Dipartimento di Informatica e
Scienze dell'Informazione

XML Schema Update: basic tecniques and extensions

A. Solimando, G. Delzanno, G. Guerrini
DISI, Università di Genova, Italy
alessandro.solimando@gmail.com {delzanno,guerrini}@disi.unige.it

Technical Reports Series **DISI-TR-2011-11-08**

DISI, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy

August 2011
http://www.disi.unige.it/

Abstract

Our world is strongly characterized by a continuous stream of data, from which we need to extract relevant information. This data are often represented using Extensible Markup Language (XML), language that have become in the last year a standard for structured and semi-structured representation of data. Most of the real use-case scenarios for XML have to deal with rapidly evolving environments, where updates are not an exception but the rule. While query languages have been very deeply investigated and can benefit from static analysis techniques, for update languages this is not the case. In the present work we want to propose a static analysis framework for a subset of XQuery Update Facility ([17]) one of the most relevant update language, starting from the proposal of a novel parallel rewriting semantic that have been used to build an algorithm capable to recognize the language generated by the $Post$ operation, that takes as its input the sequence of document update operations and the automaton related to the original regular language. With this framework it is possible to test a user-proposed sequence of document update operations thought to adapt the documents related to a schema that have just been updated by a known sequence of schema update operations: the framework can statically determine, working only with the automata related to the original and modified schema, if the document update operation sequence will preserve the validity of the documents; if so, every document valid wrt the original schema, after the application of the document update sequence will result in a new modified document valid wrt the updated schema. This avoid the usually very expensive run-time revalidation of the set of involved documents.

1 Introduction

Our world is strongly characterized by a continuous stream of data, but it is well known that the equation "data equals information" is largely false, and as many data we have, as much it becomes difficult to extract information from them, that is what we are really interested in. Data, though, because of the complex nature of the modern infrastructures and thanks to the great spread of technology, come from heterogeneous sources, that's why a normalization phase is strongly needed, both in the structure and in the content. Extensible Markup Language (XML) became a standard for structured representation of data many years ago, feature that made it used in many different fields, from the interoperability between different systems, as with SOAP and REST protocols which are used for Web Services using XML to format the data that will be exchanged, or in the definition of standards for information representation to make them usable and visualizable in the same way by every system adherent to the standard: a valid example of this scenario is the work of the *Health Level Seven International* (HL7)[1], an international association who wants to define, between others goals, a standard for representing medical data through XML. These markup language have strongly influenced the Databas scene too, where almost all the most important DBMS support and implement languages to query and update XML documents. Even if the query languages have been well formalized and developed, the schema and document update handling have not yet been fully investigated. Many practical XML scenarios, indeed, have to deal with higly evolving environments, where the updates are very frequents and the number and size of involved data are considerable: this is the reason of the need of static analysis and validation techniques that can handle updates working directly on schema, thus avoiding a very expensive run-time revalidation phase of all the involved documents.

Contributions The novel contributions of the present work are three:

- the formalization of a parallel rewriting semantic able to apply in a single step all the needed modifications, with a behaviour much more close to the update language for XML than the interleaving rewriting semantic (widely used in the literature, like in [15] and [7]), which apply modifications to a single node nondeterministically selected among the mathing ones. in order to achieve the desidered result you will need to iterate the single modification until reaching the fixed point, that is when all the matching nodes have been modified;

[1] Their HL7 official site is available at `http://www.hl7.org/`.

- the description of an algorithm able to calculate the automaton that accepts exactly the language that result from the application of $Post$ operation, defined using the parallel rewriting semantic, starting from the sequence of update operations and the automaton able to accept the starting regular language;
- proposal of *Schema Update Framework*, a framework for static analysis of sequence of updates proposed after a schema update, that works on the automata associated to the schema thus avoiding a run-time revalidation of all the document related to it.

2 Preliminaries

In this section we will introduce the notations used in the rest of the work and some useful definitions. The main source for what appear in this section is [5].

2.1 Basic definitions

String prefixes Given a string $s \in L \subseteq \Sigma^*$ the set of its prefixes, $Pref_L(s)$, is defined in this way:

$$Pref_L(s) = \{t \mid s = tu \wedge u \in L\}.$$

Prefixes language Given a language $L \subseteq \Sigma^*$ we call **prefixes language**, $Pref_L$, the set of the prefixes of the elements of L. More formally:

$$Pref(L) = \bigcup_{s \in L} Pref_L(s).$$

Prefix-closed language A language $L \subseteq \Sigma^*$ is said **prefix-closed** if $Pref(L) = L$, that is if the language contains every possible prefix of every string belonging to the set itself.

Terms The **terms** are the elements of a ranked alphabet defined as (Σ, *Arity*), where Σ is a finite and nonempty alphabet, *Arity*: $\Sigma \to \mathbb{N}$ is a function that associates a natural number, called **arity** of the symbol, to every element of Σ. The set of symbols with arity p is stated with Σ_p (often, for sake of briefness we will use a compact notation, e.g.: $f(,,)$ is a term contained in Σ_3); Σ_0 is called the set of **constants**. Let $\mathcal{X}$ be a set of **variables**, disjoint from Σ_0. The set $T(\Sigma, \mathcal{X})$ of the terms over Σ and $\mathcal{X}$ is defined in this way:

- $\Sigma_0 \subseteq T(\Sigma, \mathcal{X})$
- $\mathcal{X} \subseteq T(\Sigma, \mathcal{X})$
- If $f \in \Sigma_p$, $p > 0$ and $t_1, \ldots, t_p \in T(\Sigma, \mathcal{X})$, then $f(t_1, \ldots, t_p) \in T(\Sigma, \mathcal{X})$.

Se $\mathcal{X} = \emptyset$ then we use $T(\Sigma, \mathcal{X})$ for $T(\Sigma)$ and its elements are called **ground terms**, that is they do not have variables. **Linear terms**, instead, are the elements of $T(\Sigma, \mathcal{X})$ for which every variable occurs at most once.

String automata and tree automata Tree automata, defined as in [5] (Chapter 1, Section 1, Subsection 1) can be seen as a natural extension of string automata considering the characters of a string as unary terms (e.g.: $abcdef \to a(b(c(d(e(f(\epsilon)))))))$ and node labels of a tree analyzed by a tree automaton as n-ary terms (more formally as elements of $T(\Sigma, \mathcal{X})$). Under this point of view many string automata considerations and definitions can be adapted in a straightforward way to tree automata.

Ranked tree A finite and ordered **ranked tree** t with labels in E is a map from a prefix-closed set $\mathcal{P}os(t) \subseteq \mathbb{N}^*$ into a set of labels E, with $\mathcal{P}os(t)$ having the following properties:

1. $\mathcal{P}os(t)$ is finite, nonempty and prefix-closed.
2. $\forall p \in \mathcal{P}os(t)$, if $t(p) \in \Sigma_n$ and $n > 0$, then $\{j \mid pj \in \mathcal{P}os(t)\} = \{1, \ldots, n\}$.
3. $\forall p \in \mathcal{P}os(t)$, if $t(p) \in \Sigma_0 \cup \mathcal{X}$, then $\{j \mid pj \in \mathcal{P}os(t)\} = \emptyset$.

The elements of $\mathcal{P}os(t)$, called **positions**, can be seen as the paths to "reach", starting from the root, every node of the tree, that's why it need to be prefix-closed. The lack of a prefix for an element of $\mathcal{P}os(t)$ would let him unreachable and t would not be a tree. For instance if $121 \in \mathcal{P}os(t)$ and $12 \notin \mathcal{P}os(t)$, node $t(121)$ (that is the node of t in position 121) could not be reachable for the lack of the edge between node $t(1)$ and node $t(12)$.

The second property forces every node t to be labelled with an element of Σ_n that have exactly n children, $t_1, \ldots, t_n$, and that their positions are the concatenation of the paths of their father and a natural number in $[1, n]$, correspondant to their index (that is unique because this is an ordered tree, that is a tree having an order for all the children of a node, expressed through the index associated to them).

The third property force that only leaf nodes (that are the nodes without children) can be labelled with a variabile or a constant (that is a symbol of Σ_0).

We also define:

$\mathcal{FP}os(t) \subseteq \mathcal{P}os(t) = \{p \mid \forall j \in \mathbb{N}.pj \notin \mathcal{P}os(t)\}$, and its elements as **frontier positions**.

$\mathcal{VP}os(t) \subseteq \mathcal{P}os(t) = \{p \mid t(p) \in \mathcal{X}\}$, its elements are called **variable positions**.

$Root(t) = t(\epsilon)$, is called **root** of the tree.

Unranked tree [5] defines unranked tree as an extension of ranked trees. An **unranked tree** t with labels belonging to Σ, a set of unranked symbols, it is a map $t\colon \mathbb{N}^* \to \Sigma$ with a domanin, stated as $\mathcal{P}os(t)$, with the followings properties:

- $\mathcal{P}os(t)$ is a finite, nonempty and prefix-closed set,
- for every $p \in \mathcal{P}os(t)$ we have that $\{j \mid pj \in \mathcal{P}os(t)\} = \{1, \ldots, k\}$ for some $k > 0$.

The set of unranked tree over Σ is stated as $T(\Sigma)$. The definitions for the rest of the sections will be given only for the ranked tree because their adaptation to the unranked ones is straightforward.

Example 1: In Figure 1 we have an example of unranked tree. Note as the same label, used in different nodes, can have different number of children (it is an arbitraty but finite value). □

Terms and trees Here we define more formally the relationship between terms and trees.

A term $t \in T(\Sigma, \mathcal{X})$ can be seen as a finite and ordered tree, that is a partial function from $\mathbb{N}^*$ in $\Sigma \cup \mathcal{X}$ having $\mathcal{P}os(t)$ as its domain.

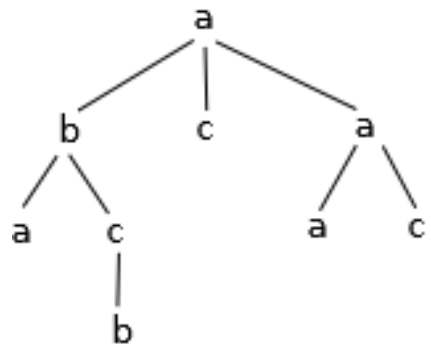

Figure 1. Example of unranked tree over the alphabet $\Sigma = \{a, b, c\}$.

Subterms The **subterm** (or subtree) $t|_p \in T(\Sigma, \mathcal{X})$ is the subterm in position p in a term (tree) $t \in T(\Sigma, \mathcal{X})$ such that:

1. $\mathcal{P}os(t|_p) = \{j \mid pj \in \mathcal{P}os(t)\}$,
2. $\forall q \in \mathcal{P}os(t|_p) \,.\, t|_p(q) = t(pq)$.

The first condition force new positions of the selected subtree to not to have p as a prefix, because it represents the path from the root of t to the one of the subtree $t|_p$, path that traverses nodes no more present in $t|_p$. Apart from the removing of the prefix p the positions of the elements of $t|_p$ are consistent with the positions of correspondant elements of t, as the second condition forces.

2.2 Functions defined over terms

Height *Height*: $T(\Sigma, \mathcal{X}) \to \mathbb{N}$ is a function defined inductively in this way:

- $Height(t) = 0$ if $t \in \mathcal{X}$,
- $Height(t) = 1$ if $t \in \Sigma_0$,
- $Height(t) = 1 + max(\{Height(t_i) \mid i \in [1, n]\})$, if $Root(t) \in \Sigma_n$.

Taking a tree as input $Height$ function returns its **height**.

Size $Size$: $T(\Sigma, \mathcal{X}) \to \mathbb{N}$ is a function defined inductively as follow:

- $Size(t) = 0$ if $t \in \mathcal{X}$,
- $Size(t) = 1$ if $t \in \Sigma_0$,
- $Size(t) = 1 + \sum_{i \in [1,n]} Size(t_i)$, if $Root(t) \in \Sigma_n$.

Taking a tree as input $Size$ function returns its **dimension**, that is the number of nodes labelled with elements of Σ.

2.3 Substitutions and contexts

Substitutions A **substitution** $\sigma\colon \mathcal{X} \to T(\Sigma, \mathcal{X})$ is a map for which $dom(\sigma) \subseteq \mathcal{X}$ and for every $x \in dom(\sigma)$ we have that $\sigma(x) \neq x$, whereas for every element $y \in \mathcal{X} \setminus dom(\sigma)$ we have that $\sigma(y) = y$. Thence a substitution $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ substitutes x_i with t_i, where $i \in [1, n]$. The substitution can be extended in this way to cover elements in $T(\Sigma, \mathcal{X})$:
$\forall f \in \Sigma_n, t_1, \ldots, t_n \in T(\Sigma, \mathcal{X})\ \sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$.

We say that a substitution is a **ground substitution** when $\sigma\colon \mathcal{X} \to T(\Sigma)$.

Contexts Let be $\mathcal{X}_n$ a set of n variables. A **context** $C \in T(\Sigma, \mathcal{X}_n)$ is a linear term, that is a term in which a variable can occur at most once. We use $C[t_1, \ldots, t_n] \in T(\Sigma)$, having $t_1, \ldots, t_n \in T(\Sigma)$, to denote a term obtained from the substitution of x_i with t_i, for $i \in [1, n]$, that is: $C[t_1, \ldots, t_n] = C[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$. We denote as $\mathcal{C}^n(\Sigma)$ the set of context defined over the variables $(x_1, \ldots, x_n)$, while $\mathcal{C}(\Sigma)$ is used instead of $\mathcal{C}^1(\Sigma)$.

A **simple** context is a context that use only one variable, that is when it represents a tree having only one node, the root, labelled with the variable used by the context. Given a context $C \in \mathcal{C}(\Sigma)$ we use C^0 to express the related simple context, C^1 as another way to express C, and, with $n > 1$, $C^n = C^{n-1}[C]$, that corresponds to n substitutions of the variable with the context.

Example 2: If we have $\mathcal{X} = \{x\}, \Sigma = \{a(), b\}$ and $C = a(x)$, then $C^3 = a(a(a(x)))$. $\square$

Example 3: Consider now the two terms $t = f(g(x_2, x_3), h, f(a, b, x_1))$ and
$t' = f(g(x_2, x_3), h, f(a, b, f(x_3, h, x_1)))$ with $x_i \in \mathcal{X}$ and with the other symbols as elements of Σ.

For t we have:

- $Height(t) = 3$ and $Size(t) = 6$,
- $\mathcal{P}os(t) = \{\epsilon, 1, 2, 3, 11, 12, 31, 32, 33\}$,
- $\mathcal{FP}os(t) = \{11, 12, 2, 31, 32, 33\}$,
- $\mathcal{VP}os(t) = \{11, 12, 33\}$,
- t is not a ground term because contains variables, but at the same time it is linear because they have a single occurrence,
- t can be seen as a context, in particular as an element of $\mathcal{C}^3(\Sigma)$, because it contains three variables.

For t' we have:

- $Height(t') = 4$ and $Size(t') = 8$,

- $\mathcal{P}os(t') = \{\epsilon, 1, 2, 3, 11, 12, 31, 32, 33, 331, 332, 333\}$,
- $\mathcal{FP}os(t') = \{11, 12, 2, 31, 32, 331, 332, 333\}$,
- $\mathcal{VP}os(t') = \{11, 12, 331, 333\}$,
- t' is neither a ground term because it contains variables, nor a linear one because variable x_1 occurs two times.

Consider $t'|_1 = g(x_1, x_2)$, a subterm of t':

- $Height(t'|_1) = 1$ and $Size(t') = 1$,
- $\mathcal{P}os(t'|_1) = \{\epsilon, 1, 2\}$,
- $\mathcal{FP}os(t'|_1) = \{1, 2\}$,
- $\mathcal{VP}os(t'|_1) = \{1, 2\}$,
- $t'|_1$ is not ground but it is linear.

Terms t and t' are represented as tree respectively in Figure 2 and Figure 3.

Let $\sigma = \{x_1 \leftarrow a, x_2 \leftarrow g(a, a), x_3 \leftarrow g(b, b)\}$ be a substitution. The application of σ to t, denoted as $\sigma(t)$ or $t\sigma$ is shown in Figure 4. □

2.4 Hedge Automata

Even if Tree Automata are used also in works regarding XML (for instance consider [11], [10]), they are more suited for ranked trees. For XML documents, indeed, the number of children of a label is not fixed, and differnt nodes sharing the same label can have a different number of sons. This feature makes XML documents **unranked tree**, for which **Hedge Automata** are best suited. More precisely XML documents are ordered unranked tree because their representation as a file force an order for the children that corresponds to the order in which we can encounter them reading the associated file sequentially from the beginning to the end. This is true even if sons' order is ininfluent in the definition of the schemas related to the documents, situation that happens using *All indicator* and *Interleave pattern* (see respectively [19] and [4]). Schemas using this features describe **commutative trees**, that is trees in which the order of the children is not relevant, for which **Sheaves Automata** [6] are needed.

2.5 Hedges and Hedge Automata

Given an unranked tree $a(t_1, \ldots, t_n)$ with $n \geq 0$, we call **hedge** the sequence $t_1, \ldots, t_n$. For $n = 0$ we have an empty sequence, represented by the symbol ϵ. The set of hedges over Σ is $H(\Sigma)$. [2] defines inductively hedges over Σ with the following definition:

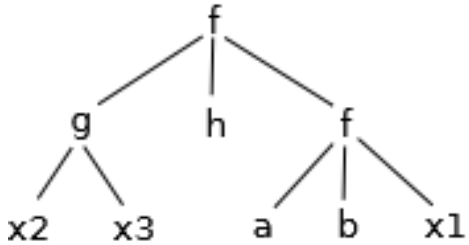

Figure 2. Term t seen as a tree.

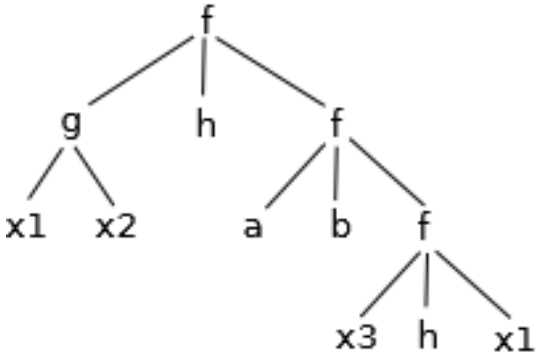

Figure 3. Term t' seen as a tree.

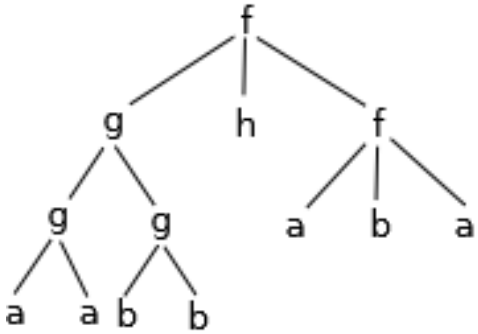

Figure 4. $\sigma(t)$ seen as a tree.

- the empty sequence ϵ is a hedge,
- if g is a hedge and $a \in \Sigma$, then $a(g)$ is a hedge too,
- if g and h are hedges, then gh is a hedge.

Note that $a(g)$ is a tree, but a single tree can be seen as a sequence of trees with only one element.

A definition that takes into account also variables is given in [12], [9] and [2] show explicitly the relationship between Tree Automata and Hedge Automata (also called Forest Automata) and that the latter are a generalization of Tree Automata. [14] presents an alternative definition with some examples.

Example 4: Given the tree t represented in Figure 1 we have in Figure 5 the hedges corresponding to the sequence of subtrees t_1, t_2, t_3, every of them having as root node the sons of $Root(t)$. □

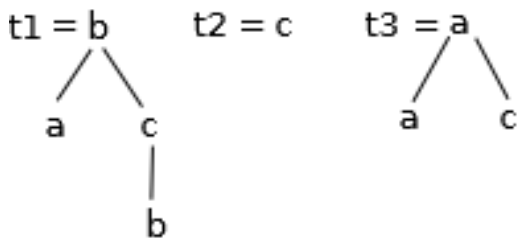

Figure 5. The hedge corresponding to the sequence t_1, t_2, t_3.

Because unranked trees don't have an a priori fixed number of children for every label is evident that tree automata are not well suited, in particular for transiction rules of the form $f(q_1, \ldots, q_n) \to q$ with $f \in \Sigma_n$. For this reason we need to use hedge automata, having transiction rules of the form $a(R) \to q$, with $R \subseteq Q^*$ a regular language over automaton's set of states. Reguler languages are not the only possibility, one can also use more expressive languages, for examples the context-free ones.

2.6 Nondeterministic Finite Hedge Automata

A **Nondeterministic Finite Hedge Automaton** (NFHA) defined over Σ is a tuple $M = (Q, \Sigma, Q_f, \Delta)$ where:

- Σ is a finite and non empty alphabet,
- Q is a finite set of states,
- $Q_f \subseteq Q$ is the set of final states, also called acceptance states,
- Δ is a finite set of transiction rules of the form $a(R) \to q$, where $a \in \Sigma$ and $R \subseteq Q^*$ is a regular language over Q, $q \in Q$.

Regular languages denoted with R that appear in rules belonging to Δ are said **horizontal languages** and can be represented with any formalism suited for regular languages. Here, following [5], we will use Nondeterministic Finite Automata (NFA).

A **computation** of M over a tree $t \in T(\Sigma)$ is a tree $M||t$ having the same domain of t and for which, for every element $p \in \mathcal{P}os(M||t)$ such that $t(p) = a$ and $M||t(p) = q$ must exists a rule $a(R) \to q$ in Δ and, if p has n successors $p1, \ldots, pn$, such that $M||t(p1) = q_1, \ldots, M||t(pn) = q_n$, then $q_1 \cdots q_n \in R$. If $n = 0$, that is considering a leaf node, the empty string ϵ must belong to the language R of the rule that we want to apply to the leaf node.

A tree t is said to be **accepted** if exists at least an accepting computation, that is a computation having the root node labelled with an accepting state, that is having as root's label a state $q \in Q_f$.

We define as **accepted language** for automaton M, denoted with $L(M) \subseteq T(\Sigma)$ the set of all the trees accepted by M. If two distinct automata accept the same language they are **equivalents**.

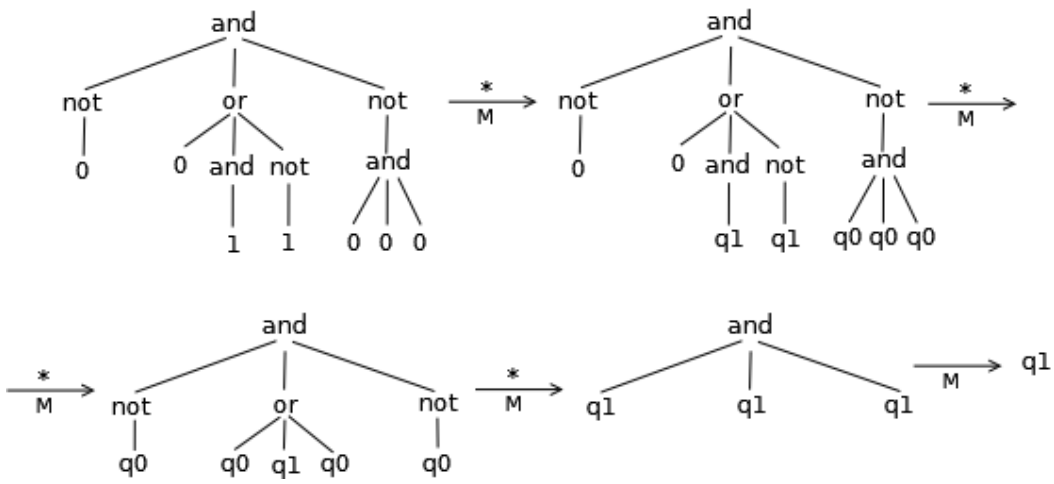

Figure 6. Application of the rewriting function associated with M over tree t.

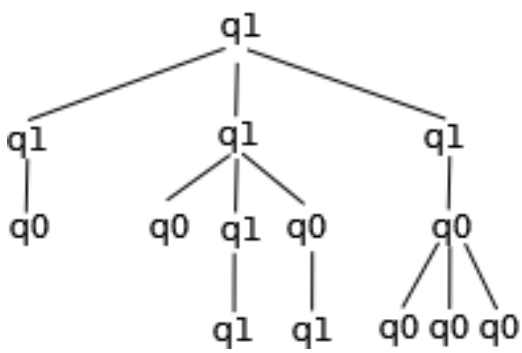

Figure 7. Accepting computation of the automaton M over tree t.

Example 5 (variant of an example of [5])**:** Let us consider the NFHA $M = (Q, \Sigma, Q_f, \Delta)$ with $Q = \{q_0, q_1\}$, $\Sigma = \{0, 1, not, and, or\}$, $Q_f = \{q_1\}$ and $\Delta = \{not(q_0) \to q_1, not(q_1) \to q_0, 1(\epsilon) \to q_1, 0(\epsilon) \to q_0, and(Q^* q_0 Q^*) \to q_0, and(q_1 q_1^*) \to q_1, or(Q^* q_1 Q^*) \to q_1, or(q_0 q_0^*) \to q_0\}$.

Despite the fact that the logical operator and and or are binary, their associativity can be used to achieve an arbitrary ariety; the automaton accepts only well formed boolean formulae that evaluate in true.

In Figure 6 we can see how tree $t = and(not(0), or(0, and(1), not(1)),$ $not(and(0,0,0)))$ which represents a well formed formula evaluated as true, is reduced by the rewriting function into the single state q_1. In Figure 7 we have the accepting computation $M||t$ (this is because $M||t(\epsilon) = q_1 \in Q_f$).

The automaton in Figure 6 operates in a bottom-up fashion, that is from the leaves to the root node. Let analyze the last step that evaluate tree $and(q_1, q_1, q_1)$ in q_1 to undertand how tha automaton works and how it applies its rules: label and has two associated rules, $r_1 = and(Q^* q_0 Q^*) \to q_0$ and $r_2 = and(q_1 q_1^*) \to q_1$; in this case we have a deterministic choice because the sequence $q_1 q_1 q_1$, associated to the sons of the considered node, belongs to the horizontal language $q_1 q_1^*$ and the rule in which appears this language, named r_2 in our example, is the only applicable, that's why we the automaton applies this rewrite $and(q_1, q_1, q_1) \underset{M}{\longrightarrow} q_1$. $\square$

A language $L' \subseteq T(\Sigma)$ having unranked trees as its elements is called **hedge recognizable** if exists an NFHA M such that $L' = L(M)$.

An NFHA $M = (Q, \Sigma, Q_f, \Delta)$ is called **complete** if for every unranked tree $t \in T(\Sigma)$ exists at lesta a state $q \in Q$ for which at least one computation of M over t is evaluated in $M||t(\epsilon) = q$. Or, in other words, for every $t \in T(\Sigma)$ we need to have at least a state $q \in Q$ such that $t \overset{*}{\underset{M}{\longrightarrow}} q$. Because string regular languages are closed for union and difference it is always possible to transform an NFHA adding a dummy state π to Q and adding the following rule for every label $p \in \Sigma$: $p(Q^* \setminus R_p) \to \pi$, where with R_p we denote the union of all the horizontal languages of the rules related to label p. Note that, after the extension of Q with π, the regular language of the new rule contains also the language $L = (Q^* \pi Q^*)$, that is necessary to ""propagate" the dummy state.

An NFHA $M = (Q, \Sigma, Q_f, \Delta)$ is called **reduced** if all of its state is reachable, that is, for every state $q \in Q$ exists at least a tree $t \in T(\Sigma)$ for which $t \overset{*}{\underset{M}{\longrightarrow}} q$.

An NFHA $M = (Q, \Sigma, Q_f, \Delta)$ is called **normalized** if for every $a \in \Sigma, q \in Q$ exists at most one rule $a(R) \to q \in \Delta$. Because, as previously said, string regular languages are closed under union is always possible to define, starting from an NFHA not normalized, the normalized equivalent one: for every couple of rules $a(R_1) \to q$ and $a(R_2) \to q$ belonging to Δ we need to substitute them with the equivalent rule $a(R_1 \cup R_2) \to q$.

Nondeterministic Finite Hedge Automata with ϵ-transition An NFHA is said **NFHA with ϵ-transition** if at least one of his rules is of the form $q \to q'$. This kind of rules don't extend the expressivity of the NFHA, it is in fact always possible, starting from an automaton $M = (Q, \Sigma, Q_f, \Delta)$ with ϵ-transition, to define an NFHA $M' = (Q, \Sigma, Q_f, \Delta')$ equivalent to the other one in this way: let $\Delta_\epsilon \subseteq \Delta$ be the set of ϵ-transitions and let $\epsilon^*(q)$ the set of states q' such that exists a rule $q' \to q$ in Δ_ϵ; $\Delta' = \{a(R) \to q' \mid a(R) \to q \in \Delta \wedge q' \in \epsilon^*(q)\}$.

2.7 Deterministic Finite Hedge Automata

A **Deterministic Finite Hedge Automaton** (DFHA) over Σ is a tuple $M = (Q, \Sigma, Q_f, \delta)$ where:

- Σ is a finite and nonempty unranked alphabet unranked,
- Q is a finite set of states,
- $Q_f \subseteq Q$ is the set of final states,
- δ is a finite set of transiction rules such that no two rules $a(R_1) \to q_1$ and $a(R_2) \to q_2$ with $R_1 \cap R_2 \neq \emptyset$ and $q_1 \neq q_2$ exist.

For every NFHA exists an equivalent DFHA, the algorithm and its proof are based on the definition of the DFHA using set of states of the NFHA instead of simple states. For sake of brevity we do not present the proof of the algorithm (provided in [5], pag. 206) but only the Algorithm 1 in pseudocode form.

Algorithm 1 DET-NFHA - Determinization of NFHA

Input: NFHA $M_1 = (\Sigma, Q_1, Q_{f1}, \Delta)$
Output: DFHA $M_2 = (\Sigma, Q_2, Q_{f2}, \delta)$

begin
  $Q_2 \leftarrow \mathcal{P}(Q_1)$, $\delta \leftarrow \emptyset$, $Q_{f2} \leftarrow \{s \in Q_2 \mid s \cap Q_{f1} \neq \emptyset\}$
  repeat
    if $s = \{q \in Q \mid \exists q_1 \in s_1, \ldots, q_n \in s_n \wedge a(R') \to q \in \Delta \wedge q_1 \ldots q_n \in R'\}$,
    $s_1 \ldots s_n \in R$ **then**
      $\delta \leftarrow \delta \cup \{a(R) \to s\}$
    end if
  until no other rule can be added to δ
  return $M_2 = (\Sigma, Q_2, Q_{f2}, \delta)$
end

2.8 Union, intersection and complement

As previously said, the class of languages recognizable by hedge automata is closed under this three operations. Here we will show how to compute hedge automata able to recognize the result of one this operations starting from the automata recognizing the starting languages.

2.8.1 Union

Given two complete and normalized NFHA $M_1 = (\Sigma, Q_1, Q_{f1}, \Delta_1)$ and $M_2 = (\Sigma, Q_2, Q_{f2}, \Delta_2)$ for which $L(M_1) = L_1$ and $L(M_2) = L_2$, the automaton $M = (\Sigma, Q, Q_f, \Delta)$ such that $L(M) = L_1 \cup L_2$ is defined in this way:

- $Q = Q_1 \times Q_2$,
- $Q_f = (Q_{f1} \times Q_2) \cup (Q_1 \times Q_{f2})$,

- $\Delta = \Delta_1 \times \Delta_2$, with $\Delta_1 \times \Delta_2 = \{a(R_1, R_2) \to (q_1, q_2) \mid a(R_1) \to q_1 \in \Delta_1 \wedge a(R_2) \to q_2 \in \Delta_2\}$, with $a \in \Sigma$, $R_1 \subseteq Q_1^*$, $R_2 \subseteq Q_2^*$, $q_1 \in Q_1$ and $q_2 \in Q_2$.

In order to apply rule $a(R_1, R_2) \to (q_1, q_2)$ to a node labelled a having as its sons $(s_1, s'_1), \ldots, (s_n, s'_n)$ for some $n \geq 0$, with $s_1, \ldots, s_n \in Q_1$ and $s'_1, \ldots, s'_n \in Q_2$ this conditions must hold, $s_1 \ldots s_n \in R_1$ and $s'_1 \ldots s'_n \in R_2$.

We require normalized automata for convenience even if it is not necessary, while completeness it is. If the starting automata are deterministic, the derived automaton will be deterministc too.

2.8.2 Complementation

Given a complete DFHA $M = (\Sigma, Q, Q_f, \delta)$ which accepts language L, the DFHA $M' = (\Sigma, Q, Q'_f, \delta)$ accepting the complementation of L is defined as follow:

- $Q'_f = Q \setminus Q_f$,
- as M for the rest.

2.8.3 Intersection

Given two normalized NFHA $M_1 = (\Sigma, Q_1, Q_{f1}, \Delta_1)$ and $M_2 = (\Sigma, Q_2, Q_{f2}, \Delta_2)$ for which hold $L(M_1) = L_1$ and $L(M_2) = L_2$, the automaton $M = (\Sigma, Q, Q_f, \Delta)$ such that $L(M) = L_1 \cap L_2$ is defined in this way:

- $Q = Q_1 \times Q_2$,
- $Q_f = Q_1 \times Q_2$,
- $\Delta = \Delta_1 \times \Delta_2$.

Here as well as for union the starting automata can be unnormalized, but here, on the contrary, completeness is optional too. Again, M will be deterministic if and only if the two automata are deterministc.

2.9 Decision problems for Hedge Automata

We will now introduce some decision problems for Hedge Automata, deterministic and nondeterministc, that will prove useful later. We will give also the relative algorithms extracted from [5], Section 8.5.

2.9.1 Uniform membership

Uniform membership is the test that, taken as input an hedge automaton $M = (\Sigma, Q, Q_f, \Delta)$ and an unranked tree $t \in T(\Sigma)$, returns true if the automaton accepts t, false otherwise. Algorithm 2 builds a map ρ: $\mathcal{P}os(t) \to \mathcal{P}(Q)$ which associates every position in the tree with the set of states (saved in the auxiliary variable S) with which it can be labelled during the execution of the automaton over the tree, using the rules in Δ. At the beginning, every position $p \in \mathcal{P}os(t)$, $\rho(p)$ is set as undefined (using symbol $\perp$ in the algorithm). After that, proceding in a bottom-up fashion, ρ is calculated for the leaves, then for the nodes for which ρ has been evaluated for every of its sons, then finally for the root node. If at least one of the associated states is a final state, then the automaton accepts the tree, otherwise it doesn't accept it.

Algorithm 2 Uniform membership test

Input: NFHA $M = (\Sigma, Q, Q_f, \Delta), t \in T(\Sigma)$
Output: true or false

```
begin
  for all p ∈ Pos(t) do
    ρ(p) ← ⊥
  end for
  while ∃p ∈ Pos(t) with ρ(p) = ⊥ and ρ(pi) ≠ ⊥ for every i ∈ [1, k] with k = max{i | pi ∈ Pos(t) ∨ i = 0}
  do
    S ← ∅
    for all a(R) → q ∈ Δ with a = t(p) do
      if ∃q_1 ∈ ρ(p1), ..., q_k ∈ ρ(pk) with q_1 ... q_k ∈ R then
        S ← S ∪ {q}
      end if
    end for
    ρ(p) ← S
  end while
  if ρ(ϵ) ∩ Q_f ≠ ∅ then
    return true
  else
    return false
  end if
end
```

2.9.2 Emptiness

Emptiness test is the decision problem which evaluates if the language accepted by a NFHA $M = (\Sigma, Q, Q_f, \Delta)$ taken as input is empty or not. The Algorithm 3 takes as its input a NFHA M and returns true if $L(M) = \emptyset$, false otherwise. We start adding to S (an auxiliary variable all the reachable states encountered) the state that can be associated to the leaves of a tree by the input automaton: S is initialized as $\emptyset$ because $\emptyset^* = \{\epsilon\}$, letting in this way to add, during the first iteration, the states associated to the leaves, which horizontal language is, by definition, the empty string. After that, it considers the states in the right part of the remainings rules not yet present in S and reachable from an arbitrary string composed by states of S, test that is made checking if the intersection between R and S^* is not empty, that is if exist some strings belonging to the regular language R build up with an arbitrary sequence of states in S; the elements in $Q \setminus S$ are not taken into account because they are not reachable in the actual phase. In the end set S will contain all the reachable states (used thus in at least a computation), if none of them is a final state, then the accepted language is empty and the algorithm returns true, otherwise exists at least one element in $L(M)$ and the algorithm returns false.

Algorithm 3 Emptiness test

Input: NFHA $M = (\Sigma, Q, Q_f, \Delta)$
Output: true or false

begin
  $S \leftarrow \emptyset$
  while $\exists a(R) \rightarrow q \in \Delta$ with $q \notin S$ and $R \cap S^* \neq \emptyset$ **do**
    $S \leftarrow S \cup q$
  end while
  if $S \cap Q_f = \emptyset$ **then**
    return true
  else
    return false
  end if
end

2.9.3 Inclusion test

Inclusion test is the decision problem that, given as input two NFHAs M_1 and M_2, returns true if $L(M_1) \subseteq L(M_2)$, false otherwise. Exploiting the following equivalence the inclusion problem is decidable through the evaluation of the intersection between the complementation of the automaton M_2 and M_1 and the *emptiness test* on its result:

$$L(M_1) \subseteq L(M_2) \Leftrightarrow L(M_1) \cap (T(\Sigma) \setminus L(M_2)) = \emptyset.$$

In Figure 8 we can see the graphical representation of the sets $T(\Sigma), L(M_1)$ and $L(M_2)$ in the case in which the inclusion test is positive, in Figure 9 we have the analogous representation for the negative case.

3 Parallel Rewriting Semantics

We'll now introduce some basic notations and definitions useful for the formalization of the parallel rewriting semantic.

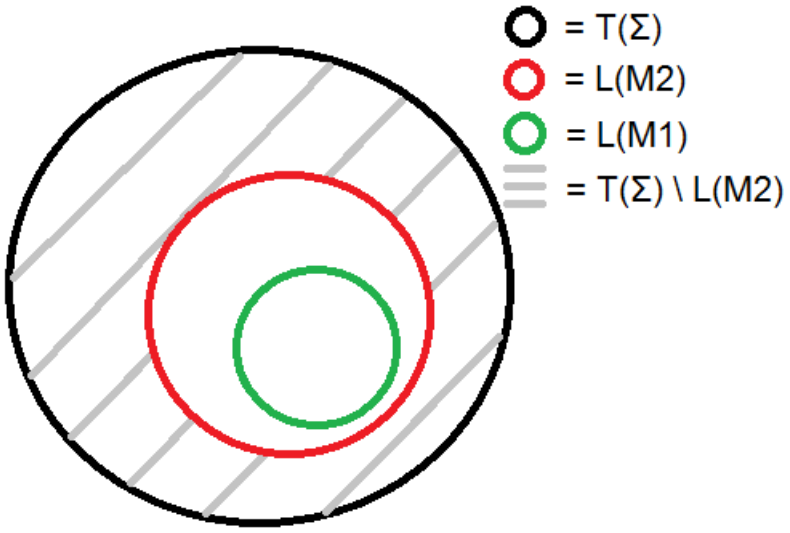

Figure 8. Example for which $L(M_1) \subseteq L(M_2)$ holds.

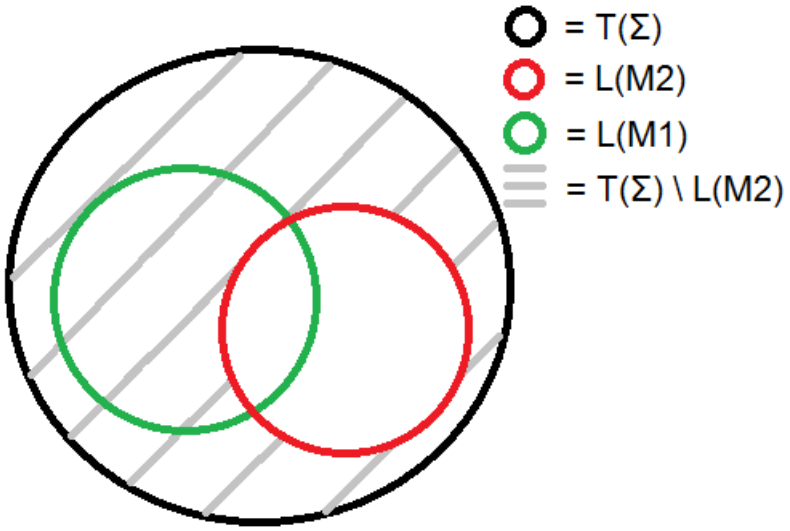

Figure 9. Example for which $L(M_1) \nsubseteq L(M_2)$ holds.

UFO	
$a(x) \to b(x)$	REN
$a(x) \to a(px)$	INS_{first}
$a(x) \to a(xp)$	INS_{last}
$a(xy) \to a(xpy)$	INS_{into}
$a(x) \to pa(x)$	INS_{before}
$a(x) \to a(x)p$	INS_{after}
$a(x) \to p$	RPL
$a(x) \to ()$	DEL

Table 1. XQuery Update Facility 1.0 primitives called UFO.

Lexicographic order over $\mathbb{N}^*$: $<_{lex}$ is the lexicographic order for strings in $\mathbb{N}^*$ defined in the following way:

$$\forall x, y \in \mathbb{N}^* \,.\, x <_{lex} y \Leftrightarrow |x| < |y| \vee \exists i \in [1, |y|] \,.\, \forall j < i \,.\, x[j] = y[j] \wedge x[i] <_{lex} y[i]$$

Over the elements of $\mathbb{N}$, string characters, $<_{lex}$ is simply the canonical order relation $<$ over natural numbers. The lexicographic order will help us to identify how to order the position of the node to modify, position that are expressed as strings over $\mathbb{N}^*$.

Target nodes: Given an unranked tree $t \in T(\Sigma)$ and a rewrite rule $r = L \to R \in UFO$, we can define the ordered list of the nodes that match the left hand side of the rule as follow:

$$\begin{aligned} Target(t, r) = (p_1 \dots p_n) \text{ s.t. } n \geq 0 \wedge \forall i \in [1, n-1] \,.\, p_i <_{lex} p_{i+1} \wedge \\ \wedge\, t|_{p_i}(\epsilon) = L(\epsilon) \wedge (\forall p \in \mathcal{P}os(t) \,.\, t|_p(\epsilon) = L(\epsilon) \wedge \exists i \in [1, n] \,.\, p = p_i) \end{aligned}$$

Subtrees of the root: The ordered sequence of the subtrees rooted in each son of the root node of an unranked tree $t \in T(\Sigma)$ is defined as follow:

$$subt(t) = (t_1, \dots, t_n) \text{ s.t. } t_i = t|_i, \text{ with } i \in [1, n] \subseteq \mathbb{N}^* \wedge i \in \mathcal{P}os(t) \text{ s.t. } \nexists j \in \mathbb{N}^* \,.\, j \neq \epsilon \wedge ji \in \mathcal{P}os(t)$$

For our convenience in the rest of the document we will split the result of this list into two sublists with a "cutoff point" in a certain position $i \in \mathbb{N}^*$, so we also define:

$$\begin{aligned} subt(t, i) = ((t_1, \dots, t_i), (t_{i+1}, \dots, t_n)) \text{ with } i \in [0, n+1], \text{ s.t. } t_j = t|_j \\ \text{with } j \in [1, n] \subseteq \mathbb{N}^* \wedge j \in \mathcal{P}os(t) \text{ s.t. } \nexists k \in \mathbb{N}^* \,.\, k \neq \epsilon \wedge kj \in \mathcal{P}os(t) \end{aligned}$$

Formal definition of the parallel rewriting semantic: In the following will follow the definition of this new semantic over UFO operations (Table 1) with some example of its application. The parallel rewriting semantic described more formally here corresponds, intuitively, to the single application of the rewriting for the positions in $Target(t, r)$, where t is the starting tree and r is the used rule, in reverse order, that is from p_n to p_1. In order to compress the formula we will fix $N = |Target(t, r)|$.

$\Rightarrow_r$: $\forall t, t' \in T(\Sigma), r = L \to R \in UFO \setminus \{INS_{into}\} \,.\, t \Rightarrow_r t' \Leftrightarrow \exists C_1, \dots, C_N \in T(\Sigma, \mathcal{X} = \{y\}), \sigma_1, \dots, \sigma_N \colon \mathcal{X}' \to H(\Sigma)$.

1. $(\forall i \in [1, N-1] \,.\, \sigma_i = \{x \leftarrow subt(C_{i+1}[R\sigma_{i+1}]|_{p_i})\}) \wedge$

2. $\sigma_N = \{x \leftarrow subt(t|_{p_N})\} \wedge$

3. $(\forall i \in [1, N-1] \,.\, (\forall z \in \mathcal{P}os(C_{i+1}[R\sigma_{i+1}]) \,.\, (\nexists y \in \mathbb{N}^* \,.\, z = p_i y)) \wedge C_i(z) = C_{i+1}[R\sigma_{i+1}](z) \wedge C_i(p_i) = y) \wedge$

4. $(\forall z \in \mathcal{P}os(t) \,.\, (\nexists y \in \mathbb{N}^* \,.\, z = p_N y) \wedge C_N(z) = t(z) \wedge C_N(p_N) = y) \wedge$

5. $t = C_N[L\sigma_N] \wedge t' = C_1[R\sigma_1]$

Let's now analyze the various numbered components of the formula in order to understand their aim:

1. this part let us build the i-th substitution which assigns to the variable x, located in both the left and right side of the considered rewrite rule, the subtrees rooted in the sons of the node in position p_i. $C_{i+1}[R\sigma_{i+1}]|_{p_i}$ is necessary to preserve any modification in the subtrees of the node with position p_i.

2. this part is used for the substitution with index N that requires directly the starting tree t.

3. this part defines the i-th context, which identifies the position p_i where the rule will be applied. The context has also the function to preserve the rest of the tree not affected by the rewriting at position p_i. The i-th context derives from the context i+1-th where the variable has been substituted by the left hand side of the rule r, on which has been previously applied the i+1-th substitution ($C_{i+1}[R\sigma_{i+1}]$) for every position outside the subtree rooted in position p_i ($\nexists y \in \mathbb{N}^* \,.\, z = p_i y$). At the position p_i of the i-th context we will have the variable proper to the context ($C_i(p_i) = y$).

4. this part defines the same modification of the previous point for the context with index N, which will corresponds to the tree t except for the subtree rooted in the node with position p_N where it presents the variable y.

5. this last part links the first and the last contexts with the involved trees.

Now we will define the corresponding formula for INS_{into} operation.

$\Rightarrow_r$**:** $\forall t, t' \in T(\Sigma), r = L \rightarrow R \in \{INS_{into}\} \,.\, t \Rightarrow_r t' \Leftrightarrow \exists C_1, \ldots, C_N \in T(\Sigma, \mathcal{X} = \{y\}), \sigma_1, \ldots, \sigma_N \colon \mathcal{X}' \rightarrow H(\Sigma)$.

1. $(\forall i \in [1, N-1] \,.\, \exists j \in \mathbb{N}^* \,.\, \sigma_i = \{x_1, x_2 \leftarrow subt(C_{i+1}[R\sigma_{i+1}]|_{p_i}, j)\}) \wedge$

2. $\exists j \in \mathbb{N}^* \,.\, \sigma_N = \{x_1, x_2 \leftarrow subt(t|_{p_N}, j)\} \wedge$

3. $(\forall i \in [1, N-1] \,.\, (\forall z \in \mathcal{P}os(C_{i+1}[R\sigma_{i+1}]) \,.\, (\nexists y \in \mathbb{N}^* \,.\, z = p_i y)) \wedge C_i(z) = C_{i+1}[R\sigma_{i+1}](z) \wedge C_i(p_i) = y) \wedge$

4. $(\forall z \in \mathcal{P}os(t) \,.\, (\nexists y \in \mathbb{N}^* \,.\, z = p_N y) \wedge C_N(z) = t(z) \wedge C_N(p_N) = y) \wedge$

5. $t = C_N[L\sigma_N] \wedge t' = C_1[R\sigma_1]$

The considerations are similar to the ones for the other operations in $UFO \setminus \{INS_{into}\}$. However here the "cutoff point" is choosen in a non-deterministc fashion, as requested by *XQuery Update Facility*, [17].

$\Rightarrow_R$: $\forall t, t' \in T(\Sigma)\ .\ t \Rightarrow_R t' \Leftrightarrow (R = (r_1, \ldots, r_n)$ s.t. $\forall i \in [1, n]\ .\ r_i \in UFO) \wedge (\exists t_1, \ldots, t_{n+1} \in T(\Sigma)\ .\ t_1 = t \wedge t_{n+1} = t' \wedge \forall i \in [1, n]\ .\ t_i \Rightarrow_{r_i} t_{i+1})$

The transitive-reflexive closure of $\Rightarrow_R$ is $\Rightarrow_R^*$.

Example 6: Now we will show an example that involves INS_{after} rule, with $t = b(c, d(c(a), a))$, $r = c(x) \rightarrow c(x)p$, INS_{after}, $p = a(b)$, $\mathcal{P}os(t) = \{\epsilon, 1, 2, 21, 22, 211\}$, $Target(t, r) = (p_1 = 1, p_2 = 21)$:

- $C_2 = b(c, d(y, a)), \sigma_2 = \{x \leftarrow (a)\}$,
- $C_2[R\sigma_2] = b(c, d(c(a), a(b), a))$,
- $C_1 = b(y, d(c(a), a(b), a)), \sigma_1 = \{x \leftarrow \epsilon\}$,
- $C_1[R\sigma_1] = b(c, a(b), d(c(a), a(b), a)) = t'$. $\square$

Example 7: Now an example that envolves RPL rule, with t, p, $\mathcal{P}os(t)$ and $Target(t, r)$ as in the previous example, while $r = c(x) \rightarrow p(RPL)$:

- $C_2 = b(c, d(y, a)), \sigma_2 = \{x \leftarrow (a)\}$,
- $C_2[R\sigma_2] = b(c, d(a(b), a))$,
- $C_1 = b(y, d(a(b), a)), \sigma_1 = \{x \leftarrow (\epsilon)\}$,
- $C_1[R\sigma_1] = b(a(b), d(a(b), a)) = t'$. $\square$

Example 8: Now an example that involves DEL rule, with t, p, $\mathcal{P}os(t)$ and $Target(t, r)$ as in the first example, while $r = c(x) \rightarrow ()(DEL)$:

- $C_2 = b(c, d(y, a)), \sigma_2 = \{x \leftarrow (a)\}$,
- $C_2[R\sigma_1] = b(c, d(a))$,
- $C_1 = b(y, d(a)), \sigma_1 = \{x \leftarrow (\epsilon)\}$,
- $C_1[R\sigma_1] = C_1[()] = b(d(a)) = t'$. $\square$

Example 9: Now an example that involves INS_{first} rule, with $t = a(b(c, d(a)), d(a(c), c))$, $r = a(x) \rightarrow a(px)$, INS_{first}, $p = c(d, b)$, $\mathcal{P}os(t) = \{\epsilon, 1, 2, 11, 12, 21, 22, 121, 211\}$, $Target(t, r) = (p_1 = \epsilon, p_2 = 21, p_3 = 121)$:

- $C_3 = a(b(c, d(y)), d(a(c), c)), \sigma_3 = \{x \leftarrow (\epsilon)\}$,
- $C_3[R\sigma_3] = a(b(c, d(a(c(b, d)))), d(a(c), c))$,
- $C_2 = a(b(c, d(a(c(d, b)))), d(y, c)), \sigma_2 = \{x \leftarrow (c)\}$,
- $C_2[R\sigma_2] = a(b(c, d(a(c(d, b)))), d(a(c(d, b), c), c))$,
- $C_1 = y, \sigma_1 = \{x \leftarrow (t_1, t_2)\}$, with $t_1 = b(c, d(a(c(d, b))))$ and $t_2 = d(a(c(d, b), c), c)$,
- $C_1[R\sigma_1] = a(c(d, b), b(c, d(a(c(d, b)))), d(a(c(d, b), c), c)) = t'$. $\square$

Example 10: Now and example that involves REN rule, with $t = c(c(c(a)))$, $r = c(x) \rightarrow c'(x)(REN)$, $\mathcal{P}os(t) = \{\epsilon, 1, 11, 111\}$, $Target(t, r) = \{p_1 = \epsilon, p_2 = 1, p_3 = 11\}$:

- $C_3 = c(c(y)), \sigma_3 = \{x \leftarrow (a)\}$,
- $C_3[R\sigma_3] = c(c(c'(a)))$,
- $C_2 = c(y), \sigma_2 = \{x \leftarrow c'(a)\}$,
- $C_2[R\sigma_2] = c(c'(c'(a)))$,
- $C_1 = y, \sigma_1 = \{x \leftarrow c'(c'(a))\}$,
- $C_1[R\sigma_1] = c'(c'(c'(a))) = t'$. $\square$

4 Automaton for Post(L)

First of all a **Parameterized Hedge Rewrite System** (SRHP) R/A, with HA $A = (\Sigma, Q, Q_f, \Delta)$, is a set of rewrite rules of the form $L \rightarrow R$, where $L \in H(\Sigma, \mathcal{X})$ and $R \in H(\Sigma \uplus Q, \mathcal{X})$.

Given an **Hedge Automata** (HA) $A = (\Sigma, P, P_f, \Theta)$ and $A_L = (\Sigma_L, Q_L, Q_L^f, \Delta_L)$ such that A and A_L are normalized automata, $P \cap Q_L = \emptyset$, $L = L(A_L)$ with L the regular language over hedges, we define the HA $A' = (\Sigma := \Sigma \cup \Sigma_L, P \cup Q_L, Q_L^f, \Delta')$ such that $L(A') = Post_{R/A}(L)$. For each $a \in \Sigma$, $q \in Q_L$, we denote with $L_{a,q}$ the horizontal language of the unique rule $a(L_{a,q}) \rightarrow q \in \Delta_L$, accepted by the **Nondeterministic Finite Automaton** (NFA) $B_{a,q} = (Q_L, S_{a,q}, i_{a,q}, \{f_{a,q}\}, \Gamma_{a,q})$. As a preliminar operation we need to expand the alphabet of each automaton that recognizes the horizontal languages, so we will have $P \cup Q_L$ instead of Q_L. For each of the following rules we assume $p \in P$, which permits only to insert hedge included in the language $L(A)$.

REN given the rule $a(x) \rightarrow b(x) \in R/A$, with $a, b \in \Sigma$, for each $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$, we define $B_{b,q} := B_{a,q}$, with a proper change of the indexes of the various elements, if holds $L(B_{b,q}) = \emptyset$, otherwise we define $L(B_{b,q}) := L(B_{a,q}) \cup L(B_{b,q})$ with a redefinition of the automaton $B_{b,q}$ as $(Q_L, S_{a,q} \uplus S_{b,q} \uplus i_{ab,q}, i_{ab,q}, \{f_{a,q}\} \uplus \{f_{b,q}\}, \Gamma_{a,q} \uplus \Gamma_{b,q} \uplus \{(i_{ab,q}, \epsilon, i_{a,q}), (i_{ab,q}, \epsilon, i_{b,q})\})$. In the end we drop all the rules of

the form $a(L_{a,q}) \to q$ from Δ_L, with $q \in Q_L$ and we add the corrisponding rule $b(L_{b,q}) \to q$ for each deleted transiction. These changes on one hand allows the automaton to accept the label b where the old automaton accepts label a, on the other hand preserves the "behaviour" of the label b, both in the related horizontal languages and in the horizontal languages where this label can be evaluated.

INS_{first} the rule $a(x) \to a(px) \in R/A$ leads to change the automaton $B_{a,q}$, for each $q \in Q_L$ such that $L_{a,q} \neq \emptyset$. A fresh state $q_{a,q}^{fresh}$ such that $q_{a,q}^{fresh} \notin S_{a,q}$ is created, then it's added to $S_{a,q}$ and used as an initial state, after that, if $\Gamma_{a,q} = \emptyset$ holds, the rule $(q_{a,q}^{fresh}, p, f_{a,q})$ is added to $\Gamma_{a,q}$, otherwise for each rule of the form $(i_{a,q}, y, q_y) \in \Gamma_{a,q}$, with $y \in P \cup Q_L$, $q_y \in S_{a,q}$, we add a rule of the form $(q_{a,q}^{fresh}, p, i_{a,q})$ to $\Gamma_{a,q}$.

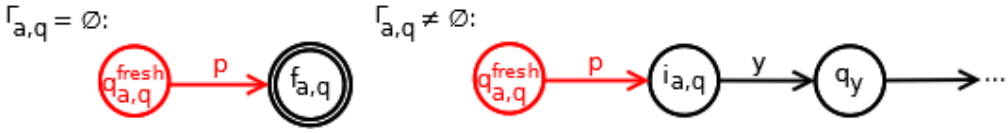

Figure 10. The changes to the horizontal automaton due to rule INS_{first} are colored in red.

INS_{last} the rule $a(x) \to a(xp) \in R/A$ leads to change the automaton $B_{a,q}$, for each $q \in Q_L$ such that $L_{a,q} \neq \emptyset$. A fresh state $q_{a,q}^{fresh}$ such that $q_{a,q}^{fresh} \notin S_{a,q}$ is created, then it's added to $S_{a,q}$ and used as initial state, after that, if $\Gamma_{a,q} = \emptyset$ holds, the rule $(i_{a,q}, p, q_{a,q}^{fresh})$ is added to $\Gamma_{a,q}$, otherwise for each rule of the form $(q_y, y, f_{a,q}) \in \Gamma_{a,q}$, with $y \in P \cup Q_L$, $q_y \in S_{a,q}$, we add a rule of the form $(f_{a,q}, p, q_{a,q}^{fresh})$ to $\Gamma_{a,q}$.

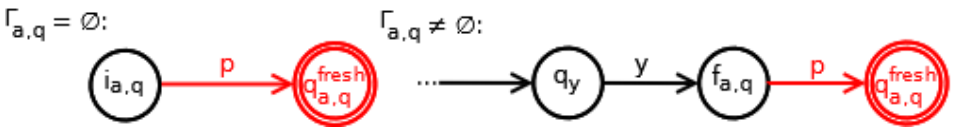

Figure 11. The changes to the horizontal automaton due to rule INS_{last} are colored in red.

INS_{into} the rule $a(xy) \to a(xpy) \in R/A$ leads to change the automaton $B_{a,q}$, for each $q \in Q_L$ such that $L_{a,q} \neq \emptyset$. A fresh state $q_{a,q}^{fresh}$ such that $q_{a,q}^{fresh} \notin S_{a,q}$ is created and added to $S_{a,q}$. At this point, for each state $s \in S_{a,q}$ reachable from $i_{a,q}$ through the transiction in $\Gamma_{a,q}$, we change the rules of the form (s, j, s') into another of the form $(s, j, q_{a,q}^{fresh})$, with $j \in P \cup Q_L$ and $s' \in S_{a,q}$, and transiction of the form $(q_{a,q}^{fresh}, p, s')$ are added to $\Gamma_{a,q}$.

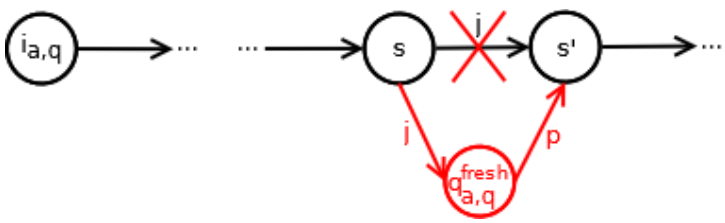

Figure 12. The changes to the horizontal automaton due to rule INS_{into} are colored in red.

INS_{before} for the rule $a(x) \to pa(x) \in R/A$ we need to modify every horizontal language in which can occur a state $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$. For each $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$ we create a fresh state $q_{a,q}^{fresh}$ such that $q_{a,q}^{fresh} \notin S_{b,z}$, for each $b \in \Sigma$ and $z \in Q_L$, and we add this new state to $S_{b,z}$ if exists at least one

transiction of the form $(s, q, s') \in \Gamma_{b,z}$, with $s, s' \in S_{b,z}$; this transictions are changed to $(s, p, q_{a,q}^{fresh})$, after that the correspondig rules $(q_{a,q}^{fresh}, q, s')$ are added to $\Gamma_{b,z}$.

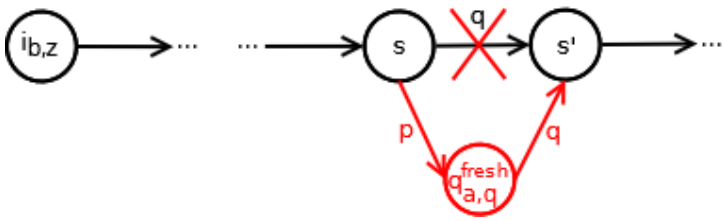

Figure 13. The changes to the horizontal automaton due to rule INS_{before} are colored in red in the picture.

INS_{after} for the rule $a(x) \to a(x)p \in R/A$ we need to modify every horizontal language in which can occur a state $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$. For every $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$ we create a fresh state $q_{a,q}^{fresh}$ such that $q_{a,q}^{fresh} \notin S_{b,z}$, for each $b \in \Sigma$ and $z \in Q_L$, and this new state is added to $S_{b,z}$ if exists at least one transiction of the form $(s, q, s') \in \Gamma_{b,z}$, with $s, s' \in S_{b,z}$; these transictions are changed into $(s, q, q_{a,q}^{fresh})$, after that we add the corresponding rules of the form $(q_{a,q}^{fresh}, p, s')$ to $\Gamma_{b,z}$.

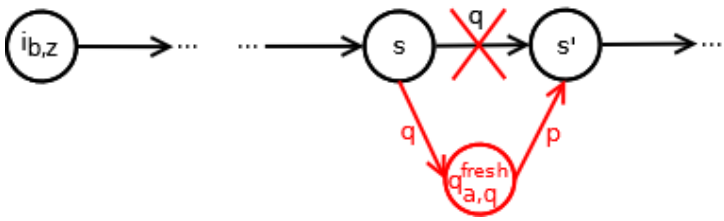

Figure 14. The changes to the horizontal automaton due to rule INS_{after} are colored in red.

RPL for the rule $a(x) \to p \in R/A$ we need to modify every horizontal language in which can occur a state $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$. Every transiction of the form (s, q, s') included in $\Gamma_{b,z}$, with $b \in \Sigma$ and $z \in Q_L$, will be changed into (s, p, s').

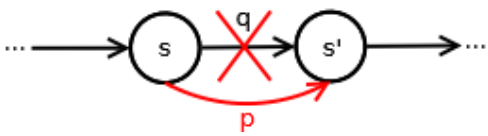

Figure 15. The changes to the horizontal automaton due to rule RPL are colored in red.

DEL for the rule $a(x) \to () \in R/A$ we need to modidy every horizontal language in which can occur a state $q \in Q_L$ such that $L(B_{a,q}) \neq \emptyset$. Every transiction of the form (s, q, s') in $\Gamma_{b,z}$, with $b \in \Sigma$ and $z \in Q_L$, will be changed into (s, ϵ, s').

For the operations INS_{before}, INS_{after}, RPL and DEL some states $q \in Q_L$ involved in a change could be shared among more than one character of Σ, or else that exist two rules $a(L_a) \to q, b(L_b) \to q \in \Delta_L$, such that $a \neq b$. To avoid an unwanted change for the symbol b we create a fresh state $q_a^{fresh} \notin P \cup Q_L$ and, for each rule in which appear simultaneously the label a and the state q, we create a copy of this state and replace q with q_a^{fresh};

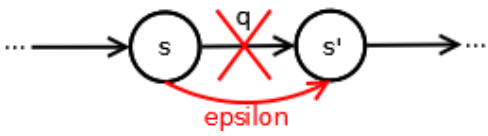

Figure 16. The changes to the horizontal automaton due to rule DEL are colored in red.

as last step we add q_a^{fresh} to Q_L and to every other alphabet belonging to the horizontal languages, while updating also their transictions. These changes must be applied before any other modification.

In the end we calculate Δ' in the following way:

$$\Delta' := \Theta \cup \{a(B_{a,q}) \to q \mid a \in \Sigma, q \in Q_L, L(B_{a,q}) \neq \emptyset)\}.$$

The transictions of Θ ensures that A' will be able to evaluate any subtree belonging to L, the other transictions are used by A' for the evaluation of the elements of $L(A)$ with the changes due to the update operations. The test $L(B_{a,q}) \neq \emptyset$ excludes unnecessary transictions.

To preserve the tree structure of an XML document we need to avoid the application of the operations INS_{before}, INS_{after} and DEL, of the form $a(x) \to pa(x)$, $a(x) \to a(x)p$ and $a(x) \to ()$, respectively, to any tree $t \in T(\Sigma)$ such that $t(\epsilon) = a$.

Example 11: Suppose we have two **Nondeterministic Finite Hedge Automata** (NFHA) $A_L = (\Sigma, Q_L, Q_L^f, \Delta_L)$ and $A = (\Sigma, P, P_f, \Theta)$ defined as follow:

- $\Sigma = \{a, b, c\}$,
- $Q_L = \{q_{a1}, q_{a2}, q_b, q_c\}$ and $P = \{g_a, g_b, g_c\}$,
- $Q_L^f = \{q_{a1}, q_{a2}\}$ and $P_f = \{g_a\}$,
- $\Delta_L = \{a({q_b}^*) \to q_{a2}, a({q_b}^* q_c) \to q_{a1}, b(\epsilon) \to q_b, c(\epsilon) \to q_c\}$,
- $\Theta = \{a({g_b}^+) \to g_a, b({g_b}^+ | g_c) \to g_b, c(\epsilon) \to g_c\}$.

The NFA used for the horizontal languages of the NFHA A_L are:

- $B_{a,q_{a1}} = (Q_L, S_{a,q_{a1}} = \{p_b, p_c\}, p_b, \{p_c\}, \Gamma_{a,q_{a1}} = \{(p_b, q_b, p_b), (p_b, q_c, p_c)\})$,
- $B_{a,q_{a2}} = (Q_L, S_{a,q_{a2}} = \{m_b\}, m_b, \{m_b\}, \Gamma_{a,q_{a2}} = \{(m_b, q_b, m_b)\})$,
- $B_{b,q_b} = (Q_L, S_{b,q_b} = \{n\}, n, \{n\}, \Gamma_{b,q_b} = \{\})$,
- $B_{c,q_c} = (Q_L, S_{c,q_c} = \{o\}, o, \{o\}, \Gamma_{c,q_c} = \{\})$.

It's clear that $L = L(A_L) = \{a(bc), a(bbc), \ldots, a(b \ldots bc), \ldots, a, a(b), a(bb), \ldots, a(b \ldots b), \ldots\}$ and that $L(A)$ is the set of the unranked tree labelled with a at the root node, with b as the label of the internal nodes and label c in the leaves.

Now we apply the sequence $s = \{REN : b(x) \to a(x), INS_{first} : c(x) \to c(g_a x), INS_{before} : c(x) \to g_a c(x)\}$ composed of update operations of R/A and we calculate the NFHA $A' = (\Sigma, P \cup Q_L, Q_L^f, \Delta')$ such that $L(A') = Post_{R/A}(L)$.

$REN : b(x) \to a(x)$**;** the NFA $B_{a,q_b} = (P \cup Q_L, S_{b,q_b} = \{n\}, n, \{n\}, \Gamma_{b,q_b} = \{\})$ is defined.

$INS_{first} : c(x) \to c(g_a x)$**;** the NFA B_{c,q_c} is changed into $(P \cup Q_L, \{q_{c,q_c}^{fresh}, o\}, q_{c,q_c}^{fresh}, \{o\}, \{(q_{c,q_c}^{fresh}, g_a, o)\})$.

$INS_{before} : c(x) \to g_a c(x)$**;** the NFA $B_{a,q_{a1}}$ is changed into $(P \cup Q_L, S_{a,q_{a1}} = \{p_b, p_c, q_{c,q_c}^{fresh}\}, p_b, \{p_c\},$
$\Gamma_{a,q_{a1}} = \{(p_b, g_a, q_{c,q_c}^{fresh}), (q_{c,q_c}^{fresh}, q_c, p_c), (p_b, q_b, p_b)\})$.

In Figure 17 we can see an example of application of the update operations REN, INS_{first} and INS_{before} that transform tree $t \in L$ into $t' \in L(A')$. In Figure 18 we can see an accepting computation of the NFHA A' related to the tree t'. □

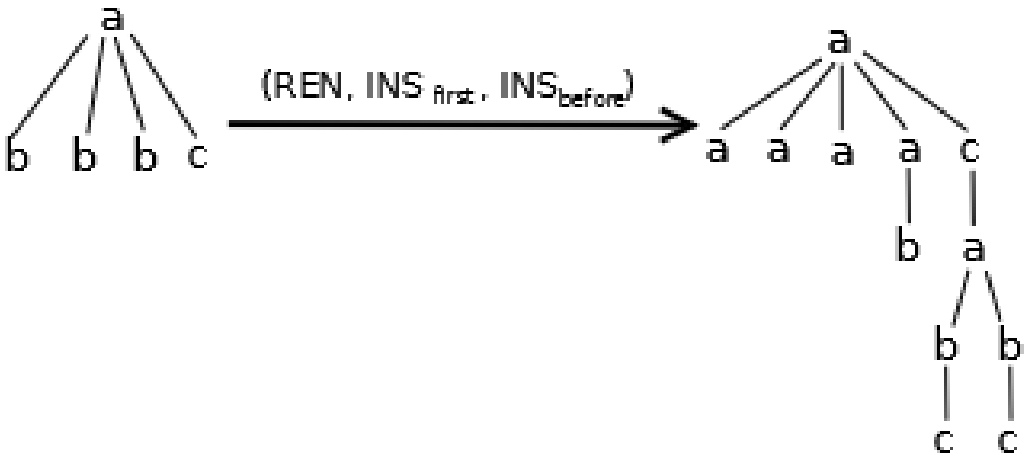

Figure 17. The picture show the change from $t \in L$ to $t' \in L(A')$ through the application of REN, INS_{first} and INS_{before}.

4.0.4 Proof of Post(L) = L(A')

Proof of $Post_{R/A}(L) \subseteq L(A')$: In order to have $Post_{R/A}(L) \subseteq L(A')$, we need to prove that $\forall op \in R/A, t = f(t_1, \ldots, t_k) \in L$, if $t \xrightarrow[\Delta_L]{n+1} q \wedge t \Rightarrow_{op \in R/A} t' = f'(t'_1, \ldots, t'_k)$ then $t' \xrightarrow[\Delta']{m} q$, with $m, n, k \in \mathbb{N}$ and $q \in Q_f$. In

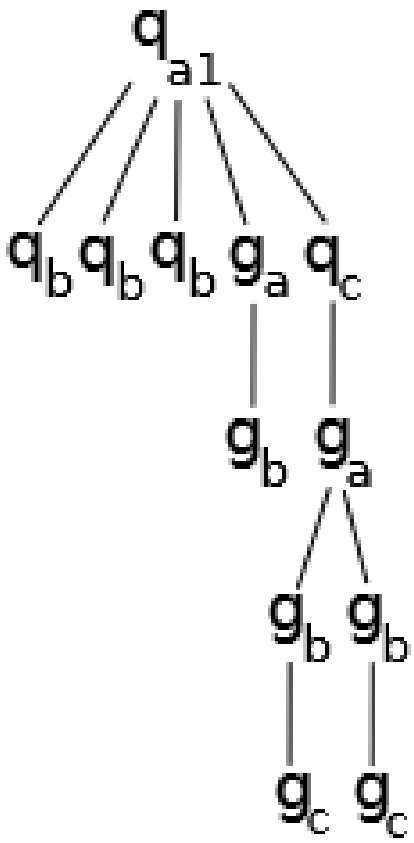

Figure 18. The accepting computation of the NFHA A' over t'.

this way $t' \in Post_{R/A}(L)$ because it is obtained through parallel rewriting from $t \in L$, using one of the operations of the (SRHP) R/A, but we have also that $t' \in L(A')$, because it's accepted by the automaton A', in other words we have that t' is evaluated in the state $q \in Q_f$ using the transictions of Δ'.

- Base case ($n = 0$): $t \xrightarrow[\Delta_L]{} q$, so $t \in \Sigma$, now we have to prove that $t' \in L(A')$ in any case, according to the possible values of *op*:

  $op = REN : a(x) \to b(x);$ then $t = a$ and $t' = b$ and due to the construction of the automaton A' we have that $\forall r_L = a(L_a) \to q_a \in \Delta_L$. $\exists r' = b(L_a) \to q_a \in \Delta'$, so for every rule $r_L \in \Delta_L$ used in $t \xrightarrow[\Delta_L]{} q$, we use the corresponding rule $r' \in \Delta'$ for $t' \xrightarrow[\Delta']{} q$.

  $op = INS_{first} : a(x) \to a(px);$ then $t = a$ and $t' = a(t'')$ with $t'' \xrightarrow[\Theta]{*} p$ ($t'' \in L(A)$ by definition), and for construction of A' we have that

  $$\begin{cases} (q_{a,q}^{fresh}, p, f_{a,q}) \in \Gamma'_{a,q} & \text{se } L_{a,q} = \{\epsilon\}, \\ \{(q_{a,q}^{fresh}, p, i_{a,q}), (i_{a,q}, y, q_y)\} \subseteq \Gamma'_{a,q} & \text{se } L_{a,q} \supset \{\epsilon\} \end{cases}$$

  and $\Theta \subseteq \Delta'$. If $a(L_{a,q}) \to q \in \Delta_L$ is used as a transiction in $t \xrightarrow[\Delta_L]{} q$, then we have $t' \xrightarrow[\Delta']{*} q$ and the last step is the application of $a(L'_{a,q}) \to q \in \Delta'$.

  $op = INS_{last} : a(x) \to a(xp);$ then $t = a$ and $t' = a(t'')$, with $t'' \xrightarrow[\Theta]{*} p$ ($t'' \in L(A)$ by definition), and by construction of A' we have

  $$\begin{cases} (i_{a,q}, p, q_{a,q}^{fresh}) \in \Gamma'_{a,q} & \text{se } L_{a,q} = \{\epsilon\}, \\ \{(q_y, y, f_{a,q}), (f_{a,q}, p, q_{a,q}^{fresh})\} \subseteq \Gamma'_{a,q} & \text{se } L_{a,q} \supset \{\epsilon\} \end{cases}$$

  and $\Theta \subseteq \Delta'$. If $a(L_{a,q}) \to q \in \Delta_L$ is used as a transiction in $t \xrightarrow[\Delta_L]{} q$, then we have $t' \xrightarrow[\Delta']{*} q$ and the last step is the application of $a(L'_{a,q}) \to q \in \Delta'$.

  $op = INS_{into} : a(xy) \to a(xpy);$ then $t = a$ and $t' = a(t'')$ $t'' \xrightarrow[\Theta]{*} p$ ($t'' \in L(A)$ by definition), and by construction of A' we have $\Theta \subseteq \Delta'$ and $\{(s, j, q_{a,q}^{fresh}), (q_{a,q}^{fresh}, p, s')\} \subseteq \Gamma'_{a,q}$ for every rule $(s, j, s') \in \Gamma_{a,q}$. If $a(L_{a,q}) \to q \in \Delta_L$ is used as a transiction in $t \xrightarrow[\Delta_L]{} q$, then we have $t' \xrightarrow[\Delta']{*} q$ and the last step is the application of $a(L'_{a,q}) \to q \in \Delta'$.

  $op = INS_{before} : a(x) \to pa(x);$ then $t = a$ and $t' = t''a$ with $t'' \xrightarrow[\Theta]{*} p$ ($t'' \in L(A)$ by definition), and by construction of A' we have $\Theta \subseteq \Delta'$ and $\{(s, p, q_{b,q}^{fresh}), (q_{b,q}^{fresh}, q', s')\} \subseteq \Gamma'_{b,q}$ for every rule (s, q', s') in $\Gamma_{b,q}$. So if $b(L_{b,q}) \to q \in \Delta_L$ is used as a transiction in $t \xrightarrow[\Delta_L]{} q$, then we have $t' \xrightarrow[\Delta']{*} q$ and the last step is the application of $b(L'_{b,q}) \to q \in \Delta'$. If t appears as a subtree we have that $\forall i \in [1, n]$. $t'_i \xrightarrow[\Delta']{} q_i \wedge t'_i(\epsilon) = a$, $q_1 \dots q_{i-1} p q_i \dots q_n \in L'_{b,q}$ if and only if $q_1 \dots q_{i-1} q_i \dots q_n \in L_{b,q}$.

  $op = INS_{after} : a(x) \to a(x)p;$ then $t = a$ and $t' = at''$ with $t'' \xrightarrow[\Theta]{*} p$ ($t'' \in L(A)$ by definition), and by construction of A' we have that $\Theta \subseteq \Delta'$ and $\{(s, q', q_{b,q}^{fresh}), (q_{b,q}^{fresh}, p, s')\} \subseteq \Gamma'_{b,q}$ for every rule (s, q', s') in $\Gamma_{b,q}$. So if $b(L_{b,q}) \to q \in \Delta_L$ is used as a transiction in $t \xrightarrow[\Delta_L]{} q$, then we have $t' \xrightarrow[\Delta']{*} q$ and the last step is the application of $b(L'_{b,q}) \to q \in \Delta'$. If t appears as a subtree we have that $\forall i \in [1, n]$. $t'_i \xrightarrow[\Delta']{} q_i \wedge t'_i(\epsilon) = a$, $q_1 \dots q_i p q_{i+1} \dots q_n \in L'_{b,q}$ if and only if $q_1 \dots q_i q_{i+1} \dots q_n \in L_{b,q}$.

$op = RPL : a(x) \to p$; then $t = a$ and $t' \xrightarrow[\Theta]{*} p$ (by definition) and, because of $\Theta \subseteq \Delta'$, we have that $t' \xrightarrow[\Delta']{*} p$. If t appears as a subtree of the root node in a generic tree t_0, with $t_0 \Rightarrow t'_0$, since by construction of A' we have that $(s, p, s') \in \Gamma'_{b,q}$ for every rule $(s, q, s') \in \Gamma_{b,q}$, we also have that if $b(L_{b,q}) \to q \in \Delta_L$ is used as the last transiction in $t_0 \xrightarrow[\Delta_L]{*} q$, it will be replaced by $b(L'_{b,q}) \to q \in \Delta'$ in $t'_0 \xrightarrow[\Delta']{*} q$.

$op = DEL : a(x) \to ()$; then $t = a$ and $t' = ()$, but we know that it's not possible to apply the operation DEL on the root node of a tree. If, on the contrary, t appears as a subtree of the root node of a generical tree t_0, with $t_0 \Rightarrow t'_0$, since by construction of A' we have that $(s, \epsilon, s') \in \Gamma'_{b,q}$ for each rule $(s, q, s') \in \Gamma_{b,q}$, we have that if $b(L_{b,q}) \to q \in \Delta_L$ is used as the last transiction in $t_0 \xrightarrow[\Delta_L]{*} q$, it will be replaced by $b(L'_{b,q}) \to q \in \Delta'$ in $t'_0 \xrightarrow[\Delta']{*} q$.

- Inductive case ($n > 0$): $t_1 \xrightarrow[\Delta_L]{n} t_2 \xrightarrow[\Delta_L]{} t_3$, we assume as inductive hypothesis that $t'_1 \xrightarrow[\Delta']{*} q_1, \ldots, t'_k \xrightarrow[\Delta']{*} q_k$ if $t^1_1 \xrightarrow[\Delta_L]{*} q_1, \ldots, t^k_1 \xrightarrow[\Delta_L]{*} q_k$, with $t_1 = f(t^1_1 \ldots t^1_k)$, from which we have $t'_1 \xrightarrow[\Delta']{*} t'_2$, then given $t_2 \xrightarrow[\Delta_L]{} t_3$ we obtain $t'_2 \xrightarrow[\Delta']{*} t'_3$ with the application of the Base case.

Proof of $L(A') \subseteq Post_{R/A}(L)$: In order to prove that $L(A') \subseteq Post_{R/A}(L)$, we start from the observation that $L(A') \subseteq Post_{R/A}(L) \equiv \forall t' \in L(A') \,.\, \exists t \in L, (u_1, \ldots, u_k) \subseteq R/A \,.\, t \Rightarrow_{(u_1,\ldots,uk)} t'$.

Given a tree t' and an accepting computation of the NFHA A' over it denoted as $A'||t'$, computation that exists by definition since we have $t' \in L(A')$, we can derive the corresponding accepting computation of the NFHA A over the tree t, denoted as $A||t$, inferring one of the possible update sequence $(u_1, \ldots, u_k)$, by which we have the thesis.

- Base case ($Height(A'||t' = 1)$):

  (REN): if $A'||t' = q_a$ and $\exists a(L_a) \to q_a \in \Delta_L$, $b(L_b) \to q_b \in \Delta' \,.\, L_a = L_b \wedge q_a = q_b \wedge a \neq b \wedge b(L_b) \to q_b \notin \Delta_L \wedge a(L_a) \to q_a \notin \Delta'$, then $A||t = A'||t'$ and $(u_1, \ldots, u_k) = REN$.

- Base case ($Height(A'||t' = 2)$):

  (DEL): if $A'||t' = q_a(q_1, \ldots, q_i, q_{i+1}, \ldots, q_n)$ and $\exists a(L_a) \to q_a \in \Delta_L$, $a(L'_a) \to q_a \in \Delta' \,.\, \exists (s, q_i, s')$, (s', ϵ, s''), $(s'', q_{i+1}, s''') \in \Gamma'_{a,q_a}$, $(s', q, s'') \in \Gamma_{a,q_a}$, then $A||t = q_a(q_1, \ldots, q_i, q, q_{i+1}, \ldots, q_n)$ and $DEL \in (u_1, \ldots, u_k)$.

  (RPL): if $A'||t' = q_a(q_1, \ldots, q_i, p, q_{i+1}, \ldots, q_n)$ and $\exists a(L_a) \to q_a \in \Delta_L$, $a(L'_a) \to q_a \in \Delta' \,.\, \exists (s, q_i, s')$, (s', p, s''), $(s'', q_{i+1}, s''') \in \Gamma'_{a,q_a}$, $(s', q, s'') \in \Gamma_{a,q_a}$, then $A||t = q_a(q_1, \ldots, q_i, q, q_{i+1}, \ldots, q_n)$ and $RPL \in (u_1, \ldots, u_k)$.

  (REN): if $\exists a(L_a) \to q_a \in \Delta_L$, $b(L_b) \to q_b \in \Delta' \,.\, L_a = L_b \wedge q_a = q_b \wedge a \neq b \wedge b(L_b) \to q_b \notin \Delta_L \wedge a(L_a) \to q_a \notin \Delta'$, then $A||t = A'||t'$ and $REN \in (u_1, \ldots, u_k)$.

  (INS_x): if none of the three preceding cases holds and $\forall e \in \mathcal{P}os(A'||t') \,.\, (A'||t')(e) = p \wedge p \in P$, then $(A||t)(e) = \epsilon$ and $INS_x \in (u_1, \ldots, u_k)$, with $x \in \{first, last, into, before, after\}$.

- Inductive case ($Height(A'||t' > 2)$): if $A'||t' = q_a(A'||t'_1, \ldots, A'||t'_n)$ through the inductive hypothesis we can derive $A||t_1$, ..., $A||t_n$ from $A'||t'_1$, ..., $A'||t'_n$, respectively, and we can also derive $A||t$ applying the Base case ($Height(A'||t' = 2)$ on the first two steps of the tree $q_a(A||t_1, \ldots, A||t_n)$.

We conclude observing that if exists an accepting computation for a given automaton over a certain tree, then that tree belongs necessarily to the language accepted by the automaton itself. Thus, since we have obtained the computation $A||t$ starting from the computation $A'||t'$, $t \Rightarrow t' \wedge t \in L(A)$ and we have the thesis.

5 Schema Update Framework

Schema Update Framework is a proposal of a framework for XML schema update that handles also updates over related documents. In Figure 19 we can see the main components of *Schema Update Framework*, below there is a detailed analisys of each of them:

- S' is the starting schema,
- S'' is the new schema obtained from the application of $(U_1, \ldots, U_n)$, a sequence of schema updates,
- A is the automaton describing the new components introduced by the updates applied to schema S',
- A' is the automaton corresponding to schema S',
- A''' is the automaton recognizing $Post_{(u_1,\ldots,u_m)}(L(A'))$, with $(u_1, \ldots, u_m) \subseteq R/A$ and $R \subseteq UFO$,
- A'' is the automaton corresponding to the schema S''.

The used schemas can be expressed through any of the main schema definition languages (DTD, XSD and RelaxNG), for each of them we can obtain the equivalent NFHA, as [13] shows proving that automata are equivalent to grammars that are in turn equivalent to schema languages. The update sequence $(U_1, \ldots, U_m)$ can be expressed through an update language compatible to the choosen schema language. The update sequence for the related documents, $(u_1, \ldots, u_m)$, is proposed by the user on the basis of the schema update sequence, $(U_1, \ldots, U_n)$; the user could be asked to suggest this sequence of document update only in few necessary cases (in the scenario shown in Example 12 only the user can choose the right instance of the rule INS_{after} by fixing the type of the tree that will be inserted); the other cases could be directly handled by an heuristic. After the initial phase in which we obtain the automata A''' and A'' we can execute the Hedge Automata *inclusion test* (described largely in Chapter 4 of the thesis) on them, if the test is succesfull we can statically ensure that the application of the proposed document update sequence $(u_1, \ldots, u_m)$ on any valid document for the schema S' will produce a valid document for the new schema S''. If the test fails we have no guarantees that a valid document with respect to S', updated following the update sequence $(u_1, \ldots, u_m)$, will be a valid document with respect to S''.

Example 12: We will now see a use case scenario for *Schema Update Framework*. In the Listing 1 we have the starting schema S', expressed using XML Schema. The symbol `(*)` used in the schema represent the point of insertion of the XML schema fragment, called S, shown in Listing 2 that will be inserted by the schema update which will generate the new schema S''. Now we obtain the three NFHA A, A' and A'', correspondig to schemas S, S' and S'', respectively.

$A = (\Sigma_L, Q, Q_f, \Theta)$, with:

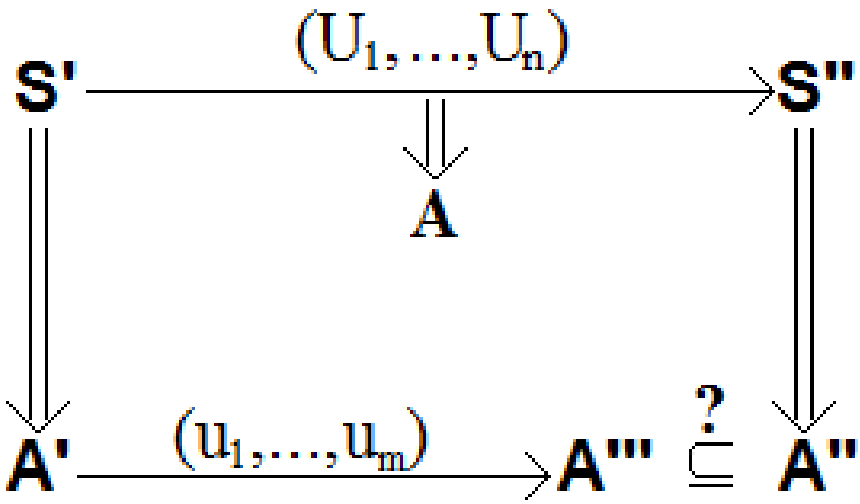

Figure 19. Summary of Schema Update Framework architecture, with $(u_1, \ldots, u_m) \subseteq R/A$ **and** $R \subseteq UFO$**.**

Listing 1. Starting schema S'**.**

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="academic-transcript">
    <xsd:complexType>
      <xsd:sequence>
        (*)
        <xsd:element name="record" minOccurs="0"
                                   maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="exam" type="xsd:string"/>
              <xsd:element name="grade" type="xsd:string"/>
              <xsd:element name="date" type="xsd:date"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 2. Schema fragment S inserted into S' by the schema update operation.

```
<xsd:element name="student-info">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="surname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- $\Sigma_L = \{'student-info', 'id', 'name', 'surname'\}$,
- $Q = \{p_{st}, p_i, p_n, p_s\}$,
- $Q_f = \{p_{st}\}$,
- $\Theta = \{id(\epsilon) \to p_i, name(\epsilon) \to p_n, surname(\epsilon) \to p_s, student-info(p_i p_n p_s) \to p_{st}\}$.

$A' = (\Sigma, Q', Q'_f, \Delta')$, with:

- $\Sigma = \{'academic-transcript', 'record', 'exam', 'grade', 'date'\}$,
- $Q' = \{q_e, q_g, q_d, q_r, q_a\}$,
- $Q'_f = \{q_a\}$,
- $\Delta' = \{exam(\epsilon) \to q_e, grade(\epsilon) \to q_g, date(\epsilon) \to q_d, record(q_e q_g q_d) \to q_r,$
  $academic-transcript(q_r^*) \to q_a\}$.

$A'' = (\Sigma := \Sigma \cup \Sigma_L, Q'', Q''_f, \Delta'')$, with:

- $\Sigma = \{'academic-transcript', 'record', 'exam', 'grade', 'date', 'student-info', 'id', 'name', 'surname'\}$,
- $Q'' = Q' \cup Q = \{q_e, q_g, q_d, q_r, q_a, p_{st}, p_i, p_n, p_s\}$,
- $Q''_f = Q'_f = \{q_a\}$,
- $\Delta'' = \Theta \cup (\Delta' \setminus \{academic-transcript(q_r^*) \to q_a\}) \cup \{academic-transcript(p_{st} q_r^*) \to q_a\}$.

Listing 3. An example of XML document D' valid with respect to schema S'.

```
<?xml version="1.0" encoding="utf-8"?>
<academic-transcript>
  (*)
  <record>
    <exam>Database</exam>
    <grade>25</grade>
    <date>2010-01-25</date>
  </record>
  <record>
    <exam>Calculus</exam>
    <grade>30</grade>
    <date>2010-02-25</date>
  </record>
</academic-transcript>
```

Listing 4. XML document fragment D.

```
<student-info>
  <id>1234ABC</id>
  <name>Alessandro</name>
  <surname>Solimando</surname>
</student-info>
```

In the Listing 3 we can see an example of an XML document, called D', valid with regard to schema S'. If we insert the XML schema fragment D (shown in Listing 4) into D' using the insertion point marked with `(*)`, we obtain a document D'' valid with respect to schema S''. In Figure 20 we can see the tree t' corresponding to the document D', while in Figure 21 we can see tree t'' corresponding to the document D''. Using our parallel rewriting semantic and the operation INS_{first} we have: $academic-transcript(x) \rightarrow academic-transcript(tx)$, with $t = student-info(id, name, surname)$ that is the tree corresponding to the XML fragment D; we can easily obtain t'' from t' ($t' \Rightarrow_{INS_{first}} t''$).

It is evident from above that the update sequence corresponding to the suggested schema update consists in this single instance of the INS_{first} operation: $academic-transcript(x) \rightarrow academic-transcript(p_{st}x) \in R/A$, with R as the SRHP composed by the rule $INS_{first} \in UFO$ and A the NFHA used to parameterize R. Let's see how to obtain the NFHA A''' accepting $Post_{R/A}(L(A'))$. $A''' = (\Sigma := \Sigma \cup \Sigma_L, Q' \cup Q, Q'_f, \Delta''')$, with:

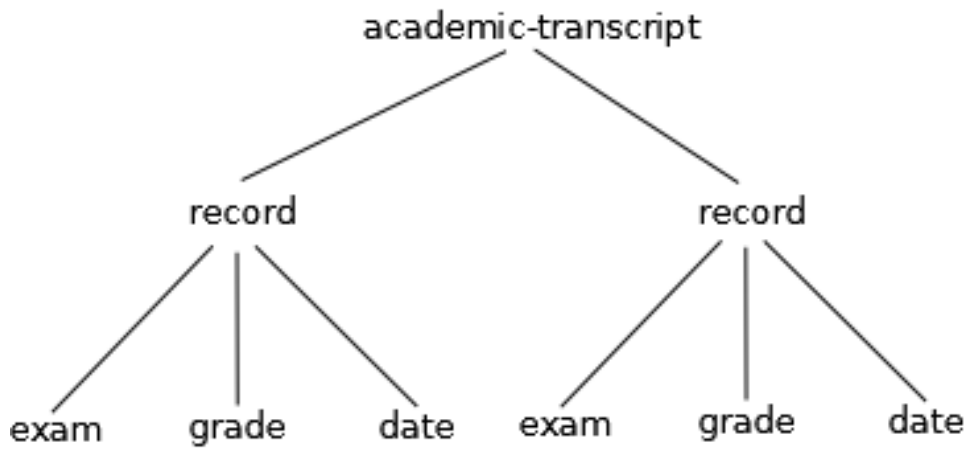

Figure 20. Tree representation t' of document D'.

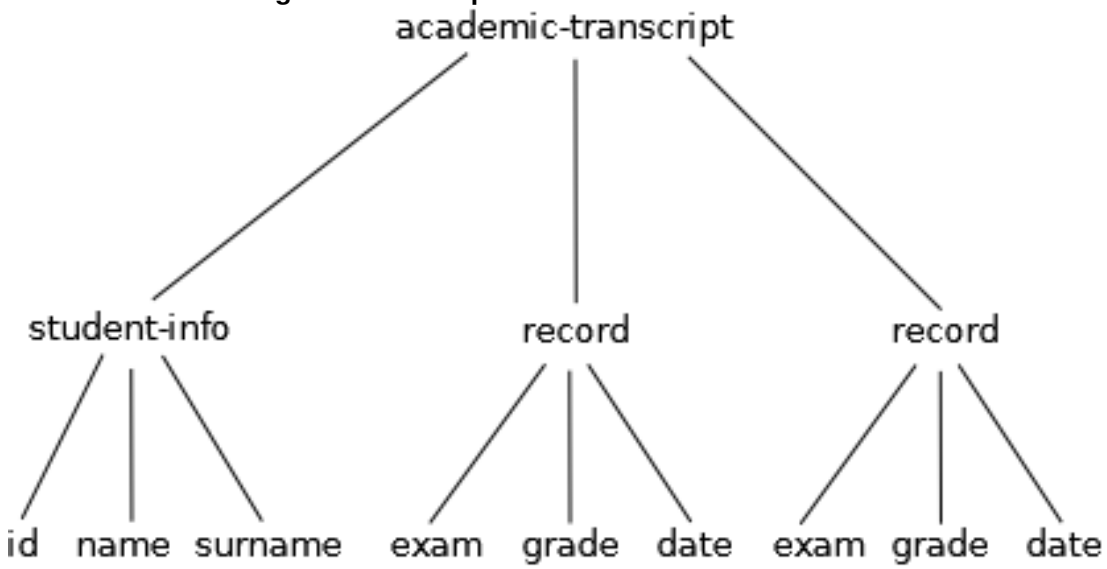

Figure 21. Tree representation t'' of document D''.

- $\Sigma = \{'academic{-}transcript', 'record', 'exam', 'grade', 'date', 'student{-}info', 'id', 'name', 'surname'\}$,
- $Q' \cup Q = \{q_e, q_g, q_d, q_r, q_a, p_{st}, p_i, p_n, p_s\}$,
- $Q'_f = \{q_a\}$.

Now we will analyze the modifications made by the algorithm to obtain Δ'''. To increase the readability we will use 'a-t' instead of 'academic-transcript'. First of all we need to analyze the NFA that will be modified by the algorithm, that is $B_{a-t,q_a} = (Q' \cup Q, \{b\}, b, \{b\}, \{(b, q_r, b)\})$. Uppermost we create a fresh state q^{fresh}_{a-t,q_a} and we add it to the states set of the automaton and we make it the starting state, finally we add the rule $(q^{fresh}_{a-t,q_a}, p_{st}, b)$ because there is rule (b, q_r, b). After these changes we have $B_{a-t,q_a} = (Q' \cup Q, \{b, q^{fresh}_{a-t,q_a}\}, q^{fresh}_{a-t,q_a}, \{b\}, \{(q^{fresh}_{a-t,q_a}, p_{st}, b), (b, q_r, b)\})$ and the horizontal language associated with the rule $a - t(L_{a-t,q_a}) \to q_a$ changes from q^*_r to $p_{st}q^*_r$. In the end we calculate $\Delta''' := \Theta \cup \{a(B_{a,q}) \to q \mid a \in \Sigma, q \in Q_L, L(B_{a,q}) \neq \emptyset)\}$, which is equal to $\Theta \cup (\Delta' \setminus \{a - t(q^*_r) \to q_a\}) \cup \{a - t(p_{st}q^*_r) \to q_a\}$, that is in turn equals to Δ''.

Since the two automata, A'' (derived from schema S'') and A''' (obtained by the algorithm that calculate $Post_{R/A}(L)$) are identical, the inclusion test will affirm that $L(A'') \subseteq L(A''')$, because $L(A'') = L(A''')$ holds, thus the proposed update sequence is correct and any of its instances will lead to a valid document with respect to S'' if the starting document is valid with respect to S', as we have already seen for the documents D' and D''.

This example is really simple and it is clear that the correct update sequence we need to apply is $(u_1, \ldots, u_m)$. In real use cases scenarios, where the schemas and documents involved are usually very large and the required update could be really complex, the user can be guaranteed by the framework that the choosen update sequence will preserve documents validity even with respect to the new schema.

Suppose we modify document D'' through update $DEL : date(x) \to ()$, obtaining so the document D''', identical to D'' but without $date$ elements; in Figure 22 we can see its tree representation called t'''. It is clear that t''' can be obtained from t'' in one step using our parallel rewriting semantic, that is $t'' \Rightarrow_{DEL} t'''$. Because of document D''' is not valid with respect to schema S'', its tree representation, called t''', is not included in the language accepted by the automaton A'' corresponding to this schema. In Figure 23 we can see the computation $A''||t'''$ of the automaton A'' related to tree t''', where the states corresponding to the evaluation of each node are marked in red. This computation cannot assign a final state to the root node of the tree because the tree isn't part of the language accepted by the considered automaton, this is caused by the absence of rules of the form $record(L) \to q_L$ whereby $q_e q_g \in L$. $\square$

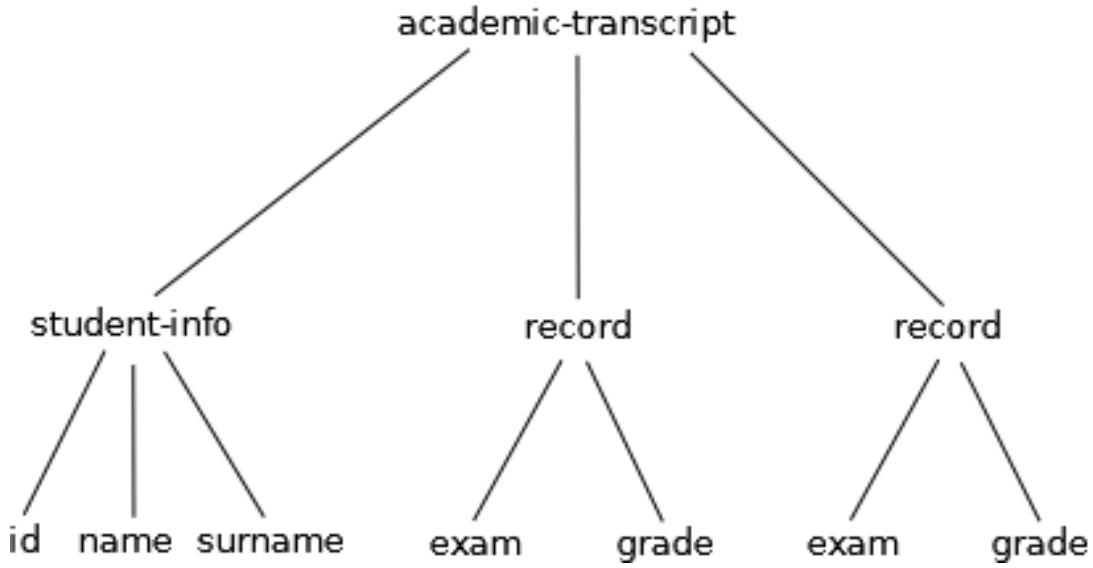

Figure 22. Tree representation t''' of the document D'''.

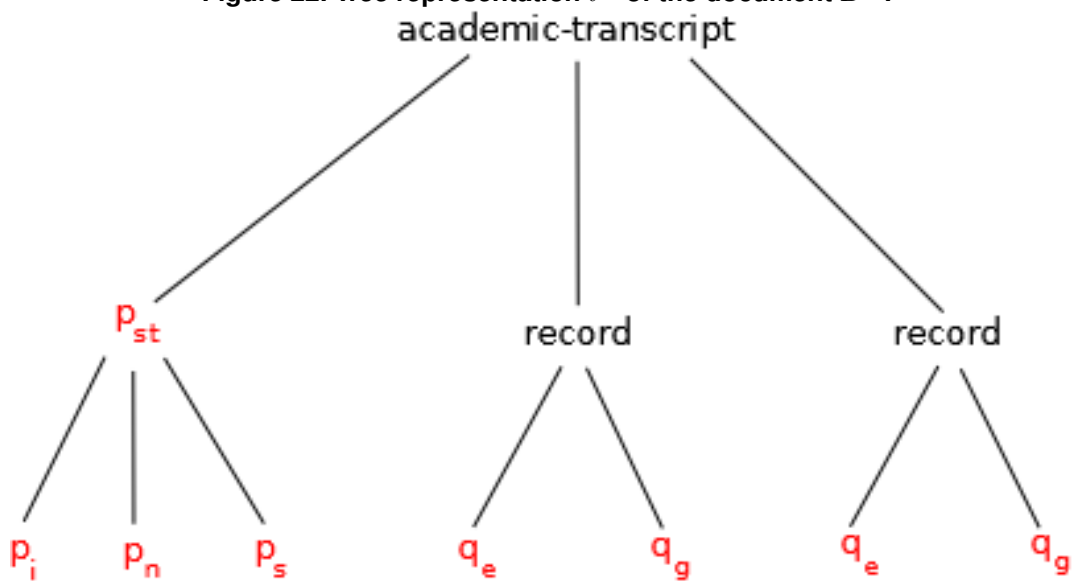

Figure 23. Computation $A''||t'''$ of the automaton A'' relative to tree t''' representing document D'''.

Listing 5. *XQuery Update* **expression which deletes the** $surname$ **element if there are more than** 10 $record$ **elements.**

```
for $student in doc("doc.xml")//academic-transcript/student-info
if (count($student/record) > 10)
then delete node $student/surname
else ()
```

Example 13: *Schema Update Framework* doesn't fully support *XQuery Update* in all its features, we will now see an example showing one of the uncovered aspects. Suppose that we want to update schema S'' showed in Example 12 in order to make the element $surname$ optional and that, through the expression *XQuery Update* reported in Listing 5, we modify every document associated with S'' in order to delete the element $surname$ if there are more than 10 $record$ elements in the document. Such an update, which uses complex selection conditions, cannot be represented using UFO rules, that's why it cannot be handled by our framework. □

6 Related Work

In this section we will analyze the limits of our works and also make a comparison with alternative techniques used for static analysis for XML updates.

6.1 Limits of our proposal

Schema Update Framework uses Hedge Automata to represent XML schemas, this fact does not allow the support for XML Schema's *all* construct and *interleaving* construct available in Relax NG, with which is possible to accept a arbitrary permutation of a sequence of elements. Sheaves Automata introduced in [6] have new transiction rules able to represent commutative trees, that is why their expressivity is strict superior than the one of the Hedge Automata. Another limitation of our approach is represented by the lack of the support of many features of *XQuery Update Facility*; in fact we only support atomic primitives that corresponds to the operations called UFO. *XQuery Update Facility*, indeed, permits to use very complex expressions, using *XPath*, procedural constructs and the execution of query with complex pattern, well beyond the simple use of the nodes label, as filtering conditions: in fact in XQuery Update Facility you can have arbitrary query as Predicates.

6.2 Alternative approaches

Our approach is not the only suitable, there others well known techniques used in statical analysis, such as Type Systems, to infer the final schema, derived from a sequence of updates: the main work for the formalization of schema update is represented by [1], where they take into account a subset of XQuery Update which doesn't deal with document attributes but only with structural conditions imposed by tags. Type inference for *XQuery Update*, without approximations, is not always possible, that come from the fact that modifications that can be produced using this language can lead to non regular schemas, that is why they cannot be captured with existing schema languages for XML; this is the reason for which [1], as well as [15], computes an over-approximation of the type

set that results from the updates. In this work, on the contrary, in order to produce an exact computation we needed to cover an inferior subset of XQuery Update's features: [1], indeed, allows to use XPath's axes to query and select nodes, making possible in this way to mix selectivity conditions with positionals constraints with the request to satisfy a given pattern. In our work, as well as in [15] and [7], we have considered only atomic update primitives, excluding thus complex expressions as well as "for loops" and "if statements", based on the result of a query, used in the first case to bind the variable, in the latter to choose one branch or the opposite. This expressions, anyway, can be translated into a sequence of atomic operations: an expression using a "for loop", for instance, repeats n times a certain atomic operation, and therefore can be simulated with a sequence of n instances of that single atomic operation. To see this let's consider what authors of [1] have done in Section called "Semantics", where is provided a translaction phase of update expressions into a list of update pending list made only of atomic operations.

Another valid approach to express XML transformations in using *Macro Tree Transducers* (MTT), as in [8]: the authors propose here a new language called *Transformation Language* (TL), based on *Monadic Second-Order logic* (MSO) as a pattern formalism, which not only generalize XPath, XQuery and XSLT, but it can also be simulated using macro tree transducers. The composition of MTT and their property of preserving recognizability for the calculation of their inverse are exploited to perform *inverse type inference*: they pre-compute in this way the pre-image of ill-formed output and perform type checking simply testing if the input type has some intersection with the pre-image or not (notice that tree languages as closed for intersection and emptyness test is decidable for them). Their system, as ours, is exact and do not approximate the calculation, but in contrast to our method there is a potential implementation problem going from MSO patterns to equivalent finite automata, on top on which most of their system is developed, even if MSO is not the only suitable pattern language that can be used with their system.

7 Conclusions

The novel contributions of the present work are the formalization of a parallel rewriting semantic, more adherent to XML's update behaviour than the interleaving semantic present in literature (as example in [7] and [15]), the definition of the *Post* operation based on our parallel semantic and on *XQuery Update Facility* primitive operations, *W3C*'s raccomandation as update language for XML; we have also formalized the algorithm that computes the new automaton able to accept the modified language, taking in input the original automaton and the sequence of applied updates. On top of these contributions we developed an update handling framework, integrable in systems using XML schema updates.

Future works The present work could be extended under the following aspects:

- Adding support for commutative trees, in which the order of the children of a node is irrelevant, feature that could permit the formalization of the *All* and *interleave* costructs of XML Schema ([19]) and Relax NG ([4], [16]), respectively. Sheaves Automata, introduced in [6], are able to recognize commutative trees, they have an expressivity strictly greater than Hedge Automata considered in this work; it should be thus useful to study the applicability of these automata in our framework.

- Given the schema update operation sequence we could develop an heuristic to extract automatically a sequence of update operations for the documents associated with the modified schema that will grant document's validity with regard to the new schema, relieving the user from suggesting the appropriate sequence.

- We could refine node's selection constraints for the update operations, for example using *XPath*'s axes ([18]) and the other features offered by *XQuery Update Facility*.

- It should be useful to implement the *Schema Update Framework* and integrate it with *XSchemaUpdate* ([3]) or other applicable languages.

- It should be interesting to investigate the complexity of the algoritm that computes the automaton recognizing $Post$ operation.

References

[1] M. Benedikt and J. Cheney. Semantics, types and effects for xml updates. In *DBPL '09: Proceedings of the 12th International Symposium on Database Programming Languages*, pages 1–17, Berlin, Heidelberg, 2009. Springer-Verlag.

[2] A. Brggemann-Klein, M. Murata, and D. Wood. Regular Tree and Regular Hedge Languages over Unranked Alphabets: Version 1, 2001.

[3] F. Cavalieri. Querying and Evolution of XML Schemas and Related Documents. Master's thesis, Universit di Genova, a.a. 2008/2009. `http://www.disi.unige.it/person/GuerriniG/doc/2009Cavalieri.pdf`.

[4] J. Clark and M. Murata. RELAX NG Specification. `http://www.relaxng.org/spec-20011203.html`, 2001.

[5] 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://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[6] S. Dal-Zilio and D. Lugiez. XML Schema, Tree Logic and Sheaves Automata. In *RTA*, pages 246–263, 2003.

[7] F. Jacquemard and M. Rusinowitch. Formal Verification of XML Updates and Access Control Policies. May 2010.

[8] S. Maneth, A. Berlea, T. Perst, and H. Seidl. Xml type checking with macro tree transducers. In *PODS*, pages 283–294, 2005.

[9] M. Murata. Forest-regular Languages and Tree-regular Languages, 1995.

[10] M. Murata. DTD Transformation by Patterns and Contextual Conditions. In *In SGML/XML '97 Conference Proceedings*, pages 325–332, 1997.

[11] M. Murata. Transformation of Documents and Schemas by Patterns and Contextual Conditions. In *PODP '96: Proceedings of the Third International Workshop on Principles of Document Processing*, pages 153–169, London, UK, 1997. Springer-Verlag.

[12] M. Murata. Hedge automata: a formal model for XML schemata, 1999.

[13] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Technol.*, 5(4):660–704, 2005.

[14] M. Raento. An introduction to forest-regular languages. Basic Research Unit, Helsinki Institute for Information Technology, Department of Computer Science, University of Helsinki.

[15] T. Touili. Computing Transitive Closures of Hedge Transformations, 2007. In: Proc. 1st Int. Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS 2007), eWIC Series. British Computer Society.

[16] E. van der Vlist. RELAX NG. `http://books.xmlschemata.org/relaxng/page2.html`, 2003.

[17] "W3C as D. Chamberlin and M. Dyck and D. Florescu and J. Melton and J. Robie and J. Simon". XQuery Update Facility 1.0. `http://www.w3.org/TR/2009/CR-xquery-update-10-20090609/`, 2009.

[18] "W3C as J. Clark, S. DeRose". XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath/`, 1999.

[19] "W3C as P. Walmsley and D.C. Fallside". XML Schema Part 0: Primer Second Edition. `http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/`, 2004.