceq sort(\tilde{t})$, (2) all $\tilde{s}_i$ with $i \in \{1,\ldots,m\} \setminus J$ are single terms, (3) $r_j = y$ for all $j \in J$ and $r_i = Y_i(y)$ with $Y_i \in \mathcal{V}_{\mathsf{R}\to sort(\tilde{s}_i)}$ fresh for all $i \in \{1,\ldots,m\} \setminus J$, (4) $\vartheta = \{X \mapsto \lambda y : \mathsf{R}.\, f(r_1,\ldots,r_m)\}$, and (5) $\tilde{u}_i = r_i\{y \mapsto \tilde{t}\vartheta\}$ for $1 \leq i \leq m$.

Termination of $\mathfrak{T}'$ follows from the fact that the search space for matchers of $\mathfrak{T}'$ is a finitely branching tree with branches of finite length, therefore it is finite.

4 Applications

For applications, we need some more flexibility that can be achieved by marking some second-order variables as function variables and by introducing anonymous variables. These variables do not add expressive power to matching. The idea behind function variables is that in substitutions they can be instantiated only by terms of the form $\lambda x : \mathsf{R}.\, f(x)$ or $\lambda x : \mathsf{R}.F(x)$ where F is again a function variable. Function variables can not be projected. The imitation rule is easy:

IFV: **Imitation for Function Variables**

$$\{F(\tilde{t}) \ll f(\tilde{s})\} \cup \Gamma;\sigma \Longrightarrow_\theta \{\tilde{t}\vartheta \ll \tilde{s}\} \cup \Gamma\vartheta;\sigma\vartheta,$$

where $F : \mathsf{R} \to \mathsf{s}$ is a function variable, $sort(f) \preceq sort(F)$ and $\vartheta = \{F \mapsto \lambda x : \mathsf{R}.\, f(x)\}$.

We allow anonymous (first-order and second-order) variables for each sort. In matching problems they behave as singleton variables of the corresponding sort. The difference is that we are not interested

in their instantiation. We use the underscore _ for anonymous variables, usually decorated with the sort. We omit the sort if it is $\top$.

A potential application of REOSM can be source code searching [9] used in, e.g., software reengineering. Assuming that a source code can be quite naturally represented as a sequence of parse trees for declarations, assignments, conditionals, loops, function calls, etc. formulated in an order-sorted language with regular expression sorts, one can formulate the search queries in the form of REOSM matching problems. For instance, to extract from a code the assignments of the form $x := x + y$ to integer variables, we write a matching problem $\{(\_, (\_ : \mathsf{asgnmt} \to \mathsf{stmt})(:= (x, +(x,y))), \_) \ll \tilde{t}\}$, where asgnmt and stmt are sorts for assignments and statements, respectively, with $\mathsf{asgnmt} \prec \mathsf{stmt}$, $\tilde{t}$ is the sequence of parse trees that corresponds the program code, $x : \mathsf{int}$, $y : \mathsf{int}$, and the second order anonymous variable $\_ : \mathsf{asgnmt} \to \mathsf{stmt}$ allows to search for the occurrences of $x := x + y$ not only on the top level but also inside blocks. Sorts of the assignment operator and of $=$ are $:=: \mathsf{int.int} \to \mathsf{asgnmt}$ and $+ : \mathsf{int.int} \to \mathsf{int}$. If $:=$ is overloaded with $:=: \mathsf{float.float} \to \mathsf{asgnmt}$ and $+$ is overloaded with $+ : \mathsf{float.float} \to \mathsf{float}$, where $\mathsf{int} \prec \mathsf{float}$, then by choosing $x : \mathsf{float}$ the same matching problem extracts assignments for float variables.

To find all invocations of a function that returns an integer value, we can write

$$\{(\_, (\_ : \mathsf{int} \to \mathsf{stmt})(F(\_)), \_) \ll \tilde{t}\}$$

where the function variable F has a sort $F : \top \to \mathsf{int}$.

Nested loops can be extracted from $\tilde{t}$ by solving the matching problem

$$\{(\_, (\_ : \mathsf{loop} \to \mathsf{stmt})(F(x, X(y), z)), \_) \ll \tilde{t}\}$$

where loop is the sort designated by us for loop statements, $\mathsf{loop} \prec \mathsf{stmt}$, $F \in \mathscr{V}_{\top \to \mathsf{loop}}$ is marked as a function variable, $X \in \mathscr{V}_{\mathsf{loop} \to \mathsf{stmt}}$, the variables $x, z \in$ have sort $\top$, and y has sort loop. The well sortedness of the left hand side of the matching equation ensures that F matches a loop constructor, and y matches a loop statement. This last example also illustrates the benefit of using marked function variables, such as F.

5 Conclusion

Order-sorted matching for terms with regular expression sorts and second-order variables (REOSM) generalizes, on the one hand, order-sorted matching for first-order terms (by adding regular expression sorts and second-order variables), context matching [10], sequence matching [6], and context sequence matching [5] (by adding order-sorted regular expression sorts). We showed that this extension leads to infinitary matching problem, designed a sound and complete procedure to enumerate its solutions, and presented the main idea for proving decidability of REOSM. We analyzed syntactic properties of terms that guarantee termination and indicated the fragments of REOSM on which the procedure terminates. These fragments cover a pretty large class of problems. Moreover, we turned the procedure into an incomplete algorithm by restricting the imitation rule. At the end, we showed some examples to illustrate potential application of REOSM in software engineering.

References

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

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

[3] R. Harper. Standard ML. http://www.cs.cmu.edu/ rwh/smlbook/online.pdf, 2009. Working draft.

[4] G. P. Huet and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Inf.*, 11:31–55, 1978.

[5] T. Kutsia. Context sequence matching for XML. *Electronic Notes on Theoretical Computer Science*, 157(2):47–65, 2006.

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

[7] T. Kutsia. Flat matching. *J. Symbolic Computation*, 43(13):58–873, 2008.

[8] The OBJ Family. http://www-cse.ucsd.edu/ goguen/sys/obj.html.

[9] S. Paul and A. Prakash. A framework for source code search using program patterns. *IEEE Trans. Software Eng.*, 20(6):463–475, 1994.

[10] M. Schmidt-Schauß and J. Stuber. The complexity of linear and stratified context matching problems. *Theory of Computing Syst.*, 37(6):717–740, 2004.

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