Static Analysis of XML Document Adaptations

Alessandro Solimando, Giorgio Delzanno, and Giovanna Guerrini

DIBRIS, Università di Genova, Italy
{alessandro.solimando,delzanno,guerrini}@unige.it

Abstract. In this paper we propose a framework for XML data and schema co-evolution that allows to check whether a user-proposed document *adaptation* (i.e., a sequence of document update operations intended to adapt the documents valid for a schema to a new schema) is guaranteed to produce a document valid for the updated schema. The proposed framework can statically determine, working only with the automata related to the original and modified schema, if the document update operation sequence will re-establish document validity, thus avoiding the very expensive run-time revalidation of the set of involved documents that is usually performed upon schema update.

1 Introduction

Practical data management scenarios are characterized by an increasing dynamicity and rapid evolution so that updates to data, as well as to their structures, are very frequent. Only an efficient support for changes to both data and structural definitions can guarantee an appropriate use of schemas [12,10]. Indeed schema updates have a strong impact on existing data. Specifically, the term co-evolution refers to the ability of managing the mutual implications of data, schema, and application changes.

This need is extremely pressing in the context of the eXtensible Markup Language (XML), which has become in the last ten years a standard for data representation and exchange and is typically employed in highly evolving environments. Upon any update at schema level, XML documents valid for the original schema are no longer guaranteed to meet the constraints described by the modified schema and might need to be *adapted* [8,2]. An automatic adaptation of associated documents to the new schema definition is possible in some cases, but it is not always obvious how to re-establish document validity broken by a schema update. More flexible, user-defined adaptations, corresponding to arbitrary document update statements, would allow the user to specify, for instance through XQuery Update Facility (XQUF) [17] expressions, ad hoc ways to convert documents valid for the old schema in documents valid for the new schema. These arbitrary, user-defined adaptations, however, are not guaranteed to produce documents valid for the new schema and dynamic revalidation is needed, which may be very expensive for large document collections.

In this paper, we propose a static analysis framework (*Schema Update Framework*) for a subset of XQUF based on *Hedge Automata*, a symbolic representation of infinite sets of XML documents given via unranked trees. Specifically, the framework allows to check whether a user-proposed document *adaptation* (i.e., a set of document update operations intended to adapt the documents related to a schema that have just been updated by a known sequence of schema update operations) is guaranteed to produce a document valid for the updated schema. The framework relies on a transformation algorithm for Hedge Automata that captures the semantic of document update operations. The transformation rules allow to reason about the impact of user-defined adaptations, potentially avoiding run-time revalidation for safe adaptations.

The proposed 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 w.r.t. the original schema, after the application of the document update sequence, will result in a new modified document valid w.r.t. the updated schema. Thus, run-time revalidation of the set of involved documents can be avoided.

The remainder of the paper is organized as follows: Section 2 introduces the preliminaries needed in the following sections, Section 3 illustrates the proposed framework, finally, in Section 4 we discuss related work, Section 5 concludes the work.

2 Preliminaries

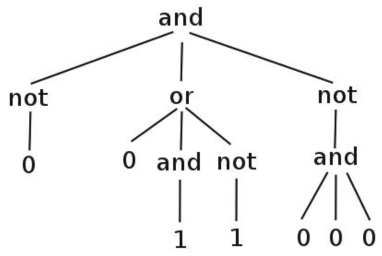
In this section we introduce Hedge Automata, as a suitable formal tool for reasoning on a representation of XML documents via unranked trees, and the update operations our framework relies on.

Hedge Automata (HA). Tree Automata are a natural generalization of finite-state automata that one can adopt to define languages over ranked finite trees. Tree Automata are used as a formal support for XML document validation. In this setting, however, it is often more convenient to consider more general classes of automata, like Hedge and Sheaves Automata, to manipulate both ranked and unranked trees. The difference between ranked and unranked trees lies in the arity of the symbols used as labels. Ranked symbols have fixed arities, that determine the branching factor of nodes labelled by them. Unranked symbols have no such a constraint. Since in XML the number of children of a node with a certain label is not fixed a priori, and different nodes sharing the same label can have a different number of children, unranked trees are more adequate for XML.

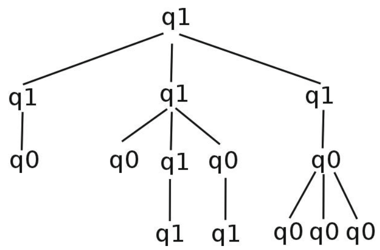
Given an unranked tree $a(t_1, \dots, t_n)$ where $n \geq 0$, the sequence $t_1, \dots, t_n$ is called **hedge**. For $n = 0$ we have an empty sequence, represented by the symbol ϵ. The set of hedges over Σ is $H(\Sigma)$. Hedges over Σ are inductively defined as follows: the empty sequence ϵ is a hedge; if g is a hedge and $a \in \Sigma$, then $a(g)$ is a hedge; if g and h are hedges, then gh is a hedge.

Example 1. *Given the tree $t = a(b(a, c(b)), c, a(a, c))$, the corresponding hedges having as root nodes the children of the root of t are $b(a(c(b)))$, c and $a(a, c)$.* □

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 accepting states, Δ is a finite set of transition rules of the form $a(R) \to q$, where $a \in \Sigma$ and $R \subseteq Q^*$ is a regular language



(a) Tree t representing a true Boolean formula.



(b) Accepting computation of the automaton M over tree t.

Fig. 1. An example of tree t (left) and the computation $M||t$ of the automaton M over t (right)

over Q, $q \in Q$. Regular languages, denoted with R, that appear in rules belonging to Δ are said **horizontal languages** and represented with Nondeterministic Finite Automata (NFA). The use of regular languages allows us to consider unranked trees. For instance, $a(q^*)$ matches a node a with any number of subtrees generated by state q.

A **computation** of M over a tree $t \in T(\Sigma)$ corresponds to a bottom-up visit of t during which node labels are rewritten into states. More precisely, consider a node with label a such that the root nodes of its children have been rewritten into the list of states $q_1 \dots q_n \in Q$. Now, if a rule $a(R) \rightarrow q$ exists with $q_1 \dots q_n \in R$, then a can be rewritten into q, and so on.

A tree t is said to be **accepted** if there exists a computation in which the root node is labelled by $q \in Q_f$. The **accepted language** for an automaton M, denoted as $L(M) \subseteq T(\Sigma)$, is the set of all the trees accepted by M.

Example 2 (from [4]). *Consider the NFHA $M = (Q, \Sigma, Q_f, \Delta)$ where $Q = \{q_0, q_1\}$, $\Sigma = \{0, 1, not, and, or\}$, $Q_f = \{q_1\}$ and $\Delta = \{not(q_0) \rightarrow q_1, not(q_1) \rightarrow q_0, 1(\epsilon) \rightarrow q_1, 0(\epsilon) \rightarrow q_0, and(Q^*q_0Q^*QQ^*) \rightarrow q_0, and(q_1q_1^*q_1) \rightarrow q_1, or(Q^*q_1Q^*QQ^*) \rightarrow q_1, or(q_0q_0^*q_0) \rightarrow q_0\}$. Fig. 1(a) and Fig. 1(b) show the tree t representing a Boolean formula and the accepting computation of the automaton M (i.e., $M||t(\epsilon) = q_1 \in Q_f$), respectively.* □

Given two NFHAs M_1 and M_2, the *inclusion test* consists in checking whether $L(M_1) \subseteq L(M_2)$. It can be reduced to the emptiness test for HA ($L(M_1) \subseteq L(M_2) \Leftrightarrow L(M_1) \cap (T(\Sigma) \setminus L(M_2)) = \emptyset$). Inclusion is decidable, since complement, intersection and emptiness of HA can be executed algorithmically [4].

XQuery Update Facility Operations as Parallel Rewriting. XQuery Update Facility (XQUF) [17] is a W3C recommendation as update language for XML. Its expressions are converted into an intermediate format called Pending Update List, that uses the primitives shown in the first column of Table 1. In the second column of Table 1, the tree rewriting rule corresponding to each primitive is shown. In the rules, a and b are labels, and p is an automaton state that can be viewed as a type declaration (defining any tree accepted by state p). For instance, consider the rule INS_{first} defined as $a(X) \rightarrow a(pX)$. Given a tree t, the rule can be applied to any node with label a. Indeed, X is a free variable that matches any sequence of subtrees. If the rule is applied to a node n

Table 1. XQUF primitives, a and b are XML tags, p is a state of an HA, X, Y are free variables that denote arbitrary sequences of trees

Update Primitives	
REN	$a(X) \to b(X)$
INS_{first}	$a(X) \to a(pX)$
INS_{last}	$a(X) \to a(Xp)$
INS_{into}	$a(XY) \to a(XpY)$
INS_{before}	$a(X) \to pa(X)$
INS_{after}	$a(X) \to a(X)p$
RPL	$a(X) \to p$
DEL	$a(X) \to ()$

with label a, the result of its application is the insertion of a tree of type p as leftmost child of n. In this paper, $\Rightarrow_r$ denotes a parallel rewriting step in which the rule r is applied in a given tree t to all occurrences of subterms that match the left-hand side of r. In the previous example, we will insert a tree of type p to the left of the children of each one of the a-labelled nodes in the term t.

Target node selection is based on the node label only. In this way, indeed, we cover the whole set of XQUF primitives and still obtain an exact static analysis. More complex selection patterns could be exploited, but as [6] shows, even using basic relations (e.g. parent and ancestor), further restrictions are necessary (their method cannot support deletion in order to keep reachability decidable).

3 Schema Update Framework

In this section we propose a framework, called Schema Update Framework, that combines schema/document updates with an automata-based static analysis of their validity. Figure 2 illustrates the main components of the framework. The framework handles document adaptations expressed through the set of primitive update operations in Table 1 and schema expressed through any of the main schema definition languages for XML (DTD, XSD and RelaxNG). For each one of them, indeed, we can extract an equivalent NFHA: automata are equivalent to grammars that are in turn equivalent to schema languages [14]. More specifically, in Figure 2:

- S' is the starting schema,
- S'' is the new schema obtained from the application of a sequence of schema updates $(U_1, \ldots, U_n)$, expressed through an update language suitable to the chosen schema language,
- 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 the language modified using the document update sequence $(u_1, \ldots, u_m)$, where u_i is one of the primitives in Table 1, with $i \in [1 \ldots m]$,
- A'' is the automaton corresponding to the schema S''.

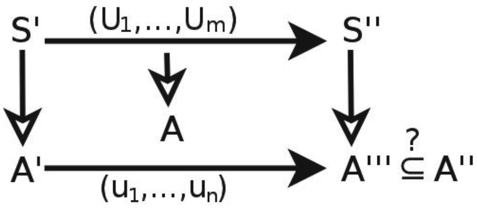

$$\begin{tikzcd} S' \arrow[rr, "(U_1,\dots,U_m)"] \arrow[d] & & S'' \arrow[d] \\ A' \arrow[rr, "(u_1,\dots,u_n)"'] & A & A''' \overset{?}{\subseteq} A'' \end{tikzcd}$$

Fig. 2. Summary of the framework

Note that in some cases no document adaptation may be needed (the schema update preserve document validity) or it can be derived from the schema update by a default adaptation strategy integrated by heuristics [8,2]. In the other cases, or if the default adaptation is not suitable, the user provides the sequence of document updates. For instance, in the update discussed in what follows, only the user can choose the right instance of the rule INS_{after} by fixing the type of the tree that will be inserted. As another example, if the schema update inserts a sibling of an optional element (resulting in a schema containing an optional sequence) document validity can be re-established by inserting the new element (default approach, that "mimics" the schema update) but also by deleting the original optional element.

In the initial phase, the automata A' and A'' are extracted from the schemas. The automaton A''' is computed using an algorithm that simulates the effect of the updates directly on the input HA. We then execute an *inclusion test* over the resulting HA. If the test succeeds, we can statically ensure that the application of the proposed document update sequence $(u_1, \dots, u_m)$ on any document valid for the starting schema S' produces a document valid for the new schema S''. If the test fails, we have no guarantees that a valid document w.r.t. S', updated following the update sequence $(u_1, \dots, u_m)$, is a valid document w.r.t. S''.

The core of the framework is the algorithm that computes the automaton A''' accepting the language obtained by the application of the document update sequence. It starts from the HA of the original schema. Depending on the operation type and parameters, it properly modifies the automaton by changing either the set of states and rules of the HA itself or the ones of the NFA accepting the horizontal languages. In this way, the semantics of the document update sequence is used to modify the language represented by the original schema, instead of the single document, allowing us to reason on the effect of the update sequence on the whole collection of documents involved. With these changes, the computed automaton accepts the original documents on which the specified sequence of updates has been applied. The algorithm and correctness proofs are available in [15]. In the following, we illustrate the behavior of the framework with the help of an example.

Listing 1.1 shows the starting schema S', expressed using XML Schema. The symbol `(*)` appearing in the schema represents the insertion point of the XML Schema fragment, referred to as S, shown in Listing 1.2. The schema update inserts the fragment S and generates the new schema S''. The three NFHA A, A' and A'', correspond to schemas S, S' and S'', respectively. For the sake of clarity, in what follows $student\text{-}info$ and $academic\text{-}transcript$ are abbreviated as si and at, respectively.

Listing 1.1. Starting schema S'.

```
<?xml version="1.0" encoding="utf−8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="at">
    <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>
```

- $A = (\Sigma_L = \{'si','id','name','surname'\}, Q = \{p_{st}, p_i, p_n, p_s\}, Q_f = \{p_{st}\}, \Theta = \{id(\epsilon) \to p_i, name(\epsilon) \to p_n, surname(\epsilon) \to p_s, si(p_i p_n p_s) \to p_{st}\})$,
- $A' = (\Sigma = \{'at','record','exam','grade','date'\}, Q' = \{q_e, q_g, q_d, q_r, q_a\}, Q'_f = \{q_a\}, \Delta' = \{exam(\epsilon) \to q_e, grade(\epsilon) \to q_g, date(\epsilon) \to q_d, record(q_e q_g q_d) \to q_r, at(q_r^*) \to q_a\})$,
- $A'' = (\Sigma := \Sigma \cup \Sigma_L = \{'at','record','exam','grade','date','si','id','name','surname'\}, Q'' = Q' \cup Q = \{q_e, q_g, q_d, q_r, q_a, p_{st}, p_i, p_n, p_s\}, Q''_f = Q'_f = \{q_a\}, \Delta'' = \Theta \cup (\Delta' \setminus \{at(q_r^*) \to q_a\}) \cup \{at(p_{st} q_r^*) \to q_a\})$.

Listing 1.2. Schema fragment S inserted into S' by the schema update operation.

```
<xsd:element name="si">
  <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>
```

Listing 1.3. An example of XML document D' valid w.r.t. 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 1.4. XML document fragment D.

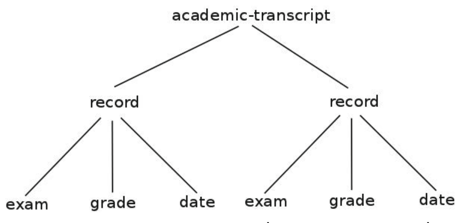
```
<si>
  <id>1234ABC</id>
  <name>Alessandro</name>
  <surname>Solimando</surname>
</si>
```

Listing 1.3 shows an example XML document, called D', valid w.r.t. schema S'. If we insert the XML schema fragment D (Listing 1.4) into D' at the insertion point marked with (*), we obtain a document D'' valid w.r.t. schema S''. Figure 3(a) shows the tree t' corresponding to document D', while in Figure 3(b) we can see tree t'' corresponding to the document D''. Using operation INS_{first} we have: $at(X) \to at(tX)$, where $t = si(id, name, surname)$ that is, the tree corresponding to the XML fragment D. t'' can be easily obtained from t' ($t' \Rightarrow_{INS_{first}} t''$).

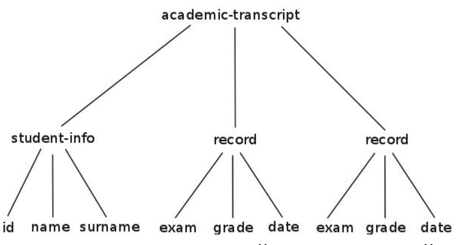
The update sequence corresponding to the suggested schema update consists in a single application of INS_{first} operation: $at(X) \to at(p_{st}X)$. The NFHA A''' accepting the language that reflects the application of the proposed document update sequence is as follows:

$A''' = (\Sigma := \Sigma \cup \Sigma_L = \{'at', 'record', 'exam', 'grade', 'date', 'si', 'id', 'name', 'surname'\}, Q' \cup Q = \{q_e, q_g, q_d, q_r, q_a, p_{st}, p_i, p_n, p_s\}, Q'_f = \{q_a\}, \Delta''')$.

Let us now see how the algorithm computes Δ'''. First of all, we need to analyze the NFA that will be modified by the algorithm, that is $B_{a-t,q_a} = (Q' \cup Q, \{b\}, b, \{b\}, \{(b, q_r, b)\})$. We then create a fresh state q^{fresh}_{a-t,q_a} and add it to the set of



(a) Tree representation t' of document D'.

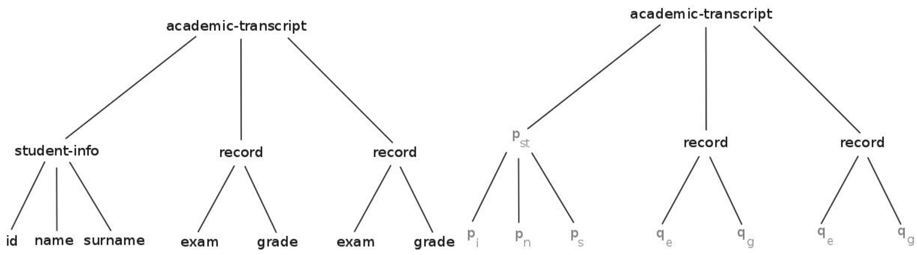


(b) Tree representation t'' of document D''.

Fig. 3. Tree representations of the XML documents D' and D''

states as a starting state. We also add the rule $(q^{fresh}_{a-t,q_a}, p_{st}, b)$, since the rule (b, q_r, b) is present. 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 $a - t(L_{a-t,q_a}) \rightarrow q_a$ changes from q_r^* to $p_{st}q_r^*$. Finally, we compute $\Delta''' := \Theta \cup \{a(B_{a,q}) \rightarrow q \mid a \in \Sigma, q \in Q_L, L(B_{a,q}) \neq \emptyset)\}$, which is equal to $\Theta \cup (\Delta' \setminus \{a - t(q_r^*) \rightarrow q_a\}) \cup \{a - t(p_{st}q_r^*) \rightarrow q_a\}$, that is, in turn, equal to Δ''.

Because the two automata, A'' (derived from schema S'') and A''' (obtained by the algorithm that calculates the language that reflects the application of the document update sequence) are identical, the inclusion test $(A'') \subseteq L(A''')$ succeeds. The proposed update sequence is thus type safe and its application on a document valid w.r.t. S' yields a document valid w.r.t. S'', as we have already seen for the documents D' and D''.



(a) Tree representation t''' of document D'''. (b) Computation $A''||t'''$ of the automaton A'' relative to tree t''' representing document D'''.

Fig. 4. Non accepting computation $A''||t'''$ (right) of automaton A'' over tree t''' (left)

Suppose now we modify document D'' through update $DEL : date(X) \rightarrow ()$, obtaining document D''', identical to D'' but without $date$ elements; its tree representation t''' is shown in Figure 4(a). Clearly t''' can be obtained from t'' as $t'' \Rightarrow_{DEL} t'''$. Since document D''' is not valid w.r.t. schema S'', its tree representation, t''', is not included in the language accepted by the automaton A'' corresponding to S''. In Figure 4(b) we can see the computation $A''||t'''$ of the automaton A'' related to tree t'''. This computation cannot assign an accepting state to the tree root node because the tree is not part of the language accepted by the considered automaton, since no rules of the form $record(L) \rightarrow q_L$, where $q_e q_g \in L$, exist.

4 Related Work

Most of the work on XML schema evolution has focused on determining the impact of schema updates on related document validity [8,2,7] and programs/queries [13,7]. Concerning document adaptations, the need of supporting both automatic and user-defined adaptations is advocated in [2], but no static analysis is performed on user-defined adaptations, so that run-time revalidation of adapted documents is needed.

Concerning related work on static analysis, the main formalization of schema updates is represented by [1], where the authors take into account a subset of XQUF which deals with structural conditions imposed by tags only, disregarding attributes.

Type inference for XQUF, without approximations, is not always possible. This follows from the fact that modifications that can be produced using this language can lead to non regular schemas, that cannot be captured with existing schema languages for XML. This is the reason why [1], as well as [16], computes an over-approximation of the type set that results from the updates. In our work, on the contrary, to produce an exact computation we need to cover a smaller subset of XQUF's features. In [1], indeed, XPath's axes can be used to query and select nodes, allowing to mix selectivity conditions with positional constraints with the request to satisfy a given pattern. In our work, as well as in [16] and [9], only update primitives have been considered, thus excluding complex expressions such as "for loops" and "if statements", based on the result of a query. These expressions, anyway, can be translated into a sequence of primitive operations.[1]

Macro Tree Transducers (MTT) [11] can also be applied to model XML updates as in a Monadic Second-Order logic (MSO)-based transformation language, that does not only generalize XPath, XQuery and XSLT, but can also be simulated using macro tree transducers. The composition of MTT and their property of preserving recognizability for the calculation of their inverses 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 whether the input type has some intersection with the pre-image. Their system, as ours, is exact and does not approximate the calculation, but our more specific approach, focused on a specific set of transformations, allows for a simpler (and more efficient) implementation.

5 Conclusions

In the paper we have presented an XML schema update framework relying on the use of hedge automata transformations for the static analysis of document adaptations. A Java prototype, based on the LETHAL Library, of the framework have been realized and tested on the XML XMark benchmark. The execution times of the automaton computation and of the inclusion test on the considered bechmarks are negligible (less than 1s), showing the potential of our proposal for a practical usage as a support for static analysis of XML updates. We plan to integrate this prototype of the framework with *EXup* ([2]) or other suitable tools as future work.

Other possible directions for extending the current work can be devised. Node selection constraints for update operations could be refined, for example using XPath axes and the other features offered by XQUF. As discussed in the paper, this would lead to approximate rather than exact analysis techniques. Support for commutative trees, in which the order of the children of a node is irrelevant, could be added. This feature would allow the formalization of the *all* and *interleave* constructs of XML Schema [18] and Relax NG [3], respectively. Sheaves Automata, introduced in [5], are able to recognize commutative trees and have an expressivity strictly greater than the HA considered in this work. The applicability of these automata in our framework needs to be investigated.

[1] The interested reader could refer to [1] (Section "Semantics"), where a translation of XQUF update expressions into a pending update list, made only of primitive operations, is provided, according to the W3C specification [17].

References

1. Benedikt, M., Cheney, J.: Semantics, Types and Effects for XML Updates. In: Gardner, P., Geerts, F. (eds.) DBPL 2009. LNCS, vol. 5708, pp. 1–17. Springer, Heidelberg (2009)
2. Cavalieri, F., Guerrini, G., Mesiti, M.: Updating XML Schemas and Associated Documents through Exup. In: Proc. of the 27th International Conference on Data Engineering, pp. 1320–1323 (2011)
3. Clark, J., Murata, M.: RELAX NG Specification (2001), `http://www.relaxng.org/spec-20011203.html`
4. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), `http://www.grappa.univ-lille3.fr/tata` (release October 12, 2007)
5. Dal-Zilio, S., Lugiez, D.: XML Schema, Tree Logic and Sheaves Automata. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 246–263. Springer, Heidelberg (2003)
6. Genest, B., Muscholl, A., Serre, O., Zeitoun, M.: Tree Pattern Rewriting Systems. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 332–346. Springer, Heidelberg (2008)
7. Geneves, P., Layaiada, N., Quint, V.: Impact of XML Schema Evolution. ACM Trans. Internet Technol. 11(1) (2011)
8. Guerrini, G., Mesiti, M., Sorrenti, M.: XML Schema Evolution: Incremental Validation and Efficient Document Adaptation. In: 5th International XML Database Symposium on Database and XML Technologies, pp. 92–106 (2007)
9. Jacquemard, F., Rusinowitch, M.: Formal Verification of XML Updates and Access Control Policies (May 2010)
10. Liu, Z., Natarajan, S., He, B., Hsiao, H., Chen, Y.: Cods: Evolving data efficiently and scalably in column oriented databases. PVLDB 3(2), 1521–1524 (2010)
11. Maneth, S., Berlea, A., Perst, T., Seidl, H.: XML type checking with macro tree transducers. In: PODS, pp. 283–294 (2005)
12. Moon, H., Curino, C., Deutsch, A., Hou, C., Zaniolo, C.: Managing and querying transaction-time databases under schema evolution. PVLDB 1(1), 882–895 (2008)
13. Moro, M., Malaika, S., Lim, L.: Preserving xml queries during schema evolution. In: WWW, pp. 1341–1342 (2007)
14. Murata, M., Lee, D., Mani, M., Kawaguchi, K.: Taxonomy of XML schema languages using formal language theory. ACM Trans. Internet Technol. 5(4), 660–704 (2005)
15. Solimando, A., Delzanno, G., Guerrini, G.: Static Analysis of XML Document Adaptations through Hedge Automata. Technical Report DISI-TR-11-08 (2011)
16. Touili, T.: Computing Transitive Closures of Hedge Transformations. In: Proc. 1st Int. Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS 2007). eWIC Series. British Computer Society (2007)
17. Chamberlin, D., Dyck, M., Florescu, D., Melton, J., Robie, J., Simon, J.: W3C. XQuery Update Facility 1.0 (2009), `http://www.w3.org/TR/2009/CR-xquery-update-10-20090609/`
18. Walmsley, P., Fallside, D.C.: W3C. XML Schema Part 0: Primer Second Edition (2004), `http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/`