Regular Hedge Language Factorization Revisited

Mircea Marin[1,⋆] and Temur Kutsia[2,⋆⋆]

[1] Department of Computer Science, University of Tsukuba, Japan
[2] RISC, Johannes Kepler University, Linz, Austria

Abstract. We consider the factorization problem of regular hedge languages. This problem is strongly related to the type checking problem in rule based transformations of valid XML documents. We propose the representation of regular hedge languages by reduced, complete, and deterministic linear hedge automata, and indicate algorithms for the computation of right factors and factor matrix.

1 Motivation

Regular hedge languages, or simply RHLs, are a natural generalization of regular languages where strings are replaced by sequences of unranked trees, also known as hedges. They were first studied by Thatcher [8,9], who developed the basic theory of unranked tree automata and investigated their regular extensions. Interest in their study was reignited by the advent of XML as the de facto standard for the exchange and manipulation of data on the Web [1], and by the recognition of the fact that RHLs are a suitable formalism to specify restrictions on the structure of XML documents. Hedge automata [7] were invented to type check (or validate) input data against specifications of RHLs, and regular expression types were introduced in XML processing languages [4] as a means to specify membership constraints to RHLs.

Several results from the theory of regular languages carry over nicely to regular hedge languages. In particular, the factorization theory of regular languages [2] has a natural generalization to regular hedge languages [6,5]. One of the motivations behind the study of regular hedge language factorizations is the type checking problem that arises in the context of XML transformation. Assume we are given a transformation rule $P \to r$ for input documents ranging over an RHL H_{in}, and we want to check if the result belongs to an output type given by an RHL H_{out}. This problem amounts to inferring the types of the pattern variables of P when matching input documents from H_{in}, using them to infer the type (or type over-approximation) H of the result r, and then checking if H is a subtype of H_{out}. We mention two possible approaches:

1. Type inference for pattern variables, used in the XML programming language XDuce [3]. Certain syntactic restrictions and a pattern matching strategy

⋆ Supported by JSPS Grant-in-Aid no. 20500025 for Scientific Research (C).
⋆⋆ Supported by EC FP6 under the project SCIEnce—Symbolic Computation Infrastructure for Europe (Contract No. 026133).

Y. Gao et al. (Eds.): DLT 2010, LNCS 6224, pp. 328–339, 2010.

are imposed in order to enable an easy static type reconstruction algorithm for the variables of P when matched against inputs of a given type. The types inferred for the pattern variables can be used to compute an over-approximation of the type of output r, and check if it is a subtype of H_{out}.

2. Type inference for the tuple of all variables of a pattern. In this case, we compute for every pattern P with variables $x_1, \ldots, x_n$ a finite set

$$\overline{H}_p = \{H_{i,1} \times \ldots \times H_{i,n} \mid i \in \{1, \ldots, p\}\}$$

of cartesian products of RHLs such that: $\{x_1 \mapsto h_1, \ldots, x_n \mapsto h_n\}$ is a matcher of P against some input from H_{in} iff $(h_1, \ldots, h_n)$ belongs to some cartesian product from $\overline{H}_p$. Alternatively, we could say that we infer the type $\sum_{i=1}^{p}(H_{i,1} \times \ldots \times H_{i,n})$ for the tuple of pattern variables $(x_1, \ldots, x_n)$. There are p possibilities for the type of $(x_1, \ldots, x_n)$, which can be used to compute an over-approximation of the type of output r and then check if it is a subtype of H_{out}.

This type inference approach works for the class of patterns[1] proposed by us in [5]. They are similar to the patterns of XDuce, but lack features such as global pattern names and pattern bindings. However, these patterns have some extra features that are desirable for XML querying: (a) they can be nonlinear, (b) variables can occur below iteration, and (c) there is no predefined matching strategy.

The first type checking approach is easier, but the second approach is more accurate and relies on factorizations of regular hedge languages, which are described in Sect. 2.1. The following example illustrates a situation when the second type checking approach is more accurate.

Example 1. Consider the type checking problem for input type $f(a^\star\, b^\star)^\star$, output type $f(a^\star b^\star a^\star b^\star a^\star)$, and transformation rule $f(x\,y)\, f(y\,x) \to f(x\,y\,x)$.

The first type checking approach infers that x is of type $a^\star b^\star$, and y is of type $a^\star b^\star$. Therefore $f(x\,y\,x)$ is of type $f(a^\star b^\star a^\star b^\star a^\star b^\star)$. Since this is not a subtype of the output type $f(a^\star b^\star a^\star b^\star a^\star)$, type checking fails.

The second approach works by noting that $f(x\,y)\, f(y\,x)$ matches an input from $f(a^\star b^\star)^\star$ if and only if both $x\,y$ and $y\,x$ match inputs from $a^\star b^\star$. Then, instead of computing the types for x and y independently of each other, we compute the type of a pair (x, y). From the type of (x, y), by easy simplifications we obtain that the type of $f(x\,y\,x)$ is $f(a^\star b^\star a^\star b^\star)$. Since this is a subtype of the output type $f(a^\star b^\star a^\star b^\star a^\star)$, type checking succeeds. The crucial step in this approach is the computation the type of the tuple (x, y), which heavily relies on factorization computations: First factorizing $f(a^\star b^\star)^\star$, then factorizing $a^\star b^\star$. We will return to this example at the end of the paper to see how the computations are done. □

Regular hedge language factorization algorithms have been described in [6,5]. In this paper we propose a significant improvement over those, by developing a new

[1] They are called regular hedge expressions in [5].

algorithm to compute the factor matrix of an RHL that is more efficient and easier to analyze. The improvement is based on a new representation of RHLs, by deterministic, reduced, and complete linear hedge automata.

The paper is structured as follows. Section 2 introduces the main notions we use in the investigation of regular hedge language factorization, and recalls some well known results. Section 3 presents a simple algorithm for the computation of automaton representations for all right factors of an RHL. In Sect. 4 we propose an algorithm to compute automata representations for the elements of the factor matrix of an RHL. The algorithm makes use of the automaton representations of the right factors of the RHL, and of their corresponding specification by linear systems of hedge language equations. Section 5 concludes by comparing the algorithms presented here with those presented in [6] and [5].

2 Preliminaries

Hedges over an unranked alphabet Σ of hedge labels and finite set of constants $\mathcal{K}$ are finite sequences of trees generated by the grammar

$$h ::= \epsilon \mid k \mid a(h)\, h$$

where ϵ denotes the empty sequence, $k \in \mathcal{K}$, and $a \in \Sigma$. A *tree* over Σ and $\mathcal{K}$ is a hedge of the form k or $a(h)$. Trees of the form $a(\epsilon)$ are abbreviated by a. We write $\mathcal{H}(\Sigma, \mathcal{K})$ for the set of hedges over Σ and $\mathcal{K}$, and $\mathcal{T}(\Sigma, \mathcal{K})$ for the set of trees over Σ and $\mathcal{K}$. Also, we abbreviate $\mathcal{H}(\Sigma, \emptyset)$ by $\mathcal{H}(\Sigma)$, and $\mathcal{T}(\Sigma, \emptyset)$ by $\mathcal{T}(\Sigma)$. A *hedge language* over Σ is a subset of $\mathcal{H}(\Sigma)$. From now on we assume implicitly that H, possibly subscripted, denotes regular hedge languages. The *concatenation* of H_1 and H_2 is the hedge language $H_1\, H_2 := \{h_1\, h_2 \mid h_1 \in H_1, h_2 \in H_2\}$; and the *asterate* of H is the hedge language $H^* := \{\epsilon\} \cup \bigcup_{n=1}^{\infty} \{h_1 \ldots h_n \mid h_1, \ldots, h_n \in H\}$.

A *regular hedge language* (RHL) over an unranked alphabet Σ is a language accepted by a *hedge automaton* (HA). According to [7], such an automaton is a tuple $M = (Q, \Sigma, F, \Delta_M)$ where: Q is a finite set of states; Δ_M is a set of transition rules of the form $a(R) \to q$ where $q \in Q$ and R is a regular language over Q; and F is a regular set over Q, called the *final state sequence set* of M. If we write $\to_M$ for the rewrite relation induced on $\mathcal{H}(\Sigma, Q)$ by the rewrite system $\{a(w) \to q \mid a(R) \to q \in \Delta_M, w \in R\}$, then the language accepted by M is $L(M) := \{h \in \mathcal{H}(\Sigma) \mid \exists w \in F.\, h \to_M^* w\}$.

In this paper we propose another representation of RHLs, by so called linear hedge automata. A *linear hedge automaton* (LHA) over an unranked alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, Q_{\mathtt{f}}, \Delta)$ where: Q is a finite set of states; Δ is a set of transition rules of the form $\epsilon \to q$ or $a(q_1)\, q_2 \to q$ with $q, q_1, q_2 \in Q$ and $a \in \Sigma$; and $Q_{\mathtt{f}} \subseteq Q$ is the set of final states of $\mathcal{A}$. An *ϵ-transition* is a transition rule of the form $\epsilon \to q$. $\mathcal{A}$ is *deterministic* if there are no two transition rules with the same left hand side.

Let $\to_{\mathcal{A}}$ be the rewrite relation induced by the rewrite system Δ on $\mathcal{H}(\Sigma, Q)$. Then the language accepted by $\mathcal{A}$ in a state $q \in Q$ is $L(\mathcal{A}, q) := \{h \in \mathcal{H}(\Sigma) \mid h \to_{\mathcal{A}}^* q\}$. For every $Q' \subseteq Q$ we define $L(\mathcal{A}, Q') := \bigcup_{q \in Q'} L(\mathcal{A}, q)$. The language

accepted by $\mathcal{A}$ is $L(\mathcal{A}) := L(\mathcal{A}, Q_{\mathtt{f}})$. A state q of an LHA $\mathcal{A}$ is *accessible* if $L(\mathcal{A}, q) \neq \emptyset$. The LHA $\mathcal{A}$ is *reduced* if all its states are accessible. $\mathcal{A}$ is *complete* if for every $a \in \Sigma$ and $q_1, q_2 \in Q$ there exists a transition rule $a(q_1)\, q_2 \to q \in \Delta$.

A convenient representation of LHAs is by a *linear system of hedge language equations* (LSH). The *LSH representation* of an LHA $\mathcal{A} = (\{q_1, \ldots, q_n\}, \Sigma, Q_{\mathtt{f}}, \Delta)$ is the pair $(Q_{\mathtt{f}}, S)$ where

$$S : \begin{cases} q_1 = c_1 + \ell_{1,1}\, q_1 + \cdots + \ell_{1,n}\, q_n \\ \vdots \\ q_n = c_n + \ell_{n,1}\, q_1 + \cdots + \ell_{n,n}\, q_n \end{cases} \quad \text{with } c_i := \begin{cases} 1 \text{ if } \epsilon \to q_i \in \Delta, \\ 0 \text{ otherwise} \end{cases}$$

and $\ell_{i,j} := \sum_{a(q)\, q_j \to q_i \in \Delta} a(q)$ for all $1 \leq i, j \leq n$. The variables of this system of equations are $q_1, \ldots, q_n$, and they correspond to the states of the LHA that is being represented. Therefore, from now on, we will refer to the elements of Q either as states of the LHA or as variables of the corresponding LSH.

The equations of S are between expressions which belong to the larger class of regular hedge expressions $\mathrm{HReg}(\Sigma, Q)$ defined by the grammar

$$e ::= 0 \mid 1 \mid q \mid a(e) \mid e + e \mid e\, e \mid e^{\star}.$$

Such expressions are interpreted with respect to an *assignment* for Q, which is a mapping $\sigma : Q \to 2^{\mathcal{H}(\Sigma)}$. The interpretation of $e \in \mathrm{HReg}(\Sigma, Q)$ with respect to an assignment σ is defined as follows: $[\![0]\!]_\sigma := \emptyset$, $[\![1]\!]_\sigma := \{\epsilon\}$, $[\![q]\!]_\sigma := \sigma(q)$, $[\![a(e)]\!]_\sigma := \{a(h) \mid h \in [\![e]\!]_\sigma\}$, $[\![e_1 + e_2]\!]_\sigma := [\![e_1]\!]_\sigma \cup [\![e_2]\!]_\sigma$, $[\![e_1\, e_2]\!]_\sigma := [\![e_1]\!]_\sigma\, [\![e_2]\!]_\sigma$, and $[\![e^{\star}]\!]_\sigma := [\![e]\!]_\sigma^{*}$. A *solution* of S is an assignment σ such that $\sigma(q_i) = [\![c_i + \ell_{i,1}\, q_1 + \ldots + \ell_{i,n}\, q_n]\!]_\sigma$ for all $1 \leq i \leq n$. We recall from [6] that an LSH S has a unique solution σ_S, and that σ_S binds the elements of Q to RHLs. In general, $L(\mathcal{A}, q) = \sigma_S(q)$ for all $q \in Q$.

Example 2. Let $\mathcal{A} = (\{q_1, q_2, q_3\}, \{a, b\}, \{q_1\}, \Delta)$ with

$$\Delta := \{\epsilon \to q_1,\ a(q_1)\, q_2 \to q_1,\ b(q_2)\, q_2 \to q_1,\ a(q_1)\, q_1 \to q_2,\\ a(q_1)\, q_2 \to q_2,\ a(q_3)\, q_2 \to q_3\}.$$

Then $\mathcal{A}$ is nondeterministic because the transition rules $a(q_1)\, q_2 \to q_1$ and $a(q_1)\, q_2 \to q_2$ have the same left hand side. The LSH representation of $\mathcal{A}$ is the pair $(\{q_1\}, S)$ where S is the LSH

$$S : \begin{cases} q_1 = 1 + (a(q_1) + b(q_2))\, q_2 \\ q_2 = 0 + a(q_1)\, q_1 + a(q_1)\, q_2 \\ q_3 = 0 + a(q_3)\, q_2 \end{cases}$$

It is not hard to see from the structure of S that state q_3 is not accessible, therefore $\mathcal{A}$ is not a reduced LHA. □

The subset construction of a deterministic, complete, and reduced finite tree automaton equivalent to a nondeterministic finite tree automaton can be easily adapted to linear hedge automata, so we can conclude the following lemma.

Lemma 1. *For every LHA $\mathcal{A}$ there exists a deterministic, complete, and reduced LHA $\mathcal{A}_{\mathrm{d}}$ such that $L(\mathcal{A}) = L(\mathcal{A}_{\mathrm{d}})$.*

Thus we can represent any RHL by a deterministic, complete, and reduced LHA.

Example 3. The determinization of the LHA $\mathcal{A}$ from Example 2 produces the deterministic, complete, and reduced LHA $\mathcal{A}_{\mathrm{d}} := (\{s_1, s_2, s_3, s_4\}, \Sigma, \{s_1, s_4\}, \Delta_{\mathrm{d}})$ whose LSH representation is $(\{s_1, s_4\}, S)$ with

$$S : \begin{cases} s_1 = 1 + (b(s_2) + b(s_4))\, s_2 + (b(s_2) + b(s_4))\, s_4 \\ s_2 = 0 + (a(s_1) + a(s_4))\, s_1 \\ s_3 = 0 + (a(s_2) + a(s_3) + \sum_{i=1}^{4} b(s_i))\, s_1 \\ \qquad + (a(s_2) + a(s_3) + b(s_1) + b(s_3))\, s_2 \\ \qquad + (\sum_{i=1}^{4} a(s_i) + \sum_{i=1}^{4} b(s_i))\, s_3 \\ \qquad + (a(s_2) + a(s_3) + b(s_1) + b(s_3))\, s_4 \\ s_4 = 0 + (a(s_1) + a(s_4))\, s_2 + (a(s_1) + a(s_4))\, s_4. \end{cases}$$

2.1 Factorizations of Regular Hedge Languages

The factorization theory of RHLs [5] is a natural generalization of the factorization theory of regular languages [2]. We recall here main definitions and results from [5]. An *n-subfactorization* of an RHL H is a tuple $(H_1, \ldots, H_n)$ of hedge languages such that the concatenation $H_1 \cdots H_n$ is a subset of H. If we define the relation

$$(H_1, \ldots, H_n) < (H'_1, \ldots, H'_n) :\Leftrightarrow \begin{array}{l} H_i \subseteq H'_i \text{ for all } i \in \{1, \ldots, n\} \text{ and} \\ H_j \neq H'_j \text{ for some } j \in \{1, \ldots, n\} \end{array}$$

then we can talk about $<$-maximal n-subfactorizations of H, also known as *n-factorizations* of H. The components of such n-factorizations are called *factors*. A *left factor* of H is the first factor of an n-factorization of H, and a *right factor* is the last factor of an n-factorization of H. For the rest of this section we assume implicitly that H, L, M are RHLs, $\mathtt{F}(H)$ is the set of factors of H, $\mathtt{LF}(H)$ is the set of left factors of H and $\mathtt{RF}(H)$ is the set of right factors of H.

Example 4. The RHL $H = \{a(\epsilon)^m\, b(\epsilon)^n a(\epsilon)^p \mid m, n, p \in \mathbb{N}\}$ over signature $\Sigma = \{a, b, c\}$ has five possible 2-factorizations (H_1, H_2): either $(\mathcal{H}(\{a, b, c\}), \emptyset)$, or $(\{a(\epsilon)^m \mid m \in \mathbb{N}\}, H)$, or $(\{a(\epsilon)^m b(\epsilon)^n \mid m, n \in \mathbb{N}\}, \{b(\epsilon)^n a(\epsilon)^p \mid n, p \in \mathbb{N}\})$, or $(H, \{a(\epsilon)^p \mid p \in \mathbb{N}\})$, or $(\emptyset, \mathcal{H}(\{a, b, c\}))$. In this case we have

$$\mathtt{LF}(H) = \{\mathcal{H}(\{a, b, c\}), \{a(\epsilon)^m \mid m \in \mathbb{N}\}, \{a(\epsilon)^m b(\epsilon)^n \mid m, n \in \mathbb{N}\}, H, \emptyset\} \text{ and}$$
$$\mathtt{RF}(H) = \{\emptyset, H, \{b(\epsilon)^n a(\epsilon)^p \mid n, p \in \mathbb{N}\}, \{a(\epsilon)^p \mid p \in \mathbb{N}\}, \mathcal{H}(\{a, b, c\})\}.$$

A remarkable fact is that the factors of an RHL H are finitely many [5]. Moreover, the factors of H can be indexed such that $\mathtt{F}(H) = \{F_{i,j} \mid 1 \leq i, j \leq p\}$ and:

- $F_{i,k}\, F_{k,j} \subseteq F_{i,j}$ for all $i, j, k \in \{1, \ldots, p\}$, and
- There exist $l, r \in \{1, \ldots, p\}$ such that $F_{l,r} = H$ and, for any k-subfactorization $(H_1, \ldots, H_k)$ of H there exist $u_1, \ldots, u_{k+1} \in \{1, \ldots, p\}$ with $u_1 = l$ and $u_{k+1} = r$, such that $H_i \subseteq F_{u_i, u_{i+1}}$ for all $i \in \{1, \ldots, k\}$.

The matrix $(F_{i,j})_{1 \leq i,j \leq p}$ is called the *factor matrix* of H.

In order to achieve a better characterization of the factors of an RHL, we introduce the following auxiliary notions:

- $h^{-1}H := \{h' \in \mathcal{H}(\Sigma) \mid h\, h' \in H\}$ is the *left quotient* of H with respect to a hedge h,
- $H\, h^{-1} := \{h' \in \mathcal{H}(\Sigma) \mid h'\, h \in H\}$ is the *right quotient* of H with respect to a hedge h,
- $L \rhd M := \{h' \in \mathcal{H}(\Sigma) \mid \forall h \in L.\, h\, h' \in M\}$ is the *product derivative* of M with respect to L,
- $M \lhd L := \{h' \in \mathcal{H}(\Sigma) \mid \forall h \in L.\, h'\, h \in M\}$ is the *product antiderivative* of M with respect to L.
- The set of *hedge derivatives* of H is $\partial(H) := \{h^{-1}H \mid h \in \mathcal{H}(\Sigma)\}$.

We recall from [5,6] that for any RHLs H, M, L we have:

- $\partial(H)$ is a finite set of RHLs,
- $\mathtt{RF}(H)$ is the closure of $\partial(H)$ under intersection, i.e., $R \in \mathtt{RF}(H)$ iff $R \in \partial(H)$ or there exist $M_1, \ldots, M_p \in \partial(H)$, $p > 1$, such that $R = M_1 \cap \cdots \cap M_p$,
- $M \lhd L$ is an RHL, and $M \lhd L = \bigcap_{h \in L}(M\, h^{-1})$,
- If $\mathtt{LF}(H) = \{L_1, \ldots, L_p\}$ then we can define $F_{i,j} := L_i \rhd L_j$,
- If $\mathtt{RF}(H) = \{R_1, \ldots, R_p\}$ then we can define $F_{i,j} := R_i \lhd R_j$.

In the rest of this paper we will investigate how to compute the factor matrix of an RHL by making use of these properties and of the representation of RHLs by deterministic, complete, and reduced LHAs. Our approach is to compute first $\{R_1, \ldots, R_p\} := \mathtt{RF}(H)$ and then to define the factor matrix by $F_{i,j} := R_i \lhd R_j$ for all $i, j \in \{1, \ldots, p\}$.

3 Computation of Right Factors

In this section we investigate the problem of computing all right factors of an RHL H accepted by a deterministic, complete, and reduced LHA $\mathcal{A}$. We solve this problem in two steps. First, we compute LHAs for the RHLs of the finite set $\partial(H)$. Then, we compute LHAs for the right factors by using the automata computed in the first step.

Suppose $\mathcal{A} = (Q, \Sigma, Q_{\mathtt{f}}, \Delta)$ with $Q = \{q_1, \ldots, q_n\}$, and let $(Q_{\mathtt{f}}, S)$ be the LSH representation of $\mathcal{A}$. To simplify the analysis of the structure of $\partial(H)$, we define for every $q, q' \in Q$, $Q' \subseteq Q$, and $a \in \Sigma$ the sets

$$\mathtt{cut}_\Delta(q, a(q')) := \{q'' \in Q \mid a(q')\, q'' \to q \in \Delta\},$$
$$\mathtt{cut}_\Delta(Q', a(q')) := \bigcup_{q \in Q'} \mathtt{cut}_\Delta(q, a(q')).$$

These notions will help us to identify a finitary characterization of the set of RHLs $\partial(H)$. The key observation is the following lemma.

Lemma 2. *$a(h)^{-1}L(\mathcal{A}, Q') = L(\mathcal{A}, \mathtt{cut}_\Delta(Q', a(q)))$ holds for all $a \in \Sigma$, $q \in Q$, $Q' \subseteq Q$, and $h \in L(\mathcal{A}, q)$.*

The following result is an easy corollary of Lemma 2.

Corollary 1. *Let $\mathcal{G}_\Delta$ be the directed graph whose nodes are the subsets of Q, and set of edges is $\{Q' \to Q'' \mid a \in \Sigma, q \in Q, Q'' = \mathtt{cut}_\Delta(Q', a(q))\}$. Then*

$$\partial(H) = \{L(\mathcal{A}, Q') \mid \textit{there exists a path from } Q_\mathtt{f} \textit{ to } Q' \textit{ in } \mathcal{G}_\Delta\}.$$

Once we know $\partial(H)$, we can compute $\mathtt{RF}(H)$ as the closure of $\partial(H)$ under intersections. At this stage, it is very useful to recall that, since $\mathcal{A}$ is deterministic, we have $L(\mathcal{A}, q) \cap L(\mathcal{A}, q') = \emptyset$ whenever $q, q' \in \mathcal{A}$ and $q \neq q'$. Therefore $L(\mathcal{A}, Q_1) \cap L(\mathcal{A}, Q_2) = L(\mathcal{A}, Q_1 \cap Q_2)$ for all subsets Q_1 and Q_2 of Q. Thus, if $\partial(H) = \{L(\mathcal{A}, Q_i) \mid 1 \leq i \leq m\}$, then

$$\mathtt{RF}(H) = \Big\{L(\mathcal{A}, \bigcap_{i \in I} Q_i) \mid \emptyset \neq I \subseteq \{1, \ldots, m\}\Big\}.$$

Example 5. Let $\mathcal{A}_\mathrm{d} = (\{s_1, s_2, s_3, s_4\}, \Sigma, \Delta_\mathrm{d}, \{s_1, s_4\})$ be the LHA from Example 3, and $H = L(\mathcal{A}_\mathrm{d})$. The set of nodes reachable from $\{s_1, s_4\}$ in $\mathcal{G}_{\Delta_\mathrm{d}}$ is $\{Q_1, Q_2, Q_3\}$ where $Q_1 := \emptyset$, $Q_2 := \{s_2, s_4\}$, and $Q_3 := \{s_1, s_2, s_4\}$. Therefore $\partial(H) = \{L(\mathcal{A}_\mathrm{d}, Q_i) \mid 1 \leq i \leq 3\}$. Since $\big\{\bigcap_{i \in I} Q_i \mid \emptyset \neq I \subseteq \{1, 2, 3\}\big\} = \{Q_1, Q_2, Q_3\}$, we conclude that $\mathtt{RF}(H) = \{L(\mathcal{A}_\mathrm{d}, Q_i) \mid 1 \leq i \leq 3\}$.

A slightly surprising fact is that $L(\mathcal{A}_\mathrm{d}, \{s_1, s_4\}) = H \in \mathtt{RF}(H)$ but $\{s_1, s_4\} \notin \{Q_1, Q_2, Q_3\}$. As it turns out, we have $L(\mathcal{A}_\mathrm{d}, \{s_1, s_4\}) = L(\mathcal{A}_\mathrm{d}, Q_3)$. □

4 Factor Matrix Computation

In this section we address the problem of computing the factor matrix of an RHL represented by an LHA. To be more specific, from now on we assume $H = L(\mathcal{A})$ where $\mathcal{A} = (Q, \Sigma, Q_\mathtt{f}, \Delta)$ with $Q = \{q_1, \ldots, q_n\}$ is a deterministic, complete, and reduced LHA. We saw in the previous section how to compute $Q_1, \ldots, Q_p \subseteq Q$ such that $\mathtt{RF}(H) = \{L(\mathcal{A}, Q_i) \mid 1 \leq i \leq p\}$. Then we can define the factor matrix $(F_{i,j})_{1 \leq i,j \leq p}$ where $F_{i,j} := L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$.

In the following two subsections we will show how to compute LHAs for the product antiderivatives $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$ when $1 \leq i, j \leq p$. First, we indicate in Subsect. 4.1 a simple algorithm to compute LHAs for the product antiderivatives $L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q)$ when $q \in Q$ and $Q' \subseteq Q$. Then, in Subsect. 4.2 we propose an algorithm that computes LHAs for the RHLs of the product antiderivatives $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$, $1 \leq i, j \leq p$ from LHAs for the product antiderivatives $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, q)$, $1 \leq i \leq p$, where $q \in Q$.

From now on we assume that the LSH representation of $\mathcal{A}$ is $(Q_{\mathtt{f}}, S)$ where

$$S : \begin{cases} q_1 = c_1 + \ell_{1,1}\, q_1 + \cdots + \ell_{1,n}\, q_n \\ \vdots \\ q_n = c_n + \ell_{n,1}\, q_1 + \cdots + \ell_{n,n}\, q_n \end{cases}$$

and that σ is the unique solution of S. Also, we assume that Q' is subset of Q.

4.1 LHA Computation for $L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q)$

Let $Q = \{q_1, \ldots, q_n\}$. We fix arbitrarily a state $q \in Q$. Then for every $1 \leq i \leq n$ and $h \in L(\mathcal{A}, q)$ we have

$$[\![q_i]\!]_\sigma\, h^{-1} = [\![c_i + \ell_{i,1}\, q_1 + \cdots + \ell_{i,n}\, q_n]\!]_\sigma\, h^{-1} = [\![d_i + \ell_{i,1}\, r_1^{(q)} + \cdots + \ell_{i,n}\, r_n^{(q)}]\!]_{\theta_q}$$

where

- $d_i = 1$ if $h \in [\![q_i]\!]_\sigma$ and $d_i = 0$ otherwise,
- θ_q is the extension of σ with the following bindings for the fresh variables $r_1^{(q)}, \ldots, r_n^{(q)}$: $\theta_q(r_i^{(q)}) := [\![q_i]\!]_\sigma\, h^{-1}$ for all $1 \leq i \leq n$.

Since $\mathcal{A}$ is deterministic, we have $L(\mathcal{A}, q_i) \cap L(\mathcal{A}, q) = \emptyset$ whenever $q_i \neq q$. Therefore $d_i = 1$ iff $h \in [\![q_i]\!]_\sigma$ iff $\emptyset \neq [\![q_i]\!]_\sigma \cap [\![q]\!]_\sigma$ iff $q_i = q$. Thus, if we define for all q', q'' and $1 \leq i \leq n$:

$$\rho_q(q_i) := r_i^{(q)}, \quad \rho_q(Q') := \{\rho_q(q') \mid q' \in Q'\}, \quad \delta_{q',q''} := \begin{cases} 1 \text{ if } q' = q'' \\ 0 \text{ otherwise} \end{cases}, \quad \text{and}$$

$$S_q : \begin{cases} r_l^{(q)} = \delta_{q_l,q} + \ell_{l,1}\, r_1^{(q)} + \cdots + \ell_{l,n}\, r_n^{(q)} & (1 \leq l \leq n) \\ q_l = c_l \quad\quad + \ell_{l,1}\, q_1 + \cdots + \ell_{l,n}\, q_n & (1 \leq l \leq n) \end{cases}$$

then $(S_q, \rho_q(Q'))$ is an LSH representation for $L(\mathcal{A}, Q')\, h^{-1}$. Since

$$L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q) = \bigcap_{h \in L(\mathcal{A},q)} L(\mathcal{A}, Q')\, h^{-1} \quad \text{and}$$

$$L(\mathcal{A}, Q')\, h_1^{-1} = L(\mathcal{A}, Q')\, h_2^{-1} \text{ for all } h_1, h_2 \in L(\mathcal{A}, q),$$

we learn that $L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q) = L(\mathcal{A}, Q')\, h^{-1}$. Let $R_q := \{r_1^{(q)}, \ldots, r_n^{(q)}\}$ and $\mathcal{A}_q := (Q \cup R_q, \Sigma, \rho_q(Q_{\mathtt{f}}), \Delta_q)$ be the LHA whose LSH representation is (Q, S_q). Then $L(\mathcal{A}, Q') \lhd L(\mathcal{A}, q) = L(\mathcal{A}_q, \rho_q(Q'))$.

Note that the computation of the LSH representation of $\mathcal{A}_q$ involves only duplications of the equations of S followed by some trivial variable renamings and alterations of their constant parts. Another useful observation is that every LHA $\mathcal{A}_{q_i}$ with $1 \leq i \leq n$ is *almost* deterministic, because the only transition rules of Δ_{q_i} with same left hand side are $\epsilon \to r_i^{(q_i)}$ and the ϵ-transition of Δ. The following lemma is an easy consequence of this observation.

Lemma 3. *$L(\mathcal{A}_q, r_i^{(q)}) \cap L(\mathcal{A}_q, r_j^{(q)}) = \emptyset$ for all $i, j \in \{1, \ldots, n\}$ and $i \neq j$.*

4.2 LHA Computation for $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$

In order to compute an LHA for $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$, we can proceed as follows. If $Q_j = \emptyset$ then $L(\mathcal{A}, Q_j) = \emptyset$ and $(L(\mathcal{A}, Q_i) \lhd \emptyset) = \mathcal{H}(\Sigma) = L(\mathcal{A}, Q)$, where the last equality follows from the fact that $\mathcal{A}$ is complete. If $Q_j \neq \emptyset$ then

$$L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j) = \bigcap_{q \in Q_j} (L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, q)) = \bigcap_{q \in Q_j} L(\mathcal{A}_q, \rho_q(Q_i)).$$

Let $Q_j := \{q_{k_1}, \ldots, q_{k_d}\}$ with $d \geq 1$, and $\mathtt{I}_i := \{l \mid q_l \in Q_i\}$. For every $q \in Q_j$, the system of equations S_q can be rewritten as follows:

$$S_q : \begin{cases} r_l^{(q)} = \delta_{q_l,q} + \sum_{a \in \Sigma} \sum_{u=1}^{n} a(q_u)\, e_{a(q_u),l,q} & (1 \leq l \leq n) \\ q_l = c_l \qquad\qquad +\ell_{l,1}\, q_1 + \cdots + \ell_{l,n}\, q_n & (1 \leq l \leq n) \end{cases}$$

where $e_{a(q_u),l,q} = \sum_{r \in \mathtt{cut}_{\Delta_q}(r_l^{(q)}, a(q_u))} r$. Suppose θ_q is the unique solution of S_q for every $q \in Q_j$. Then

$$\begin{aligned} L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j) &= \textstyle\bigcap_{q \in Q_j} L(\mathcal{A}_q, \rho_q(Q_i)) \\ &= \textstyle\bigcap_{q \in Q_j} \bigcup_{l \in \mathtt{I}_i} [\![r_l^{(q)}]\!]_{\theta_q} = \bigcup_{(l_1,\ldots,l_d) \in \mathtt{I}_i^d} \bigcap_{s=1}^{d} [\![r_{l_s}^{(q_{k_s})}]\!]_{\theta_{q_{k_s}}}. \end{aligned}$$

where $\mathtt{I}_i^d = \underbrace{\mathtt{I}_i \times \cdots \times \mathtt{I}_i}_{d \text{ times}}$. At this stage, we can start generating an LSH representation for the product antiderivative $L(\mathcal{A}, Q_i) \lhd L(\mathcal{A}, Q_j)$. Our algorithm works in stages, by keeping track of (1) a list E of equations generated so far, (2) a set $V_{\mathtt{p}}$ of variables produced so far, and (3) a set $V_{\mathtt{a}}$ of variables assigned so far. In addition to the variables from Q and the equations of S, the algorithm produces fresh variables of the form $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ with $l_1, \ldots, l_d \in \{1, \ldots, n\}$, and corresponding equations such that the solution of the new system of equations binds $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ to $\bigcap_{s=1}^{d} [\![r_{l_s}^{(q_{k_s})}]\!]_{\theta_{q_{k_s}}}$. The main goal is to produce equations for all variables of the set $\{r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})} \mid (l_1, \ldots, l_d) \in \mathtt{I}_i^d\}$ and the other fresh variables that show up in the construction process. The initial values of $V_{\mathtt{p}}$ and $V_{\mathtt{a}}$ are $V_{\mathtt{p}} = Q \cup \{r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})} \mid (l_1, \ldots, l_d) \in \mathtt{I}_i^d\}$, $V_{\mathtt{a}} = Q$, and E consists of the equations of S. At every stage, the algorithm enlarges $V_{\mathtt{p}}$, $V_{\mathtt{a}}$, and E by performing the following operations:

- Select $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})} \in V_{\mathtt{p}} \setminus V_{\mathtt{a}}$; $V_{\mathtt{a}} := V_{\mathtt{a}} \cup \{r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}\}$
- Produce an equation for $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ such that E extended with it has a solution that binds $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ to $\bigcap_{s=1}^{d} [\![r_{l_s}^{(q_{k_s})}]\!]_{\theta_{q_{k_s}}}$. The right hand side of the equation for $r_{(l_1,\ldots,l_d)}^{(q_{k_1},\ldots,q_{k_d})}$ is produced by intersecting the right hand sides of the equations

$$\begin{aligned} &(r_{l_1}^{(q_{k_1})} = \delta_{q_{l_1},q_{k_1}} + \textstyle\sum_{a \in \Sigma} \sum_{u=1}^{n} a(q_u)\, e_{a(q_u),l_1,q_{k_1}}) \in S_{q_{k_1}} \text{ with solution } \theta_{k_1} \\ &\quad\vdots \\ &(r_{l_d}^{(q_{k_d})} = \delta_{q_{l_d},q_{k_d}} + \textstyle\sum_{a \in \Sigma} \sum_{u=1}^{n} a(q_u)\, e_{a(q_u),l_d,q_{k_d}}) \in S_{q_{k_d}} \text{ with solution } \theta_{k_d} \end{aligned}$$

and using the facts that $[\![a(q_u)\, e_{a(q_u),l_s,q_{k_s}}]\!]_{\theta_{q_{k_s}}} = [\![a(q_u)]\!]_\sigma\, [\![e_{a(q_u),l_s,q_{k_s}}]\!]_{\theta_{q_{k_s}}}$, and $[\![a(q)]\!]_\sigma \cap [\![b(q')]\!]_\sigma = \emptyset$ whenever $(a,q) \neq (b,q')$. It follows that $[\![a(q)\, e]\!]_{\theta_{q_{k_i}}} \cap [\![b(q')\, e']\!]_{\theta_{q_{k_j}}} = \emptyset$ whenever $(a,q) \neq (b,q')$. We obtain

$$r^{(q_{k_1},\ldots,q_{k_d})}_{(l_1,\ldots,l_d)} = \prod_{s=1}^{d} \delta_{q_{l_s},q_{k_s}} + \sum_{a\in\Sigma}\sum_{u=1}^{n} a(q_u)\Big(\sum_{(m_1,\ldots,m_d)\in \mathtt{S}^{a(q_u)}_{(l_1,\ldots,l_d)}} r^{(q_{k_1},\ldots,q_{k_d})}_{(m_1,\ldots,m_d)}\Big) \quad (1)$$

where

$$\mathtt{S}^{a(q_u)}_{(l_1,\ldots,l_d)} = \Big\{(m_1,\ldots,m_d) \mid \forall s \in \{1,\ldots,d\}.\, r^{(q_{k_s})}_{m_s} \in \mathtt{cut}_{\Delta_{q_{k_s}}}(r^{(q_{k_s})}_{l_s}, a(q_u))\Big\}.$$

- Append equation (1) to E;
  $V_{\mathtt{p}} := V_{\mathtt{p}} \cup \bigcup_{a\in\Sigma}\bigcup_{u=1}^{n}\{r^{(q_{k_1},\ldots,q_{k_d})}_{(m_1,\ldots,m_d)} \mid (m_1,\ldots,m_d) \in \mathtt{S}^{a(q_u)}_{(l_1,\ldots,l_d)}\}$.
- If $V_{\mathtt{p}} = V_{\mathtt{a}}$, then stop and return $(\{r^{(q_{k_1},\ldots,q_{k_d})}_{(l_1,\ldots,l_d)} \mid (l_1,\ldots,l_d) \in \mathtt{I}^d_i\}, E)$. This is the LSH representation of an LHA for $L(\mathcal{A},Q_i) \lhd L(\mathcal{A},Q_j)$.

The algorithm will eventually stop because $V_{\mathtt{p}}$ and $V_{\mathtt{a}}$ are subsets of the finite set $Q \cup \{r^{(q_{k_1},\ldots,q_{k_d})}_{(l_1,\ldots,l_d)} \mid l_1,\ldots,l_d \in \{1,\ldots,n\}\}$, therefore they can not increase forever. □

Let us now look back at Example 1 and see how factorization theory helps there to infer the type of the tuple (x,y) of variables of pattern $f(x\,y)\,f(y\,x)$ when matched against inputs of type $f(a^\star b^\star)^\star$. Let's write $P \ll T$ for the type inference problem of the tuple $(x_1,\ldots,x_n)$ of variables of a pattern P when matched against inputs of type T. This problem can be solved by recursion on the structure of P. In this example, we have $P_1\,P_2 \ll T$ where $P_1 = f(x\,y)$, $P_2 = f(y\,x)$, and $T = f(a^\star\,b^\star)^\star$. This problem can be reduced to solving the problems $P_1 \ll T_1$ and $P_2 \ll T_2$, where T_1, T_2 are nonempty and $\subseteq$-maximal types such that $T_1\,T_2 \subseteq T$. Such tuples (T_1,T_2) are called 2-factorizations of T, and can be computed effectively. The only such 2-factorization of $f(a^\star b^\star)^\star$ is $(T_1,T_2) = (f(a^\star b^\star)^\star, f(a^\star b^\star)^\star)$, therefore $P \ll f(a^\star b^\star)^\star$ is reduced to $f(x\,y) \ll f(a^\star b^\star)^\star$ and $f(y\,x) \ll f(a^\star b^\star)^\star$. The first subproblem is equivalent to $x\,y \ll a^\star b^\star$, and the second subproblem is equivalent to $y\,x \ll a^\star\,b^\star$. To solve these subproblems, we look for nonempty types T_3, T_4 such that (T_3,T_4) is a 2-factorization of $a^\star b^\star$. There are two possibilities: either $(T_3,T_4) = (a^\star, a^\star b^\star)$, or $(T_3,T_4) = (a^\star b^\star, b^\star)$. Thus $x\,y \ll a^\star b^\star$ reduces to ($x \ll T_3$ and $y \ll T_4$) where $(T_3,T_4) \in \{(a^\star, a^\star b^\star), (a^\star b^\star, b^\star)\}$. This means (x,y) is of type $(a^\star \times a^\star b^\star) + (a^\star b^\star \times b^\star)$ where $\times$ denotes cartesian product and $+$ denotes union. Similarly, the second subproblem $y\,x \ll a^\star b^\star$ reduces to the fact that (x,y) is of type $(a^\star b^\star \times a^\star) + (b^\star \times a^\star b^\star)$. We conclude that (x,y) is of type

$$\begin{aligned}&((a^\star \times a^\star b^\star) + (a^\star b^\star \times b^\star)) \cap ((a^\star b^\star \times a^\star) + (b^\star \times a^\star b^\star))\\ &= (a^\star \times a^\star) + (1 \times a^\star b^\star) + (a^\star b^\star \times 1) + (b^\star \times b^\star).\end{aligned}$$

From here, one can easily compute an over-approximation of the type of $x\,y\,x$ as $a^\star a^\star a^\star + 1\,a^\star b^\star\,1 + a^\star b^\star\,1 a^\star b^\star + b^\star\,b^\star\,b^\star = a^\star b^\star a^\star b^\star$, therefore $f(x\,y\,x)$ is of type $f(a^\star b^\star a^\star b^\star)$.

5 Discussion

The approach to compute the factor matrix of an RHL proposed in this paper is more efficient than that from our previous works [5,6]. Our main insight is that, if we want to compute the factor matrix of an RHL, it is better to start with a representation by a deterministic, reduced, and complete LHA. Next, we noticed that the computation of the factor matrix of an RHL is easier to compute via computations of product antiderivatives between right factors than via product derivatives between left factors.

The following discussion aims at explaining the main differences between our former algorithms and those given in this paper. In [5,6], we considered an RHL H with an LSH representation of the form $(\{x_1\}, S)$. Such a representation corresponds to a nondeterministic LHA with one final state. The computation of the factor matrix is achieved in 3 steps. Step 1 computes representations for the elements of the finite set $\{H\, h^{-1} \mid h \in \mathcal{H}(\Sigma)\}$. Step 2 computes representations for the elements of $\mathtt{LF}(H)$ by using the fact that $\mathtt{LF}(H)$ coincides with the closure of $\{H\, h^{-1} \mid h \in \mathcal{H}(\Sigma)\}$ under intersection. Step 3 makes use of the fact that, if we know $\{L_1, \ldots, L_p\} := \mathtt{LF}(H)$, then the factor matrix of H is $(F_{i,j})_{1 \leq i,j \leq p}$ with $F_{i,j} := L_i \rhd L_j$. Step 1 amounts to the computation of the least fixed point of a monotone operator on the set $2^{2^{\mathcal{X}} \times 2^{\mathcal{X}}}$ where $\mathcal{X}$ is the set of variables of S. The construction of LSHs for the elements of $\mathtt{LF}(H)$ performs intersections of equations in a similar way as the construction of product antiderivatives of right factors, except for the fact that it must account for the possible nonempty intersections $[\![a(x)]\!]_\sigma \cap [\![b(x')]\!]_\sigma$ when $(a, x) \neq (b, x')$. Finally, step 3 finds the LSH representations of the factor matrix of H via product derivative computations between the left factors of H.

Our new approach is to compute the factor matrix of H starting from a reduced, complete, and deterministic LHA $\mathcal{A} = (Q, \Sigma, Q_{\mathtt{f}}, \Delta)$ for H. To achieve a fair comparison with our former approach, we should account for the fact that the determinization algorithm of a nondeterministic LHA with n states has, in the worst case, 2^n states. Thus, we assume the worst situation: we compare the former approach when $\mathcal{X}$ has n elements, with the new approach when Q has 2^n states. The newly proposed computation of the factor matrix is carried out in 3 steps. The first step computes a set $\{Q_1, \ldots, Q_p\}$ of subsets of Q such that $\partial(H) = \{L(\mathcal{A}, Q_i) \mid i \in \{1, \ldots, p\}\}$. The sets $\{Q_1, \ldots, Q_p\}$ are the nodes reachable from $Q_{\mathtt{f}}$ in a directed graph $\mathcal{G}_\Delta$ whose nodes are the subsets of Q, thus, a graph with 2^{2^n} nodes.

We claim that new step 1 is more efficient than former step 1 because

- Former step 1 computes at most 2^n LSH representations for the RHLs in $\{H\, h^{-1} \mid h \in \mathcal{H}(\Sigma)\}$ via the computation of the least fixed point of a monotone operator over a set of size $2^{2^{2n}}$.
- New step 1 computes at most 2^n sets $Q_1, \ldots, Q_p \subseteq Q$ such that $\partial(H) = \{L(\mathcal{A}, Q_i) \mid i \in \{1, \ldots, p\}\}$.

– The computation of $Q_1, \ldots, Q_p$ is straightforward, whereas the operations required to compute the LSH representations for $\{H\, h^{-1} \mid h \in \mathcal{H}(\Sigma)\}$ are much more involved.

Also, we claim that new step 2 is more efficient then former step 2 because

– Former step 2 computes LSHs for left factors as intersections of at most 2^n RHLs represented by the nondeterministic LSHs computed in former step 1.
– In new step 2, a right factor is $L(\mathcal{A}, Q')$ where Q' is an intersection of sets from $\{Q_1, \ldots, Q_p\}$. Since $p \leq 2^n$, Q' is obtained by intersecting at most 2^n sets. This is more efficient than intersecting at most 2^n RHLs represented by nondeterministic LHSs.

New step 3 computes the factor matrix of H via product antiderivatives of right factors, whereas former step 3 computes it via product derivatives of left factors. The new approach is more efficient than the former one, mainly because both of them rely on computing representations for intersections of RHLs, but these computations can be simplified under the assumption $L(\mathcal{A}, q) \cap L(\mathcal{A}, q') = \emptyset$ whenever $q \neq q'$, which follows from the fact that $\mathcal{A}$ is deterministic.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers, San Francisco (2000)
2. Conway, J.H.: Regular Algebra and Finite Machines. Mathematics series. Chapman and Hall, Boca Raton (1971)
3. Hosoya, H., Pierce, B.C.: Regular expression pattern matching for XML. Journal of Functional Programming 13(6), 961–1004 (2002)
4. Hosoya, H., Pierce, B.C.: XDuce: A statically typed XML processing language. ACM Trans. Internet Techn. 3(2), 117–148 (2003)
5. Marin, M., Crăciun, A.: Factorizations of regular hedge languages. In: Proceedings of 11th Intl. Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2009), Timişoara, Romania. IEEE, Los Alamitos (September 2009), `http://www.score.cs.tsukuba.ac.jp/~mmarin/synasc2009.pdf`
6. Marin, M., Kutsia, T.: Linear systems for regular hedge languages. In: Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Novickis, L. (eds.) ADBIS 2009 Workshops. LNCS, vol. 5968, pp. 104–112. Springer, Heidelberg (2010)
7. Murata, M.: Extended path expressions for XML. In: Proceedings of the 20th Symposium on Principles of Database Systems (PODS 2001), Santa Barbara, California, USA, pp. 126–137. ACM, New York (2001)
8. Thatcher, J.: There is a lot more to finite automata theory than you would have thought. Technical Report RC-2852 (#13407), IBM Thomas J. Watson Research Center, Yorktown, New York (1970)
9. Thatcher, J., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory 2(1), 57–81 (1968)