Matching with Regular Constraints

Temur Kutsia[1]* and Mircea Marin[2]**

[1] Research Institute for Symbolic Computation
Johannes Kepler University, A-4040 Linz, Austria
tkutsia@risc.uni-linz.ac.at
[2] Graduate School of Systems and Information Engineering
University of Tsukuba, Tsukuba 305-8573, Japan
mmarin@cs.tsukuba.ac.jp

Abstract. We describe a sound, terminating, and complete matching algorithm for terms built over flexible arity function symbols and context, function, sequence, and individual variables. Context and sequence variables allow matching to move in term trees to arbitrary depth and breadth, respectively. The values of variables can be constrained by regular expressions which are not necessarily linear. We describe heuristics for optimization, and discuss applications.

1 Introduction

We describe an algorithm to solve matching problems for terms built over flexible arity function symbols and context, function, sequence, and individual variables. Context and sequence variables can be constrained by regular expressions. These four kinds of variables, together with regular constraints, make the term tree traversal and subterm extraction process very flexible: The algorithm can explore terms in a uniform way in vertical (via function and context variables) and in horizontal (via individual and sequence variables) directions.

Context variables may be instantiated with a context—a term with a hole, while function variables match a single function symbol. Hence, context variables support "vertical movement" in the tree in arbitrary depth, and function variables do the same in one depth level only. Sequence and individual variables can be seen as the "horizontal counterparts" for context and function variables: Sequence variables match arbitrarily long sequences of terms, and individual variables match only a single term.

Sequence variables can be constrained by regular expressions over terms. The values of constrained variables are required to be elements of the corresponding regular word language. Context variables are constrained by regular expressions over contexts. The values of constrained context variables should be elements of

* Supported by the Austrian Science Foundation (FWF) under the Project SFB F1302 and F1322.

** Supported by the JSPS Grant-in-Aid no. 17700025 for Scientific Research sponsored by the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT).

the corresponding regular tree language (it extends the result from [29] where context variables have been restricted by regular expressions over function symbols). Moreover, regular expressions are not limited to be linear. This gives a powerful data extraction mechanism. On the other hand, we do not allow recursion in constraints. The algorithm with regular constraints is sound, terminating, and complete. We show how to optimize the algorithm by early failure detection and branching reduction heuristics, and discuss possible applications.

The paper is organized as follows: Preliminary notions are introduced in Section 2. In Section 3 we describe the CSM algorithm and its optimizations. CSM with regular expressions is addressed in Section 4. Applications are discussed in Section 5. Related work is reviewed in Section 6. Section 7 concludes.

Due to space limitations, proofs are given in a technical report [30].

2 Preliminaries

We assume the following mutually disjoint sets of symbols fixed: individual variables $\mathcal{V}_{\mathrm{Ind}}$, sequence variables $\mathcal{V}_{\mathrm{Seq}}$, function variables $\mathcal{V}_{\mathrm{Fun}}$, context variables $\mathcal{V}_{\mathrm{Con}}$, and function symbols $\mathcal{F}$. The sets $\mathcal{V}_{\mathrm{Ind}}$, $\mathcal{V}_{\mathrm{Seq}}$, $\mathcal{V}_{\mathrm{Fun}}$, and $\mathcal{V}_{\mathrm{Con}}$ are countable. The set $\mathcal{F}$ is finite or countably infinite. All the symbols in $\mathcal{F}$ except a distinguished constant $\circ$ (called a *hole*) have flexible arity. We will use x, y, z for individual variables, $\overline{x}, \overline{y}, \overline{z}$ for sequence variables, F, G, H for function variables, $\overline{C}, \overline{D}, \overline{E}$ for context variables, and a, b, c, f, g, h for function symbols. We may use these meta-variables with indices as well.

Terms are constructed using the following grammar:

$$t ::= x \mid \overline{x} \mid \circ \mid f(t_1, \ldots, t_n) \mid F(t_1, \ldots, t_n) \mid \overline{C}(t).$$

In $\overline{C}(t)$ the term t can not be a sequence variable. We will write a for the term $a()$ where $a \in \mathcal{F}$. The meta-variables s, t, r, maybe with indices, will be used for terms. A function symbol f is called the *head* of $f(t_1, \ldots, f_n)$. A *ground* term is a term without variables. A *context* is a term with a single occurrence of the hole constant $\circ$. To emphasize that a term t is a context we will write $t[\circ]$. A context $t[\circ]$ may be applied to a term s that is not a sequence variable, written $t[s]$, and the result is the term consisting of t with $\circ$ replaced by s. We will use C and D, with or without indices, for contexts.

A *substitution* is a mapping from individual variables to those terms which are not sequence variables and contain no holes, from sequence variables to finite, possibly empty sequences of terms without holes, from function variables to function variables and symbols, and from context variables to contexts, such that all but finitely many individual and function variables are mapped to themselves, all but finitely many sequence variables are mapped to themselves considered as singleton sequences, and all but finitely many context variables are mapped to themselves applied to the hole. For example, the mapping $\{x \mapsto f(a, \overline{y}), \overline{x} \mapsto \ulcorner\urcorner, \overline{y} \mapsto \ulcorner a, \overline{C}(f(b)), x\urcorner, F \mapsto g, \overline{C} \mapsto g(\circ)\}$ is a substitution.[3] We will use lower

[3] To improve readability we write sequences between the symbols $\ulcorner$ and $\urcorner$.

case Greek letters $\sigma, \vartheta, \varphi$, and ε for substitutions, where ε denotes the empty substitution. As usual, indices may be used with the meta-variables.

Substitutions are extended to terms: $v\sigma = \sigma(v)$ for $v \in \mathcal{V}_{\text{Ind}} \cup \mathcal{V}_{\text{Seq}}$, $\overline{C}(t)\sigma = \sigma(\overline{C})[t\sigma]$, $F(t_1, \ldots, t_n)\sigma = \sigma(F)(t_1\sigma, \ldots, t_n\sigma)$, $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.

A substitution σ is *more general* than ϑ, denoted $\sigma \leq \vartheta$, if there exists a φ such that $\sigma\varphi = \vartheta$. A substitution σ is *more general* than ϑ *on a set of variables* $\mathcal{V}$, denoted $\sigma \leq^{\mathcal{V}} \vartheta$, if there exists a φ such that $v\sigma\varphi = v\vartheta$ for all $v \in \mathcal{V}$. A Csm *problem* is a finite multiset of term pairs (Csm *equations*), written $\{s_1 \ll t_1, \ldots, s_n \ll t_n\}$, where the s's and the t's contain no holes, the s's are not sequence variables, and the t's are ground. We will also call the s's the *query* and the t's the *data*. Substitutions are extended to Csm equations and problems in the usual way. A substitution σ is called a *matcher* of the Csm problem $\{s_1 \ll t_1, \ldots, s_n \ll t_n\}$ if $s_i\sigma = t_i$ for all $1 \leq i \leq n$. We will use Γ and Δ to denote Csm problems. A *complete set of matchers* of a Csm problem Γ is a set of substitutions S such that (i) each element of S is a matcher of Γ, and (ii) for each matcher ϑ of Γ there exist a substitution $\sigma \in S$ such that $\sigma \leq \vartheta$. The set S is a *minimal complete set of matchers* of Γ if it is a complete set and two distinct elements of S are incomparable with respect to $\leq$.

Example 1. The minimal complete set of matchers for the context sequence matching problem $\{\overline{C}(f(\overline{x})) \ll g(f(a,b), h(f(a), f))\}$ consists of three elements: $\{\overline{C} \mapsto g(\circ, h(f(a), f)),\ \overline{x} \mapsto \ulcorner a, b \urcorner\}$, $\{\overline{C} \mapsto g(f(a,b), h(\circ, f)),\ \overline{x} \mapsto a\}$, and $\{\overline{C} \mapsto g(f(a,b), h(f(a), \circ)),\ \overline{x} \mapsto \ulcorner\urcorner\}$.

3 Matching Algorithm

We now present inference rules for deriving solutions for Csm problems. A *system* is either the symbol $\bot$ (failure) or a pair $\Gamma; \sigma$, where Γ is a Csm problem and σ is a substitution. The inference system $\mathfrak{I}$ consists of the transformation rules listed below. The indices n and m are non-negative unless otherwise stated.

T: **Trivial**

$$\{t \ll t\} \cup \Gamma;\ \sigma \Longrightarrow \Gamma;\ \sigma.$$

IVE: **Individual Variable Elimination**

$$\{x \ll t\} \cup \Gamma;\ \sigma \Longrightarrow \Gamma\vartheta;\ \sigma\vartheta, \qquad \text{where } \vartheta = \{x \mapsto t\}.$$

FVE: **Function Variable Elimination**

$$\{F(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)\} \cup \Gamma;\ \sigma \\ \Longrightarrow \{f(s_1\vartheta, \ldots, s_n\vartheta) \ll f(t_1, \ldots, t_m)\} \cup \Gamma\vartheta;\ \sigma\vartheta, \quad \text{where } \vartheta = \{F \mapsto f\}.$$

PD: **Partial Decomposition**

$$\{f(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)\} \cup \Gamma;\ \sigma \\ \Longrightarrow \{s_1 \ll t_1, \ldots, s_{k-1} \ll t_{k-1}, f(s_k, \ldots, s_n) \ll f(t_k, \ldots, t_m)\} \cup \Gamma;\ \sigma,$$

if $f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_m)$, $s_k \in \mathcal{V}_{\text{Seq}}$ for some $1 < k \leq \min(n, m)+1$, and $s_i \notin \mathcal{V}_{\text{Seq}}$ for all $1 \leq i < k$.

TD: **Total Decomposition**

$\{f(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_n)\} \cup \Gamma;\ \sigma \Longrightarrow \{s_1 \ll t_1, \ldots, s_n \ll t_n\} \cup \Gamma;\ \sigma$,
if $f(s_1, \ldots, s_n) \neq f(t_1, \ldots, t_n)$ and $s_i \notin \mathcal{V}_{\mathrm{Seq}}$ for all $1 \leq i \leq n$.

SVD: **Sequence Variable Deletion**

$\{f(\overline{x}, s_1, \ldots, s_n) \ll t\} \cup \Gamma;\ \sigma \Longrightarrow \{f(s_1\vartheta, \ldots, s_n\vartheta) \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta$,
where $\vartheta = \{\overline{x} \mapsto \ulcorner\urcorner\}$.

W: **Widening**

$\{f(\overline{x}, s_1, \ldots, s_n) \ll f(t, t_1, \ldots, t_m)\} \cup \Gamma;\ \sigma$
$\Longrightarrow \{f(\overline{x}, s_1\vartheta, \ldots, s_n\vartheta) \ll f(t_1, \ldots, t_m)\} \cup \Gamma\vartheta;\ \sigma\vartheta$, where $\vartheta = \{\overline{x} \mapsto \ulcorner t, \overline{x}\urcorner\}$.

CVD: **Context Variable Deletion**

$\{\overline{C}(s) \ll t\} \cup \Gamma;\ \sigma \Longrightarrow \{s\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta$, where $\vartheta = \{\overline{C} \mapsto \circ\}$.

D: **Deepening**

$\{\overline{C}(s) \ll f(t_1, \ldots, t_m)\} \cup \Gamma;\ \sigma \Longrightarrow \{\overline{C}(s\vartheta) \ll t_j\} \cup \Gamma\vartheta;\ \sigma\vartheta$,
where $\vartheta = \{\overline{C} \mapsto f(t_1, \ldots, t_{j-1}, \overline{C}(\circ), t_{j+1}, \ldots, t_m)\}$ for some $1 \leq j \leq m$, and $m > 0$.

SC: **Symbol Clash**

$\{f(s_1, \ldots, s_n) \ll g(t_1, \ldots, t_m)\} \cup \Gamma;\ \sigma \Longrightarrow \bot$, if $f \notin \mathcal{V}_{\mathrm{Con}} \cup \mathcal{V}_{\mathrm{Fun}}$ and $f \neq g$.

AD: **Arity Disagreement**

$\{f(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)\} \cup \Gamma;\ \sigma \Longrightarrow \bot$,
if $m \neq n$ and $s_i \notin \mathcal{V}_{\mathrm{Seq}}$ for all $1 \leq i \leq n$, or $m = 0$ and $s_i \notin \mathcal{V}_{\mathrm{Seq}}$ for some $1 < i \leq n$.

We may use the rule name abbreviations as subscripts, e.g. $\Gamma_1; \sigma_1 \Longrightarrow_{\mathsf{T}} \Gamma_2; \sigma_2$ for the Trivial rule. SVD, W, CVD, and D are non-deterministic rules. A *derivation* is a sequence $\Gamma_1; \sigma_1 \Longrightarrow \Gamma_2; \sigma_2 \Longrightarrow \cdots$ of system transformations.

Definition 1. *A* CSM algorithm $\mathfrak{M}$ *is any program that takes a system* $\Gamma; \varepsilon$ *as input and uses the rules in* $\mathfrak{I}$ *to generate a complete tree of derivations, called the* matching tree for Γ*, in the following way:*

1. *The root of the tree is labeled with* $\Gamma; \varepsilon$.
2. *Each branch of the tree is a derivation. The nodes in the tree are systems.*
3. *If several transformation rules, or different instances of the same transformation rule are applicable to a node in the tree, they are applied concurrently. No rules are applicable to the leaves.*

The algorithm $\mathfrak{M}$ was first introduced in [29]. The leaves of a matching tree are labeled either with the systems of the form $\emptyset; \sigma$ or with $\bot$. The branches that end with $\emptyset; \sigma$ are *successful branches*, and those that end with $\bot$ are *failed branches*. We denote by $\mathcal{S}ol_{\mathfrak{M}}(\Gamma)$ the solution set of Γ generated by $\mathfrak{M}$, i.e., the set of all σ's such that $\emptyset; \sigma$ is a leaf of the matching tree for Γ.

Theorem 1. *The matching algorithm* $\mathfrak{M}$ *terminates for any input problem* Γ *and generates a minimal complete set of matchers of* Γ.

Moreover, $\mathfrak{M}$ never computes the same matcher twice. If we are not interested in bindings for certain variables, we can replace them with the anonymous variables: "$\_$" for any individual or function variable, and "$\_\_$" for any sequence or context variable. It is straightforward to adapt the rules in $\mathfrak{I}$ to such cases: If an anonymous variable occurs in the rule IVE, FVE, SVD, W, CVD, or D then the substitution ϑ in the same rule is ε. Strictly speaking, if $\{s \ll t\}$ is a CSM problem where s contains anonymous variables and ϑ is a solution computed by the adapted version of the algorithm then $s\vartheta$ is not identical to t (because it still contains anonymous variables) but is embedded in t.

We can use (the adapted form of) $\mathfrak{M}$ for multi-slot information extraction from data by nonlinear queries (cf. e.g. [38]):

Example 2. Solving the CSM problem

$$\{\overline{C}(F(\_\_, \overline{D}(f(x)), \_\_, \overline{E}(f(x)), \_\_)) \ll f(g(b, f(a), f(a)), f(b), f(a))\}$$

by $\mathfrak{M}$ gives three solutions:

$$\{\overline{C} \mapsto \circ, \overline{D} \mapsto g(b, \circ, f(a)), \overline{E} \mapsto \circ, F \mapsto f, x \mapsto a\},$$
$$\{\overline{C} \mapsto \circ, \overline{D} \mapsto g(b, f(a), \circ), \overline{E} \mapsto \circ, F \mapsto f, x \mapsto a\},$$
$$\{\overline{C} \mapsto f(\circ, f(b), f(a)), \overline{D} \mapsto \circ, \overline{E} \mapsto \circ, F \mapsto g, x \mapsto a\}.$$

It extracts contexts under which two equal subtrees of the form $f(x)$ are located. With the help of function variables one can also extract contexts under which two equal leaves lie: $\{\overline{C}(F(\_\_, \overline{D}(G()), \_\_, \overline{E}(G()), \_\_)) \ll f(g(a, b), a)\}$ returns $\{\overline{C} \mapsto \circ, \overline{D} \mapsto g(\circ, b), \overline{E} \mapsto \circ, F \mapsto f, G \mapsto a\}$ (remember that $a() = a$).

The algorithm $\mathfrak{M}$ can be further optimized by detecting failure early and avoiding branching whenever possible. Below we consider some of the methods to achieve this. Let $s \ll t$ be a CSM equation where $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_m)$. Then $s \ll t$ fails if any of the following matching pretests succeeds:

1. The number of symbol occurrences N different from context and sequence variables in s is greater than that in t. For instance, if $s = f(\overline{C}(a), F(x), \overline{y})$ and $t = f(a, a)$, then $N(s) = 4$, $N(t) = 3$ and, hence, $s \ll t$ fails.
2. If s contains a function symbol that does not occur in t like, for instance, for $s = f(\overline{x}, \overline{C}(a), b)$ and $t = f(c, b)$ where a does not occur in t.
3. If the sequence of heads of s's is not a subsequence of the sequence of heads of t's. This is the case, for instance, for $s = f(\overline{C}(a), g(x), \overline{x}, g(y))$ and $t = f(a, g(a), f(a))$, where the sequence g, g is not a subsequence of a, g, f.
4. If the minimum depth of s is greater than the depth of t. The minimum depth of a term is computed as the depth without context variables. For instance, the minimum depth of $s = f(f(\overline{C}(F(\overline{x}, f(a)))), g(a, f(x)))$ is 4, and s does not match $t = f(f(a, f(a)), g(a, f(b)))$ whose depth is 3.

Various such pretests are known in the term indexing literature; see, e.g. [42].

Branching is caused by context and sequence variables that permit multiple bindings. It happens in the rules SVD, W, CVD, and D. In certain cases backtracking can be avoided if we can detect the right binding early enough. For instance, for the matching equation $f(\overline{x}) \ll f(a,b,c)$ we can compute the solution $\{\overline{x} \mapsto \ulcorner a,b,c \urcorner\}$ immediately instead of applying the rule W three times and then SVD once. Therefore, a good heuristics would be first, to select such equations as early as possible, and second, to facilitate generating such equations. To achieve the latter whenever possible, we introduce the following two rules:

Sp: **Splitting**

$$\{f(\overline{x}, s_1, \ldots, s_i, \ldots, s_n) \ll f(t_1, \ldots, t_j, \ldots, t_m)\} \cup \Gamma;\ \sigma \Longrightarrow$$
$$\{f(\overline{x}, s_1, \ldots, s_{i-1}) \ll f(t_1, \ldots, t_{j-1}), s_i \ll t_j,$$
$$f(s_{i+1}, \ldots, s_n) \ll f(t_{j+1}, \ldots, t_m)\} \cup \Gamma;\ \sigma, \text{ where } \mathit{head}(s_i) = \mathit{head}(t_j).$$

TID: **Tail Decomposition**

$$\{f(\overline{x}, s_1, \ldots, s_{i-1}, \overline{y}, s_{i+1}, \ldots, s_n) \ll f(t_1, \ldots, t_j, \ldots, t_m)\} \cup \Gamma;\ \sigma \Longrightarrow$$
$$\{f(\overline{x}, s_1, \ldots, s_{i-1}, \overline{y}) \ll f(t_1, \ldots, t_j), s_{i+1} \ll t_{j+1}, \ldots, s_n \ll t_m\} \cup \Gamma;\ \sigma,$$

if $s_k \notin \mathcal{V}_{\mathrm{Seq}}$ for all $i < k \leq n$ and $n - i = m - j$.

Note that Sp still introduces branching because there can be several choices of s_i and t_j. (Branching factor can be reduced by tailoring early failure pretests into Sp.) Applying Sp and TID eagerly together with early failure detection tests and the deterministic rules from $\mathfrak{I}$ eventually generates CSM problems where sequence variables occur in the equations like $f(\overline{x}) \ll t$ and $f(\overline{x}, s_1, \ldots, s_n, \overline{y}) \ll t$. Here s's are variables or have function or context variables in the topmost position. The equations of the former type can be solved immediately, while the latter ones can be attacked either by SVD and W rules, or by eliminating sequence variables by Diophantine techniques. It can be done as follows: Let $f(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)$ be a CSM problem, where $\overline{x}_1, \ldots, \overline{x}_k$ are all sequence variables among s's, and N_i is the number of occurrences of $\overline{x}_i$ (at the topmost level). We associate a linear Diophantine equation $\sum_{i=1}^{k} N_i X_i = m - n + k$ to each such CSM problem and solve it for X's over naturals. If the equation is unsolvable then the matching attempt fails. Otherwise, a solution l_i for each X_i specifies the length of sequence the variable $\overline{x}_i$ can be bound with. Therefore, we replace $f(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)$ with new matching problems $f(s_i) \ll f(t_{j_i}, \ldots, t_{j_i + k_i})$ for each $1 \leq i \leq n$, where $j_1 = 1, j_{i+1} = j_i + k_i + 1, j_n + k_n = m$, $k_i = l_i - 1$ if s_i is a sequence variable, and $k_i = 0$ otherwise. Since linear Diophantine equations can have several solutions, this technique introduces a branching point. For instance, the matching problem $\{f(\overline{x}, \overline{y}) \ll f(a,b)\}$ will lead either to $\{f(\overline{x}) \ll f(), f(\overline{y}) \ll f(a,b)\}$, to $\{f(\overline{x}) \ll f(a), f(\overline{y}) \ll f(b)\}$, or to $\{f(\overline{x}) \ll f(a,b), f(\overline{y}) \ll f()\}$.

Although solving linear Diophantine equations over naturals is NP-complete, in practice it may still be useful to apply this technique for certain problems. Hence, in this way a CSM problem can essentially be reduced to matching with individual, context, and function variables. For such problems we can easily adapt context matching optimization techniques from [41] and add them to $\mathfrak{M}$.

4 Matching Algorithm with Regular Constraints

Regular expressions provide a powerful mechanism for restricting data values. The classical approach to regular expression matching is based on automata. In this section we show that regular expression matching can be easily incorporated into the rule-based framework of CSM.

Regular expressions on terms are defined by the following grammar:

$$\mathtt{R} ::= t \mid \ulcorner\urcorner \mid \ulcorner \mathtt{R}_1, \mathtt{R}_2 \urcorner \mid \mathtt{R}_1 | \mathtt{R}_2 \mid \mathtt{R}^*,$$

where t is a term without holes, $\ulcorner\urcorner$ is the empty sequence, "," is concatenation, "|" is choice, and "*" is repetition (Kleene star). The operators are right-associative; "*" has the highest precedence, followed by "," and "|".

Substitutions are extended to regular expressions on terms in the usual way: $\ulcorner\urcorner\sigma = \ulcorner\urcorner$, $\ulcorner \mathtt{R}_1, \mathtt{R}_2 \urcorner\sigma = \ulcorner \mathtt{R}_1\sigma, \mathtt{R}_2\sigma \urcorner$, $(\mathtt{R}_1|\mathtt{R}_2)\sigma = \mathtt{R}_1\sigma|\mathtt{R}_2\sigma$, and $\mathtt{R}^*\sigma = (\mathtt{R}\sigma)^*$. Each regular expression on terms $\mathtt{R}$ defines the corresponding regular language $L(\mathtt{R})$.

Regular expressions on contexts are defined as follows:

$$\mathtt{Q} ::= C \mid \ulcorner \mathtt{Q}_1, \mathtt{Q}_2 \urcorner \mid \mathtt{Q}_1 | \mathtt{Q}_2 \mid \mathtt{Q}^*.$$

Like for regular expressions on terms, substitutions are extended to regular expressions on contexts in the usual way. Each regular expression on contexts $\mathtt{Q}$ defines the corresponding regular tree language $L(\mathtt{Q})$ as follows:

$$\begin{aligned} L(C) &= \{C\}. \\ L(\ulcorner \mathtt{Q}_1, \mathtt{Q}_2 \urcorner) &= \{C_1[C_2] \mid C_1 \in L(\mathtt{Q}_1) \text{ and } C_2 \in L(\mathtt{Q}_2)\}. \\ L(\mathtt{Q}_1|\mathtt{Q}_2) &= L(\mathtt{Q}_1) \cup L(\mathtt{Q}_2). \\ L(\mathtt{Q}^*) &= \{\circ\} \cup L(\ulcorner \mathtt{Q}, \mathtt{Q}^* \urcorner). \end{aligned}$$

Membership atoms are atoms of the form $Ts \ \mathsf{in}\ \mathtt{R}$ or $Cv \ \mathsf{in}\ \mathtt{Q}$, where Ts is a finite, possibly empty, sequence of terms, and Cv is either a context or a context variable. *Regular constraints* are pairs $(p, \mathtt{f})$ where p is a membership atom and $\mathtt{f}$ is a flag that is a boolean expression (with the possible values 0 or 1). The intuition behind the regular constraint $(Ts \ \mathsf{in}\ \mathtt{R}, \mathtt{f})$ is that $Ts \in L(\mathtt{R}) \setminus \{\ulcorner\urcorner\}$ for $\mathtt{f} = 1$ and $Ts \in L(\mathtt{R})$ for $\mathtt{f} = 0$.[4] Similarly, the intuition behind $(Cv \ \mathsf{in}\ \mathtt{Q}, \mathtt{g})$ is that $Cv \in L(\mathtt{Q}) \setminus \{\circ\}$ for $\mathtt{g} = 1$ and $Cv \in L(\mathtt{Q})$ for $\mathtt{g} = 0$. It will be needed later to guarantee that the regular matching algorithm terminates. Substitutions are extended to regular constraints in the usual way. A *regular* CSM *problem* is a multiset of matching equations and regular constraints of the form:

$$\begin{aligned} &\{s_1 \ll t_1, \ldots, s_n \ll t_n,\ (\overline{x}_1 \ \mathsf{in}\ \mathtt{R}_1, \mathtt{f}_1), \ldots, (\overline{x}_m \ \mathsf{in}\ \mathtt{R}_m, \mathtt{f}_m), \\ &\ (\overline{C}_1 \ \mathsf{in}\ \mathtt{Q}_1, \mathtt{g}_1), \ldots, (\overline{C}_k \ \mathsf{in}\ \mathtt{Q}_k, \mathtt{g}_k)\}, \end{aligned}$$

[4] Note that $(Ts \ \mathsf{in}\ \mathtt{R}^*, 1)$ does not have the same meaning as $(Ts \ \mathsf{in}\ \ulcorner \mathtt{R}, \mathtt{R}^* \urcorner, 0)$: Just take a^* as $\mathtt{R}$.

where all $\overline{x}$'s and all $\overline{C}$'s are distinct and do not occur in $\mathtt{R}$'s and $\mathtt{Q}$'s.[5] We will assume that all $\overline{x}$'s and $\overline{C}$'s occur in the matching equations. A substitution σ is called a *regular matcher* for such a problem if $s_i\sigma = t_i$, $\mathtt{f}_j\sigma \in \{0,1\}$, $\mathtt{Q}_l\sigma \in \{0,1\}$, $\overline{x}_j\sigma \in L(\mathtt{R}_j\sigma)_{\mathtt{f}_j\sigma}$, and $\overline{C}_l\sigma \in L(\mathtt{Q}_l\sigma)_{\mathtt{g}_l\sigma}$ for all $1 \le i \le n$, $1 \le j \le m$, and $1 \le l \le k$, where $L(\mathtt{R})_0 = L(\mathtt{R})$, $L(\mathtt{R})_1 = L(\mathtt{R}) \setminus \{\ulcorner\urcorner\}$, $L(\mathtt{Q})_0 = L(\mathtt{Q})$, and $L(\mathtt{Q})_1 = L(\mathtt{Q}) \setminus \{\circ\}$.

A straightforward way to solve regular CSM problems would be first computing matchers and then testing whether the values of constrained variables satisfy the corresponding constraints. Testing can be done by automata constructed from regular expressions for each computed matcher. (Since regular expressions contain variables that get instantiated during the matching process, the automata would be different for each matcher.) Below we propose a different approach that saves the effort of solution testing. We construct an algorithm that computes the correct answers directly. Another advantage of this approach is that we are not restricted to linear regular expressions.

We define the inference system $\mathfrak{I}_R$ to solve regular CSM problems. It operates on systems $\Gamma;\sigma$ where Γ is a regular CSM problem and σ is a substitution. The system $\mathfrak{I}_R$ includes all the rules from the system $\mathfrak{I}$, but SVD, W, CVD, and D need an extra condition on applicability: For the variables $\overline{x}$ and $\overline{C}$ in those rules there should be no regular constraint $(\overline{x} \text{ in } \mathtt{R},\mathtt{f})$ and $(\overline{C} \text{ in } \mathtt{Q},\mathtt{g})$ in the matching problem. There are additional rules in $\mathfrak{I}_R$ for the variables constrained by regular constraints listed below. For the function symbols `NonEmptySeq`, `NonEmptyCtx`, and $\oplus$ used in these rules the following equalities hold: $\mathtt{NonEmptySeq}() = 0$ and $\mathtt{NonEmptySeq}(r_1,\dots,r_n) = 1$ if $r_i \notin \mathcal{V}_{\mathrm{Seq}}$ for some $1 \le i \le n$; $\mathtt{NonEmptyCtx}(\circ) = 0$ and $\mathtt{NonEmptyCtx}(C) = 1$ if the context C contains at least one symbol different from context variables and the hole constant; $0 \oplus 0 = 1 \oplus 1 = 0$ and $1 \oplus 0 = 0 \oplus 1 = 1$.

ESRET: **Empty Sequence in a Regular Expression for Terms**

$$\{f(\overline{x},s_1,\dots,s_n) \ll t,\ (\overline{x} \text{ in } \ulcorner\urcorner,\mathtt{f})\} \cup \Gamma;\ \sigma \quad\Longrightarrow\quad \begin{cases} \{f(\overline{x},s_1,\dots,s_n)\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta, \text{ with } \vartheta = \{\overline{x} \mapsto \ulcorner\urcorner\} & \text{if } \mathtt{f} = 0, \\ \bot & \text{if } \mathtt{f} = 1. \end{cases}$$

TRET: **Term in a Regular Expression for Terms**

$$\{f(\overline{x},s_1,\dots,s_n) \ll t,\ (\overline{x} \text{ in } s,\mathtt{f})\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\overline{x},s_1,\dots,s_n)\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta, \quad \text{where } \vartheta = \{\overline{x} \mapsto s\} \text{ and } s \notin \mathcal{V}_{\mathrm{Seq}}.$$

SVRET: **Sequence Variable in a Regular Expression for Terms**

$$\{f(\overline{x},s_1,\dots,s_n) \ll t,\ (\overline{x} \text{ in } \overline{y},\mathtt{f})\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\overline{x},s_1,\dots,s_n)\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{x} \mapsto \overline{y}\}$ if $\mathtt{f} = 0$. If $\mathtt{f} = 1$ then $\vartheta = \{\overline{x} \mapsto \ulcorner y,\overline{y}\urcorner,\ \overline{y} \mapsto \ulcorner y,\overline{y}\urcorner\}$ where y is a fresh variable.

ChRET: **Choice in a Regular Expression for Terms**

$$\{f(\overline{x},s_1,\dots,s_n) \ll t,\ (\overline{x} \text{ in } \mathtt{R}_1|\mathtt{R}_2,\mathtt{f})\} \cup \Gamma;\ \sigma \Longrightarrow \{f(\overline{x},s_1,\dots,s_n) \ll t,\ (\overline{x} \text{ in } \mathtt{R}_i,\mathtt{f})\} \cup \Gamma;\ \sigma, \text{ for } i = 1,2.$$

[5] This restriction can be relaxed allowing occurrences without cycles.

CRET: **Concatenation in a Regular Expression for Terms**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x}\ \mathsf{in}\ \ulcorner \mathtt{R}_1, \mathtt{R}_2 \urcorner, \mathtt{f})\} \cup \Gamma;\ \sigma$$
$$\Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t,\ (\overline{y}_1\ \mathsf{in}\ \mathtt{R}_1, \mathtt{f}_1), (\overline{y}_2\ \mathsf{in}\ \mathtt{R}_2, \mathtt{f}_2)\} \cup \Gamma\vartheta;\ \sigma\vartheta,$$

where $\overline{y}_1$ and $\overline{y}_2$ are fresh variables, $\vartheta = \{\overline{x} \mapsto \ulcorner \overline{y}_1, \overline{y}_2 \urcorner\}$, and $\mathtt{f}_1$ and $\mathtt{f}_2$ are computed as follows: If $\mathtt{f} = 0$ then $\mathtt{f}_1 = \mathtt{f}_2 = 0$ else $\mathtt{f}_1 = 0$ and $\mathtt{f}_2 = \mathtt{NonEmptySeq}(\overline{y}_1) \oplus 1$.

RRET1: **Repetition in a Regular Expression for Terms 1**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x}\ \mathsf{in}\ \mathtt{R}^*, 0)\} \cup \Gamma;\ \sigma$$
$$\Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta,\ \text{where } \vartheta = \{\overline{x} \mapsto \ulcorner \urcorner\}.$$

RRET2: **Repetition in a Regular Expression for Terms 2**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x}\ \mathsf{in}\ \mathtt{R}^*, \mathtt{f})\} \cup \Gamma;\ \sigma$$
$$\Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t,\ (\overline{y}\ \mathsf{in}\ \mathtt{R}, 1),\ (\overline{x}\ \mathsf{in}\ \mathtt{R}^*, 0)\} \cup \Gamma\vartheta;\ \sigma\vartheta,$$

where y is a fresh variable and $\vartheta = \{\overline{x} \mapsto \ulcorner \overline{y}, \overline{x} \urcorner\}$.

HREC: **Hole in a Regular Expression for Contexts**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \circ, \mathtt{g})\} \cup \Gamma;\ \sigma$$
$$\Longrightarrow \begin{array}{ll} \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta,\ \text{with } \vartheta = \{\overline{C} \mapsto \circ\} & \text{if } \mathtt{g} = 0, \\ \bot & \text{if } \mathtt{g} = 1. \end{array}$$

CxREC: **Context in a Regular Expression for Contexts**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ C, \mathtt{g})\} \cup \Gamma;\ \sigma \Longrightarrow \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta,$$

where $C \neq \circ$, $\mathit{head}(C) \notin \mathcal{V}_{\mathrm{Con}}$, and $\vartheta = \{\overline{C} \mapsto C\}$.

CVREC: **Context Variable in a Regular Expression for Contexts**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \overline{D}(\circ), \mathtt{g})\} \cup \Gamma;\ \sigma \Longrightarrow \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{C} \mapsto \overline{D}(\circ)\}$ if $\mathtt{g} = 0$. If $\mathtt{g} = 1$ then $\vartheta = \{\overline{C} \mapsto F(\overline{x}, \overline{D}(\circ), \overline{y}),\ \overline{D} \mapsto F(\overline{x}, \overline{D}(\circ), \overline{y})\}$, where $F, \overline{x}$, and $\overline{y}$ are fresh variables.

ChREC: **Choice in a Regular Expression for Contexts**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}_1|\mathtt{Q}_2, \mathtt{g})\} \cup \Gamma;\ \sigma \Longrightarrow \{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}_i, \mathtt{g})\} \cup \Gamma;\ \sigma,$$

for $i = 1, 2$.

CREC: **Concatenation in a Regular Expression for Contexts**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \ulcorner \mathtt{Q}_1, \mathtt{Q}_2 \urcorner, \mathtt{g})\} \cup \Gamma;\ \sigma$$
$$\Longrightarrow \{\overline{C}(s)\vartheta \ll t,\ (\overline{D}_1\ \mathsf{in}\ \mathtt{Q}_1, \mathtt{g}_1), (\overline{D}_2\ \mathsf{in}\ \mathtt{Q}_2, \mathtt{g}_2)\} \cup \Gamma\vartheta;\ \sigma\vartheta,$$

where $\overline{D}_1$ and $\overline{D}_2$ are fresh variables, $\vartheta = \{\overline{C} \mapsto \overline{D}_1(\overline{D}_2(\circ))\}$, and $\mathtt{g}_1$ and $\mathtt{g}_2$ are computed as follows: If $\mathtt{g} = 0$ then $\mathtt{g}_1 = \mathtt{g}_2 = 0$ else $\mathtt{g}_1 = 0$ and $\mathtt{g}_2 = \mathtt{NonEmptyCtx}(\overline{D}_1) \oplus 1$.

RREC1: **Repetition in a Regular Expression for Contexts 1**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, 0)\} \cup \Gamma;\ \sigma$$
$$\Longrightarrow \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma\vartheta;\ \sigma\vartheta,\ \text{where } \vartheta = \{\overline{C} \mapsto \circ\}.$$

RREC2: **Repetition in a Regular Expression for Contexts 2**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, \mathtt{g})\} \cup \Gamma;\ \sigma$$
$$\Longrightarrow \{\overline{C}(s)\vartheta \ll t,\ (\overline{D}\ \mathsf{in}\ \mathtt{Q}, 1),\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, 0)\} \cup \Gamma\vartheta;\ \sigma\vartheta,$$

where $\overline{D}$ is a fresh variable and $\vartheta = \{\overline{C} \mapsto \overline{D}(\overline{C}(\circ))\}$.

A regular CSM algorithm $\mathfrak{M}_R$ is defined in a similar way to the algorithm $\mathfrak{M}$ (Definition 1) with the only difference that the rules of $\mathfrak{I}_R$ are used instead of the rules of $\mathfrak{I}$. From the beginning, each flag in the input problem is set either to 0 or to 1. Note that the rules in $\mathfrak{I}_R$ work either on a selected matching equation, or on a selected pair of a matching equation and a regular constraint. No rule selects a regular constraint alone. We denote by $\mathcal{S}ol_{\mathfrak{M}_R}(\Gamma)$ the solution set of Γ generated by $\mathfrak{M}_R$. The following theorems show that $\mathfrak{M}_R$ is sound, terminating, and complete.

Theorem 2 (Soundness of $\mathfrak{M}_R$). *Let Γ be a regular* CSM *problem. Then every substitution $\sigma \in \mathcal{S}ol_{\mathfrak{M}_R}(\Gamma)$ is a regular matcher of Γ.*

Theorem 3 (Termination of $\mathfrak{M}_R$). *$\mathfrak{M}_R$ terminates on any input.*

Theorem 4 (Completeness of $\mathfrak{M}_R$). *Let Γ be a regular* CSM *problem, ϑ be a regular matcher of Γ, and $\mathcal{V}$ be a variable set of Γ. Then there exists a substitution $\sigma \in \mathcal{S}ol_{\mathfrak{M}_R}$ such that $\sigma \leq^{\mathcal{V}} \vartheta$.*

We can adapt $\mathfrak{M}_R$ to anonymous variables like we did for $\mathfrak{M}$. However, a remark has to be made about using anonymous variables in regular expressions with Kleene star. There they behave differently from named singleton variables and play a similar role as, for instance, the pattern `Any` in [24]. The reason is that the variables that had only one occurrence in the matching problem (in an expression with Kleene star) will have two occurrences after the application of the RRET2 and RREC2 rules, while duplicated anonymous variables are not considered to be the same. It affects solvability. For instance, the regular CSM problem $\{f(\overline{x}) \ll f(g(a), g(b)), (\overline{x} \text{ in } g(\_)^*, 0)\}$ has a solution $\{\overline{x} \mapsto \ulcorner g(a), g(b) \urcorner\}$ while the problem $\{f(\overline{x}) \ll f(g(a), g(b)), (\overline{x} \text{ in } g(x)^*, 0)\}$ is unsolvable because it is reduced to $\{f(\overline{x}) \ll f(g(b)), (\overline{x} \text{ in } g(a)^*, 0)\}$. In general, the notion of a *regular matcher* for regular CSM problems with anonymous variables has to be redefined: First, we write $s \preccurlyeq t$ iff the term s (maybe with holes) whose only variables are anonymous variables can be made identical to the ground term t (maybe with holes) by replacing anonymous variables in s with the corresponding expressions (terms, term sequences, function symbols, contexts) and applying contexts as long as possible. For instance, $f(\_, \_\_(\_(\circ, \_\_, a)), \_\_) \preccurlyeq f(a, f(b, g(\circ, \circ, b, a)), c)$. Next, we write $\ulcorner t_1, \ldots, t_n \urcorner \dot{\in} S$ iff there exists $\ulcorner s_1, \ldots, s_n \urcorner \in S$ such that $s_i \preccurlyeq t_i$ for each $1 \leq i \leq n$. Now, let a regular CSM problem be $\{s_1 \ll t_1, \ldots, s_n \ll t_n, (\overline{x}_1 \text{ in } \mathtt{R}_1, \mathtt{f}_1), \ldots, (\overline{x}_m \text{ in } \mathtt{R}_m, \mathtt{f}_m), (\overline{C}_1 \text{ in } \mathtt{Q}_1, \mathtt{g}_1), \ldots, (\overline{C}_k \text{ in } \mathtt{Q}_k, \mathtt{g}_k)\}$, where s's, $\mathtt{R}$'s, and $\mathtt{Q}$'s may contain anonymous variables. A substitution σ is a regular matcher for such a problem if $s_i\sigma \preccurlyeq t_i$, $\mathtt{f}_j\sigma \in \{0, 1\}$, $\mathtt{Q}_l\sigma \in \{0, 1\}$, $\overline{x}_j\sigma \dot{\in} L(\mathtt{R}_j\sigma)_{\mathtt{f}_j\sigma}$, and $\overline{C}_l\sigma \dot{\in} L(\mathtt{Q}_l\sigma)_{\mathtt{g}_l\sigma}$ for all $1 \leq i \leq n$, $1 \leq j \leq m$, and $1 \leq l \leq k$, where the only variables in $s_i\sigma, \mathtt{R}_j\sigma$, and in $\mathtt{Q}_l\sigma$ are anonymous variables. For instance, $\{\overline{x} \mapsto \ulcorner g(a), g(b) \urcorner, x \mapsto c, \overline{C} \mapsto f(g(\circ))\}$ is a regular matcher for the matching problem $\{f(\overline{x}, \overline{C}(x), \_) \ll f(g(a), g(b), f(g(c)), d), (\overline{x} \text{ in } g(\_)^*, 0), (\overline{C} \text{ in } f(\_\_, g(\circ), \_\_), 0)\}$.

Special failure detection tests can be incorporated into $\mathfrak{M}_R$. For instance, we can add the rule $\{f(\overline{x}, s_1, \ldots, s_n) \ll f(), (\overline{x} \text{ in } \mathtt{R}, 1)\} \cup \Gamma; \sigma \Longrightarrow \bot$.

Note that for a problem Γ there might be $\sigma, \vartheta \in \mathcal{S}ol_{\mathfrak{M}_R}(\Gamma)$ such that $v\sigma = v\vartheta$ for all v in the set of variables of Γ. This is the case, for instance, for $\{f(\overline{x}) \ll f(a,b,b,a), (\overline{x} \text{ in } \ulcorner a^*, b^{*\urcorner *}, 0)\}$ and $\{\overline{C}(a) \ll f(g(a), f(a)), (\overline{C} \text{ in } (f(\_\_, \circ, \_\_)^* | g(\_\_, \circ, \_\_)^*)^*, 0)\}$. It can be avoided by replacing regular expressions with the equivalent "disambiguated" ones like, e.g. star normal forms [5]. Such an equivalent formulation for the matching problems above are $\{f(\overline{x}) \ll f(a,b,b,a), \overline{x} \text{ in } ((a|b)^*, 0)\}$ and $\{\overline{C}(a) \ll f(g(a), f(a)), (\overline{C} \text{ in } (f(\_\_, \circ, \_\_) | g(\_\_, \circ, \_\_))^*, 0)\}$.

As syntactic sugar for regular context expressions, we let function symbols, function variables, and context variables be used as the basic building blocks for regular expressions. Such regular expressions are understood as abbreviations for the corresponding regular expressions on contexts. For example, $\ulcorner F, f | \ulcorner \overline{C}, g^{\urcorner *\urcorner}$ abbreviates $\ulcorner F(\_\_, \circ, \_\_), f(\_\_, \circ, \_\_) | \ulcorner \overline{C}(\circ), g(\_\_, \circ, \_\_)^{\urcorner *\urcorner}$. Answer substitutions can also be modified correspondingly. In this way $\mathfrak{M}_R$ will understand the regular path expression syntax.

5 Applications

CSM is the main pattern matching mechanism in the rule-based programming system ρLog [33, 35]. ρLog supports strategic programming with deterministic (labeled) conditional transformation rules, matching with regular constraints, and is built on top of the Mathematica system. As an example, we show a ρLog clause (in a conventional notation) that implements rewriting: $\overline{C}(x) \rightarrow_{\text{rewrite}(z)} \overline{C}(y) \Leftarrow x \rightarrow_z y$. Assume that we have another clause $a \rightarrow_r b$ that defines the rule labeled by r. Then the query $f(a,a) \rightarrow_{\text{rewrite}(r)} x$ (read: find such an x to which $f(a,a)$ can be rewritten by r) succeeds twice: with $x = f(b,a)$ and $x = f(a,b)$. The order in which these answers are generated (and, hence, the term traversal strategy) is defined by the order of matching rules in CSM that compute bindings for $\overline{C}$.

Another ρLog example is the program that from a given term selects subterms whose nodes are all labeled with a. It consists of the following three clauses $\_\_(x) \rightarrow_{a\text{-subt}} x \Leftarrow x \rightarrow_{\texttt{NF}[a\text{'s}]} \mathit{true}$, $a \rightarrow_{a\text{'s}} \mathit{true}$, $\overline{C}(a(a, \overline{x})) \rightarrow_{a\text{'s}} \overline{C}(a(\overline{x}))$, where NF is the ρLog strategy for a normal form computation.

CSM can be used to achieve more control on rewriting, to match program schemata with programs (cf. semi-unification [11], see also [9]), in Web site verification (e.g. in a rewriting-based framework similar to [1]), in XML querying, transformation, schema matching, and related areas. For this purpose (especially for XML related applications) we would need to extend our matching algorithm for *orderless* function symbols. (The orderless property generalizes commutativity for flexible arity function symbols.) Such functions are important for XML querying because the users often are not concerned with the actual order of elements in an XML document. A straightforward but inefficient way of dealing with orderless functions is to consider all possible permutations of their arguments and applying the CSM. To achieve a better performance one can carry over some known techniques from AC-matching to CSM with orderless functions.

In our opinion, a (conditional) rewriting-based query language that implements CSM with orderless functions would possess the advantages of both navigational (path-based) and positional (pattern-based) types of XML query languages. (See [18] for a recent survey on this topic.) It would easily support, for instance, a wide range of queries (selection and extraction, reduction, negation, restructuring, combination), parent-child and sibling relations and their closures, access by position, unordered matching, order-preserving result, partial and total queries, multiple results, and other properties. Moreover, the rule-based paradigm would provide a clean declarative semantics. As an example, we show how to express a reduction query. Reduction is one of the query operations described as desiderata for XML query languages in [32] and, according to [4], is a bottleneck for many of them. Let the XML data (translated into our syntax) consist of the elements of the form:

$$\begin{aligned}&\mathit{manufacturer}(\mathit{mn\text{-}name}(\mathit{Mercury}), \mathit{year}(\mathit{1999}),\\ &\quad \mathit{model}(\mathit{mo\text{-}name}(\mathit{SLT}), \mathit{front\text{-}rating}(\mathit{3.84}), \mathit{side\text{-}rating}(\mathit{2.14}), \mathit{rank}(\mathit{9})), \ldots).\end{aligned}$$

The reduction query operation is formulated as follows: From the *manufacturer* elements drop those *model* sub-elements whose *rank* is greater than 10, and elide the *front-rating* and *side-rating* elements from the remaining models. It can be expressed as a rule $\mathit{manufacturer}(\overline{x}) \to_{\mathtt{NF}[\mathrm{Reduce}]} y$ that evaluates as follows: Its left hand side matches the data, the obtained instance is rewritten into the normal form with respect to the rule Reduce, and the result is returned in y. Reduce is defined by two conditional rewrite rules:

$$\begin{aligned}&\mathit{manufacturer}(\overline{x}_1, \mathit{model}(\_\_, \mathit{rank}(x), \_\_), \overline{x}_2)\\ &\quad \to_{\mathrm{Reduce}} \mathit{manufacturer}(\overline{x}_1, \overline{x}_2) \Leftarrow x > \mathit{10}.\\ &\mathit{manufacturer}(\overline{x}_1, \mathit{model}(\overline{y}_1, \mathit{front\text{-}rating}(\_), \mathit{side\text{-}rating}(\_), \mathit{rank}(x), \overline{y}_2), \overline{x}_2)\\ &\quad \to_{\mathrm{Reduce}} \mathit{manufacturer}(\overline{x}_1, \mathit{model}(\overline{y}_1, \mathit{rank}(x), \overline{y}_2), \overline{x}_2) \Leftarrow x \leq \mathit{10}.\end{aligned}$$

In general, we believe that such a language would be a good candidate to meet many of the requirements for versatile Web query languages [7]. At least, the core principles of referential transparency and answer-closedness, and incomplete queries and answers can be easily supported. As for dealing with nonhierarchical relations provided by, e.g. ID/IDREF links (that naturally asks for the graph data model), one could apply techniques of equational CSM to query such data. As an equational theory we could specify (oriented) equalities between constants representing IDREFs and terms that correspond to IDs. If such a theory can be turned into a convergent rewrite system, it would mean that the data it represents contains no cycles via ID/IDREFs. It would be interesting to study equational CSM in more details. Another interesting and useful future work would be to identify the types of matching problems that CSM can solve efficiently.

6 Related Work

Solving equations with context variables has been intensively investigated in the recent years; see e.g, [13, 14, 31, 39–41]. Context matching is NP-complete.

Decidability of context unification is still an open question. Sequence matching and unification was addressed, for instance, in [3, 20, 23, 26–28, 34]. Sequence unification (and, hence, matching as well) is decidable.

There is a rich literature on matching with regular expressions, especially in the context of general-purpose programming languages and semistructured data querying. Regular expressions are supported in Perl, Emacs-Lisp, XDuce [25], CDuce [2], Xtatic [19], and in the languages based on XPath [12], just to name a few. Various automata-based approaches have been proposed for XML querying; see, e.g. [36, 6, 37, 16, 10]. Context matching is closely related to the evaluation of conjunctive queries over trees [22].

Hosoya and Pierce [25] propose regular expression pattern matching for developing convenient programming constructs for tree manipulation in a statically typed setting. Similar in spirit to ML style pattern matching, their algorithm uses regular expression types to dynamically match values. Patterns can be recursive (under certain restrictions that guarantee that the language remains regular). Recursion allows to write patterns that match, for instance, trees whose nodes are labeled with the same label. CSM does not allow recursion in regular constraints. That is why we needed three ρLog clauses above to solve the problem of selecting terms with all a-labeled nodes. Patterns of Hosoya and Pierce are restricted to be linear. We do not have such a restriction. In general, non-linearity is one of the main difficulties for tree automata-based approaches [15]. Niehren et al [38] use tree automata for multi-slot information extraction from semistructured data. The automata are restricted to be unambiguous that limits n-ary queries to finite unions of Cartesian closed queries (Cartesian products of monadic queries), but this restricted case is processed efficiently. For monadic queries an efficient and expressive information extraction approach, monadic Datalog, was proposed by Gottlob and Koch [21].

Simulation unification [8] uses the *descendant* construct that is similar to context variables in the sense that it allows us to descend in terms to arbitrary depth, but it does not allow regular expressions along it. Also, sequence variables are not present there. However, it can process unordered and incomplete queries, and it is a full scale unification, not a matching.

Our technique of using flags in constraints to guarantee termination is similar to that of Frisch and Cardelli [17] for dealing with ambiguity in matching sequences against regular expressions.

7 Conclusions

We described a sound, complete and terminating matching algorithm for terms built over flexible arity function symbols and context, sequence, function, and individual variables. Values of some context and sequence variables can be constrained by regular expressions. The constraints are not restricted to be linear. We discussed ways to optimize the main algorithm as well as some of the possible applications. Interesting future developments would be the complexity analysis of the algorithm and extending CSM for equational case.

References

1. M. Alpuente, D. Ballis, and M. Falaschi. A rewriting-based framework for web sites verification. *Electr. Notes on Theoretical Comp. Science*, 124(1):41–61, 2005.
2. V. Benzaken, G. Castagna, and A. Frisch. CDuce: an XML-centric general-purpose language. In *Proc. of ICFP'03*, pages 51–63. ACM, 2003.
3. H. Boley. *A Tight, Practical Integration of Relations and Functions*, volume 1712 of *LNAI*. Springer, 1999.
4. A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *ACM SIGMOD Record*, 29(1):68–79, 2000.
5. A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.
6. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Technical Report HKUST-TCSC-2001-05, Hong Kong University of Science and Technology, 2001.
7. F. Bry, Ch. Koch, T. Furche, S. Schaffert, L. Badea, and S. Berger. Querying the web reconsidered: Design principles for versatile web query languages. *Int. J. Semantic Web Inf. Syst.*, 1(2):1–21, 2005.
8. F. Bry and S. Schaffert. Towards a declarative query and transformation language for XML and semistructured data: Simulation unification. In *Proc. of ICLP*, number 2401 in LNCS, Copenhagen, Denmark, 2002. Springer.
9. B. Buchberger and A. Crăciun. Algorithm synthesis by Lazy Thinking: Examples and implementation in Theorema. *Electr. Notes Theor. Comput. Sci.*, 93:24–59, 2004.
10. J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In V. van Oostrom, editor, *Proc. of RTA'04*, volume 3091 of *LNCS*, pages 105–118. Springer, 2004.
11. E. Chasseur and Y. Deville. Logic program schemas, constraints and semi-unification. In *Proc. of LOPSTR'97*, volume 1463 of *LNCS*, pages 69–89. Springer, 1998.
12. J. Clark and S. DeRose, editors. *XML Path Language (XPath) Version 1.0*. W3C, 1999. Available from: `http://www.w3.org/TR/xpath/`.
13. H. Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *J. Symbolic Computation*, 25(4):397–419, 1998.
14. H. Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *J. Symbolic Computation*, 25(4):421–453, 1998.
15. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available from: `http://www.grappa.univ-lille3.fr/tata`, 1997.
16. M. Frick, M. Grohe, and Ch. Koch. Query evaluation on compressed trees. In *Proc. of LICS'03*, pages 188–198. IEEE Computer Society, 2003.
17. A. Frisch and L. Cardelli. Greedy regular expression matching. In *Proc. of ICALP'04*, pages 618–629, 2004.
18. T. Furche, F. Bry, S. Schaffert, R. Orsini, I. Horroks, M. Kraus, and O. Bolzer. Survey over existing query and transformation languages. Available from: `http://rewerse.net/deliverables/i4-d1.pdf`, 2004.
19. V. Gapeyev and B. C. Pierce. Regular object types. In L. Cardelli, editor, *Proc. of ECOOP'03*, volume 2743 of *LNCS*, pages 151–175. Springer, 2003.
20. M. Ginsberg. The MVL theorem proving system. *SIGART Bull.*, 2(3):57–60, 1991.
21. G. Gottlob and Ch. Koch. Monadic Datalog and the expressive power of languages for web information retrieval. *J. ACM*, 51(1):74–113, 2004.

22. G. Gottlob, Ch. Koch, and K. Schulz. Conjunctive queries over trees. In A. Deutsch, editor, *Proc. of PODS'04*, pages 189–200. ACM, 2004.
23. M. Hamana. Term rewriting with sequences. In: Proc. of the First Int. *Theorema* Workshop. Technical report 97–20, RISC, Johannes Kepler University, Linz, 1997.
24. H. Hosoya. Regular expression pattern matching—a simpler design. Manuscript, 2003.
25. H. Hosoya and B. Pierce. Regular expression pattern matching for XML. *J. Functional Programming*, 13(6):961–1004, 2003.
26. T. Kutsia. *Solving and Proving in Equational Theories with Sequence Variables and Flexible Arity Symbols*. PhD thesis, Johannes Kepler University, Linz, 2002.
27. T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Proc. of Joint AISC'2002—Calculemus'2002 Conference*, volume 2385 of *LNAI*, pages 290–304. Springer, 2002.
28. T. Kutsia. Solving equations involving sequence variables and sequence functions. In B. Buchberger and J. A. Campbell, editors, *Proc. of AISC'04*, volume 3249 of *LNAI*, pages 157–170. Springer, 2004.
29. T. Kutsia. Context sequence matching for XML. In M. Alpuente, S. Escobar, and M. Falaschi, editors, *Proc. of WWV'05*, pages 103–119, 2005. (Full version to appear in ENTCS).
30. T. Kutsia and M. Marin. Matching with regular constraints. Technical Report 05-05, RISC, Johannes Kepler University, Linz, 2005.
31. J. Levy and M. Villaret. Linear second-order unification and context unification with tree-regular constraints. In L. Bachmair, editor, *Proc. of RTA'2000*, volume 1833 of *LNCS*, pages 156–171. Springer, 2000.
32. D. Maier. Database desiderata for an XML query language. Available from: `http://www.w3.org/TandS/QL/QL98/pp/maier.html`, 1998.
33. M. Marin. Introducing ρLog. Available from: `http://www.score.is.tsukuba.ac.jp/~mmarin/RhoLog/`, 2005.
34. M. Marin and D. Ţepeneu. Programming with sequence variables: The *Sequentica* package. In *Proc. of the 5th Int.* Mathematica *Symposium*, pages 17–24, 2003.
35. M. Marin and T. Ida. Progress of ρLog, a rule-based programming system. In *7th Intl. Mathematica Symposium (IMS'05)*, Perth, Australia, 2005. To appear.
36. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In *Proc. of FSTTCS'98*, volume 1530 of *LNCS*, pages 134–145. Springer, 1998.
37. F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.
38. J. Niehren, L. Planque, J.-M. Talbot, and S. Tison. N-ary queries by tree automata. In *Proc. of DBPL'05*, 2005.
39. M. Schmidt-Schauß. A decision algorithm for stratified context unification. *J. Logic and Computation*, 12(6):929–953, 2002.
40. M. Schmidt-Schauß and K. U. Schulz. Solvability of context equations with two context variables is decidable. *J. Symbolic Computation*, 33(1):77–122, 2002.
41. M. Schmidt-Schauß and J. Stuber. On the complexity of linear and stratified context matching problems. *Theory Comput. Systems*, 37:717–740, 2004.
42. R. C. Sekar, I. V. Ramakrishnan, and A. Voronkov. Term indexing. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1853–1964. Elsevier and MIT Press, 2001.