Rewrite-Based Verification of XML Updates

Florent Jacquemard, Michael Rusinowitch

Rewrite-Based Verification of XML Updates *

Florent Jacquemard
INRIA Saclay - IdF & LSV UMR
61 av. du pdt Wilson 94230 Cachan, France
florent.jacquemard@inria.fr

Michael Rusinowitch
INRIA Nancy - Grand Est & LORIA UMR
615 rue du Jardin Botanique
54602 Villers-les-Nancy, France
rusi@loria.fr

Abstract

We propose a model for XML update primitives of the W3C XQuery Update Facility as parameterized rewriting rules of the form: "insert an unranked tree from a regular tree language L as the first child of a node labeled by a". For these rules, we give type inference algorithms, considering types defined by several classes of unranked tree automata. These type inference algorithms are directly applicable to XML static typechecking, which is the problem of verifying whether, a given document transformation always converts source documents of a given input type into documents of a given output type. We show that typechecking for arbitrary sequences of XML update primitives can be done in polynomial time when the unranked tree automaton defining the output type is deterministic and complete, and that it is EXPTIME-complete otherwise.
We then apply the results to the verification of access control policies for XML updates. We propose in particular a polynomial time algorithm for the problem of local consistency of a policy, that is, for deciding the non-existence of a sequence of authorized update operations starting from a given document that simulates a forbidden update operation.

Categories and Subject Descriptors D.2.4 [*SOFTWARE ENGINEERING*]: Software/Program VerificationFormal methods, model checking]

General Terms Verification, Languages, Theory, Security

Keywords XML Updates, Static Typechecking, XML Access Control Policies, Term Rewriting, Hedge Automata

1. Introduction

XQuery language has been extended to XQuery Update Facility [8] in order to provide convenient means of modifying XML documents or data. The language is a candidate recommendation from W3C and adds imperative operations that permit one e.g. to update some parts of a document while leaving the rest unchanged. This includes rename, insert, replace and delete operations at the node level. Compared to other transformation languages (such a XSLT), XQuery Update Facility is considered to offer concise, readable solutions.

A central problem in XML document processing is *static typechecking*. This problem amounts to verifying at compile time that every output XML document which is the result of a specified query or transformation applied to an input document with a valid input type has a valid output type. However for transformation languages such as the one provided by XQuery Update Facility, the output type of (iterated) applications of update primitives are not easy to predict. Another important issue for XML data processing is the specification and enforcement of access policies. A large amount of work has been devoted to secure XML querying. But most of the work focus on read-only rights, and very few have considered update rights for a model based on XQuery Update Facility operations e.g. [6, 17].

In the domain of infinite state systems and program verification, several approaches such as regular model checking rely on algorithms computing the rewrite closure of tree automata languages, see e.g. [5, 15, 19]. It seems natural to consider such tree automata techniques for verification problems related to the typing of XML documents and XML transformations, in particular XML updates [8]. Indeed, XML documents are commonly represented as finite labeled unranked trees, and most of the typing formalisms currently used for XML are based on finite tree automata [30, 35].

A standard approach to XML typechecking is forward (resp. backward) *type inference*, that is, the computation of an output (resp. input) XML type given an input (resp. output) type and a tree transformation. Then the typechecking itself can be reduced to the verification of inclusion of the computed type in the given output or input type, see [28] for an example of backward type inference procedure.

In this paper, we consider the problem of typechecking arbitrary sequences of operations taken in a given set of atomic update primitives. We propose a modeling of (possibly infinite) sets of primitive update operations of the W3C XQuery Update Facility proposal [8] in terms of rewrite rules with parameters and XPath expressions for the selection of the rewrite positions. The update operations include renaming, insertion, deletion and replacement in XML documents, and some extensions, like the deletion of one single node (preserving its descendant) instead of the deletion of a whole subtree. For several subclasses of these operations, we derive algorithms of synthesis of unranked tree automata, yielding both forward and backward type inference results. Since update operations, beside relabeling document nodes, can create and delete entire XML fragments, modifying a document's structure, it is not obvious how to infer the type of updated documents. Former tree automata completion constructions like [15] work for automata computing on ranked trees. Here, we consider unranked ordered trees, and our constructions are non trivial adaptations of former tree automata completion

* This work has been supported by the INRIA ARC 2010 project ACCESS, the FP7-ICT-2007-1 Project no. 216471, AVANTSSAR and the FET-Open grant agreement FOX no. FP7-ICT-23359.

procedures, where, starting from an initial automaton, new transitions rules are added and existing transition rules are recursively modified. Moreover, we show that some update operations do not preserve regular tree languages (i.e. languages of hedge automata) and that for the type inference for these operations, we need to consider a larger and less mainstream class of decidable unranked tree recognizers called context-free hedge automata.

One of our motivations for this study is the static analysis of access control policies (ACP) for XML updates. We consider two approaches for this problem. The first approach addresses *rule-based* specifications of ACPs, where the operations allowed, resp. forbidden, to a user are specified as two sets of atomic update primitives [6, 17]. We show in particular how to apply our type inference results to the verification of local consistency of ACPs, i.e. whether no sequence of allowed updates starting from a given document can achieve an explicitly forbidden update. Such situations may lead to serious security breaches which are challenging to detect according to [17]. In the second approach (*DTD-based* XML ACPs) the ACP is defined by adding security annotations to a DTD D, as in [14, 17]. In this case, it is required to check the validity of the document wrt D before applying every update operation. We show that under this restriction typechecking becomes undecidable.

Related work: Many works have employed tree automata to compute sets of descendants for standard (ranked) term rewriting (see e.g. [15]). Regular model checking [4] is extended to hedge rewriting and hedge automata in [39], which gives a procedure to compute reachability sets *approximations*. Here we compute exact reachability sets for some classes of hedge rewrite systems. For some results we need context-free hedge automata, a more general class than the regular hedge automata of [39].

When considering real programming languages like XDuce or CDuce [3] for writing transformations, typechecking is generally undecidable and approximations must be applied. In order to obtain exact algorithms, several approaches define conveniently abstract formalisms for representing transformations. Let us cite for instance TL (the transformation language) [25] and macro tree transducers (MTT) [26, 34], and k-pebble tree transducers (k-PTT) [28], a powerful model defined so as to cover relevant fragments of XSLT [22] and other XML transformation languages. Some restrictions on schema languages and on top down tree transducers (on which transformations are based) have also been studied [12, 27] in order to obtain PTIME typechecking procedures. [40] propose a backward type inference algorithm (based on tree automata techniques) for an XSLT fragment without XPath but with recursive calls. In a comparable approach, [16] propose a backward type inference algorithm for MTTs based on alternating tree automata, optimized towards practicability.

In this paper, we consider unrestricted applications of updates, unlike e.g. top-down transductions in [27]. It is shown in [28] that the set of output trees of a k-PTT for a fixed input tree is a regular tree language. In contrast, we shall see (Example 4 below) that it is not the case for the iteration of some update operations, and therefore that such transformation are not expressible as k-PTT. In Theorem 2, we show that the output language of the iteration of these updates for a regular input language is recognizable by a context-free hedge automata. This can be related to the result of [13], used in [26] in the context of typechecking XML transformations, and stating that the output language of a linear stay MTT can be characterized by a context-free tree grammar (in the case of ranked trees). Theorem 2 implies that the output languages of the iteration of updates can be described by MTTs, as MTT can generate all context-free tree languages. On the other hand, each of the primitive update operations can be solely modeled by a MTT. It is however not clear whether the finite (but unbounded) iterations of updates operations can be easily expressed as a MTT relation.

In [2] the authors investigate the problem of synthesizing an output schema describing the result of an update applied to a given input schema. They show how to infer safe over-approximations for the results of both queries and updates. Recent works have also applied local Hoare reasoning to simple tree update and even to a significant subset of the XML update library in W3C Document Object Model [18]. As far as we know this approach is not automated.

The first access control model for XML was proposed by [10] and was extended to secure updates in [7]. Static analysis has been applied to XML Access Control in [32] to determine if a query expression is guaranteed not to access to elements that are forbidden by the policy. In [17] the authors propose the XACU language. They study policy consistency and show that it is undecidable in their setting. On the positive side [6] considers policies defined in term of annotated non recursive XML DTDs and gives a polynomial algorithm for checking consistency.

Organization of the paper: we introduce the needed formal background about terms, hedge automata and rewriting systems in Section 2. Then we present XML update as parameterized rewriting rules and the type synthesis algorithms in Section 3. In Section 4 we study an extension of our rewriting rules by XPath expressions specifying the nodes where the rules can be applied. Finally we give applications to Access Control Policies verification in Section 5.

2. Definitions

2.1 Unranked Ordered Trees

Terms and Hedges. We consider a finite alphabet Σ and an infinite set of variables $\mathcal{X}$. The symbols of Σ are generally denoted $a, b, c \ldots$ and the variables $x, y \ldots$ We define recursively a *hedge* over Σ and $\mathcal{X}$ as a finite (possibly empty) sequence of terms and a *term* as either a single node n labeled by a variable of $x \in \mathcal{X}$ or the application of a node n labeled by a symbol $a \in \Sigma$ to a hedge h. The term is denoted x in the first case and $a(h)$ in the second case, and n is called the *root* of the term in both cases. The empty sequence is denoted $()$ and when h is empty, the term $a(h)$ will be simply denoted by a. The root node of $a(h)$ is called the *parent* of every root of h and every root of h is called a *child* of the root of $a(h)$. A root of a hedge $(t_1 \ldots t_n)$ is a root node of one of $t_1, ..., t_n$. A leaf of a hedge $(t_1 \ldots t_n)$ is a leaf (node without child) of one of the terms $t_1, ..., t_n$. A *path* is a sequence of nodes $n_0, \ldots, n_p$ such that for all $i < p$, n_{i+1} is a child of n_i. In this case, n_p is called a *descendant* of n_0. As usual, we can see a hedge $h \in \mathcal{H}(\Sigma, \mathcal{X})$ as a function from its set of nodes $dom(h)$ into labels in $\Sigma \cup \mathcal{X}$. The label of the node $n \in dom(h)$ is denoted by $h(n)$.

The set of hedges and terms over Σ and $\mathcal{X}$ are respectively denoted $\mathcal{H}(\Sigma, \mathcal{X})$ and $\mathcal{T}(\Sigma, \mathcal{X})$. We will sometimes consider a term as a hedge of length one, *i.e.* consider that $\mathcal{T}(\Sigma, \mathcal{X}) \subset \mathcal{H}(\Sigma, \mathcal{X})$. The sets of ground terms (terms without variables) and ground hedges are respectively denoted $\mathcal{T}(\Sigma)$ and $\mathcal{H}(\Sigma)$. The set of variables occurring in a hedge $h \in \mathcal{H}(\Sigma, \mathcal{X})$ is denoted $var(h)$. A hedge $h \in \mathcal{H}(\Sigma, \mathcal{X})$ is called *linear* if every variable of $\mathcal{X}$ occurs at most once in h.

Substitutions. A *substitution* σ is a mapping of finite domain from $\mathcal{X}$ into $\mathcal{H}(\Sigma, \mathcal{X})$. The application of a substitution σ to terms and hedges (written with postfix notation) is defined recursively by $x\sigma := \sigma(x)$ when $x \in dom(\sigma)$, $y\sigma := y$ when $y \in \mathcal{X} \setminus dom(\sigma)$, $(t_1 \ldots t_n)\sigma := (t_1\sigma \ldots t_n\sigma)$ for $n \geq 0$, and $a(h)\sigma := a(h\sigma)$.

Contexts. A *context* is a hedge $u \in \mathcal{H}(\Sigma, \mathcal{X})$ with a distinguished variable x_u linear (with exactly one occurrence) in u. The application of a context u to a hedge $h \in \mathcal{H}(\Sigma, \mathcal{X})$ is defined by $u[h] := u\{x_u \mapsto h\}$. It consists in inserting h into a hedge in

u in place of the node labelled by x_u. Sometimes, we write $t[s]$ in order to emphasize that s is a subterm (or subhedge) of t.

2.2 Hedge Automata and Context-Free Hedge Automata

We consider two kind of types for XML documents, defined as two classes of automata for unranked trees. The first one is the class of hedge automata [30], denoted HA. It captures the expressive strength of almost all popular type formalisms for XML [31]. The second and perhaps lesser known class is the context-free hedge automata, denoted CF-HA and introduced in [33]. CF-HA are strictly more expressive than HA and we shall see that they are of interest for typing certain update operations.

DEFINITION 1. *A* hedge automaton *(resp.* context-free hedge automaton*) is a tuple $\mathcal{A} = (\Sigma, Q, Q^{\mathrm{f}}, \Delta)$ where Σ is an finite unranked alphabet, Q is a finite set of states disjoint from Σ, $Q^{\mathrm{f}} \subseteq Q$ is a set of final states, and Δ is a set of transitions of the form $a(L) \to q$ where $a \in \Sigma$, $q \in Q$ and $L \subseteq Q^*$ is a regular word language (resp. a context-free word language).*

When Σ is clear from the context it is omitted in the tuple specifying $\mathcal{A}$. We define the move relation between ground hedges $h, h' \in \mathcal{H}(\Sigma \cup Q)$ as follows: $h \xrightarrow[\mathcal{A}]{} h'$ iff there exists a context $u \in \mathcal{H}(\Sigma, \{x_C\})$ and a transition $a(L) \to q \in \Delta$ such that $h = u[a(q_1 \dots q_n)]$, with $q_1 \dots q_n \in L$ and $h' = u[q]$. The relation $\xrightarrow[\mathcal{A}]{*}$ is the transitive closure of $\xrightarrow[\mathcal{A}]{}$.

Collapsing Transitions. We consider the extension of HA and CF-HA with so called *collapsing transitions* which are special transitions of the form $L \to q$ where $L \subseteq Q^*$ is a context-free language and q is a state. The move relation for the extended set of transitions generalizes the above definition with the case $u[q_1 \dots q_n] \xrightarrow[\mathcal{A}]{} u[q]$ if $L \to q$ is a collapsing transition of $\mathcal{A}$ and $q_1 \dots q_n \in L$. Note that we do not exclude the case $n = 0$ in this definition, i.e. L may contain the empty word in $L \to q$. Collapsing transitions with a singleton language L containing a length one word (i.e. transitions of the form $q \to q'$, where q and q' are states) correspond to *ε-transitions* for tree automata.

Languages. The language of a HA or CF-HA $\mathcal{A}$ in one of its states q, denoted by $L(\mathcal{A}, q)$ and also called set of hedges of type q, is the set of ground hedges $h \in \mathcal{H}(\Sigma)$ such that $h \xrightarrow[\mathcal{A}]{*} q$. We say sometimes that a hedge of $L(\mathcal{A}, q)$ has type q (when $\mathcal{A}$ is clear from context). A hedge is accepted by $\mathcal{A}$ if there exists $q \in Q^{\mathrm{f}}$ such that $h \in L(\mathcal{A}, q)$. The language of $\mathcal{A}$, denoted by $L(\mathcal{A})$ is the set of hedge accepted by $\mathcal{A}$.

Note that without collapsing transitions, all the hedges of $L(\mathcal{A}, q)$ are terms. Indeed, by applying standard transitions of the form $a(L) \to a$, one can only reduce length-one hedges into states. But collapsing transitions permit to reduce a ground hedge of length more than one into a single state.

The ε-transitions of the form $q \to q'$ do not increase the expressiveness HA or CF-HA (see [9] for HA and the proof for CF-HA is similar). But it is not the case in general for collapsing transitions: collapsing transitions strictly extend HA in expressiveness, and even collapsing transitions of the form $L \to q$ where L is finite (hence regular).

EXAMPLE 1. *[21]. The extended HA $\mathcal{A} = \big(\{q, q_a, q_b, q_{\mathrm{f}}\}, \{g, a, b\}, \{q_{\mathrm{f}}\}, \{a \to q_a, b \to q_b, g(q) \to q_{\mathrm{f}}, q_a\, q\, q_b \to q\}\big)$ recognizes $\{g(a^n b^n) \mid n \geq 1\}$ which is not a HA language.*

However, collapsing transitions can be eliminated from CF-HA, when restricting to the recognition of terms.

LEMMA 1 ([21]). *For every extended CF-HA over Σ with collapsing transitions $\mathcal{A}$, there exists a CF-HA $\mathcal{A}'$ without collapsing transitions such that $L(\mathcal{A}') = L(\mathcal{A}) \cap \mathcal{T}(\Sigma)$.*

Properties. It is known that for both classes of HA and CF-HA, the membership and emptiness problems are decidable in PTIME [9, 30, 33]. Moreover HA languages are closed under Boolean operations, but CF-HA are not closed under intersection and complementation. The intersection of a CF-HA language and a HA language is a CF-HA language. All these results are effective, with PTIME (resp. EXPTIME) constructions of automata of polynomial (resp. exponential) sizes for the closures under union and intersection (resp. complement). We call a HA or CF-HA $\mathcal{A} = (\Sigma, Q, Q^{\mathrm{f}}, \Delta)$ *normalized* if for every $a \in \Sigma$ and every $q \in Q$, there is at most one transition rule $a(L_{a,q}) \to q$ in Δ. Every HA (resp. CF-HA) can be transformed into a normalized HA (resp. CF-HA) in polynomial time by replacing every two rules $a(L_1) \to q$ and $a(L_2) \to q$ by $a(L_1 \cup L_2) \to q$.

A CF-HA $\mathcal{A} = (Q, Q^{\mathrm{f}}, \Delta)$ is called *deterministic* iff for all two transitions rules $a(L_1) \to q_1$ and $a(L_2) \to q_2$ in Δ, either $L_1 \cap L_2 = \emptyset$ or $q_1 = q_2$. It is called *complete* if for all $a \in \Sigma$ and and $w \in Q^*$, there exists at least one rule $a(L) \to q \in \Delta$ such that $w \in L$. When $\mathcal{A}$ is deterministic (resp. complete), for all $t \in \mathcal{T}(\Sigma)$, there exists at most (resp. at least) one state $q \in Q$ such that $t \in L(\mathcal{A}, q)$. Every HA can transformed into a deterministic and complete HA recognizing the same language (see *e.g.* [9]). CF-HA can be completed but not determinized.

2.3 Term Rewriting Systems

We use below term rewriting rules for modeling XML update operations. For this purpose, we propose a non-standard definition of term rewriting, extending the classical one [11] in two ways: the application of rewrite rules is extended from ranked terms to unranked terms and second, the rules are parameterized by HA languages (i.e. each parameterized rule can represent an infinite number of unparameterized rules).

Unranked Term Rewriting Systems. A term rewriting system $\mathcal{R}$ over a finite unranked alphabet Σ (TRS) is a set of *rewrite rules* of the form $\ell \to r$ where $\ell \in \mathcal{H}(\Sigma, \mathcal{X}) \setminus \mathcal{X}$ and $r \in \mathcal{H}(\Sigma, \mathcal{X})$; ℓ and r are respectively called left- and right-hand-side (*lhs* and *rhs*) of the rule. Note that we do not assume the cardinality of $\mathcal{R}$ to be finite.

The rewrite relation $\xrightarrow[\mathcal{R}]{}$ of a TRS $\mathcal{R}$ is the smallest binary relation on $\mathcal{H}(\Sigma, \mathcal{X})$ containing $\mathcal{R}$ and closed by application of substitutions and contexts. In other words, $h \xrightarrow[\mathcal{R}]{} h'$, iff there exists a context u, a rule $\ell \to r$ in $\mathcal{R}$ and a substitution σ such that $h = u[\ell\sigma]$ and $h' = u[r\sigma]$. The reflexive and transitive closure of $\xrightarrow[\mathcal{R}]{}$ is denoted $\xrightarrow[\mathcal{R}]{*}$.

Parameterized Term Rewriting Systems. Let $\mathcal{A} = (\Sigma, Q, Q^{\mathrm{f}}, \Delta)$ be a HA. A term rewriting system over Σ parameterized by $\mathcal{A}$ (**PTRS**) is given by a finite set, denoted $\mathcal{R}/\mathcal{A}$, of rewrite rules $\ell \to r$ where $\ell \in \mathcal{H}(\Sigma, \mathcal{X})$ and $r \in \mathcal{H}(\Sigma \uplus Q, \mathcal{X})$ and symbols of Q can only label leaves of r ($\uplus$ stands disjoint union, hence we implicitly assume that Σ and Q are disjoint sets). In this notation, $\mathcal{A}$ may be omitted when it is clear from context or not necessary. The rewrite relation $\xrightarrow[\mathcal{R}/\mathcal{A}]{}$ associated to a PTRS $\mathcal{R}/\mathcal{A}$ is defined as the rewrite relation $\xrightarrow[\mathcal{R}[\mathcal{A}]]{}$ where the TRS $\mathcal{R}[\mathcal{A}]$ is the (possibly infinite) set of all rewrite rules obtained from rules $\ell \to r$ in $\mathcal{R}/\mathcal{A}$ by replacing in r every state $q \in Q$ by a ground term of $L(\mathcal{A}, q)$. Several examples of parameterized rewrite rules can be found in Figure 1 below. We will consider in Sections 4 and 5.2 two extensions of PTRS, called controlled PTRS and PTRS with global constraints.

Problems. Given a set $L \subseteq \mathcal{H}(\Sigma, \mathcal{X})$ and a PTRS $\mathcal{R}/\mathcal{A}$, we define $post^*_{\mathcal{R}/\mathcal{A}}(L) := \{h' \in \mathcal{H}(\Sigma, \mathcal{X}) \mid \exists h \in L, h \xrightarrow[\mathcal{R}/\mathcal{A}]{*} h'\}$ and $pre^*_{\mathcal{R}/\mathcal{A}}(L) := \{h \in \mathcal{H}(\Sigma, \mathcal{X}) \mid \exists h' \in L, h \xrightarrow[\mathcal{R}/\mathcal{A}]{*} h'\}$.

Reachability is the problem to decide, given two hedges $h, h' \in \mathcal{H}(\Sigma)$ and a PTRS $\mathcal{R}/\mathcal{A}$ whether $h \xrightarrow[\mathcal{R}/\mathcal{A}]{*} h'$. Reachability problems for ground ranked term rewriting have been investigated in e.g. [20]. C. Löding [23] has obtained results in a more general setting where rules of type $L \to R$ specify the replacement of any element of a regular language L by any element of a regular tree language R. Then [24] has extended some of these works to unranked tree rewriting for the case of *subtree and flat prefix rewriting* which is a combination of standard ground tree rewriting and prefix word rewriting on the ordered leaves of subtrees of height 1. *Typechecking* (see e.g. [28]) is the problem to decide, given two sets of terms τ_{in} and τ_{out} called input and output types (generally presented as HA) and a PTRS $\mathcal{R}/\mathcal{A}$ whether $post^*_{\mathcal{R}/\mathcal{A}}(\tau_{in}) \subseteq \tau_{out}$ or equivalently $\tau_{in} \cap pre^*_{\mathcal{R}/\mathcal{A}}(\overline{\tau_{out}}) = \emptyset$ (where $\overline{\tau_{out}}$ is the complement of τ_{out}). One related problem, called *forward* (resp. *backward*) *type inference*, is, given a PTRS $\mathcal{R}/A$ and a HA or CF-HA language L, to construct a HA or CF-HA recognizing $post^*_{\mathcal{R}/\mathcal{A}}(L)$ (resp. $pre^*_{\mathcal{R}/\mathcal{A}}(L)$).

3. Forward and Backward Type Inference for Update Operations

In this section, we study the problem of type inference for arbitrary finite sequences of primitive update operations taken in a given set. More precisely, we propose a definition in term of PTRS rules (Section 3.1) of infinite sets of update primitive operations of the XQuery update facility [8] and some extensions. Then, we present constructions of HA and CF-HA for forward and backward type inference in these settings (Sections 3.2–3.4).

3.1 Primitive Update Facility Operations

We assume given an unranked alphabet Σ and a HA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$. Figure 1 displays PTRS rules, parameterized by states p, p_1,..,p_n of $\mathcal{A}$, representing infinite sets of atomic operations of the XQuery update facility [8], and some restrictions or extensions. We call $\mathsf{UFO}+$ the class of PTRS rules in Figure 1.

The following rules correspond to the update primitives of [8] except for the possibility in [8] to select by XQuery the nodes to be inserted (called *content* nodes in [8]) from the document one is working on.

REN renames a node: it changes its label from a into b. Such a rule leaves the structure of the term unchanged. $\mathsf{INS}_{\mathsf{first}}$ inserts a term of type p at the first position below a node labeled by a. $\mathsf{INS}_{\mathsf{last}}$ inserts at the last position and $\mathsf{INS}_{\mathsf{into}}$ at an arbitrary position below a node labeled by a. $\mathsf{INS}_{\mathsf{before}}$ (resp. $\mathsf{INS}_{\mathsf{after}}$) inserts a term of type p at the left (resp. right) sibling position to a node labeled by a. DEL deletes a whole subterm whose root node is labeled by a and RPL replaces a subterm by a sequence of terms of respective types $p_1, \ldots, p_n$.

EXAMPLE 2. *The patient data in a hospital are stored in an XML document whose DTD can be characterized by an HA $\mathcal{A}$ with transition rules:*

$$\begin{array}{rcl@{\quad}rcl} \mathsf{hospital}(\{p_{\mathsf{pa}}, p_{\mathsf{epa}}\}^*) & \to & p_{\mathsf{h}}, & \mathsf{name}(p_c^*) & \to & p_{\mathsf{n}}, \\ \mathsf{patient}(p_{\mathsf{n}}\, p_{\mathsf{t}}) & \to & p_{\mathsf{pa}}, & \mathsf{drug}(p_c^*) & \to & p_{\mathsf{dr}}, \\ \mathsf{patient}(p_{\mathsf{n}}) & \to & p_{\mathsf{epa}}, & \mathsf{diagnosis}(p_c^*) & \to & p_{\mathsf{dia}}, \\ \mathsf{treatment}(p_{\mathsf{dr}}\, p_{\mathsf{dia}}\, p_{\mathsf{da}}) & \to & p_{\mathsf{t}}, & \mathsf{date}(p_c^*) & \to & p_{\mathsf{da}}, \\ \multicolumn{6}{l}{\mathsf{a} \to p_c, \quad \mathsf{b} \to p_c, \quad \mathsf{c} \to p_c \ldots} \end{array}$$

The state p_{h} is the entry point *of the DTD* i.e. *it represents the type of the root element.*

A DEL *rule* $\mathsf{patient}(x) \to ()$ *will delete a* patient *in the base, and a* $\mathsf{INS}_{\mathsf{last}}$ *rule* $\mathsf{hospital}(x) \to \mathsf{hospital}(x\, p_{\mathsf{pa}})$ *will insert a new* patient*, at the last position below the root node* hospital*. We can ensure that the patient newly added has an empty* treatments *list (to be completed later) using* $\mathsf{hospital}(x) \to \mathsf{hospital}(x\, p_{\mathsf{epa}})$*. A* $\mathsf{INS}_{\mathsf{after}}$ *rule* $\mathsf{name}(x) \to \mathsf{name}(x)\, p_{\mathsf{t}}$ *can be used to insert later a* treatment *next to the patient's* name*.*

We propose also in Figure 1 some other operations not in [8]. The rules RNS_* combine the application of the corresponding insert operations INS_* and of a node renaming REN. The rule RPL_1 is a restriction of RPL to $n = 1$ (note that DEL is also a special cases of RPL, with $n = 0$). Finally, $\mathsf{DEL}_{\mathsf{s}}$ deletes a single node n whose arguments inherit the position. In other words, it replaces a term with the hedge containing its children. This operation is employed to build user views of XML documents e.g. in [14], and can also be useful for updates as well.

EXAMPLE 3. *Assume that some patients of the hospital of Example 2 are grouped in one department like in*

$$\mathsf{hospital}(\ldots \mathsf{surgery}(p^*_{\mathsf{pa}}) \ldots),$$

and that we want to suppress the department surgery *while keeping its patients. This can be done with the* $\mathsf{DEL}_{\mathsf{s}}$ *rule* $\mathsf{surgery}(x) \to x$*.*

We will see in Section 3.3 that allowing the operations RNS_*, $\mathsf{DEL}_{\mathsf{s}}$ or RPL has important consequences w.r.t. type inference. Indeed, the subclass of operations in the first column of Figure 1, called $\mathsf{UFO}_{\mathsf{reg}}$ preserves languages of HA whereas the operations in the second column may transform a HA language into a CF-HA language.

3.2 Forward Type Inference for $\mathsf{UFO}_{\mathsf{reg}}$ Rules

We want to characterize the sets of terms which can be obtained, from terms of a given type, by arbitrary application of updates operations defined as PTRS rules. For this purpose, we shall study the recognizability (by HA and CF-HA) of the forward closure ($post^*$) of automata languages under the above rewrite rules.

THEOREM 1. *For all HA $\mathcal{A}$ on Σ, PTRS $\mathcal{R}/\mathcal{A} \in \mathsf{UFO}_{\mathsf{reg}}$, and HA language L, $post^*_{\mathcal{R}/\mathcal{A}}(L)$ is the language of an HA of size polynomial and which can be constructed in PTIME in the size of $\mathcal{R}/\mathcal{A}$ and of an HA recognizing L.*

In the following proofs, we describe *finite automata* for the horizontal languages of HA transitions as tuples $B = (Q, S, i, F, \Gamma)$, where Q is the finite input alphabet, S is a finite set of states, i is the initial state, $F \subseteq S$ is the set of final states and $Q \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times S$ is the set of transitions and ε-transitions. Every transition (s, q, s') will be denoted $s \xrightarrow{q} s'$. For $s, s' \in S$, we write $s \xrightarrow[B]{\varepsilon} s'$ to express that s' can be reached from s by a (possibly empty) sequence of ε-transitions of B, and $s \xrightarrow{a_1 \ldots a_n} s'$, for $a_1, \ldots, a_n \in Q$, if there exists $2(n+1)$ states $s_0, s'_0, \ldots, s_n, s'_n \in S$ with $s_0 = s$, $s_n \xrightarrow[B]{\varepsilon} s'$ and $0 \leq i < n$, $s_i \xrightarrow[B]{\varepsilon} s'_i$ and $(s'_i, a_{i+1}, s_{i+1}) \in \Gamma$.

Proof. Let $\mathcal{A} = (\Sigma, P, P^{\mathsf{f}}, \Theta)$ and let $\mathcal{A}_L = (\Sigma, Q_L, Q^{\mathsf{f}}_L, \Delta_L)$ recognizing L. We assume that both $\mathcal{A}$ and $\mathcal{A}_L$ are normalized and that their state sets P and Q_L are disjoint. We construct a HA $\mathcal{A}' = (\Sigma, P \uplus Q_L, Q^{\mathsf{f}}_L, \Delta')$ recognizing $post^*_{\mathcal{R}/\mathcal{A}}(L)$. For each $a \in \Sigma$, $q \in Q_L$, let $L_{a,q}$ be the regular language in the transition (assumed unique) $a(L_{a,q}) \to q \in \Delta_L$, and let $B_{a,q} = \big(Q_L, S_{a,q}, i_{a,q}, \{f_{a,q}\}, \Gamma_{a,q}\big)$ be a finite automaton recognizing $L_{a,q}$. The sets of states $S_{a,q}$ are assumed pairwise disjoint. Let S be the disjoint union of all $S_{a,q}$ for all $a \in \Sigma$ and $q \in Q_L$.

For the construction of Δ', we develop a set of transition rules $\Gamma' \subseteq S \times (P \cup Q_L) \times S$. Initially, we let Γ' be the union Γ_0 of all $\Gamma_{a,q}$ for $a \in \Sigma$, $q \in Q_L$, and we complete Γ' iteratively by analyzing the different cases of update rules of $\mathcal{R}/\mathcal{A}$. At each step, for each $a \in \Sigma$ and $q \in Q_L$, we let $B'_{a,q}$ be the automaton $(P \cup Q_L, S, i_{a,q}, \{f_{a,q}\}, \Gamma')$. For the sake of conciseness we make no distinction between an automaton $B'_{a,q}$ and its language $L(B'_{a,q})$.

UFO+							
$\mathsf{UFO}_{\mathsf{reg}}$							
$a(x)$	$\to$	$b(x)$	REN				
$a(x)$	$\to$	$a(p\,x)$	$\mathsf{INS}_{\mathsf{first}}$	$a(x)$	$\to$	$b(p\,x)$	$\mathsf{RNS}_{\mathsf{first}}$
$a(x)$	$\to$	$a(x\,p)$	$\mathsf{INS}_{\mathsf{last}}$	$a(x)$	$\to$	$b(x\,p)$	$\mathsf{RNS}_{\mathsf{last}}$
$a(xy)$	$\to$	$a(x\,p\,y)$	$\mathsf{INS}_{\mathsf{into}}$				
$a(x)$	$\to$	$p\,a(x)$	$\mathsf{INS}_{\mathsf{before}}$				
$a(x)$	$\to$	$a(x)\,p$	$\mathsf{INS}_{\mathsf{after}}$				
$a(x)$	$\to$	p	RPL_1	$a(x)$	$\to$	$p_1 \dots p_n$	RPL
$a(x)$	$\to$	$()$	DEL	$a(x)$	$\to$	x	$\mathsf{DEL}_{\mathsf{s}}$

Figure 1. PTRS Rules for the Primitive XQuery Update Facility Operations and Extensions

REN for every $a(x) \to b(x) \in \mathcal{R}/\mathcal{A}$ and $q \in Q_L$, we add two ε-transitions $(i_{b,q}, \varepsilon, i_{a,q})$ and $(f_{a,q}, \varepsilon, f_{b,q})$ to Γ'.

$\mathsf{INS}_{\mathsf{first}}$ for every $a(x) \to a(p\,x) \in \mathcal{R}/\mathcal{A}$ and $q \in Q_L$, we add one looping transition $(i_{a,q}, p, i_{a,q})$ to Γ'.

$\mathsf{INS}_{\mathsf{last}}$ for every $a(x) \to a(x\,p) \in \mathcal{R}/\mathcal{A}$ and $q \in Q_L$, we add one looping transition rule $(f_{a,q}, p, f_{a,q})$ to Γ'.

$\mathsf{INS}_{\mathsf{into}}$ for every $a(xy) \to a(x\,p\,y) \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$ and $s \in S$ reachable from $i_{a,q}$ using the transitions of Γ', we add one looping transition rule (s, p, s) to Γ'.

$\mathsf{INS}_{\mathsf{before}}$ for every $a(x) \to p\,a(x) \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$ and state $s \in S$ such that $L(B'_{a,q}) \neq \emptyset$ and there exists a transition $(s, q, s') \in \Gamma'$, we add one looping transition (s, p, s) to Γ'.

$\mathsf{INS}_{\mathsf{after}}$ for every $a(x) \to a(x)\,p \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$ and $s' \in S$ such that $L(B'_{a,q}) \neq \emptyset$ and there exists a transition $(s, q, s') \in \Gamma'$, we add one looping transition (s', p, s') to Γ'.

RPL_1 for every $a(x) \to p \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$, and $s, s' \in S$ such that $L(B'_{a,q}) \neq \emptyset$, and there exists a transition $(s, q, s') \in \Gamma'$, we add one transition (s, p, s') to Γ'.

DEL for every $a(x) \to () \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$, and $s, s' \in S$ such that $L(B'_{a,q}) \neq \emptyset$, and there exists a transition $(s, q, s') \in \Gamma'$, we add one ε-transition (s, ε, s') to Γ'.

Note that some of the above new transitions summarize several insertions. Such a construction are comparable to *acceleration* techniques used in model checking.

We iterate the above operations until a fixpoint is reached (only a finite number of transitions can be added to Γ' this way). Finally, we let

$$\Delta' := \Theta \cup \big\{a\big(B'_{a,q}\big) \to q \;\big|\; a \in \Sigma, q \in Q, L(B'_{a,q}) \neq \emptyset\big\}.$$

We show in the long version that $L(\mathcal{A}') = post^*_{\mathcal{R}/\mathcal{A}}(L)$. □

COROLLARY 1. *Typechecking is EXPTIME-complete for $\mathsf{UFO}_{\mathsf{reg}}$ and PTIME-complete when the output type is given by a deterministic and complete HA.*

Proof. Let τ_{in} and τ_{out} be two HA languages (resp. input and output types), and let $\mathcal{R}/\mathcal{A}$ by a PTRS. We want to know whether $post^*_{\mathcal{R}/\mathcal{A}}(\tau_{in}) \subseteq \tau_{out}$. Following Theorem 1, $post^*_{\mathcal{R}/\mathcal{A}}(\tau_{in})$ is a HA language. Hence $post^*_{\mathcal{R}/\mathcal{A}}(\tau_{in}) \cap \overline{\tau_{out}}$ is a HA language. The size of the HA for the complement $\overline{\tau_{out}}$ can be exponential in the size of the HA for τ_{out} if this latter HA is non-deterministic, and it is polynomial otherwise. Testing the emptiness of the above intersection language solves the problem.

Regarding the lower bounds, the EXPTIME-hardness follows the fact that the inclusion problem is already EXPTIME-complete for ranked tree automata [36], and the PTIME-hardness from the fact that the inclusion problem is PTIME-hard for deterministic HA. □

Regarding the problem of type synthesis, if we are given $\mathcal{R}/\mathcal{A}$ and an input type τ_{in} as a HA, Theorem 1 provides in PTIME an output type presented as a HA of polynomial size.

3.3 Forward Type Inference for UFO+ Rules

Theorem 1 is not true for all the rules of UFO+: the rules of UFO+ $\setminus$ $\mathsf{UFO}_{\mathsf{reg}}$ do not preserve HA languages in general. It is evident for RPL, and the examples below show that it is also the case for RNS_* and $\mathsf{DEL}_{\mathsf{s}}$. However, we prove in Theorem 2 that the rules of UFO+ preserve the larger class of CF-HA language.

EXAMPLE 4. *Let $\Sigma = \{a, b, c, c'\}$ and let $\mathcal{R}$ be the finite TRS containing the two $\mathsf{RNS}_{\mathsf{first}}$ and $\mathsf{RNS}_{\mathsf{last}}$ rules $c(x) \to c'(ax), c'(x) \to c(xb)$. We have $post^*_{\mathcal{R}}\big(\{c\}\big) \cap \mathcal{H}(\Sigma) = \{c(a^n b^n) \mid n \geq 0\}$, and this set is not a HA language. It follows that $post^*_{\mathcal{R}}\big(\{c\}\big)$ is not a HA language.*

Let $\Sigma = \{a, b, c\}$, let $\mathcal{R}$ be the finite TRS with one $\mathsf{DEL}_{\mathsf{s}}$ rule $c(x) \to x$ and let L be the HA language containing exactly the terms $c(ac(a \dots c \dots b)b)$; it is recognized by the HA with the set of transition rules $\big\{a \to q_a, b \to q_b, c\big(\{(), q_a\, q\, q_b\}\big) \to q\big\}$. We have $post^*_{\mathcal{R}}(L) \cap c\big(\{a, b\}^*\big) = \{c(a^n b^n) \mid n \geq 0\}$, hence $post^*_{\mathcal{R}}(L)$ is not a HA language.

THEOREM 2. *For all HA $\mathcal{A}$ on Σ, PTRS $\mathcal{R}/\mathcal{A} \in$ UFO+, and CF-HA language L, $post^*_{\mathcal{R}/\mathcal{A}}(L)$ is the language of a CF-HA of size polynomial and which can be constructed in PTIME in the size of $\mathcal{R}/\mathcal{A}$ and of an CF-HA recognizing L.*

Proof. Let $\mathcal{A} = (\Sigma, P, P^{\mathsf{f}}, \Theta)$ and let us assume that it is normalized. Let $\mathcal{A}_L = (\Sigma, Q_L, Q^{\mathsf{f}}_L, \Delta_L)$ be a CF-HA recognizing L, normalized and without collapsing transitions. The state sets P and Q_L are assumed disjoint.

We shall construct a CF-HA extended with collapsing transitions $\mathcal{A}' = (\Sigma, P \uplus Q_L, Q^{\mathsf{f}}_L, \Delta')$ recognizing $post^*_{\mathcal{R}/\mathcal{A}}(L)$. It follows that $post^*_{\mathcal{R}/\mathcal{A}}(L)$ is a CF-HA language thanks to Lemma 1.

Very roughly, we define new CFG $\mathcal{G}'_{a,q}$ for the horizontal languages of the transitions of $\mathcal{A}'$, like in Theorem 1, starting from the CFG for the transitions of $\mathcal{A}_L$ and adding a new initial non-terminal $I'_{a,q}$ and new production rules, according to cases of rewrite rules in $\mathcal{R}/\mathcal{A}$.

More formally, The set of transitions Δ' is constructed starting from $\Delta_L \cup \Theta$ and analysing the different cases of update rules.

For each $a \in \Sigma$, $q \in Q_L$, let $L_{a,q}$ be the context-free language in the transition (assumed unique) $a(L_{a,q}) \to q \in \Delta_L$, and let $\mathcal{G}_{a,q} = (Q_L, \mathcal{N}_{a,q}, I_{a,q}, \Gamma_{a,q})$ be a CF grammar in Chomski normal form generating $L_{a,q}$. The sets of non-terminals $\mathcal{N}_{a,q}$ are assumed pairwise disjoint.

Let us consider one new non-terminal $I'_{a,q}$ for each $a \in \Sigma$ and $q \in Q_L$. Each of these non terminals aims at becoming the initial non terminal of the CF grammar in the transition associated to a

and q in Δ'. For technical convenience, we also add one new non terminal X_p for each $p \in P$.

For the construction of Δ', we construct first below

- a set C' of collapsing transitions, and
- a set Γ' of production rules of CF grammar over the set of terminal symbols in $P \cup Q_L$ and the set of non terminals

$$\mathcal{N} = \bigcup_{a\in\Sigma, q\in Q} \big(\mathcal{N}_{a,q} \cup \{I'_{a,q}\}\big) \cup \{X_p \mid p \in P\}.$$

Initially, we let $C' = \emptyset$ and

$$\begin{aligned}\Gamma' = \Gamma'_0 \quad := \quad & \bigcup_{a\in\Sigma, q\in Q} \big(P_{a,q} \cup \{I'_{a,q} := I_{a,q}\}\big) \\ & \cup \{X_p := p \mid p \in P\}.\end{aligned}$$

We now proceed by analysis of the rewrite rules of $\mathcal{R}/\mathcal{A}$ for the completion of Γ' and C'. At each step, for each $a \in \Sigma$ and $q \in Q_L$, we let $\mathcal{G}'_{a,q}$ be the CF grammar $(P \cup Q_L, \mathcal{N}, I'_{a,q}, \Gamma')$, and let $L'_{a,q} = L(\mathcal{G}'_{a,q})$. The production rules of Γ' remain in Chomski normal form after each completion step.

REN for every $a(x) \to b(x) \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$, we add one production rule $I'_{b,q} := I'_{a,q}$ to Γ'.

RNS$_{\mathsf{first}}$ for every $a(x) \to b(p\,x) \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$, we add one production rule $I'_{b,q} := X_p I'_{a,q}$ to Γ'.

RNS$_{\mathsf{last}}$ for every $a(x) \to b(x\,p) \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$, we add one production rule $I'_{b,q} := I'_{a,q} X_p$ to Γ'.

INS$_{\mathsf{into}}$ for every $a(xy) \to a(x\,p\,y) \in \mathcal{R}/\mathcal{A}$, $q \in Q_L$ and every $N \in \mathcal{N}$ reachable from $I'_{a,q}$ using the rules of Γ', we add two production rules $N := NX_p$ and $N := X_pN$.

INS$_{\mathsf{before}}$ for every $a(x) \to p\,a(x) \in \mathcal{R}/\mathcal{A}$, and $q \in Q_L$ such that $L'_{a,q} \neq \emptyset$, we add one collapsing transition $p\,q \to q$ to C'.

INS$_{\mathsf{after}}$ for every $a(x) \to a(x)\,p \in \mathcal{R}/\mathcal{A}$, and $q \in Q_L$ such that $L'_{a,q} \neq \emptyset$, we add one collapsing transition $q\,p \to q$ to C'.

RPL for every $a(x) \to p_1 \dots p_n \in \mathcal{R}/\mathcal{A}$, with $n \geq 0$, and $q \in Q_L$ such that $L'_{a,q} \neq \emptyset$, we add one collapsing transition $p_1 \dots p_n \to q$ to C'.

DEL for every $a(x) \to () \in \mathcal{R}/\mathcal{A}$ and $q \in Q_L$ such that $L'_{a,q} \neq \emptyset$, we add one collapsing transition $() \to q$ to C'.

Note that INS$_{\mathsf{first}}$, INS$_{\mathsf{last}}$, RPL$_1$ are special cases of respectively RNS$_{\mathsf{first}}$, RNS$_{\mathsf{last}}$, RPL.

We iterate the above operations until a fixpoint is reached. Indeed, only a finite number of production and collapsing rules can be added. Finally, we let

$$\begin{aligned}\Delta' \quad := \quad & \Theta \cup \big\{a(L'_{a,q}) \to q \;\big|\; a \in \Sigma, q \in Q, L'_{a,q} \neq \emptyset\big\} \cup C' \\ & \cup \{L'_{a,q} \to q \mid a(x) \to x \in \mathcal{R}/\mathcal{A}, L'_{a,q} \neq \emptyset\}.\end{aligned}$$

We show in the long version that $L(\mathcal{A}') = post^*_{\mathcal{R}/\mathcal{A}}(L)$.

The proof of the direction $\subseteq$ is by induction on the number of application of collapsing transitions in a reduction by $\mathcal{A}'$. For the base case (no collapsing transition applied), we make a second induction on the number of application of production rules of $\Gamma' \setminus \Gamma_0$ in the derivations, by the grammars $\mathcal{G}'_{a,q_0}$, for the generations of the sequences of states $q_1 \dots q_n \in Q^*$ used in moves of $\mathcal{A}'$ of the form $C[a(q_1 \dots q_n)] \to C[q_0]$ in the reduction $t \xrightarrow[\mathcal{A}']{*} q$. Intuitively every application of such production rule corresponds to a rewrite step with a rule of $\mathcal{R}/\mathcal{A}$.

The proof of the direction $\supseteq$ is by induction on the length of a rewrite sequence $u \xrightarrow[\mathcal{R}/\mathcal{A}]{*} t$ for $u \in L(\mathcal{A})$. It follows that $post^*_{\mathcal{R}/\mathcal{A}}(L)$ is a CF-HA language by Lemma 1. □

COROLLARY 2. *Typechecking is EXPTIME-complete for UFO+ and PTIME-complete when the output type is given by a deterministic and complete HA.*

Proof. The proof for the upper bound works as in Corollary 1, because the intersection of a CF-HA and a HA language is a CF-HA language (there is an effective PTIME construction of an CF-HA of polynomial size), and emptiness of CF-HA is decidable in PTIME. The arguments of Corollary 1 for lower bounds are still valid here because HA are special cases of CF-HA. □

Regarding the problem of type synthesis for a $\mathcal{R}/\mathcal{A}$ in UFO+, if an input type τ_{in} is given as a HA or CF-HA, then Theorem 2 provides in PTIME an output type, presented as a CF-HA of polynomial size. Unlike HA, CF-HA are not popular type schemes, but HA solely do not permit to extend the results of Theorem 1, in particular for the operation RPL of [8], as we have seen above.

Note that $post^*_{\mathcal{R}/\mathcal{A}}(L)$ can already be a CF-HA language when the given L is a HA language (see Example 4). One may wonder to what extent the CF-HA produced by Theorem 1, given a HA for L and a $\mathcal{R}/\mathcal{A}$, is actually an HA. This problem is actually undecidable, since the problem of knowing whether a given CF language is regular is undecidable.

3.4 Backward Type Inference for UFO+ Rules

Since UFO+ Rules do not preserve HA languages, as for k-pebble tree transducer [28] we may attempt to perform typechecking using pre^* computations (backward type inference). The next theorem shows that this is indeed possible, though EXPTIME, since the class of HA languages is preserved by pre^* when using UFO+ rules.

THEOREM 3. *Given a HA $\mathcal{A}$ on Σ and a PTRS $\mathcal{R}/\mathcal{A} \in$ UFO+, for all HA language L, $pre^*_{\mathcal{R}/\mathcal{A}}(L)$ is the language of a HA of size exponential and which can be constructed in EXPTIME in the size of $\mathcal{R}/\mathcal{A}$ and of an HA recognizing L.*

Proof. We consider a normalized and complete HA $\mathcal{A}_L = (\Sigma, Q_L, Q^{\mathsf{f}}_L, \Delta_L)$ recognizing L. Like in the proof of Theorem 1, we assume given, for each $a \in \Sigma$, $q \in Q_L$, a finite automaton $B_{a,q} = (Q_L, S_{a,q}, i_{a,q}, \{f_{a,q}\}, \Gamma_{a,q})$ recognizing the regular language $L_{a,q}$ in the transition $a(L_{a,q}) \to q \in \Delta_L$.

Unlike the proofs of Theorems 1 and 2, we will incrementally add transitions to $\mathcal{A}_L$, according to the rules of $\mathcal{R}/\mathcal{A}$, until a fixpoint automaton is reached which recognizes $pre^*_{\mathcal{R}/\mathcal{A}}(L)$. Every transition added has the form $a_0(B) \to q_0$ where B belongs to the smallest set $\mathcal{C}$ defined below.

More precisely, we construct a finite sequence sequence of HA $\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_k$ whose final element's language is $pre^*_{\mathcal{R}/\mathcal{A}}(L)$, where for all $i \leq k$, $\mathcal{A}_i = (\Sigma, Q_L, Q^{\mathsf{f}}_L, \Delta_i)$. For the construction of the transition sets Δ_i, we consider the set $\mathcal{C}$ of finite automata over Q_L defined as the smallest set such that:

- $\mathcal{C}$ contains every $B_{a,q}$ for $a \in \Sigma$, $q \in Q_L$,
- for all $B \in \mathcal{C}$, $B = (Q_L, S, i, F, \Gamma)$ and all states $s, s' \in S$, the automaton $B_{s,s'} := (Q_L, S, s, \{s'\}, \Gamma)$ is in $\mathcal{C}$,
- for all $B \in \mathcal{C}$, $B = (Q_L, S, i, F, \Gamma) \in \mathcal{C}$, $q \in Q_L$ and all states $s, s' \in S$, the automata $(Q_L, S, i, F, \Gamma \cup \{\langle s, q, s'\rangle\})$ and $(Q_L, S, i, F, \Gamma \cup \{\langle s, \varepsilon, s'\rangle\})$, respectively denoted by $B + \langle s, q, s'\rangle$ and $B + \langle s, \varepsilon, s'\rangle$ also belong to $\mathcal{C}$.

Note that $\mathcal{C}$ is finite with this definition, though exponential.

For the sake of conciseness, we make no distinction below between an automaton $B \in \mathcal{C}$ and the language $L(B)$ recognized by B. Moreover, we assume that every $B \in \mathcal{C}$ has a unique final state denoted f_B and its initial state is denoted i_B.

First, we let $\Delta_0 = \Delta_L$. The other Δ_i are constructed recursively by iteration of the following case analysis until a fixpoint is reached (only a finite number of transition can be added in the construction). In the construction we use an extension of the move relation of HA, from states to set of states (single states are considered as singleton sets): $a(L_1, \ldots, L_n) \hookrightarrow_{\Delta_i} q$ (where $L_1, \ldots, L_n \subseteq Q_L$ and $q \in Q_L$) iff there exists a transition $a(L) \to q \in \Delta_i$ such that $L_1 \ldots L_n \subseteq L$.

REN if $a(x) \to b(x) \in \mathcal{R}/\mathcal{A}$, $B \in \mathcal{C}$ and $q \in Q_L$, such that $b(B) \hookrightarrow_{\Delta_i} q$, then let $\Delta_{i+1} := \Delta_i \cup \{a(B) \to q\}$.

$\mathsf{RNS}_{\mathsf{first}}$ if $a(x) \to b(p\,x) \in \mathcal{R}/\mathcal{A}$, $B \in \mathcal{C}$ and $q, q_p \in Q_L$, such that $L(\mathcal{A}_i, q_p) \cap L(\mathcal{A}, p) \neq \emptyset$ and $b(q_p B) \hookrightarrow_{\Delta_i} q$, then $\Delta_{i+1} := \Delta_i \cup \{a(B) \to q\}$.

$\mathsf{RNS}_{\mathsf{last}}$ if $a(x) \to b(x\,p) \in \mathcal{R}/\mathcal{A}$, $B \in \mathcal{C}$ and $q, q_p \in Q_L$, such that $L(\mathcal{A}_i, q_p) \cap L(\mathcal{A}, p) \neq \emptyset$ and $b(B\,q_p) \hookrightarrow_{\Delta_i} q$, then $\Delta_{i+1} := \Delta_i \cup \{a(B) \to q\}$.

$\mathsf{INS}_{\mathsf{into}}$ if $a(xy) \to a(x\,p\,y) \in \mathcal{R}/\mathcal{A}$, $B \in \mathcal{C}$, s, s' are states of B, and $q, q_p \in Q_L$, such that $L(\mathcal{A}_i, q_p) \cap L(\mathcal{A}, p) \neq \emptyset$, $s \xrightarrow[B]{q_p} s'$, and $a(B) \hookrightarrow_{\Delta_i} q$ then $\Delta_{i+1} := \Delta_i \cup \{a(B + \langle s, \varepsilon, s'\rangle) \to q\}$.

$\mathsf{INS}_{\mathsf{before}}$ if $a(x) \to p\,a(x) \in \mathcal{R}/\mathcal{A}$, $b \in \Sigma$, $B, B' \in \mathcal{C}$, s, s' are states of B, and $q, q_p, q' \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B') \hookrightarrow_{\Delta_i} q'$, $L(\mathcal{A}_i, q_p) \cap L(\mathcal{A}, p) \neq \emptyset$, $s \xrightarrow[B]{q_p q'} s'$, then $\Delta_{i+1} := \Delta_i \cup \{b(B + \langle s, q', s'\rangle) \to q\}$.

$\mathsf{INS}_{\mathsf{after}}$ if $a(x) \to a(x)\,p \in \mathcal{R}/\mathcal{A}$, $b \in \Sigma$, $B, B' \in \mathcal{C}$, s, s' are states of B, and $q, q_p, q' \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B') \hookrightarrow_{\Delta_i} q'$, $L(\mathcal{A}_i, q_p) \cap L(\mathcal{A}, p) \neq \emptyset$, $s \xrightarrow[B]{q' q_p} s'$, then $\Delta_{i+1} := \Delta_i \cup \{b(B + \langle s, q', s'\rangle) \to q\}$.

RPL if $a(x) \to p_1 \ldots p_n \in \mathcal{R}/\mathcal{A}$, $b \in \Sigma$, $B, B' \in \mathcal{C}$, s, s' are states of B, and $q, q', q_1, \ldots, q_n \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B') \hookrightarrow_{\Delta_i} q'$, $L(\mathcal{A}_i, q_j) \cap L(\mathcal{A}, p_j) \neq \emptyset$ for all $1 \leq j \leq n$, $s \xrightarrow[B]{q_1 \ldots q_n} s'$ then $\Delta_{i+1} := \Delta_i \cup \{b(B + \langle s, q', s'\rangle)) \to q\}$.

DEL if $a(x) \to () \in \mathcal{R}/\mathcal{A}$, $b \in \Sigma$, $B, B' \in \mathcal{C}$, s is a state of B, $q, q' \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B') \hookrightarrow_{\Delta_i} q'$, then $\Delta_{i+1} := \Delta_i \cup \{b(B + \langle s, q', s\rangle) \to q\}$.

$\mathsf{DEL}_{\mathsf{s}}$ if $a(x) \to x \in \mathcal{R}/\mathcal{A}$, $b \in \Sigma$, $B \in \mathcal{C}$, s, s' are states of B, $q, q' \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B_{s,s'}) \hookrightarrow_{\Delta_i} q'$, then $\Delta_{i+1} := \Delta_i \cup \{b(B + \langle s, q', s'\rangle) \to q\}$.

Note that $\mathsf{INS}_{\mathsf{first}}$, $\mathsf{INS}_{\mathsf{last}}$, RPL_1 are special cases of respectively $\mathsf{RNS}_{\mathsf{first}}$, $\mathsf{RNS}_{\mathsf{last}}$, RPL. Since no state is added to the original automaton $\mathcal{A}_L$ and all the transitions added involve horizontal languages of the set $\mathcal{C}$, which is finite, the iteration of the above operations terminates with an automaton $\mathcal{A}'$. We show in the long version of this paper that $L(\mathcal{A}') = pre^*_{\mathcal{R}/\mathcal{A}}(L)$. □

4. Selection of Target Nodes

In general, an XML update operation is applied to nodes (called *target* nodes in [8]) selected in a document using specified XPath 2.0 or XQuery expressions. In this section, we study an extension of PTRS which permits to model such a feature.

4.1 Controlled Rewriting

We consider a fragment XP of Regular XPath [38] with the following path expressions (where $a \in \Sigma$):

$$p \quad := \quad . \mid a \mid .. \mid p/p \mid p \cup p \mid p^* \mid p[q]$$

and the node expressions:

$$q := p \mid \mathsf{lab}(a) \mid q \wedge q \mid q \vee q \mid \neg q$$

$h, (n, n')$	$\Vdash$	.	iff $n = n'$
$h, (n, n')$	$\Vdash$	a	iff n' is a child of n and $h(n') = a$
$h, (n, n')$	$\Vdash$	..	iff n is a child of n'
$h, (n, n')$	$\Vdash$	p/p'	iff $\exists m \in dom(h)$ such that $h, (n, m) \Vdash p$ and $h, (m, n') \Vdash p'$
$h, (n, n')$	$\Vdash$	$p \cup p'$	iff $h, (n, n') \Vdash p$ or $h, (n, n') \Vdash p'$
$h, (n, n')$	$\Vdash$	p^*	iff $\exists n_0, \ldots, n_k, n_0 = n, n_k = n'$, and $h, (n_i, n_{i+1}) \Vdash p$ for all $i < k$
$h, (n, n')$	$\Vdash$	$p[q]$	iff $h, (n, n') \Vdash p$ and $h, n' \models q$
h, n	$\models$	p	iff $\exists n',\ h, (n, n') \Vdash p$
h, n	$\models$	$\mathsf{lab}(a)$	iff $h(n) = a$
h, n	$\models$	$q \vee q'$	iff $h, n \models q$ or $h, n \models q'$
h, n	$\models$	$\neg q$	iff $h, n \not\models q$

Figure 2. Semantics of Path and Node Expressions

The satisfaction of a path expression p by a hedge h and a pair of nodes $n, n' \in dom(h)$, denoted by $h, (n, n') \Vdash p$, and of a node expression q by a hedge h and one node $n \in dom(h)$, denoted $h, n \models q$, are defined in Figure 2. Given a path expression p, we use below the abbreviation $//p$ for the path expression $(a_1 \cup \ldots \cup a_k)^*/p$ (assuming $\Sigma = \{a_1, \ldots, a_k\}$) and we shall omit a . at the beginning of an expression.

A *controlled term rewriting system* over Σ is a set $\mathcal{R}$ of controlled rewrite rules of the form $\ell \to r$ at ϕ where $\ell, r \in \mathcal{H}(\Sigma, \mathcal{X})$ and ϕ is a path expression of XP. The rewrite relation of $\mathcal{R}$ is defined as the rewrite relation of uncontrolled systems (see Section 2.3) by furthermore restricting the rewrite nodes to nodes defined by ϕ. More precisely, $h \xrightarrow[\mathcal{R}]{} h'$, iff there exists a controlled rule $\ell \to r$ at ϕ in $\mathcal{R}$, a substitution σ, and a context u such that the node n labelled by the variable x_u in the context u is *selected* by ϕ, i.e. there exists a root n_0 of h such that $h, (n_0, n) \Vdash \phi$, and $h = u[\ell\sigma]$, $h' = u[r\sigma]$. Note that for applying a rule $\ell \to r$ at ϕ it is expected for the path expression ϕ and the lhs ℓ to match the same labels.

A *controlled term rewriting system parameterized* by a HA (**CPTRS**) over Σ is a finite set of controlled and parameterized rewrite rules $\ell \to r$ at ϕ, where ℓ and r are like in the definition of PTRS in Section 2.3 and ϕ is as above. The rewrite relation of a CPTRS parameterized by $\mathcal{A}$ is defined as the rewrite relation of the associated CTRS $\mathcal{R}[\mathcal{A}]$ like in Section 2.3.

4.2 Selection by Label

The PTRS rewrite rules of Section 3 permit to define a minimal criteria for the selection of rewrite nodes (node where the updates operations are applied), by specifying the label of the selected node. Indeed, all the left-hand-sides of rules have the form $a(x)$ (or $a(xy)$ for $\mathsf{INS}_{\mathsf{into}}$). For instance, in the case of a rule of $\mathsf{INS}_{\mathsf{first}}$: $a(x) \to a(p\,x)$, a term of type p (w.r.t. to the given HA $\mathcal{A}$) can only be inserted below a node labeled with a. For a rule $\mathsf{INS}_{\mathsf{after}}$: $a(x) \to a(x)\,p$, a term of type p (w.r.t. to the given HA $\mathcal{A}$) can only be inserted at the sibling position next to a node labeled with a, and for DEL: $a(x) \to ()$, the term to be deleted must have a label a at its root node. It means that a PTRS rule $a(x) \to r$ is semantically equivalent to the CPTRS rule $a(x) \to r$ at $//a$.

4.3 Selection by Label and Parent's Label

For the rules with a hedge at right-hand-side (like $\mathsf{INS}_{\mathsf{before}}$, $\mathsf{INS}_{\mathsf{after}}$, RPL_1, DEL, $\mathsf{DEL}_{\mathsf{s}}$...), we can refine the selection by furthermore constraining the label at the parent of the node where the update is performed, obtaining the generalized rules of Figure 3. Indeed, every PTRS rule of the form $b(y\,a(x)\,z) \to b(y\,r\,z)$ in Figure 3 is semantically equivalent to the CPTRS rule $a(x) \to r$ at $//b/a$.

EXAMPLE 5. *The* DEL′ *rule*

$$\mathsf{hospital}(y\,\mathsf{patient}(x)\,z) \to \mathsf{hospital}(y\,z)$$

can be used to delete a patient *only if it is located under a* hospital *node selected by the path expression* //hospital[./patient]. *It corresponds to the CPTRS rule* $\mathsf{patient}(x) \to ()\mathtt{at}//\mathsf{hospital}/\mathsf{patient}$.

Let us call UFO′+ the class of all PTRS with rules in UFO+ or of a kind described in Figure 3. The result of Theorem 3 for backward type inference can be straightforwardly extended from UFO+ to UFO′+. For instance, during the iteration, if the PTRS $\mathcal{R}/\mathcal{A}$ contains a rule $\mathsf{INS}'_{\mathsf{before}}$ $b(y\,a(x)\,z) \to b(y\,p\,a(x)\,z)$, and if $B, B' \in \mathcal{C}$, s, s' are states of B, and $q, q_p, q' \in Q_L$ such that $s \xrightarrow[B]{q_p q'} s'$, $b(B) \to q$ is one of the current transitions and $a(B')$ can reach q' and some term of $L(\mathcal{A}, p)$ can reach q_p using the current transitions, then we add the transition $b\big(B \cup \{\langle s, q', s'\rangle\}\big) \to q$.

THEOREM 4. *Given a HA $\mathcal{A}$ on Σ and a PTRS $\mathcal{R}/\mathcal{A} \in$ UFO′+, for all HA language L, $pre^*_{\mathcal{R}/\mathcal{A}}(L)$ is the language of a HA of size exponential and which can be constructed in EXPTIME in the size of $\mathcal{R}/\mathcal{A}$ and of an HA recognizing L.*

Proof. The proof is very close to the proof of Theorem 3. Indeed, in the above construction for Theorem 3, we consider the applications of rules $\mathsf{INS}_{\mathsf{before}}$, $\mathsf{INS}_{\mathsf{after}}$, RPL, DEL and $\mathsf{DEL}_{\mathsf{s}}$ under any symbol $b \in \Sigma$. Here instead, we can restrict the construction to the application under the symbol specified in the lhs of the rewrite rules. More precisely, let us just detail below the cases of the construction which are modified.

- $\mathsf{INS}'_{\mathsf{before}}$ if $b(y\,a(x)\,z) \to b(y\,p\,a(x)\,z) \in \mathcal{R}/\mathcal{A}$, $B, B' \in \mathcal{C}$, s, s' are states of B, and $q, q_p, q' \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B') \hookrightarrow_{\Delta_i} q'$, $L(\mathcal{A}_i, q_p) \cap L(\mathcal{A}, p) \neq \emptyset$, $s \xrightarrow[B]{q_p q'} s'$, then $\Delta_{i+1} := \Delta_i \cup \big\{b(B + \langle s, q', s'\rangle) \to q\big\}$.
- $\mathsf{INS}'_{\mathsf{after}}$ if $b(y\,a(x)\,z) \to b(y\,a(x)\,p\,z) \in \mathcal{R}/\mathcal{A}$, $B, B' \in \mathcal{C}$, s, s' are states of B, and $q, q_p, q' \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B') \hookrightarrow_{\Delta_i} q'$, $L(\mathcal{A}_i, q_p) \cap L(\mathcal{A}, p) \neq \emptyset$, $s \xrightarrow[B]{q' q_p} s'$, then $\Delta_{i+1} := \Delta_i \cup \big\{b(B + \langle s, q', s'\rangle) \to q\big\}$.
- RPL′ if $b(y\,a(x)\,z) \to b(y\,p_1 \ldots p_n\,z) \in \mathcal{R}/\mathcal{A}$, $B, B' \in \mathcal{C}$, s, s' are states of B, and $q, q', q_1, \ldots, q_n \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B') \hookrightarrow_{\Delta_i} q'$, $L(\mathcal{A}_i, q_j) \cap L(\mathcal{A}, p_j) \neq \emptyset$ for all $1 \leq j \leq n$, $s \xrightarrow[B]{q_1 \ldots q_n} s'$ then $\Delta_{i+1} := \Delta_i \cup \big\{b(B + \langle s, q', s'\rangle)) \to q\big\}$.
- DEL′ if $b(y\,a(x)\,z) \to b(yz) \in \mathcal{R}/\mathcal{A}$, $B, B' \in \mathcal{C}$, s is a state of B, $q, q' \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B') \hookrightarrow_{\Delta_i} q'$, then $\Delta_{i+1} := \Delta_i \cup \big\{b(B + \langle s, q', s\rangle) \to q\big\}$.
- $\mathsf{DEL}'_{\mathsf{s}}$ if $b(y\,a(x)\,z) \to b(yxz) \in \mathcal{R}/\mathcal{A}$, $B \in \mathcal{C}$, s, s' are states of B, $q, q' \in Q_L$ such that $b(B) \to q \in \Delta_i$, $a(B_{s,s'}) \hookrightarrow_{\Delta_i} q'$, then $\Delta_{i+1} := \Delta_i \cup \big\{b(B + \langle s, q', s'\rangle) \to q\big\}$.

The rest of the proof is the same as for Theorem 3. □

4.4 Selection by XPath Expressions

Allowing more navigation axis, like the parent axis, in the control expressions ϕ of the CPTRS rules leads to the undecidability of reachability, hence of typechecking.

More precisely, let XP_1 be the following fragment of path expressions of XP (where $a \in \Sigma$):

$$p_1 := . \mid a \mid .. \mid p_1/p_1 \mid p_1 \cup p_1 \mid p_1[\mathsf{lab}(a)]$$

THEOREM 5. *Reachability is undecidable for CPTRS with rules of the form $\ell \to r$ at ϕ with $\ell \to r \in$ UFO$_{\mathsf{reg}}$ of type* REN *or* RPL_1, *and $\phi \in \mathsf{XP}_1$.*

Proof. The proof is very close to the proof of undecidability of inconsistency of update ACPs in [17]. We reduce the halting problem of a deterministic Turing Machine (TM) $\mathcal{M}$ that work on half a tape (unbounded on the right). Let $\Gamma = \{0, 1, \flat\}$ be the tape alphabet ($\flat$ is the blank symbol) and $S = \{s_1, s_2, \ldots, s_n\}$ be the state set of $\mathcal{M}$.

We consider the alphabet $\Sigma := \{g\} \cup \Gamma \cup (S \times \Sigma) \cup (S \times \Sigma)'$ for representing the configurations of $\mathcal{M}$ as binary terms. A symbol of the form $\langle s, a\rangle$ with $s \in S$ and $a \in \Gamma$ will be used to indicate the position of the head of $\mathcal{M}$. For instance, the TM configuration with tape $abcde\,\flat\,\flat\ldots$, symbol d under head, and state s will be represented by the following binary term of $\mathcal{T}(\Sigma)$: $g(a\,g(b\,g(c\,g(\langle s, d\rangle\,g(eg(\flat\,\flat))))))$.

We also use a trivial HA automaton $\mathcal{A} = (\Sigma, Q', Q', \delta)$ to recognize some particular terms: every term of the form $g(\langle r, \flat\rangle', \flat)$ (with $r \in S$) will be recognized in a state $p_{g(\langle r,\flat\rangle',\flat)} \in Q'$, and it is the only term recognized in this state.

We define a CPTRS $\mathcal{R}/\mathcal{A}$ such that every transition of $\mathcal{M}$ can be simulated by a sequence of (at most three) rewrite steps with $\mathcal{R}/\mathcal{A}$.

For each TM instruction of type: *"In state s reading a go to state r and write b"*, we define the following uncontrolled PTRS rule (of type REN): $\langle s, a\rangle(x) \to \langle r, b\rangle(x)$.

For each TM instruction of type: *"In state s reading a go to state r and move left"*, we define the following CPTRS rules:

1. $b(x) \to \langle r, b\rangle'(x)$ at $//\langle s, a\rangle/../../b$ (for all $b \in \{0, 1\}$), (the symbol b at the left of the head - marked by $\langle s, a\rangle$ - is renamed into the temporary symbol $\langle r, b\rangle'$)
2. $\langle s, a\rangle(x) \to a(x)$ at $//\langle r, b\rangle'/g/\langle s, a\rangle$ ($\langle s, a\rangle$ is renamed into a if it has $\langle r, b\rangle'$ at its left),
3. $\langle r, b\rangle'(x) \to \langle r, b\rangle(x)$ at $//a/../../\langle r, b\rangle'$ ($\langle r, b\rangle'$ is renamed into $\langle r, b\rangle$, which marks the new position of the head).

Note the use of the XPath expressions (selecting rewrite nodes) for checking the neighbor symbol and ensuring a correct chaining of the rewrite steps. Note also that for the first rule, if a is the first symbol of the tape, then the rule cannot be applied because of the path expression, this corresponds to the fact that the Turing machine cannot move to the left of the beginning of the tape.

For a transition of $\mathcal{M}$ moving to the right, we also add a RPL rule for moving the $\flat$ marker. More precisely, for each instruction of type: *"In state s reading a go to state r and move right"*, we define the following CPTRS rules of type REN and RPL_1 (we recall that $p_{g(\langle r,\flat\rangle',\flat)}$ is a state of $\mathcal{A}$):

$b(x)$	$\to$	$\langle r, b\rangle'(x)$	at $//\langle s, a\rangle/../g/g/b$ for all $b \in \{0, 1\}$
$\flat(x)$	$\to$	$p_{g(\langle r,\flat\rangle',\flat)}$	at $//\langle s, a\rangle/../g/\flat$
$\langle s, a\rangle(x)$	$\to$	$a(x)$	at $//\langle r, b\rangle'/../../\langle s, a\rangle$
$\langle r, b\rangle'(x)$	$\to$	$\langle r, b\rangle(x)$	at $//a/../g/g/\langle r, b\rangle'$

The TM instruction will be executed in three rewrite steps: first the symbol at position at the right of the head (marked by $\langle s, a\rangle$) is renamed from b into the temporary symbol $\langle r, b\rangle'$. Next $\langle s, a\rangle$ is renamed into a and finally $\langle r, b\rangle'$ is renamed into $\langle r, b\rangle$, which marks the new position of the head. The tests in the path expressions for the selection of rewrite nodes will ensure a correct chaining of the rewrite steps: at each step, we check the neighbor position in order to test that the previous step has been applied.

For all couple of TM configurations T_1, T_2 and their respective term encodings t_1, t_2, there is a sequence of transitions from T_1 to T_2 with $\mathcal{M}$ iff $t_1 \xrightarrow[\mathcal{R}/\mathcal{A}]{*} t_2$.

Assuming (wlog) that $\mathcal{M}$ has unique initial and final configurations, we can conclude. □

$b(y\,a(x)\,z)$	$\to$	$b(y\,p\,a(x)\,z)$	$\mathsf{INS}'_{\mathsf{before}}$				
$b(y\,a(x)\,z)$	$\to$	$b(y\,a(x)\,p\,z)$	$\mathsf{INS}'_{\mathsf{after}}$				
$b(y\,a(x)\,z)$	$\to$	$b(y\,p\,z)$	RPL'_1	$b(y\,a(x)\,z)$	$\to$	$b(y\,p_1\ldots p_n\,z)$	RPL'
$b(y\,a(x)\,z)$	$\to$	$b(y\,z)$	DEL'	$b(y\,a(x)\,z)$	$\to$	$b(y\,x\,z)$	$\mathsf{DEL}'_{\mathsf{s}}$

Figure 3. PTRS rules for Update Facility Operations with Control of Parent's Label

5. ACP for XML Updates

In this last section we study some models of Access Control Policies (ACP) for the update operations defined in Section 3, and verification problems for these ACP. We consider two kind of formalisms from the literature for the specification of XML ACPs. The first formalism is the most widespread. It consists in defining an ACP as a set of updates rules, partitioned into authorized and forbidden operations. The second one is a most recent proposal of [17] where the ACP is defined by adding security annotations to a DTD.

5.1 Local Consistency of Rule-based ACPs

An ACP for XML updates can be defined by a pair $(\mathcal{R}_{\mathsf{a}}/\mathcal{A}, \mathcal{R}_{\mathsf{f}}/\mathcal{A})$ of PTRS, where $\mathcal{R}_{\mathsf{a}}$ contains allowed operations and $\mathcal{R}_{\mathsf{f}}$ contains forbidden operations (see e.g. [6]). Such an ACP is called *inconsistent* [6, 17] if some forbidden operation can be simulated through a sequence of allowed operations, *i.e.* if there exists $t, u \in \mathcal{T}(\Sigma)$ such that $t \xrightarrow[\mathcal{R}_{\mathsf{f}}/\mathcal{A}]{} u$ and $t \xrightarrow[\mathcal{R}_{\mathsf{a}}/\mathcal{A}]{*} u$.

EXAMPLE 6. *Assume that in the* hospital *document of example 2, it is forbidden to rename a* patient*, that is the following update of* RPL'_1 *is forbidden:* $\mathsf{patient}(y\,\mathsf{name}(x)\,z) \to \mathsf{patient}(y\,p_{\mathsf{n}}\,z)$*. If the following updates are allowed:* $\mathsf{patient}(x) \to ()$ *for deleting a* patient*, and* $\mathsf{hospital}(x) \to \mathsf{hospital}(x\,p_{\mathsf{pa}})$ *to insert a* patient*, then we have an inconsistency in the sense of [6] since the effect of the forbidden update can be obtained by a combination of allowed updates.*

Using the results of Section 3, we can decide the above problems individually for terms of $\mathcal{T}(\Sigma)$. More precisely, we solve the following problem called *local inconsistency*: given a HA $\mathcal{A}$ over Σ and a term $t \in \mathcal{T}(\Sigma)$, an ACP $(\mathcal{R}_{\mathsf{a}}/\mathcal{A}, \mathcal{R}_{\mathsf{f}}/\mathcal{A})$ is locally inconsistent if there exists $u \in \mathcal{T}(\Sigma)$ such that $t \xrightarrow[\mathcal{R}_{\mathsf{f}}/\mathcal{A}]{} u$ and $t \xrightarrow[\mathcal{R}_{\mathsf{a}}/\mathcal{A}]{*} u$?

THEOREM 6. *Local inconsistency is decidable in PTIME for* $\mathsf{UFO}+$ *ACPs.*

Proof. It can be easily shown that the set $\{u \in \mathcal{T}(\Sigma) \mid t \xrightarrow[\mathcal{R}_{\mathsf{f}}/\mathcal{A}]{} u\}$ is the language of a HA of size polynomial and constructed in PTIME on the sizes of $\mathcal{A}$, $\mathcal{R}_{\mathsf{f}}$ and t. By Theorem 2, $post^*_{\mathcal{R}_{\mathsf{a}}/\mathcal{A}}(\{t\})$ is the language of a CF-HA of polynomial size and constructed in polynomial time on the sizes of $\mathcal{A}$, $\mathcal{R}_{\mathsf{a}}$ and t. The ACP is locally inconsistent w.r.t. t iff the intersection of the two above language is not empty, and this property can be tested in PTIME. □

Inconsistency is undecidable [17] for an ACP defined by a pair $(\mathcal{R}_{\mathsf{a}}/\mathcal{A}, \mathcal{R}_{\mathsf{f}}/\mathcal{A})$ of CPTRS of Section 4. Moreover, in this setting, the problem of reachability (whether a given term t can be obtained from a given term s using instances of rules of $\mathcal{R}_{\mathsf{a}}/\mathcal{A}$ which are not in $\mathcal{R}_{\mathsf{f}}/\mathcal{A}$) is also undecidable [29], therefore local consistency is undecidable as well. It is an open question whether inconsistency is decidable or not for PTRS of type $\mathsf{UFO}_{\mathsf{reg}}$ or $\mathsf{UFO}+$.

5.2 Local Consistency of DTD-based ACPs

Following the principle of DTD-based ACPs [14], [17] have proposed the language $\mathsf{XACU}_{\mathsf{annot}}$ for the definition of ACP for XML updates in presence of a DTD D. The idea is to add to D some security annotations specifying the authorizations for the update operations for XML documents valid for D. In [14], the annotations are mapping from pairs of DTD elements types (b, a) to authorization, specifying the right to access a nodes which are children of b nodes. Such annotations can be compared to the rewrite node selection presented in Section 4.3.

The formalism of [14, 17] imposes the condition that every document t to which we want to apply an update operation (under the given ACP) must be valid for the DTD D.

In our rewrite-based formalism, the above condition may be expressed by adding global constraints to the parameterized rewrite rules of Section 2.3. These global constraints restrict the rewrite relation to terms in a given HA language. Theorem 7 below shows that, unfortunately, adding such constraints to parameterized rewrite rules of type REN or RPL makes the reachability undecidable.

Given a HA $\mathcal{A} = (\Sigma, Q, Q^{\mathsf{f}}, \Delta)$, a term rewriting system over Σ, parameterized by $\mathcal{A}$ and with global constraints (**PGTRS**) is given by a PTRS, denoted $\mathcal{R}/\mathcal{A}$, (see Section 2.3) and $L \subseteq \mathcal{T}(\Sigma)$ an HA language. We say that L is the constraint of R. The rewrite relation generated by the PGTRS is defined as the restriction of the relation defined in Section 2.3 to ground terms such that for the application of a rule $\ell \to r \in \mathcal{R}/\mathcal{A}$ to a term t, we require that $t \in L$.

THEOREM 7. *Reachability is undecidable for PGTRS's with rules in* $\mathsf{UFO}_{\mathsf{reg}}$ *and constraint given by a non recursive DTD.*

Proof. The proof is a variant of the one given by A. Spelten [37] for subterm and flat prefix rewriting. Like in the proof of Theorem 5, we will reduce the halting problem of a Deterministic Turing Machine (TM) $\mathcal{M}$ that work on half a tape (unbounded on the right). However, configurations are now encoded as flat terms.

We consider the same tape alphabet $\Gamma = \{0, 1, \flat\}$, ($\flat$ is the blank symbol) and state set $S = \{s_1, s_2, \ldots, s_n\}$ of $\mathcal{M}$ as in the proof of Theorem 5, and the following alphabet Σ for the representation of the configurations of $\mathcal{M}$.

$$\Sigma := \{g\} \cup \{0, 1, \flat\} \cup (S \times \Sigma) \cup (S \times \Sigma)'\}.$$

For instance, the TM configuration with tape $abcde\flat\flat\ldots$, symbol d under head, state s, will be represented by the following flat term of $\mathcal{T}(\Sigma)$: $g(abc\langle s, d\rangle e\,\flat\,\flat)$.

We shall also use a trivial HA automata $\mathcal{A} = (\Sigma, Q', Q', \delta\})$ (as in the proof of Theorem 5) which recognizes only constant symbols by taking $Q' = \{p_\sigma | \sigma \in \Sigma\}$ and $\delta = \{\sigma \to p_\sigma \mid \sigma \in \Sigma\}$.

We define a PGTRS $\mathcal{R}/\mathcal{A}$ such that every transition of $\mathcal{M}$ can be simulated by a sequence of (at most three) rewrite steps with $\mathcal{R}/\mathcal{A}$. Let us first introduce some standard auxiliary PTRS rules and some word regular languages for controlling rule applications.

For each instruction of $\mathcal{M}$ of type: *"In state s reading a go to state r and write b"*, we define the following TRS rule:

$$\langle s, a\rangle(x) \to \langle r, b\rangle(x)$$

We also define the regular word language

$$L_{\langle s,a\rangle} = \Gamma^*\langle s, a\rangle\,\Gamma^*.$$

For each instruction of $\mathcal{M}$ of type: *"In state s reading a go to state r and move right"*, we define the following PTRS rules of

types REN and $\mathsf{INS}_{\mathsf{after}}$ (note that $p_\flat$ is a state of $\mathcal{A}$):

$$\begin{array}{lcl} b(x) & \to & \langle r,b\rangle'(x) \text{ for all } b \in \{0,1,\flat\} \\ \flat(x) & \to & \flat(x)\ p_\flat \\ \langle s,a\rangle(x) & \to & a(x) \\ \langle r,b\rangle'(x) & \to & \langle r,b\rangle(x) \text{ for all } b \in \Gamma. \end{array}$$

We also define the regular word languages:

$$\begin{array}{l} L_{\langle s,a\rangle} = \Gamma^*\langle s,a\rangle\,\Gamma^* \\ L_{\langle s,a\rangle'} = \Gamma^*\langle s,a\rangle'\,\Gamma^* \\ L_{\langle s,a\rangle\langle r,b\rangle'} = \Gamma^*\langle s,a\rangle\langle r,b\rangle'\,\Gamma^* \text{ for all } b \in \Gamma. \end{array}$$

For each instruction of $\mathcal{M}$ of type: *"In state s reading a go to state r and move left"*, we define the following TRS rules:

$$\begin{array}{lcl} b(x) & \to & \langle r,b\rangle'(x) \text{ for all } b \in \{0,1\} \\ \langle s,a\rangle(x) & \to & a(x) \\ \langle r,b\rangle'(x) & \to & \langle r,b\rangle(x) \text{ for all } b \in \{0,1\} \end{array}$$

We also define the regular word languages:

$$\begin{array}{lcl} L_{\langle s,a\rangle} & = & \Gamma^*\langle s,a\rangle\,\Gamma^* \\ L_{\langle s,a\rangle'} & = & \Gamma^*\langle s,a\rangle'\,\Gamma^* \\ L_{\langle r,b\rangle'\langle s,a\rangle} & = & \Gamma^*\langle r,b\rangle'\langle s,a\rangle\,\Gamma^* \text{ for all } b \in \{0,1\}. \end{array}$$

The constraint of the PGTRS will be defined by the non recursive DTD $D : g \to L$ where L is the finite union of the regular languages associated to the instructions of $\mathcal{M}$ as above. Since the machine to be simulated is deterministic, the union is disjoint.

Our final PGTRS is given by $\mathcal{R}/\mathcal{A}$ and L so that the rewrite rules in $\mathcal{R}/\mathcal{A}$ can only be applied to terms satisfying the DTD D. With the above constraint, the PGTRS rules of $\mathcal{R}/\mathcal{A}$ can only be applied to terms valid for the DTD D, ensuring a correct chaining for the application of these rules.

By case inspection we can show that for any couple of TM configurations T_1, T_2 and their respective term encodings t_1, t_2, there is a sequence of transitions from T_1 to T_2 iff $t_1 \xrightarrow[\mathcal{R}/\mathcal{A}]{*} t_2$. The theorem follows. □

Note that the above result also holds for PGTRS's with rules are ground (without variables nor parameters): in the above rewrite rules, every variable x could be replaced by the empty hedge $()$, and every parameter such as $p_\flat$ could be replaced by the corresponding ground term $\flat$. Hence the above result can be contrasted with the decidability of reachability for ground rewriting [20].

In [1] the author study the more general problem of *satisfiability* for active XML documents in the context and unranked unordered terms. This property is shown decidable for insertions constrained by an unordered DTD, but undecidable when they are constrained by an unordered HA.

COROLLARY 3. *Local inconsistency is undecidable for PGTRS with rules in UFO+ and with constraint given by a non recursive DTD.*

6. Conclusion

We have proposed a model for the primitive XML updates operations of [8] based on term rewriting systems parameterized by hedge automata (PTRS), and studied the problems of type inference and typechecking for arbitrary long sequences of such operations. We have also studied some extensions of the model for selecting the rewrite positions with XPath expressions (CPTRS) and restricting of the application of update operations to documents conforming to a fixed non recursive DTD (PGTRS). Finally, we have shown how to apply our results to show the decidability of the property of local inconsistency of access control policies for XML updates.

One of our main results of forward type inference (Theorem 2) requires to use CF-HA (a strict extension of hedge automata) for output types. One may wonder whether this result could be adapted to compute regular over-approximations of output types, leading to an approximating forward type inference algorithm, in an approach similar to e.g. [39].

Reachability is undecidable for CPTRS rules controlled with XPath expressions with child and parent axis. The cases of CPTRS rules controlled with a downward XPath fragment, or a downward regular XPath fragment, deserve to be considered. Indeed, a decidability result for typechecking in these settings should give a novel approach (using CPTRS) to known problems on other tree transformations formalisms (like MTTs or XSLT).

The W3C recommendation [8] defines some priorities for the application of update operations (for instance REN has higher priority than DEL). The influence of such restriction on type inference should be investigated. Finally, it could also be interesting to apply a similar approach for studying updates of unranked unordered trees.

Acknowledgments

The authors wish to thank Serge Abiteboul, Pierre Bourhis, Sebastian Maneth and Luc Segoufin for discussions discussions about XML updates and access control, and the anonymous referees for their numerous comments and suggestions.

References

[1] S. Abiteboul, P. Bourhis, and B. Marinoiu. Satisfiability and relevance for queries over active documents. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 87–96. ACM, 2009.

[2] M. Benedikt and J. Cheney. Semantics, Types and Effects for XML Updates. In *Proceedings of the 12th International Symposium, Database Programming Languages (DBPL)*, volume 5708 of *LNCS*, pages 1–17, Springer, 2009.

[3] V. Benzaken , G. Castagna and A. Frisch. CDuce: an XML-centric general-purpose language. In *Proceedings of the 8th ACM SIGPLAN International conference on Functional programming*, pages 51-63, ACM, 2003.

[4] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular Model Checking. In *Proceedings of the 12th Int. Conference on Computer Aided Verification (CAV)*, volume 1855 of *LNCS*, pages 403–418. Springer, 2000.

[5] A. Bouajjani and T. Touili. On computing reachability sets of process rewrite systems. In *Proceedings 16th International Conference Term Rewriting and Applications (RTA)*, volume 3467 of *LNCS*, pages 484–499. Springer, 2005.

[6] L. Bravo, J. Cheney, and I. Fundulaki. ACCOn: Checking Consistency of XML Write-Access Control Policies. In *Proceedings 11th International Conference on Extending Database Technology (EDBT)*, volume 261 of *ACM International Conference Proceeding Series*, pages 715–719. ACM, 2008.

[7] S. C. Lim and S. H. Son. Access Control of XML Documents Considering Update Operations. In *Proceedings of ACM Workshop on XML Security*, ACM, 2003.

[8] D. Chamberlin and J. Robie. XQuery Update Facility 1.0. W3C Candidate Recommendation. `http://www.w3.org/TR/xquery-update-10/`, 2009.

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

[10] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Securing XML Documents. In *Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*, volume 1777 of *LNCS*, pages 121–135. Springer, 2000.

[11] N. Dershowitz and J. P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science (Vol. B: Formal Models and Semantics)*, pages 243–320, Amsterdam, North-Holland, 1990.

[12] J. Engelfriet, S. Maneth, and H. Seidl. Deciding Equivalence of Top-Down XML Transformations in Polynomial Time. *J. Comput. Syst. Sci.*, 75(5):271–286, 2009.

[13] J. Engelfriet and H. Vogler. Macro Tree Transducers. *J. Comp. Syst. Sci.*, 31:71–146, 1985.

[14] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML Querying with Security Views. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 587–598, ACM, 2004.

[15] G. Feuillade, T. Genet, and V. Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *Journal of Automated Reasoning*, 33 (3-4):341–383, 2004.

[16] A. Frisch and H. Hosoya. Towards Practical Typechecking for Macro Tree Transducers. In *Proceedings of the 11th International Symposium on Database Programming Languages (DBPL)*, volume 4797 of *LNCS*, pages 246–260. Springer, 2007.

[17] I. Fundulaki and S. Maneth. Formalizing XML Access Control for Update Operations. In *Proceedings of the 12th ACM symposium on Access control models and technologies (SACMAT)*, pages 169–174, ACM, 2007.

[18] P. A. Gardner, G. D. Smith, M. J. Wheelhouse, and U. D. Zarfaty. Local Hoare Reasoning about DOM. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 261–270, ACM, 2008.

[19] T. Genet and V. Rusu. Equational approximations for tree automata completion. Journal of Symbolic Computation, 45 (5):574–597, 2010.

[20] R. Gilleron. Decision problems for term rewrite systems and recognizable tree languages. In *$8^t h$ Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 480 of *LNCS*, pages 148–159, Springer, 1991.

[21] F. Jacquemard and M. Rusinowitch. Closure of Hedge-Automata Languages by Hedge Rewriting. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA)*, volume 5117 of *LNCS*, pages 157–171, Springer, 2008.

[22] M. Kay. XSL Transformations (XSLT) 2.0. W3C working draft, World Wide Web Consortium, 2003. Available at `http://www.w3.org/TR/xslt20`.

[23] C. Löding. Ground Tree Rewriting Graphs of Bounded Tree Width. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2285 of *LNCS*, pages 559–570. Springer, 2002.

[24] C. Löding and A. Spelten. Transition Graphs of Rewriting Systems over Unranked Trees. In *Proceedings 32nd International Symposium on Mathematical Foundations of Computer Science (MFCS)* volume 4708 of *LNCS*, pages 67–77, Springer, 2007.

[25] S. Maneth, A. Berlea, T. Perst, and H. Seidl. XML Type Checking with Macro Tree Transducers. In *24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, pages 283–294, ACM, 2005.

[26] S. Maneth, T. Perst, and H. Seidl. Exact XML Type Checking in Polynomial Time. In *Proceedings of the 11th International Conference on Database Theory (ICDT)*, volume 4353 of *LNCS*, pages 254–268, Springer, 2007.

[27] W. Martens and F. Neven. Frontiers of Tractability for Typechecking Simple XML Transformations. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 23–34, ACM, 2004.

[28] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML Transformers. *J. of Comp. Syst. Sci.*, 66(1):66–97, 2003.

[29] N. Moore. The Halting Problem and Undecidability of Document Generation under Access Control for Tree Updates, In *Proceedings of the 3d International Conference on Language and Automata Theory and Applications (LATA)*, volume 5457 of *LNCS*, pages 601-613, Springer, 2009.

[30] M. Murata. “Hedge Automata: a Formal Model for XML Schemata”. Web page, 2000.

[31] M. Murata, D. Lee, and M. Mani. Taxonomy of XML Schema Languages using Formal Language Theory. In *Extreme Markup Languages*, 2000.

[32] M. Murata, A. Tozawa, M. Kudo, and S. Hada. XML Access Control using Static Analysis. *ACM Trans. Inf. Syst. Secur.*, 9(3):292–324, 2006.

[33] H. Ohsaki, H. Seki, and T. Takai. Recognizing Boolean Closed A-tree languages with Membership Conditional Rewriting Mechanism. In *Proc. of the 14th Int. Conference on Rewriting Techniques and Applications (RTA)*, volume 2706 of *LNCS*, pages 483–498. Springer, 2003.

[34] T. Perst and H. Seidl. Macro Forest Transducers. *Information Processing Letters*, 89:141–149, 2004.

[35] T. Schwentick. Automata for XML - A Survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.

[36] H. Seidl. Deciding Equivalence of Finite Tree Automata. *SIAM Journal of Computing*, 19(3):424–437, 1990.

[37] A. Spelten. Rewriting Systems over Unranked Trees. Master’s thesis, Diplomarbeit, RWTH Aachen, 2006.

[38] B. ten Cate. The Expressivity of XPath with Transitive Closure. In *Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 328–337, ACM, 2006. ISBN 1-59593-318-2.

[39] T. Touili. Computing Transitive Closures of Hedge Transformations. In *In Proceedings of the 1st International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS)*, eWIC Series, British Computer Society, 2007.

[40] A. Tozawa. Towards Static Type Checking for XSLT. In *Proceedings of the 2001 ACM Symposium on Document engineering (DocEng)*, pages 18–27, ACM, 2001.