Can Context Sequence Matching Be Used for XML Querying?

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 matching algorithm for terms built over flexible arity function symbols and context, function, sequence, and individual variables. The algorithm is called a context sequence matching algorithm. Context variables allow matching to descend in term-trees to arbitrary depth. Sequence variables allow matching to move in term-trees in arbitrary breadth. The ability to explore terms in two orthogonal directions in a uniform way may be useful for querying data available as a large term, like XML documents. We extend the algorithm to process regular constraints and discuss its possible application in XML querying.

1 Introduction

We describe a context sequence matching algorithm and discuss its possible application in querying XML [27]. Context variables may be instantiated with a context—a term with a hole. They permit matching to descend to arbitrary depth in a term represented as a tree. Sequence variables may be instantiated with a finite (maybe empty) sequence of terms. They are normally used with flexible arity function symbols and permit matching to move to arbitrary breadth. Thus, context and sequence variables together allow exploring terms in two orthogonal directions in a uniform way which may be useful for querying data available as a large term, like XML documents.

Besides context and sequence variables we have function and individual variables. Function variables may be instantiated with a single function symbol or with another function variable. Individual variables may be bound with a single term. Like context and sequence variables, functional and individual variables can be used to traverse terms in depth and breadth, respectively, but only in one level.

* Temur Kutsia has been supported by the Austrian Science Foundation (FWF) under Project SFB F1302 and F1322.

In this paper first we describe a minimal and complete rule-based algorithm for context sequence matching. It operates on terms built using flexible arity function symbols and involving context, sequence, function, and individual variables. Then we show how to use context sequence matching in a declarative XML query language. At the end, we extend the algorithm to deal with regular constraints, both in depth and in breadth. Context sequence matching, with or without regular restrictions, is finitary. Regular expressions provide a powerful mechanism for restricting data values in XML. Many languages have support for them.

Context matching and unification have been intensively investigated in the recent past years, see e.g, [9, 10, 20, 24–26]. Context matching is decidable. Decidability of context unification is still an open question. Schmidt-Schauß and Stuber in [26] gave a context matching algorithm and noted that it can be used similar to XPath [7] matching for XML documents. Sequence matching and unification was addressed, for instance, in [2, 12, 13, 16–18, 22]. Both matching and unification with sequence variables are decidable. Sequence unification procedure described in [17, 18] was implemented in the constraint logic programming language CLP(Flex) [8] and was used for XML processing.

Simulation unification [4] implemented in the Xcerpt language has a 'descendant' construct that is similar to context variables in the sense that it allows 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. Having sequence variables in a full scale unification would make it infinitary (see e.g., [18]).

In our opinion, context sequence matching can serve as a computational mechanism for a declarative, rule-based language to query and transform XML. Such a query language would have advantages of both path-based and pattern-based languages that form two important classes of XML query languages. Path-based languages usually allow to access a single set of nodes of the graph or tree representing an XML data. The access is based on relations with other nodes in the graph or tree specified in the path expression. Pattern-based languages allow access to several parts of the graph or tree at once specifying the relations among the accessed nodes by tree or graph patterns. (For a recent survey over query and transformation languages see [11].) Moreover, with context sequence matching we can achieve improved control on rewriting that can be useful for rewriting-based web site specification and verification techniques [1]. In our opinion, a system like ρLog [23] can be extended to a prototype of such a query language. ρLog is a rule-based language whose computational mechanism uses sequence matching with function variables. It supports single and multiple query answers, non-deterministic computations, and has a clean declarative semantics. However, implementation issues for such a query language is not a subject of this paper.

Another possible application area for context sequence matching is mathematical knowledge management. For instance, it can retrieve algorithms or problems from the schema library [5] of the Theorema system [6].

The results of this paper extend those of [19] by considering regular expressions on full contexts instead of functions only.

The paper is organized as follows: In Section 2 we introduce preliminary notions. In Section 3 we describe the context sequence matching algorithm. Its application in XML querying is discussed in Section 4. Section 5 is about regular expression matching for context and sequence variables. Section 6 concludes.

2 Preliminaries

We assume fixed pairwise disjoint sets of symbols: 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 countable. 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 *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. We will use lower case Greek letters $\sigma, \vartheta, \varphi$, and ε for substitutions, where ε will denote the empty substitution. As usual, indices may be used with the meta-variables.

Substitutions are extended to terms as follows:

$$\begin{aligned} x\sigma &= \sigma(x) \\ \overline{x}\sigma &= \sigma(\overline{x}) \\ f(t_1, \ldots, t_n)\sigma &= f(t_1\sigma, \ldots, t_n\sigma) \\ F(t_1, \ldots, t_n)\sigma &= \sigma(F)(t_1\sigma, \ldots, t_n\sigma) \\ \overline{C}(t)\sigma &= \sigma(\overline{C})[t\sigma] \end{aligned}$$

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 *context sequence matching problem* is a finite multiset of term pairs (*matching 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 matching equations and matching problems in the usual way. A substitution σ is called a *matcher* of the matching 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 matching problems. A *complete set of matchers* of a matching 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 $\leq$. For solvable problems this set is finite, i.e. context sequence matching is finitary.

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 matching problems. A *system* is either the symbol $\bot$ (representing failure) or a pair $\Gamma;\sigma$, where Γ is a matching problem and σ is a substitution. The inference system $\mathfrak{I}$ consists of the transformation rules on systems listed below. We assume that 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**

$$\{x \ll t\} \cup \Gamma';\ \sigma \Longrightarrow \Gamma'\vartheta;\ \sigma \cup \vartheta, \qquad \text{where } \vartheta = \{x \mapsto t\}.$$

FVE: **Function Variable Elimination**

$$\{F(s_1, \ldots, s_n) \ll f(t_1, \ldots, t_m)\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(s_1\vartheta, \ldots, s_n\vartheta) \ll f(t_1, \ldots, t_m)\} \cup \Gamma'\vartheta;\ \sigma \cup \vartheta,$$

where $\vartheta = \{F \mapsto f\}$.

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 $s_i \notin \mathcal{V}_{\text{Seq}}$ for all $1 \leq i \leq n$.

PD: **Partial Decomposition**

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

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

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**

$\{f(\overline{x}, s_1, \ldots, s_n) \ll f(t, t_1, \ldots, t_m)\} \cup \Gamma';\ \sigma$
$\quad \Longrightarrow \{f(\overline{x}, s_1\vartheta, \ldots, s_n\vartheta) \ll f(t_1, \ldots, t_m)\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$

where $\vartheta = \{\overline{x} \mapsto \ulcorner t, \overline{x}\urcorner\}$.

CVD: **Context Variable Deletion**

$\{\overline{C}(s) \ll t\} \cup \Gamma';\ \sigma \Longrightarrow \{s\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta, \qquad$ 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* context sequence matching algorithm $\mathfrak{M}$ *is any program that takes a system* $\Gamma; \varepsilon$ *as an 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 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 (Main Theorem). *The matching algorithm $\mathfrak{M}$ terminates for any input problem Γ and generates a minimal complete set of matchers of Γ.*

Proof. See Appendix A. □

Moreover, note that $\mathfrak{M}$ 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 anonymous variables: If an anonymous variable occurs in the rule IVE, FVE, SVD, W, CVD, or D then the substitution ϑ in the same rule is the empty substitution ε. It is interesting to note that a context sequence matching equation $s \ll t$ whose all variables are anonymous variables can be considered as a problem of computing simulations of s in t that can be efficiently solved by the algorithm described in [14].

4 Querying XML

We assume the existence of a declarative, rule-based query and transformation language for XML that uses the context sequence matching to answer queries. We refer to this language as $\mathcal{L}$. Queries in $\mathcal{L}$ are expressed as (conditional) rules *pattern* $\rightarrow$ *result* `if` *condition*. We do not go into the details, just mention that conditions can be effectively checked. In particular, arithmetic formulae, matchability tests, and queries can be used in conditions, but special care has to be taken about variable occurrences. Note that conditions can also be omitted (assumed to be true). The *pattern* matches the data in the root position. One can choose between getting all the results or only one of them.

To put more syntactic sugar on queries, we borrow some notation from [4]. We write $f\{s_1, \ldots, s_n\}$ if the order of arguments $s_1, \ldots, s_n$ does not matter. The following (rather inefficient) rule relates a matching problem in which the curly bracket construct occurs, to the standard matching problems:

Ord: **Orderless**

$\{f\{s_1, \ldots, s_n\} \ll t\} \cup \Gamma';\ \sigma \Longrightarrow \{f(s_{\pi(1)}, \ldots, s_{\pi(n)}) \ll t\} \cup \Gamma';\ \sigma,$
if $f(s_1, \ldots, s_n) \neq t$ and π is a permutation of $1, \ldots, n$.

Moreover, we can use the double curly bracket notation $f\{\!\{s_1, \ldots, s_n\}\!\}$ for the term $f\{\_\_\_, s_1, \_\_\_, \ldots, \_\_\_, s_n, \_\_\_\}$, and the double bracket notation $f(\!(s_1, \ldots, s_n)\!)$ for $f(\_\_\_, s_1, \_\_\_, \ldots, \_\_\_, s_n, \_\_\_)$. The matching algorithm can be easily modified to work directly (and more efficiently) on such representations.

Now we show how in this language the query operations given in [21] can be expressed. (This benchmark was used to compare five XML query languages in [3].) The case study is that of a car dealer office, with documents from different auto dealers and brokers. The `manufacturer` documents list the manufacturers name, year, and models with their names, front rating, side rating, and rank; the `vehicle` documents list the vendor, make, year, color and price. We consider XML data of the form:

```
<manufacturer>
       <mn-name>Mercury</mn-name>
       <year>1999</year>
       <model>
          <mo-name>Sable LT</mo-name>
          <front-rating>3.84</front-rating>
          <side-rating>2.14</side-rating>
          <rank>9</rank>
       </model>
       <model>...</model>
       ...
</manufacturer>
```

while the dealers and brokers publish information in the form

```
<vehicle>
       <vendor>Scott Thomason</vendor>
       <make>Mercury</make>
       <model>Sable LT</model>
       <year>1999</year>
       <color>metallic blue</color>
       <option opt="sunroof"/>
       <option opt="A/C"/>
       <option opt="lthr seats"/>
       <price>26800</price>
</vehicle>.
```

Translating the data into our syntax is pretty straightforward. For instance, the manufacturer element can be written as:

$$manufacturer(mn\text{-}name(Mercury), year(1999),\\ model(mo\text{-}name(SableLT), front\text{-}rating(3.84), side\text{-}rating(2.14), rank(9))).$$

The query operations and their encoding in our syntax are given below.

Selection and Extraction: We want to select and extract <`manufacturer`> elements where some <`model`> has <`rank`> less or equal to 10:

$$\_(\!(manufacturer(\overline{x}_1, model(\overline{y}_1, rank(x), \overline{y}_2), \overline{x}_2))\!)\\ \rightarrow manufacturer(\overline{x}_1, model(\overline{y}_1, rank(x), \overline{y}_2), \overline{x}_2) \texttt{ if } x \leq 10.$$

Reduction: From the <`manufacturer`> elements, we want to drop those <`model`> sub-elements whose <`rank`> is greater than 10. We also want to elide the <`front rating`> and <`side rating`> elements from the remaining models.

$$\begin{aligned} &\_\langle\!\langle \mathit{manufacturer}(\overline{x}_1, \\ &\qquad \mathit{model}(\overline{y}_1, \mathit{front\text{-}rating}(\_), \mathit{side\text{-}rating}(\_), \mathit{rank}(x), \overline{y}_2), \overline{x}_2)\rangle\!\rangle \\ &\rightarrow \mathit{manufacturer}(\overline{x}_1, \mathit{model}(\overline{y}_1, \mathit{rank}(x), \overline{y}_2), \overline{x}_2) \texttt{ if } x \leq 10. \end{aligned}$$

Joins: We want our query to generate pairs of <`manufacturer`> and <`vehicle`> elements where <`mn-name`>=<`make`>, <`mo-name`>=<`model`>, and <`year`>=<`year`>.

$$\begin{aligned} &\_\{\!\{ \mathit{manufacturer}(\overline{x}_1, \mathit{mn\text{-}name}(x_1), \overline{x}_2, \mathit{year}(x_2), \overline{x}_3, \\ &\qquad \overline{C}(\mathit{mo\text{-}name}(y_1)), \overline{x}_4), \\ &\qquad \mathit{vehicle}(\overline{z}_1, \mathit{make}(x_1), \overline{z}_2, \mathit{model}(y_1), \overline{z}_3, \mathit{year}(x_2), \overline{z}_4)\}\!\} \\ &\rightarrow \mathit{pair}(\mathit{manufacturer}(\overline{x}_1, \mathit{mn\text{-}name}(x_1), \overline{x}_2, \mathit{year}(x_2), \overline{x}_3, \\ &\qquad \overline{C}(\mathit{mo\text{-}name}(y_1)), \overline{x}_4), \\ &\qquad \mathit{vehicle}(\overline{z}_1, \mathit{make}(x_1), \overline{z}_2, \mathit{model}(x_2), \overline{z}_3, \mathit{year}(y_1), \overline{z}_4)). \end{aligned}$$

Restructuring: We want our query to collect <`car`> elements listing their make, model, vendor, rank, and price, in this order:

$$\begin{aligned} &\_\{\!\{ \mathit{vehicle}(\langle\!\langle \mathit{vendor}(y_1), \mathit{make}(y_2), \mathit{model}(y_3), \mathit{year}(y_4), \mathit{price}(y_5)\rangle\!\rangle), \\ &\qquad \mathit{manufacturer}(\langle\!\langle \overline{C}(\mathit{rank}(x_1))\rangle\!\rangle)\}\!\} \\ &\rightarrow \mathit{car}(\mathit{make}(y_2), \mathit{model}(y_3), \mathit{vendor}(y_1), \mathit{rank}(x_1), \mathit{price}(y_5)). \end{aligned}$$

Hence, all these operations can be easily expressed in our framework.

At the end of this section we give an example how to extract elements from an XML document that do not meet certain requirements (e.g., miss certain information). Such problems arise in web site verification tasks discussed in [1].

We want our query to select from the <`manufacturer`> elements those <`model`> sub-elements which miss the <`rank`> information:

$$\_\langle\!\langle \mathit{manufacturer}(\langle\!\langle \mathit{model}(\overline{y})\rangle\!\rangle)\rangle\!\rangle \rightarrow \mathit{model}(\overline{y}) \texttt{ if } \mathit{model}(\langle\!\langle \mathit{rank}(\langle\!\langle\rangle\!\rangle)\rangle\!\rangle) \not\ll \mathit{model}(\overline{y}).$$

The condition in the query requires the term $\mathit{model}(\langle\!\langle \mathit{rank}(\langle\!\langle\rangle\!\rangle)\rangle\!\rangle)$ not to match $\mathit{model}(\overline{y})$. The variable $\overline{y}$ gets instantiated while matching the query pattern $\_\langle\!\langle \mathit{manufacturer}(\langle\!\langle \mathit{model}(\overline{y})\rangle\!\rangle)\rangle\!\rangle$ against the data. Since context sequence matching is decidable, the condition can be effectively checked.

5 Regular Expressions

Regular expressions provide a powerful mechanism for restricting data values in XML. Many languages have support for them. In [15] regular expression pattern

matching is proposed as a core feature of programming languages for manipulating XML. The classical approach uses finite automata for regular expression matching. In this section we show that regular expressions matching can be easily incorporated into the rule-based framework of context sequence matching.

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 symbols "$\ulcorner$" and "$\urcorner$" are there just for the readability purposes. 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 $\mathtt{R}$ define 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 `in` $\mathtt{R}$ or Cv `in` $\mathtt{Q}$, where Ts is a finite, possibly empty, sequence of terms, and Cv is either a context or a context variable. Membership-pairs are pairs $(p, \mathtt{f})$ where p is a membership atom and $\mathtt{f}$ is a flag that is boolean expression (with the possible values 0 or 1). The intuition behind the membership-pair $(\overline{x}$ `in` $\mathtt{R}, \mathtt{f})$ is that if $\mathtt{f} = 0$ then $\overline{x}$ is allowed to be replaced with $\ulcorner\urcorner$ if $\mathtt{R}$ permits. If $\mathtt{f} = 1$ then the replacement is impossible, even if the corresponding regular expression permits. Similarly, the intuition behind $(\overline{C}$ `in` $\mathtt{Q}, \mathtt{g})$ is that if $\mathtt{g} = 0$ then $\overline{C}$ is allowed to be replaced with $\circ$ if $\mathtt{Q}$ permits. If $\mathtt{g} = 1$ then the replacement is impossible, even if the corresponding regular expression permits. It will be needed later to guarantee that the regular matching algorithm terminates. Substitutions are extended to membership-pairs in the usual way.

Now, we can extend the query language $\mathcal{L}$ allowing (with some care) membership-pairs in conditions. Then such conditions can be checked using finite (tree) automata. We can also tailor this check into the matching process itself, and this is what we discuss in details below.

A *context sequence regular matching problem* is a multiset of matching equations and membership-pairs of the form:

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

where all $\overline{x}$'s and all $\overline{C}$'s are distinct and do not occur in $\mathtt{R}$'s and $\mathtt{Q}$'s. 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 $L(\mathtt{Q})_1 = L(\mathtt{Q}) \setminus \{\circ\}$.

We define the inference system $\mathfrak{I}_R$ to solve context sequence regular matching problems. It operates on systems $\Gamma; \sigma$ where Γ is a regular matching 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 membership-pair $(\overline{x} \ \mathsf{in}\ \mathtt{R}, \mathtt{f})$ and $(\overline{C} \ \mathsf{in}\ \mathtt{Q}, \mathtt{g})$ in the matching problem. There are additional rules in $\mathfrak{I}_R$ for the variables constrained by membership-pairs listed below. The meta-functions `NonEmptySequence`, `NonEmptyContext`, and $\oplus$ used in these rules are defined as follows: $\mathtt{NonEmptySequence}() = 0$ and $\mathtt{NonEmptySequence}(r_1, \ldots, r_n) = 1$ if $r_i \notin \mathcal{V}_{\mathrm{Seq}}$ for some $1 \le i \le n$; $\mathtt{NonEmptyContext}(\circ) = 0$ and $\mathtt{NonEmptyContext}(C) = 1$ 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**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x} \ \mathsf{in}\ \ulcorner\urcorner, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \begin{cases} \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta, & \text{if } \mathtt{f} = 0, \\ \text{where } \vartheta = \{\overline{x} \mapsto \ulcorner\urcorner\} & \\ \bot & \text{if } \mathtt{f} = 1. \end{cases}$$

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

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x} \ \mathsf{in}\ s, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

where $\vartheta = \{\overline{x} \mapsto s\}$ and $s \notin \mathcal{V}_{\mathrm{Seq}}$.

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

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x} \ \mathsf{in}\ \overline{y}, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

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**

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x} \ \mathsf{in}\ \mathtt{R}_1|\mathtt{R}_2, \mathtt{f})\} \cup \Gamma';\ \sigma \\ \Longrightarrow \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t,\ (\overline{y}_i \ \mathsf{in}\ \mathtt{R}_i, \mathtt{f})\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

for $i = 1, 2$, where y_i is a fresh variable and $\vartheta = \{\overline{x} \mapsto \overline{y}_i\}$.

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 $\mathtt{f}_2 = \mathtt{NonEmptySequence}(\overline{y}_1) \oplus 1$.

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

$$\{f(\overline{x}, s_1, \ldots, s_n) \ll t,\ (\overline{x}\ \mathsf{in}\ \mathtt{R}^*, \mathtt{f})\} \cup \Gamma';\ \sigma$$
$$\Longrightarrow \begin{cases} \{f(\overline{x}, s_1, \ldots, s_n)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta, & \text{if } \mathtt{f} = 0, \\ \text{where } \vartheta = \{\overline{x} \mapsto \ulcorner \urcorner\} & \\ \bot & \text{if } \mathtt{f} = 1. \end{cases}$$

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**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \circ, \mathtt{g})\} \cup \Gamma';\ \sigma$$
$$\Longrightarrow \begin{cases} \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta, & \text{if } \mathtt{g} = 0, \\ \text{where } \vartheta = \{\overline{C} \mapsto \circ\} & \\ \bot & \text{if } \mathtt{g} = 1. \end{cases}$$

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 $C \neq \circ$, $\mathcal{H}ead(C) \notin \mathcal{V}_{\mathrm{Con}}$, 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**

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}_1|\mathtt{Q}_2, \mathtt{g})\} \cup \Gamma';\ \sigma$$
$$\Longrightarrow \{\overline{C}(s)\vartheta \ll t,\ (\overline{D}_i\ \mathsf{in}\ \mathtt{Q}_i, \mathtt{g})\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

for $i = 1, 2$, where $\overline{D}_i$ is a fresh variable and $\vartheta = \{\overline{C} \mapsto \overline{D}_i(\circ)\}$.

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 and $\vartheta = \{\overline{C} \mapsto \overline{D}_1(\overline{D}_2(\circ))\}$, 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 $\mathtt{g}_2 = \mathtt{NonEmptyContext}(\overline{D}_1) \oplus 1$.

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

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, \mathsf{g})\} \cup \Gamma';\ \sigma \Longrightarrow \begin{cases} \{\overline{C}(s)\vartheta \ll t\} \cup \Gamma'\vartheta;\ \sigma\vartheta, & \text{if } \mathsf{g} = 0, \\ \text{where } \vartheta = \{\overline{C} \mapsto \circ\} & \\ \bot & \text{if } \mathsf{g} = 1. \end{cases}$$

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

$$\{\overline{C}(s) \ll t,\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, \mathsf{g})\} \cup \Gamma';\ \sigma \Longrightarrow \{\overline{C}(s)\vartheta \ll t,\ (\overline{D}\ \mathsf{in}\ \mathtt{Q}, 1),\ (\overline{C}\ \mathsf{in}\ \mathtt{Q}^*, 0)\} \cup \Gamma'\vartheta;\ \sigma\vartheta,$$

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

A context sequence regular matching algorithm $\mathfrak{M}_R$ is defined in the similar way as 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 membership-pair. No rule selects a membership-pair alone. We denote by $\mathcal{S}ol_{\mathfrak{M}_R}(\Gamma)$ the solution set of Γ generated by $\mathfrak{M}_R$.

Theorem 2. *The algorithm $\mathfrak{M}_R$ is sound, terminating and complete.*

Proof. See Appendix B. □

Note that we can extend the system $\mathfrak{I}_R$ with some more rules that facilitate an early detection of failure, e.g., $\{f(\overline{x}, s_1, \ldots, s_n) \ll f(), (\overline{x}\ \mathsf{in}\ \mathtt{R}, 1)\} \cup \Gamma'; \sigma \Longrightarrow \bot$ would be one of such rules.

Turning back to the query language $\mathcal{L}$, now the queries that contain membership atoms in conditions can be resolved by context sequence regular matching, forming the matching problem with the pattern from the query against the data, and the corresponding membership pairs from the condition.

As a syntactic sugar on regular expressions on contexts, 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. We demonstrate the correspondence on the example: The regular expression $\ulcorner F, f|^{\ulcorner}\overline{C}, g^{\urcorner *\urcorner}$ abbreviates the regular context expression $\ulcorner F(\overline{x}_1, \circ, \overline{y}_1), f(\overline{x}_2, \circ, \overline{y}_2)|^{\ulcorner}\overline{C}(\circ), g(\overline{x}_3, \circ, \overline{y}_3)^{\urcorner *\urcorner}$, where $\overline{x}$'s and $\overline{y}$'s are fresh variables. In this way, the query language $\mathcal{L}$ will understand also the regular path expression syntax.

6 Conclusions

We showed how to use context sequence matching to explore terms (represented as trees) in two orthogonal directions: in depth (by context variables) and in breadth (by sequence variables). We developed a minimal complete rule-based

algorithm for context sequence matching. Moreover, we showed that regular restrictions can be easily incorporated in the rule-based matching framework, and sketched proofs of soundness, termination and completeness of such an extension.

Context sequence matching can serve as a computational mechanism for a declarative rule-based XML query and transformation language. In our opinion, an advantage of such a language would be its flexibility and expressiveness: It would combine in itself the features of both path-based and pattern-based languages, and 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, rule-based paradigm would provide a clean declarative semantics.

References

1. M. Alpuente, D. Ballis, and M. Falaschi. A rewriting-based framework for web sites verification. *Electronic Notes on Theoretical Computer Science*, 124(1):41–61, 2005.
2. H. Boley. *A Tight, Practical Integration of Relations and Functions*, volume 1712 of *LNAI*. Springer, 1999.
3. A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *ACM SIGMOD Record*, 29(1):68–79, 2000.
4. F. Bry and S. Schaffert. Towards a declarative query and transformation language for XML and semistructured data: Simulation unification. In *Proc. of International Conference on Logic Programming (ICLP)*, number 2401 in LNCS, Copenhagen, Denmark, 2002. Springer.
5. B. Buchberger and A. Crăciun. Algorithm synthesis by lazy thinking: Examples and implementation in Theorema. In *Proc. of the Mathematical Knowledge Management Symposium*, volume 93 of *Electronic Notes on Theoretical Computer Science*, pages 24–59, 2003.
6. B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The Theorema project: A progress report. In M. Kerber and M. Kohlhase, editors, *Proc. of Calculemus'2000 Conference*, pages 98–113, 2000.
7. J. Clark and S. DeRose, editors. *XML Path Language (XPath) Version 1.0*. W3C, 1999. Available from: `http://www.w3.org/TR/xpath/`.
8. J. Coelho and M. Florido. CLP(Flex): Constraint logic programming applied to XML processing. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE. Proc. of Confederated Int. Conferences*, volume 3291 of *LNCS*, pages 1098–1112. Springer, 2004.
9. H. Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *J. Symbolic Computation*, 25(4):397–419, 1998.
10. H. Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *J. Symbolic Computation*, 25(4):421–453, 1998.
11. 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.

12. M. L. Ginsberg. The MVL theorem proving system. *SIGART Bull.*, 2(3):57–60, 1991.
13. M. Hamana. Term rewriting with sequences. In: Proc. of the First Int. *Theorema* Workshop. Technical report 97–20, RISC, Johannes Kepler University, Linz, Austria, 1997.
14. M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 453–462. IEEE Computer Society Press, 1995.
15. H. Hosoya and B. Pierce. Regular expression pattern matching for XML. *J. Functional Programming*, 13(6):961–1004, 2003.
16. T. Kutsia. *Solving and Proving in Equational Theories with Sequence Variables and Flexible Arity Symbols.* PhD thesis, Johannes Kepler University, Linz, Austria, 2002.
17. T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors, *Artificial Intelligence, Automated Reasoning and Symbolic Computation. Proc. of Joint AISC'2002 – Calculemus'2002 Conference*, volume 2385 of *LNAI*, pages 290–304. Springer, 2002.
18. T. Kutsia. Solving equations involving sequence variables and sequence functions. In B. Buchberger and J. A. Campbell, editors, *Artificial Intelligence and Symbolic Computation. Proc. of AISC'04 Conference*, volume 3249 of *LNAI*, pages 157–170. Springer, 2004.
19. T. Kutsia. Context sequence matching for XML. In M. Alpuente, S. Escobar, and M. Falaschi, editors, *Proc. of First International Workshop on Automated Specification and Verification of Web Sites (WWV'05)*, pages 103–119, Valencia, Spain, March 14–15 2005. (Full version to appear in Elsevier ENTCS).
20. J. Levy and M. Villaret. Linear second-order unification and context unification with tree-regular constraints. In L. Bachmair, editor, *Proc. of the 11th Int. Conference on Rewriting Techniques and Applications (RTA'2000)*, volume 1833 of *LNCS*, pages 156–171. Springer, 2000.
21. D. Maier. Database desiderata for an XML query language. Available from: `http://www.w3.org/TandS/QL/QL98/pp/maier.html`, 1998.
22. M. Marin and D. Ţepeneu. Programming with sequence variables: The *Sequentica* package. In P. Mitic, P. Ramsden, and J. Carne, editors, *Challenging the Boundaries of Symbolic Computation. Proc. of 5th Int.* Mathematica *Symposium*, pages 17–24, London, 2003. Imperial College Press.
23. M. Marin and T. Kutsia. Programming with transformation rules. *Analele Universitatii de Vest din Timisoara*, XVI:163–177, 2003.
24. M. Schmidt-Schauß. A decision algorithm for stratified context unification. *J. Logic and Computation*, 12(6):929–953, 2002.
25. 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.
26. M. Schmidt-Schauß and J. Stuber. On the complexity of linear and stratified context matching problems. Research Report 4923, INRIA-Lorraine, France, 2003.
27. World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0. Second edition. Available from: `http://www.w3.org/`, 1999.

A Proof of Theorem 1

Theorem 1 is an immediate consequence of Soundness, Termination, Completeness, and Minimality theorems for $\mathfrak{M}$ given below.

Theorem 3 (Soundness of $\mathfrak{M}$). *Let Γ be a matching problem. Then every substitution $\sigma \in \mathcal{S}ol_{\mathfrak{M}}(\Gamma)$ is a matcher of Γ.*

Proof. (Sketch) Inspecting the rules in $\mathfrak{I}$ one can conclude that for a derivation $\Gamma; \varepsilon \Longrightarrow^{+} \emptyset; \sigma$ the problems $\Gamma\sigma$ and $\emptyset$ have the same set of matchers. It implies that σ is a matcher of Γ. □

Theorem 4 (Termination of $\mathfrak{M}$). *The algorithm $\mathfrak{M}$ terminates on any input.*

Proof. With each matching problem Δ we associate a complexity measure as a triple of non-negative integers $\langle n_1, n_2, n_3 \rangle$, where n_1 is the number of distinct variables in Δ, n_2 is the number of symbols in the ground sides of matching equations in Δ, and n_3 is the number of subterms in Δ of the form $f(s_1, \ldots, s_n)$, where s_1 is not a sequence variable. Measures are compared lexicographically. Every non-failing rule in $\mathfrak{I}$ strictly decreases the measure. Failing rules immediately lead to termination. Hence, $\mathfrak{M}$ terminates on any input. □

Theorem 5 (Completeness of $\mathfrak{M}$). *Let Γ be a matching problem and let ϑ be a matcher of Γ. Then there exists a derivation $\Gamma; \varepsilon \Longrightarrow^{+} \emptyset; \sigma$ such that $\sigma \preceq \vartheta$.*

Proof. We construct the derivation recursively. For the base case $\Gamma_1; \sigma_1 = \Gamma; \varepsilon$ we have $\varepsilon \preceq \vartheta$. Now assume that the system $\Gamma_n; \sigma_n$, where $n \geq 1$ and $\Gamma_n \neq \emptyset$, belongs to the derivation and find a system $\Gamma_{n+1}; \sigma_{n+1}$ such that $\Gamma_n; \sigma_n \Longrightarrow \Gamma_{n+1}; \sigma_{n+1}$ and $\sigma_{n+1} \preceq \vartheta$. We have $\sigma_n \preceq \vartheta$. Therefore, there exists φ such that $\sigma_n\varphi = \vartheta$ and φ is a matcher of Γ_n. Without loss of generality, we pick an arbitrary matching equation $s \ll t$ from Γ_n and represent Γ_n as $\{s \ll t\} \cup \Gamma'_n$. Depending on the form of $s \ll t$, we have three cases:

Case 1. The terms s and t are the same. We extend the derivation with the step $\Gamma_n; \sigma_n \Longrightarrow_{\mathsf{T}} \Gamma'_n; \sigma_n$. Therefore, $\sigma_{n+1} = \sigma_n \preceq \vartheta$.

Case 2. The term s is an individual variable x. Then $x\varphi = t$. Therefore, for $\psi = \{x \mapsto t\}$ we have $\psi\varphi = \varphi$ and, hence, $\sigma_n\psi\varphi = \vartheta$. We extend the derivation with the step $\Gamma_n; \sigma_n \Longrightarrow_{\mathsf{IVE}} \Gamma'_n; \sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi \preceq \vartheta$.

Case 3. The terms s and t are not the same and s is a compound term. The only non-trivial cases are those when the first argument of s is a sequence variable, or when the head of s is a context variable. If the first argument of s is a sequence variable $\overline{x}$ then φ must contain a binding $\overline{x} \mapsto \ulcorner t_1, \ldots, t_k \urcorner$ for $\overline{x}$, where $k \geq 0$ and t_i's are ground terms. If $k = 0$ then we take $\psi = \{\overline{x} \mapsto \ulcorner\urcorner\}$ and extend the derivation with the step $\Gamma_n; \sigma_n \Longrightarrow_{\mathsf{SVD}} \Gamma'_n; \sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi$. If $k > 0$ then we take $\psi = \{\overline{x} \mapsto \ulcorner t_1, \overline{x} \urcorner\}$ and extend the derivation with the step $\Gamma_n; \sigma_n \Longrightarrow_{\mathsf{W}} \Gamma'_n; \sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi$. In both cases we have $\sigma_{n+1} = \sigma_n\psi \preceq \sigma_n\varphi = \vartheta$. If the head of s is a context variable $\overline{C}$ then φ must contain a binding $\overline{C} \mapsto C$ for $\overline{C}$, where C is a ground context. If $C =$

$\circ$ then we take $\psi = \{\overline{C} \mapsto \circ\}$ and we extend the derivation with the step $\Gamma_n;\sigma_n \Longrightarrow_{\mathsf{CVD}} \Gamma'_n;\sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi$. If $C \neq \circ$ then C should have a form $f(t_1,\ldots,t_{j-1},D,t_{j+1},\ldots,t_m)$, where D is a context and $f(t_1,\ldots,t_m) = t$. Then we take $\psi = \{\overline{C} \mapsto f(t_1,\ldots,t_{j-1},\overline{C}(\circ),t_{j+1},\ldots,t_m)\}$ and extend the derivation with the step $\Gamma_n;\sigma_n \Longrightarrow_{\mathsf{W}} \Gamma'_n;\sigma_{n+1}$, where $\sigma_{n+1} = \sigma_n\psi$. In both cases $\sigma_{n+1} = \sigma_n\psi \leq \sigma_n\varphi = \vartheta$. □

Theorem 6 (Minimality). *Let Γ be a matching problem. Then $\mathcal{S}ol_{\mathfrak{M}}(\Gamma)$ is a minimal set of matchers of Γ.*

Proof. For any matching problem Δ the set

$$S(\Delta) = \{\varphi \mid \Delta;\varepsilon \Longrightarrow \Phi;\varphi \text{ for some } \Phi\}$$

is minimal. Moreover, every substitution ϑ in $S(\Delta)$ preserves minimality: If $\{\sigma_1,\ldots,\sigma_n\}$ is a minimal set of substitutions then so is the set $\{\vartheta\sigma_1,\ldots,\vartheta\sigma_n\}$. It implies that $\mathcal{S}ol_{\mathfrak{M}}(\Gamma)$ is minimal. □

B Proof of Theorem 2

Theorem 2 follows from Soundness, Termination, and Completeness theorems for $\mathfrak{M}_R$ given below.

Theorem 7 (Soundness of $\mathfrak{M}_R$). *Let Γ be a regular matching problem. Then every substitution $\sigma \in \mathcal{S}ol_{\mathfrak{M}_R}(\Gamma)$ is a regular matcher of Γ.*

Proof. (Sketch) Inspecting the rules in $\mathfrak{I}_R$ one can conclude that for a derivation $\Gamma;\varepsilon \Longrightarrow^+ \emptyset;\sigma$ every regular matcher of $\emptyset$ is also a regular matcher of $\Gamma\sigma$. It implies that σ is a regular matcher of Γ. □

Theorem 8 (Termination of $\mathfrak{M}_R$). *The algorithm $\mathfrak{M}_R$ terminates on any input.*

Proof. The tricky part of the proof is related with patterns containing the star "*". A derivation that contains an application of the RRET2 rule on a system with a selected matching equation and membership-pair $s_0 \ll t_0, (\overline{x} \text{ in } \mathtt{R}_0^*, \mathtt{f})$ either fails or eventually produces a system that contains a matching equation $s_1 \ll t_1$ and a membership-pair $(\overline{x} \text{ in } \mathtt{R}_1^*, 0)$ where $\mathtt{R}_1$ is an instance of $\mathtt{R}_0$ and $\overline{x}$ is the first argument of s_1:

$$\begin{aligned}&\{s_0 \ll t_0, (\overline{x} \text{ in } \mathtt{R}_0^*, \mathtt{f})\} \cup \Gamma;\sigma\\ &\quad\Longrightarrow_{\mathsf{RRET2}} \{s_0\vartheta \ll t_0, (\overline{y} \text{ in } \mathtt{R}_0, 1), (\overline{x} \text{ in } \mathtt{R}_0^*, \mathtt{f})\} \cup \Gamma\vartheta;\sigma\vartheta\\ &\quad\Longrightarrow^+ \{s_1 \ll t_1, (\overline{x} \text{ in } \mathtt{R}_1^*, 0)\} \cup \Delta;\varphi.\end{aligned}$$

Hence, the rule RRET2 can apply again on $\{s_1 \ll t_1, (\overline{x} \text{ in } \mathtt{R}_1^*, 0)\} \cup \Delta;\varphi$. The important point is that the total size of the ground sides of the matching equations strictly decreases between these two applications of RRET2: In $\{s_1 \ll t_1\} \cup \Delta$

it is strictly smaller than in $\{s_0 \ll t_0\} \cup \Gamma$. This is guaranteed by the fact that $(\overline{y} \text{ in } \mathtt{R}_0, 1)$ does not allow the variable $\overline{y}$ to be bound with the empty sequence. The same argument applies to derivations that contain an application of the RREC2 rule. Applications of the other rules also lead to a strict decrease of the size of the ground sides after finitely many steps. Since no rule increases the size of the ground sides, the algorithm $\mathfrak{M}_R$ terminates.

Theorem 9 (Completeness of $\mathfrak{M}_R$). *Let Γ be a regular matching 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 $\sigma \leq^{\mathcal{V}} \vartheta$.*

Proof. Similar to the proof of Theorem 5. □