Evolving Schemas for Streaming XML

Maryam Shoaran and Alex Thomo

University of Victoria, Victoria, Canada
{maryam,thomo}@cs.uvic.ca

Abstract. In this paper we model schema evolution for XML by defining formal language operators on Visibly Pushdown Languages (VPLs). Our goal is to provide a framework for efficient validation of streaming XML in the realistic setting where the schemas of the exchanging parties evolve and thus diverge from one another. We show that Visibly Pushdown Languages are closed under the defined language operators and this enables us to expand the schemas (for XML) in order to account for flexible or constrained evolution.

1 Introduction

The ubiquitous theme in the modern theory of software systems is that evolution is unavoidable in real-world systems. The force of this fact is increasingly prominent today when software systems have numerous online interconnections with other systems and are more than ever under the user pressure for new changes and enhancements. It is often noted that no system can survive without being agile and open to change.

In this paper we propose ways to evolve schemas for XML in an online, streaming setting. As XML is by now the omnipresent standard for representing data and documents on the Web, there is a pressing need for having the ability to smoothly adapt schemas for XML to deal with changes to business requirements, and exchange standards.

One important use of schemas for XML is the validation of documents, which is checking whether or not a document conforms to a given schema. Notably, the validation is the basis of any application involving data-exchange between two or more parties.

Now in a scenario where the schemas of the exchanging parties possibly diverge from each other due to various changing business requirements, we need to expand the schemas making them more "tolerant" against incoming XML documents. To illustrate, suppose that there are parties exchanging patient records such as for example:

```
<record>
  <hospital> Victoria General </hospital>
  <patient>
    <name> Smith Brown </name>
    <address> 353 Douglas Str
      <phone> 250-234-5678 </phone>
    </address>
    <test> Complete blood count </test>
  </patient>
</record>
```

Suppose now that some sending party decides to suppress sending addresses containing the patient's phone number, apparently for privacy reasons. What we want is the system to adapt and continue to function in the face of this change. Namely, the schemas of the receiving parties need to evolve and be tolerant against the change. If we consider the XML documents as nested words (strings), and the schemas as languages of such words, then what we need is to expand the schemas with new words obtained from the original ones after deleting all the subwords of the form `<address>`w_1`<phone>`w_2`</phone>`w_3`</address>`, where w_1, w_2, and w_3 are properly nested words. Evidently, words `<address>`w_1`<phone>` w_2`</phone>`w_3`</address>` form a language, say D, and the problem becomes that of "deleting D from a schema language L." Similar arguments can also be made for expanding schemas by "inserting a language I into a schema language L."

Interestingly, language deletions and insertions have been studied as operators for regular languages in representing biological computations (cf.[12, 6]). In this paper, we investigate instead the deletion and insertion operators as means for evolving languages of nested words capturing the common schema formalism for XML.

We, also consider constrained variants of these language operators. Specifically, we provide means for specifying that we want to allow an operation to apply only at certain elements of the XML documents. For instance, in our example, we could specify that the above deletion is allowed to take place only inside the `patient` element and not anywhere else.

Schemas for XML. When it comes to XML schema specifications, the most popular ones are Document Type Definition (DTD), XML Schema ([19]) and Relax NG ([4]). Notably, all these schema formalisms can be captured by Extended Document Type Definitions (EDTDs) (cf. [14, 15, 17, 5]). It is well known that the tree languages specified by EDTDs coincide with (unranked) regular tree languages (cf. [5]).

In this paper, we will represent XML schemas by Visibly Pushdown Automata (VPAs) introduced in [1]. VPAs are in essence pushdown automata, whose push or pop mode can be determined by looking at the input only (hence their name). VPAs recognize Visibly Pushdown Languages (VPLs), which form a well-behaved and robust family of context-free languages. VPLs enjoy useful closure properties and several important problems for them are decidable. Furthermore, VPLs have been shown to coincide with the class of (word-encoded) regular tree languages, i.e. VPAs are equivalent in power with EDTDs. Recent work [13] has also shown that EDTDs can be directly compiled into VPAs.

Now, the validation problem reduces to the problem of accepting or rejecting the XML document (string) using a VPA built for the given schema. Notably, a VPA accepts or rejects an XML document without building a tree representation for it, and this is a clear advantage in a streaming setting, where transforming and storing the XML into a tree representation is a luxury we do not have.

Another reason for preferring VPAs over tree automata for XML is that VPAs are often more natural and exponentially more succinct than tree automata when it comes to "semi-formally" specify documents using pattern-based conditions on the global linear order of XML (cf. [2, 24]).

Also, considering the schemas for XML as word languages opens the way for a natural extension of deletion and insertion operations, thus making the schemas evolve in a similar spirit to biological computing artifacts.

We show that the deletion and insertion operations can be efficiently computed for VPLs, and furthermore they can be combined with useful constraints determining the scope of their applications.

Contributions. More specifically, our contributions in this paper are as follows.

1. We show that VPLs are closed under the language operations of deletion and insertion. This is in contrast to Context-Free Languages which are not closed under deletion, but only under insertion.
2. We introduce the extended operations of k-bounded deletion and insertion which allow the deletion and insertion, respectively, of k words in parallel. It is exactly these operations that are practical to use for evolving schemas for XML documents containing (or need to contain) multiple occurrences of words to be deleted (or inserted). For instance, a patient record might contain not only his/her own address, but the also the doctor's address, and all these addresses might need to be deleted. We show that the VPLs are closed under these extended operations as well.
3. We present an algorithm, which, given a schema VPL L, two sets $\mathcal{D}$ and $\mathcal{I}$ of allowed language deletions and insertions, respectively, and a positive integer k, produces a (succinctly represented) expanded language L' by applying not more than k operations in *parallel* from $\mathcal{D} \cup \mathcal{I}$ on L. Language L' contains *all* the possible "k-evolution" of L using operations from $\mathcal{D} \cup \mathcal{I}$. The difference from the k-bounded deletion and insertion is that now we allow these operations to be intermixed together.
4. We enhance the deletion and insertion operations by constrains that specify the allowed scope of the operations. We express these constraints by using XPath expressions which select the XML elements of interest. We present an algorithm which computes the "k-evolution" of a given schema VPL L under this constrained setting. The challenge is to be able to first mark non intrusively all the candidate spots for applying the operations, and then apply them. This is because applying an operation could possibly change the structure of the words and thus harm matching of the other constraints.

Organization. The rest of the paper is organized as follows. In Section 2 we discuss related work. Section 3 reviews VPAs and VPTs (Visibly Pushdown Transducers). In Section 4 we study the deletion operation for VPLs. The k-bounded deletion for VPLs is also introduced there. In Section 5 we study the insertion operation for VPLs. The k-bounded insertion for VPLs is also introduced there. In Section 6 we present an algorithm for evolving a schema VPL by applying at most k language operations in parallel. In Section 7 we introduce constrained operations, and in Section 8 we present an algorithm to evolve schema VPLs using such operations. Finally, Section 9 concludes the paper.

2 Related Work

The first to propose using pushdown automata for validating streaming XML are Segufin and Vianu in [18]. The notion of auxiliary space for validating streaming XML is also defined in this work. Auxiliary space is the stack space needed to validate an XML document and is proportional to the depth of the document.

VPLs and their recognizing devices, VPAs, are introduced in [1]. In [13], it is argued that VPAs are the apt device for the validation of streaming XML and a direct construction is given for going from EDTDs to equivalent VPAs.

The problem of error-tolerant validation has been studied in several works (cf. [3, 21, 7, 23]). These works use edit operations to modify the XML and possibly make it fit the schema. The difference of our work from these works is that we consider language operations rather than edit operations on XML trees. We note that performing edit operations might not correspond naturally to the user intention of changing an XML document or schema. For example to delete a complex `address` element we need several delete edit operations rather than just one language operation as in our setting. The latter, we believe, corresponds better to a user intention for deleting such an element in one-shot. Furthermore, with our language operations, the user is given the opportunity to specify the structure of the elements to be deleted or inserted, which as illustrated in the Introduction, is useful in practice.

Using edit operations on regular languages is studied in [9–11]. They revolve around the problem of finding paths in graph databases that approximately spell words in a given regular language.

The language operations of deletion and insertion are studied in [12] for regular and context free languages. As shown there, the regular languages are closed under deletion and insertion while context free languages are not closed under deletion, but closed under insertion.

Visibly Pushdown Transducers (VPTs) are introduced in [23] and [16]. The latter showed that VPLs are closed under transductions of VPTs which refrain from erasing open or close symbols. In this paper we will use this class of VPTs for some auxiliary marking operations on VPLs.

3 Visibly Pushdown Automata and Transducers

3.1 Visibly Pushdown Automata

VPAs were introduced in [1] and are a special case of pushdown automata. Their alphabet is partitioned into three disjoint sets of call, return and local symbols, and their push or pop behavior is determined by the consumed symbol. Specifically, while scanning the input, when a call symbol is read, the automaton pushes one stack symbol onto the stack; when a return symbol is read, the automaton pops off the top of the stack; and when a local symbol is read, the automaton only moves its control state.

Formally, a *visibly pushdown automaton* (VPA) A is a 6-tuple $(Q, (\Sigma, f), \Gamma, \tau, q_0, F)$, where

1. Q is a finite set of states.
2. – Σ is the alphabet partitioned into the (sub) alphabets Σ_c, Σ_l and Σ_r of call, local and return symbols respectively.
   – f is a one-to-one mapping $\Sigma_c \to \Sigma_r$. We denote $f(a)$, where $a \in \Sigma_c$, by $\bar{a}$, which is in Σ_r.[1]
3. Γ is a finite stack alphabet that (besides other symbols) contains a special “bottom-of-the-stack” symbol $\perp$.
4. q_0 is the initial state.
5. F is the set of final states.
6. $\tau = \tau_c \cup \tau_r \cup \tau_l \cup \tau_\epsilon$ is the transition relation and τ_c, τ_l, τ_r and τ_ϵ are as follows.
   – $\tau_c \subseteq Q \times \Sigma_c \times Q \times (\Gamma \setminus \perp)$
   – $\tau_r \subseteq Q \times \Sigma_r \times \Gamma \times Q$
   – $\tau_l \subseteq Q \times \Sigma_l \times Q$
   – $\tau_\epsilon \subseteq Q \times \{\epsilon\} \times Q$

We note that the ϵ-transitions do not affect the stack and they behave like in an NFA. They can be easily removed by an ϵ-removal procedure similar to the standard one for NFAs. However, we consider ϵ-transitions as they make expressing certain constructions more convenient.

A run of VPA A on word $w = x_0 \ldots x_{k-1}$ is a sequence $\rho = (q_i, \sigma_i), \ldots, (q_{i+k}, \sigma_{i+k})$, where $q_{i+j} \in Q$, $\sigma_{i+j} \in (\Gamma \setminus \{\perp\})^* \cdot \{\perp\}$, and for every $0 \leq j \leq k-1$, the following holds:

– If x_j is a call symbol, then for some $\gamma \in \Gamma$, $(q_{i+j}, x_j, q_{i+j+1}, \gamma) \in \tau_c$ and $\sigma_{i+j+1} = \gamma \cdot \sigma_{i+j}$ (Push γ).
– If x_j is a return symbol, then for some $\gamma \in \Gamma \setminus \{\perp\}$, $(q_{i+j}, x_j, \gamma, q_{i+j+1}) \in \tau_r$ and $\sigma_{i+j} = \gamma \cdot \sigma_{i+j+1}$ (Pop γ).
– If x_j is a local symbol, then $(q_{i+j}, x_j, q_{i+j+1}) \in \tau_l$ and $\sigma_{i+j+1} = \sigma_{i+j}$.

[1] When referring to arbitrary elements of Σ_r, we will use $\bar{a}, \bar{b}, \ldots$ in order to emphasize that these elements correspond to $a, b, \ldots$ elements of Σ_c.

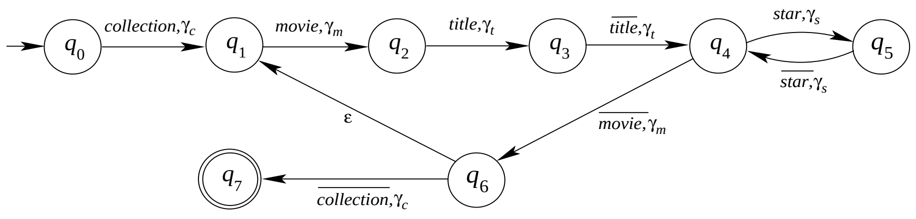
A run is *accepting* if $q_i = q_0$, $q_{i+k} \in F$, and $\sigma_{i+k} = \perp$. A word w is accepted by a VPA if there is an accepting run in the VPA which spells w. A language L is a *visibly pushdown language* (VPL) if there exists a VPA that accepts all and only the words in L. The VPL accepted by a VPA A is denoted by $L(A)$.

When reasoning about XML structure and validity, the local symbols are not important, and thus, we consider the languages of XML schemas as VPLs on the alphabet $\Sigma_c \cup \Sigma_r$. Furthermore, we note that here, we are asking for an empty stack in the end of an accepting run because we are interested in VPLs of properly nested words.

Example 1. Suppose that we want to build a VPA accepting XML documents about movie collections. Such documents will have a *collection* element nesting any number of *movie* elements in them. Each *movie* element will nest a *title* element and any number of *star* elements. A VPA accepting well-formed documents of this structure is $A = (Q, (\Sigma, f), \Gamma, \tau, q_0, F)$, where

$$\begin{aligned}
&Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\},\\
&\Sigma = \Sigma_c \cup \Sigma_r =\\
&\qquad \{\mathit{collection}, \mathit{movie}, \mathit{title}, \mathit{star}\} \cup \{\overline{\mathit{collection}}, \overline{\mathit{movie}}, \overline{\mathit{title}}, \overline{\mathit{star}}\},\\
&\qquad f \text{ maps the } \Sigma_c \text{ elements into their ``bar''-ed counterparts in } \Sigma_r,\\
&\Gamma = \{\gamma_c, \gamma_m, \gamma_t, \gamma_s\} \cup \{\perp\},\\
&F = \{q_7\},\\
&\tau = \{(q_0, \mathit{collection}, q_1, \gamma_c), (q_1, \mathit{movie}, q_2, \gamma_m), (q_2, \mathit{title}, q_3, \gamma_t),\\
&\qquad (q_3, \overline{\mathit{title}}, \gamma_t, q_4), (q_4, \mathit{star}, q_5, \gamma_s), (q_5, \overline{\mathit{star}}, \gamma_s, q_4),\\
&\qquad (q_4, \overline{\mathit{movie}}, \gamma_m, q_6), (q_6, \overline{\mathit{collection}}, \gamma_c, q_7), (q_6, \epsilon, q_1)\}.
\end{aligned}$$

We show this VPA in Fig. 1.



q_0
collection,γ_c
q_1
movie,γ_m
q_2
title,γ_t
q_3
$\overline{\text{title}}$,γ_t
q_4
star,γ_s
q_5
$\overline{\text{star}}$,γ_s
ε
$\overline{\text{movie}}$,γ_m
q_7
q_6
$\overline{\text{collection}}$,$\gamma_c$

Fig. 1. Example of a VPA.

Processing a document with a VPA. As explained in [13], given a schema specification VPA $A = (Q, (\Sigma, f), \Gamma, \tau, q_0, F)$, the (exact) validation of an XML document (word) w amounts to accepting or rejecting w using A.

Intersection with Regular Languages. It can be shown that VPLs are closed under the intersection with regular languages. The construction is similar to the one showing closure of CFLs under intersection with regular languages. Formally we have

Theorem 1. *Let L be a VPL and R a regular language. Then, $L \cap R$ is a VPL.*

Proof Sketch. Let $A = (Q, (\Sigma, f), \Gamma, \tau^A, q_0, F)$ be an ϵ-free VPA for L, and $B = (P, \Sigma, \tau^B, p_0, G)$ an ϵ-free NFA for R. Now, language $L \cap R$ is accepted by the product VPA $C = (Q \times P, (\Sigma, f), \Gamma, \tau^C, q_0, F \times G)$, where

$$\begin{aligned}\tau^C = \{&((q,p), a, (q',p'), \gamma) : (q, a, q', \gamma) \in \tau^A \text{ and } (p, a, p') \in \tau^B\} \cup \\ \{&((q,p), \bar{a}, \gamma, (q',p')) : (q, \bar{a}, \gamma, q') \in \tau^A \text{ and } (p, a, p') \in \tau^B\} \cup \\ \{&((q,p), l, (q',p')) : (q, l, q') \in \tau^A \text{ and } (p, l, p') \in \tau^B\}.\end{aligned}$$

□

The Language of All Properly Nested Words. In our constructions we will often use the language of all properly nested words which we denote by PN. All the VPLs of properly nested words are subsets of it. We consider the empty word ϵ to also be in PN.

3.2 Visibly Pushdown Transducers

A *visibly pushdown transducer* (VPT) T is a 7-tuple $(P, (I, f), (O, g), \Gamma, \tau, p_0, F)$, where

1. P is a finite set of states.
2. – I is the input alphabet partitioned into the (sub) alphabets I_c and I_r of input call and return symbols.
   – f is a one-to-one mapping $I_c \to I_r$. We denote $f(a)$, where $a \in I_c$, by $\bar{a}$.
3. – O is the output alphabet partitioned into the (sub) alphabets O_c and O_r of output call and return symbols respectively.
   – g is a one-to-one mapping $O_c \to O_r$. We denote $g(b)$, where $b \in O_c$, by $\bar{b}$.
4. Γ is a finite stack alphabet that (besides other symbols) contains a special "bottom-of-the-stack" symbol $\bot$.
5. p_0 is the initial state.
6. F is the set of final states.
7. $\tau = \tau_c \cup \tau_r \cup \tau_\epsilon$, where
   – $\tau_c \subseteq P \times I_c \times O_c \times P \times \Gamma$
   – $\tau_r \subseteq P \times I_r \times O_r \times \Gamma \times P$
   – $\tau_\epsilon \subseteq P \times \{\epsilon\} \times \{\epsilon\} \times P$.

We define an *accepting run* for T similarly as for VPAs. Now, given a word $u \in I^*$, we say that a word $w \in O^*$ is an *output of T for u* if there exists an accepting run in T spelling u as input and w as output.[2]

[2] In other words, we get u and w when concatenating the transitions' input and output components respectively.

A transducer T might produce more than one output for a given word u. We denote the set of all outputs of T for u by $T(u)$. For a language $L \subseteq I^*$, we define the *image of L through T* as $T(L) = \bigcup_{u \in L} T(u)$.

We note that in our definition of VPTs we disallow transitions which transduce a call or return symbol to ϵ. As [16] showed, VPLs are closed under the transductions of such non-erasing VPTs.

4 Language Deletion

In this section we present the language operation of deletion and show that VPLs are closed under this operation.

Let L and D be languages on Σ. The *deletion of D from L*, denoted by $L \longrightarrow D$, removes from the words of L one occurrence of some word in D. For example if $L = \{abcd, ab\}$ and $D = \{bc, cd, a\}$, then $L \longrightarrow D = \{ad, ab, bcd, b\}$.

Formally, the *deletion of D from L* is defined as:

$$L \longrightarrow D = \{w_1 w_2 : w_1 v w_2 \in L \text{ and } v \in D\}.$$

Kari (in [12]) showed that the regular languages are closed under deletion, whereas context-free languages are not. We show here that VPLs are closed under deletion.

Theorem 2. *If L and D are VPLs over Σ, then $L \longrightarrow D$ is a VPL as well.*

Proof.
Construction. Let $A = (Q, (\Sigma, f), \Gamma, \tau, q_0, F)$, where $Q = \{q_0, \ldots, q_{n-1}\}$, be a VPA that accepts L. For every two states q_i and q_j in Q define the VPA

$$A_{ij} = (Q, (\Sigma, f), \Gamma, \tau, q_i, \{q_j\}),$$

which is the same as A, but with initial and final states being q_i and q_j, respectively. The language $L(A_{ij})$ (which we also denote by L_{ij}) is a VPL for each $q_i, q_j \in Q$. Consider now the VPA $A' = (Q, (\Sigma', f), \Gamma, \tau', q_0, F)$, with

1. $\Sigma' = \Sigma \cup \{\dagger\}$, where $\dagger$ is a fresh local symbol,
2. $\tau'_c = \tau_c$,
3. $\tau'_r = \tau_r$, and
4. $\tau'_l = \{(q_i, \dagger, q_j) : q_i, q_j \in Q \text{ and } L_{ij} \cap D \neq \emptyset\}$.

We then define language $L' = L(A') \cap (\Sigma^* \cdot \{\dagger\} \cdot \Sigma^*)$. This intersection extracts from $L(A')$ all the words marked by one $\dagger$. Such words are derived from the words of L containing some properly nested subword which is a word in D.

Now we obtain VPL L'' by substituting ϵ for $\dagger$ in L'. This is achieved by replacing the *local* $\dagger$ transitions, in the VPA for L', by ϵ transitions. We have that

Lemma 1. $L \longrightarrow D = L''$.

Proof. "$\subseteq$". Let $w \in L \longrightarrow D$. There exists $u \in L$, $v \in D$ such that $u = w_1 v w_2$ and $w = w_1 w_2$. Hence, there exists an accepting run of A for u:

$$\rho_u = (q_0, \sigma_0), \ldots, (q_i, \sigma_i), \ldots, (q_j, \sigma_j), \ldots, (q_f, \sigma_f),$$

where $(q_0, \sigma_0), \ldots, (q_i, \sigma_i)$ is a sub-run for w_1, $(q_i, \sigma_i), \ldots, (q_j, \sigma_j)$ is a sub-run for v, $(q_j, \sigma_j), \ldots, (q_f, \sigma_f)$ is a sub-run for w_2, and $q_f \in F$. As $v \in D$ is a properly nested word, we have that $\sigma_i = \sigma_j$.

Since $v \in L_{ij} \cap D$, the transition $(q_i, \dagger, q_j)$ exists in τ' and we have the following run in A':

$$\rho = (q_0, \sigma_0), \ldots, (q_i, \sigma_i), (q_j, \sigma_j), \ldots, (q_f, \sigma_f),$$

where the sub-run $(q_i, \sigma_i), (q_j, \sigma_j)$ reads symbol $\dagger$ and $\sigma_i = \sigma_j$ since $\dagger$ is a local symbol. It is a key point that $\sigma_i = \sigma_j$ in both of the sub-runs $(q_i, \sigma_i), \ldots, (q_j, \sigma_j)$ for v and $(q_i, \sigma_i), (q_j, \sigma_j)$ for $\dagger$. Thus, $w_1 \dagger w_2 \in L(A')$ and also $w_1 \dagger w_2 \in L' = L(A') \cap (\Sigma^* \cdot \{\dagger\} \cdot \Sigma^*)$. This proves that $w = w_1 w_2 \in L''$.

"$\supseteq$". Let $w \in L''$. As such, there exists a word $w' \in L' = L(A') \cap (\Sigma^*.\{\dagger\}.\Sigma^*)$ which is of the form $w_1 \dagger w_2$, where $w_1, w_2 \in \Sigma^*$. For w' there exists an accepting run in A':

$$\rho_{w'} = (q_0, \sigma_0), \ldots, (q_i, \sigma_i), (q_j, \sigma_j), \ldots, (q_f, \sigma_f),$$

where $(q_0, \sigma_0), \ldots, (q_i, \sigma_i)$ is a sub-run for w_1, $(q_j, \sigma_j), \ldots, (q_f, \sigma_f)$ is a sub-run for w_2, and $(q_i, \sigma_i), (q_j, \sigma_j)$ is a sub-run for $\dagger$ with $\sigma_i = \sigma_j$ (since $\dagger$ is a local symbol). This run indicates the existence of transition $(q_i, \dagger, q_j)$ in A', which means $L_{ij} \cap D \neq \emptyset$. This also implies that there exists some properly nested word $v \in D$ corresponding to a run $(q_i, \sigma_i), \ldots, (q_j, \sigma_j)$ in automaton A with $\sigma_i = \sigma_j$.

Since w' contains only one $\dagger$ symbol, there is only one transition from $\tau' \setminus \tau$ that has been traversed in run $\rho_{w'}$. Now it can be concluded that the following accepting run exists for A:

$$\rho = (q_0, \sigma_0), \ldots, (q_i, \sigma_i), \ldots, (q_j, \sigma_j), \ldots, (q_f, \sigma_f),$$

which indicates that $w_1 v w_2 \in L_1$. As $v \in D$, we have that $w = w_1 w_2 \in (w_1 v w_2 \longrightarrow v) \subseteq (L \longrightarrow D)$,
i.e. $w \in L \longrightarrow D$. □

Finally, the claim of the theorem follows from the above lemma and the fact that the visibly pushdown languages are closed under intersection. □

Based on the construction of the above theorem we have that

Theorem 3. *Computing $L \longrightarrow D$ can be done in PTIME.*

***k*-Bounded Deletion.** We can also define a variant of deletion which allows for up to k deletions where k is a given positive integer. This is useful when we want to evolve a given schema language L by allowing the words of a given language

D to be deleted (in parallel) from k-locations in the words of L rather than just one.

Let L and D be languages on Σ. The *k-bounded deletion of D from L* is defined as:

$$L \xrightarrow{\leq k} D = \{w_1w_2 \ldots w_kw_{k+1} : w_1v_1w_2 \ldots w_kv_kw_{k+1} \in L \text{ and } v_1, \ldots, v_k \in D\}.$$

Observe that $L \xrightarrow{\leq k} D$ cannot be obtained by iterative applications of k single deletions. For example, assume $L = \{abcc\bar{c}\bar{b}\bar{a}\}$ and $D = \{b\bar{b}, c\bar{c}\}$. The language resulting from 2-bounded deletion of D from L is $\{ab\bar{b}\bar{a}\}$. On the other hand, by applying 2 successive deletions of D from L we have $(L \longrightarrow D) \longrightarrow D = \{a\bar{a}\}$.

We show now that VPLs are closed under the k-bounded deletion.

Theorem 4. *If L and D are VPLs over Σ, then $L \xrightarrow{\leq k} D$ is a VPL as well.*

Proof sketch.
Construction. Let $A = (Q, (\Sigma, f), \Gamma, \tau, q_0, F)$, where $Q = \{q_0, \ldots, q_{n-1}\}$, be a VPA that accepts L. We build a VPA A' as explained in the proof of Theorem 2. Now we set

$$L' = L(A') \cap \bigcup_{1 \leq h \leq k} (\Sigma^* \cdot \{\dagger\} \cdot \Sigma^*)^h.$$

This intersection extracts from $L(A')$ all the words marked by up to k $\dagger$'s. Such words are derived from the words of L containing properly nested subwords which are words in D.

Now we obtain VPL L'' by substituting symbols $\dagger$ in L' by ϵ. This is achieved by replacing the local $\dagger$ transitions, in the VPA for L', by ϵ transitions. We can show that

Lemma 2. $L \xrightarrow{\leq k} D = L''$.

Based on this lemma the claim of the theorem follows. □

Based on the construction of the above theorem we have that

Theorem 5. *Computing $L \xrightarrow{\leq k} D$ can be done in PTIME if k is constant.*

On the other hand, if k is not constant, then the complexity is pseudo-polynomial in k. This is because of the intersection with $\bigcup_{1 \leq h \leq k} (\Sigma^* \cdot \{\dagger\} \cdot \Sigma^*)^h$ in the proof of Theorem 4.

5 Language Insertion

Before defining the insertion operation, we define the *Insertion-for-Local-Symbol (ILS)* operation. Let $\#$ be a local symbol and L and I be VPLs on $\Sigma_\# = \Sigma \cup \{\#\}$ and Σ, respectively. The *ILS of I into L* is defined as:

$$L \longleftarrow_\# I = \{w_1vw_2 : w_1\#w_2 \in L \text{ and } v \in I\}.$$

Here, we show that VPLs are closed under the ILS operation.

Theorem 6. *If L and I are VPLs over $\Sigma_\#$ and Σ, respectively, then $L \longleftarrow_\# I$ is a VPL over $\Sigma_\#$.*

Proof.
Construction. Let $A = (Q, (\Sigma_\#, f), \Gamma, \tau, q_0, F)$ and $A^I = (P, (\Sigma, f), \Gamma, \tau^I, p_0, F^I)$ be VPA's accepting L and I, respectively. Starting with A, we construct VPA A' by adding for each (local) transition $(q_i, \#, q_j)$ a fresh copy of A^I connected with q_i and q_j with the following new transitions,

$$\{(q_i, \diamond, p_0)\} \cup \{(p_f, \epsilon, q_j) : p_f \in F^I\},$$

where $\diamond$ is a new local symbol.

In essence, A' will accept all the words accepted by A with an arbitrary number of # replaced by the words of language $\{\diamond\} \cdot I$. In order to have only one # replacement, we construct

$$L' = L(A') \cap (\Sigma_\#^* \cdot \{\diamond\} \cdot \Sigma_\#^*).$$

Then, we replace the single $\diamond$ by ϵ in L', achieved by changing the $(\_, \diamond, \_)$ transitions to $(\_, \epsilon, \_)$. Let L'' be the language obtained in this way. Now, we show that:

Lemma 3. $L \longleftarrow_\# I = L''$.

Proof. Let A'' be the VPA of L''.

"$\subseteq$". Let $w \in L \longleftarrow_\# I$. There exists $u \in L$, $v \in I$ such that $u = w_1 \# w_2$ and $w = w_1 v w_2$. Hence, there exists an accepting run:

$$\rho_u = (q_0, \sigma_0), \ldots, (q_i, \sigma_i), (q_j, \sigma_j), \ldots, (q_f, \sigma_f),$$

in A for u, where $(q_0, \sigma_0), \ldots, (q_i, \sigma_i)$ is a sub-run for w_1, $(q_i, \sigma_i), (q_j, \sigma_j)$ is a sub-run for symbol #, $(q_j, \sigma_j), \ldots, (q_f, \sigma_f)$ is a sub-run for w_2, and $q_f \in F$. As # is a local symbol, we have that $\sigma_i = \sigma_j$.

According to the construction in Theorem 6, we now have some accepting runs in A'' of the form:

$$\rho = (q_0, \sigma_0), \ldots, (q_i, \sigma_i), (p_0, \sigma_{i+1}), \ldots, (p_f, \sigma_{j-1}), (q_j, \sigma_j), \ldots, (q_f, \sigma_f),$$

where $p_f \in F^I$ and so $(p_0, \sigma_{i+1}), \ldots, (p_f, \sigma_{j-1})$ is a sub-run for some properly nested word $v \in I$. Since this sub-run is an accepting run in VPL A^I, we have that $\sigma_{i+1} = \sigma_{j-1}$. The (local) transitions (q_i, ϵ, p_0) and (p_f, ϵ, q_j) implies that $\sigma_i = \sigma_{i+1}$ and $\sigma_{j-1} = \sigma_j$, respectively. Therefore, we have that $\sigma_i = \sigma_j$ in the sub-run $(q_i, \sigma_i), (p_0, \sigma_{i+1}), \ldots, (p_f, \sigma_{j-1}), (q_j, \sigma_j)$ of ρ reading v. This is a key point that shows the sub-run $(q_j, \sigma_j), \ldots, (q_f, \sigma_f)$ in ρ can still read w_2 after reading v. Therefore, $w_1 v w_2 \in L''$.

"$\supseteq$". Let $w \in L''$. For w there exists an accepting run in A'':

$$\rho_w = (q_0, \sigma_0), \ldots, (q_i, \sigma_i), (p_0, \sigma_{i+1}), \ldots, (p_f, \sigma_{j-1}), (q_j, \sigma_j), \ldots, (q_f, \sigma_f),$$

where $q_f \in F$ and $p_f \in F^I$. The existence of sub-run $(p_0, \sigma_{i+1}), \ldots, (p_f, \sigma_{j-1})$ in ρ_w implies that w contains a subword, say v, belonging to I. Hence, w can be written as $w = w_1 v w_2$, where w_1 and w_2 correspond, respectively, to sub-runs $(q_0, \sigma_0), \ldots, (q_i, \sigma_i)$ and $(q_j, \sigma_j), \ldots, (q_f, \sigma_f)$. Since $(p_0, \sigma_{i+1}), \ldots, (p_f, \sigma_{j-1})$ in ρ_w is an accepting run in A^I, we have that $\sigma_{i+1} = \sigma_{j-1}$. Regarding the ϵ transitions (q_i, ϵ, p_0) and (p_f, ϵ, q_j) in A', we conclude $\sigma_i = \sigma_{i+1}$, $\sigma_{j-1} = \sigma_j$, and finally $\sigma_i = \sigma_j$. According to this equality and the way A'' is built, there exists a transition $(q_i, \#, q_j)$ in A, which indicates the existence of the following accepting run in A:

$$\rho = (q_0, \sigma_0), \ldots, (q_i, \sigma_i), (q_j, \sigma_j), \ldots, (q_f, \sigma_f),$$

Hence, $w_1 \# w_2 \in L$. As $v \in I$, we have that $w = w_1 v w_2 \in (w_1 \# w_2 \longleftarrow_{\#} v) \subseteq (L \longleftarrow_{\#} I)$, i.e. $w \in L \longleftarrow_{\#} I$. □

Finally, the claim of the theorem follows from the above lemma and the fact that the visibly pushdown languages are closed under intersection. □

5.1 Insertion

Let L and I be languages on Σ. The *insertion of I into L*, denoted by $L \longleftarrow I$, inserts into the words of L some word in I. For example if $L = \{ab, cd\}$ and $I = \{eg\}$, then $L \longleftarrow I = \{egab, aegb, abeg, egcd, cegd, cdeg\}$.

Formally, the *insertion of I into L* is defined as

$$L \longleftarrow I = \{w_1 v w_2 : w_1 w_2 \in L, \text{ and } v \in I\}.$$

Kari (in [12]) showed that the regular languages and context free languages are closed under insertion. We show here that VPLs are closed under insertion, too.

Theorem 7. *If L and I are VPLs over Σ, then $L \longleftarrow I$ is a VPL as well.*

Proof sketch.
Construction. Let $A = (Q, (\Sigma, f), \Gamma, \tau, q_0, F)$ be a VPA that accepts L. Consider now the VPA $A' = (Q, (\Sigma', f), \Gamma, \tau', q_0, F)$, with

1. $\Sigma' = \Sigma \cup \{\#\}$, where # is a fresh local symbol,
2. $\tau'_c = \tau_c$,
3. $\tau'_r = \tau_r$, and
4. $\tau'_l = \{(q_i, \#, q_i) : q_i \in Q\}$.

We then define language $L' = L(A') \cap (\Sigma^* \cdot \{\#\} \cdot \Sigma^*)$. This intersection extracts from $L(A')$ all the words marked by only *one* #. It can be verified now that

Lemma 4. $L \longleftarrow I = L' \longleftarrow_{\#} I$.

Finally the claim of the theorem follows from the above lemma, Theorem 6, and the fact that VPLs are closed under intersection. □

Based on the constructions of theorems 7 and 6, we have that

Theorem 8. *Computing $L \longleftarrow I$ can be done in PTIME.*

***k*-Bounded Insertion.** We can also define a variant of insertion which allows for up to k insertions where k is a given positive integer. This is useful when we want to evolve a given schema language L by allowing the words of a given language I to be inserted (in parallel) into k-locations in the words of L rather than just one.

Before, we define the k-bounded ILS operation. Let L and I be languages on $\Sigma_{\#}$ and Σ, respectively.

The *k-bounded ILS of I into L* is defined as:

$$L \xleftarrow{\leq k}_{\#} I = \{w_1 v_1 w_2 \ldots w_k v_k w_{k+1} : w_1 \# w_2 \ldots w_k \# w_{k+1} \in L \text{ and } v_1, \ldots, v_k \in I\}.$$

It can be verified that VPLs are closed under the k-bounded ILS operation. Namely, we have

Theorem 9. *If L and I are VPLs over $\Sigma_{\#}$ and Σ, respectively, then $L \xleftarrow{\leq k}_{\#} I$ is a VPL over $\Sigma_{\#}$.*

Now, let L and I be languages on Σ. The *k-bounded insertion of I into L* is defined as:

$$L \xleftarrow{\leq k} I = \{w_1 v_1 w_2 \ldots w_k v_k w_{k+1} : w_1 w_2 \ldots w_k w_{k+1} \in L \text{ and } v_1, \ldots, v_k \in I\}.$$

We show here that VPLs are closed under the operation of k-bounded insertion.

Theorem 10. *If L and I are VPLs over Σ, then $L \xleftarrow{\leq k} I$ is a VPL as well.*

Proof sketch.
Construction. Let $A = (Q, (\Sigma, f), \Gamma, \tau, q_0, F)$, where $Q = \{q_0, \ldots, q_{n-1}\}$, be a VPA that accepts L. We build a VPA A' as explained in the proof of Theorem 7. Then we define language

$$L' = L(A') \cap \bigcup_{1 \leq h \leq k} (\Sigma^* \cdot \{\#\} \cdot \Sigma^*)^h.$$

The above intersection extracts from $L(A')$ all the words marked by one up to k #'s. It can be verified now that

Lemma 5. *$L \xleftarrow{\leq k} I = L' \xleftarrow{\leq k}_{\#} I$.*

From this the claim of the theorem follows. $\square$

Based on theorems 10 and 9, we have that

Theorem 11. *Computing $L \xleftarrow{\leq k} I$ can be done in PTIME if k is constant.*

On the other hand, if k is not constant, then the complexity is pseudo-polynomial in k. This is because of the intersection with $\bigcup_{1 \leq h \leq k} (\Sigma^* \cdot \{\#\} \cdot \Sigma^*)^h$ in the proof of Theorem 10.

6 Transforming a VPL with Language Operations

In practice it is more useful to allow the schema transformation to be achieved by a set of deletion and insertion operations. For example, we can define a set $\mathcal{D} = \{D_1, \ldots, D_m\}$ and $\mathcal{I} = \{I_1, \ldots, I_n\}$ of allowed language deletions and insertions, respectively. With slight abuse of notation we will consider $D_1, \ldots, D_m$ and $I_1, \ldots, I_n$ to also denote their corresponding delete and insert operations, respectively. What we would like now is to apply (in parallel) up to k operations from $\mathcal{D} \cup \mathcal{I}$ on a given schema language L.

For this, given a VPA A for L, we extend the constructions described in the constructions of theorems 2 and 7. Specifically, let VPA $A = (Q, (\Sigma, f), \Gamma, \tau, q_0, F)$ have the states numbered as in Theorem 2. Also, for every two states q_i and q_j in Q define VPA A_{ij} and its accepted language L_{ij} as described in the same theorem. We construct now the VPA $A' = (Q, (\Sigma', f), \Gamma, \tau', q_0, F)$, with

1. $\Sigma' = \Sigma_c \cup \Sigma_r \cup \{\dagger_1, \ldots, \dagger_m\} \cup \{\#_1, \ldots, \#_n\}$
2. $\tau'_c = \tau_c$,
3. $\tau'_r = \tau_r$, and
4. $\tau'_l = \{(q_i, \dagger_x, q_j) : q_i, q_j \in Q \text{ and } L_{ij} \cap D_x \neq \emptyset, \text{ for } 1 \leq x \leq m\} \cup$
   $\{(q_i, \#_y, q_i) : q_i \in Q \text{ and } 1 \leq y \leq n\}$.

VPA A' will accept language L' containing words with an arbitrary number of special local symbols. Each $\dagger_x$ represents a deletion corresponding to D_x, and each $\#_y$ represents an insertion corresponding to I_y. What we want, though, is to extract only those words of L', whose total number of the special symbols is not more than k. For this we construct the following intersection

$$L'' = L' \cap \bigcup_{1 \leq h \leq k} (\Sigma^* \cdot \{\dagger_1, \ldots, \dagger_m, \#_1, \ldots, \#_n\} \cdot \Sigma^*)^h.$$

Language L'' will contain all the words of L' with not more than k special symbols. Then, we obtain language L''' by replacing all the $\dagger_x$ for $1 \leq x \leq m$ by ϵ and performing n (one after the other) k-bounded ILS operations corresponding to each of $I_1, \ldots, I_n$ languages. It can be verified that

Theorem 12. *L''' is the result of applying from one up to k operations from $\mathcal{D} \cup \mathcal{I}$ on L.*

Taking the union $L \cup L'''$ gives us the new expanded schema language.

Based on the above construction, we have that

Theorem 13. *Computing L''' can be done in PTIME if k is constant.*

On the other hand, if k is not constant, then the complexity is pseudo-polynomial in k. This is because of the intersection with $\bigcup_{1 \leq h \leq k} (\Sigma^* \cdot \{\dagger_1, \ldots, \dagger_m, \#_1, \ldots, \#_n\} \cdot \Sigma^*)^h$ for obtaining L''.

7 Constrained Deletions and Insertions

Often we do not like the deletions and insertions to be performed in *unrestricted* places in the words of schemas for XML. Rather we would like them to apply only at certain parts of the words. For instance, taking the example given in the Introduction, one might want to apply operations only within a `patient` XML element.

For this, we assume that there is a given set of constraining rules that specify the conditions under which the operations can be applied on L.

We propose to express the conditions in the form of XPath expressions[3]. The alphabet of the XPath expressions is the set of XML elements corresponding to Σ_c (or Σ_r). Formally this alphabet is $\Sigma_e = \{\tilde{a} : a \in \Sigma_c\}$. As the validation problem considers the structure of XML only, we do not have data values in the XPath expressions.

Definition 1 *A deletion rule is a tuple (π, D), where π is an XPath expression, and D is a VPL.*

Such a rule implies that the words of nested language D can be deleted if they correspond to elements reached by XPath expression π. Of course, such elements need to further satisfy the structure imposed by D. An example of a deletion rule is $(/\tilde{a}/\tilde{b}/\tilde{c}, PN)$. Using this rule we can delete *all* the $\tilde{c}$ elements that are children of $\tilde{b}$ elements which in turn are children of $\tilde{a}$ elements. Surely, specifying $D = PN$ is a useful case in practice. However, we can further qualify the D language to allow only the deletion of those $\tilde{c}$ elements which contain some particular child, say an element $\tilde{d}$. In such a case, we set $D = \{c\} \cdot PN \cdot \{d\} \cdot PN \cdot \{\bar{d}\} \cdot PN \cdot \{\bar{c}\}$.

We denote by $\mathcal{D}$ the set of deletion rules.

Definition 2 *An insertion rule is a tuple (π, I), where π is an XPath expression, and I is a VPL.*

Such a rule implies that the words of nested language I can only be inserted as children of elements reached by XPath expression π. An example of an insertion rule is $(/\tilde{a}/\tilde{b}, PN)$.

We denote by $\mathcal{I}$ the set of insertion rules.

As shown in [8, 22], a unary (Core) XPath query can be represented by a VPA with a single output variable attached to some call transitions. The set of query answers consists of all the elements whose call (open) symbol binds to the variable during an accepting run.

Let π be an XPath expression and A_π a VPA for it. Let τ'_c be the subset of the A_π transitions with the output variable attached to them. It is easy to identify with the help of the stack the corresponding return transitions, τ'_r. Since we are not interested in outputting the result of the XPath queries, but rather just locate the elements of interest, we will assume that instead of having a

[3] We consider in fact unary CoreXPath expressions.

variable attached to the transitions in τ_c', we have them simply marked by a dot $(\dot{\ })$. Also, we will similarly assume the transitions in τ_r' are marked as well.

We consider the alphabet of this VPA to be

$$(\Sigma_c \cup \dot{\Sigma}_c) \cup (\Sigma_r \cup \dot{\Sigma}_r),$$

where $\dot{\Sigma}_c$ and $\dot{\Sigma}_r$ are copies of Σ_c and Σ_r, respectively, with the symbols being marked by $(\dot{\ })$.

When a rule is applied on a given word (XML document), we do not initially perform deletion or insertion on the word, but only color the related symbols. This is due to the fact that other rules can be further applied on the same document, and the XPath expressions for those rules were written considering the original version of that document in the given XML schema.

For example, suppose we have a set of deletion rules $\mathcal{D} = \{(/\tilde{a}/\tilde{b}, PN), (/\tilde{a}, \{\tilde{a}\tilde{b}\})\}$, and a given word $w = ab\bar{b}\bar{a}c\bar{c}$. Both of these rules can be applied on w. But if we apply the first rule and truly delete $b\bar{b}$, then the second rule can no longer be applied on the result word, which is $w' = a\bar{a}c\bar{c}$. On the other hand, only coloring b and $\bar{b}$, does not prevent the second rule from being applied on the word.

In the following we construct coloring VPTs based on the VPAs for the given rules. Then, we apply these VPTs on a schema language L.

7.1 Coloring VPTs for Deletion Rules

Let (π_x, D_x) be a deletion rule in $\mathcal{D}$. For each such rule we choose a distinct red color, r_x. In the following construction, we use alphabets Σ_c, Σ_r, $\Sigma_c^{r_x}$, $\Sigma_r^{r_x}$, where the last two are copies of the first two, respectively, having the symbols colored in r_x. For the sake of the discussion, we will consider the symbols of Σ_c, Σ_r as being colored in black.

Using A_{π_x} we construct the r_x-coloring VPT $T_{\pi_x}^{r_x}$, which has exactly the same states as A_{π_x}, and transitions:

1. (q, a, a, p, γ_a) for (q, a, p, γ_a) in A_{π_x},
2. $(q, \bar{a}, \bar{a}, \gamma_a, p)$ for $(q, \bar{a}, \gamma_a, p)$ in A_{π_x},
3. $(q, a, a^{r_x}, p, \gamma_a^{r_x})$ for $(q, \dot{a}, p, \gamma_{\dot{a}})$ in A_{π_x},
4. $(q, \bar{a}, \bar{a}^{r_x}, \gamma_a^{r_x}, p)$ for $(q, \dot{\bar{a}}, \gamma_{\dot{a}}, p)$ in A_{π_x}.

Intuitively, the "un-marked" transitions of A_{π_x} become "leave-unchanged" transitions in $T_{\pi_x}^{r_x}$, whereas the "marked" transitions of A_{π_x} become "black to red" transitions in $T_{\pi_x}^{r_x}$.

7.2 Coloring VPTs for Insertion Rules

Let (π_y, I_y) be an insertion rule in $\mathcal{I}$. For each such rule we choose a distinct green color, g_y. In the following construction, we use alphabets Σ_c, Σ_r, $\Sigma_c^{g_y}$, $\Sigma_r^{g_y}$, where the last two are copies of the first two, respectively, having the symbols colored in g_y.

Using A_{π_y} we construct the g_y-coloring VPT $T^{g_y}_{\pi_y}$, which has exactly the same states as A_{π_y}, and transitions:

1. (q, a, a, p, γ_a) for (q, a, p, γ_a) in A_{π_y},
2. $(q, \bar{a}, \bar{a}, \gamma_a, p)$ for $(q, \bar{a}, \gamma_a, p)$ in A_{π_y},
3. $(q, a, a^{g_y}, p, \gamma_a^{g_y})$ for $(q, \dot{a}, p, \gamma_{\dot{a}})$ in A_{π_y},
4. $(q, \bar{a}, \bar{a}^{g_y}, \gamma_a^{g_y}, p)$ for $(q, \dot{\bar{a}}, \gamma_{\dot{a}}, p)$ in A_{π_y}.

Intuitively, the "un-marked" transitions of A_{π_y} become "leave-unchanged" transitions in $T^{g_y}_{\pi_y}$, whereas the "marked" transitions of A_{π_x} become "black to green" transitions in $T^{g_y}_{\pi_y}$.

7.3 Color-Tolerant VPTs

Coloring VPTs presented in the two previous subsections can be applied only on black (normal) words. When we want to apply a coloring VPT on a word more than once or when we apply coloring VPTs for deletions and insertions one after the other, the VPTs have to be applicable also to words which have parts already colored. For example, suppose word $w = ab\bar{b}\bar{a}cd\bar{d}\bar{c}$ has the $b\bar{b}$ part already colored in a red (being so ready for deletion). Word w might be needed next to have $d\bar{d}$ colored in a green color (to become ready for an insertion). In order for a coloring VPT for insertion to be able to color $d\bar{d}$ in green, it has to be "color-tolerant" while reading the prefix $ab\bar{b}\bar{a}c$ of w.

Let T_{π_z} be a coloring VPT for a deletion as described in Subsection 7.1. Now we make T_{π_z} color-tolerant by adding the following colored copies of its transitions.

1. $(q, a^{r_x}, a^{r_x}, p, \gamma_a^{r_x})$ and $(q, a^{g_y}, a^{g_y}, p, \gamma_a^{g_y})$, for each transition (q, a, a, p, γ_a), and for every color r_x and g_y.
2. $(q, \bar{a}^{r_x}, \bar{a}^{r_x}, \gamma_a^{r_x}, p)$ and $(q, \bar{a}^{g_y}, \bar{a}^{g_y}, \gamma_a^{g_y}, p)$, for each transition $(q, \bar{a}, \bar{a}, \gamma_a, p)$, and for every color r_x and g_y.
3. $(q, a^{r_x}, a^{r_z}, p, \gamma_a^{r_z})$ and $(q, a^{g_y}, a^{r_z}, p, \gamma_a^{r_z})$, for each transition $(q, a, a^{r_z}, p, \gamma_a^{r_z})$, and for every color $r_x \neq r_z$ and g_y.
4. $(q, \bar{a}^{r_x}, \bar{a}^{r_z}, \gamma_a^{r_z}, p)$ and $(q, \bar{a}^{g_y}, \bar{a}^{r_z}, \gamma_a^{r_z}, p)$, for each transition $(q, \bar{a}, \bar{a}^{r_z}, \gamma_a^{r_z}, p)$, and for every color $r_x \neq r_z$ and g_y.

Finally, we mention that color-tolerant VPTs for insertions can be constructed in a similar way. Specifically, wherever there is an r_z superscript there will be a g_z one.

8 Transforming a VPL with Constrained Operations

Let $\mathcal{D} = \{(\pi_1, D_1), \ldots, (\pi_m, D_m)\}$ and $\mathcal{I} = \{(\pi'_1, I_1), \ldots, (\pi'_n, I_n)\}$ be the sets of rules for the allowed deletions and insertions, respectively. What we want is to apply up to k operations corresponding to the rules in $\mathcal{D} \cup \mathcal{I}$ on a given schema language L.

We start by constructing coloring VPT for each of the rules in $\mathcal{D} \cup \mathcal{I}$, as described in subsections 7.1, 7.2, and 7.3. Next, we transduce L by iteratively applying these VPTs one after the other (in no particular order) k times each. The result of this multiple transduction will be a language that is the same as L but with the words being colored to indicate the allowed places for deletions and insertions. For simplicity let us continue to use L for this colored version of the schema language.

Let A be a VPA for (the colored) L. Let A_{ij}, and its accepted language L_{ij}, be defined as in theorems 2 and 7. Also, let β be a transformation that uncolors words and languages. This transformation can be easily realized by a VPT. Now from A we build VPA A' keeping the same states and transitions, but adding the following transitions labeled by special local symbols.

$$\{(q_i, \dagger_x, q_j) : \beta(L_{ij} \cap (\varSigma_c^{r_x} \cdot PN \cdot \varSigma_r^{r_x})) \cap D_x \neq \emptyset,\ \text{for } 1 \leq x \leq m\} \cup$$
$$\{(q_j, \#_y, q_j) : \text{ there exists a transition } (\_, \_^{g_y}, q_i, \_) \text{ and } L_{ij} \neq \emptyset\}.$$

The first set is for † transitions between the pairs of states connected by properly nested words in D_x. The words of interest in L_{ij} are those with a first and last symbol colored in red (r_x). We determine these words by intersecting with ($\varSigma_c^{r_x} \cdot PN \cdot \varSigma_r^{r_x}$). Then, we apply the β transformation in order to uncolor the resulting language and proceed with intersecting with D_x.

The second set indicates that if there is a call transition $(\_, \_^{g_y}, q_i, \_)$ colored by g_y (due to an insertion rule (π'_y, I_y)) in A, then in A' we have a transition labeled by $\#_y$ added in all of the states q_j reachable from q_i such that A_{ij} accepts a non-empty properly nested language including the empty word.

VPA A' will accept language L' containing words with at most k special symbols of each kind ($\dagger_1, \ldots, \dagger_m, \#_1, \ldots, \#_n$). Each $\dagger_x$ represents a deletion corresponding to D_x, and each $\#_y$ represents an insertion corresponding to I_y. What we want, though, is to extract only those words of L', whose *total* number of the special symbols is not more than k. For this we construct the following intersection

$$L'' = L' \cap \bigcup_{1 \leq h \leq k} (\varSigma^* \cdot \{\dagger_1, \ldots, \dagger_m, \#_1, \ldots, \#_n\} \cdot \varSigma^*)^h.$$

Language L'' will contain all the words of L' with not more than k special symbols. Then, we obtain language L''' by replacing all the $\dagger_x$ for $1 \leq x \leq m$ by ϵ and performing m (one after the other) k-bounded ILS operations corresponding to each of $I_1, \ldots, I_n$ languages. Based on all the above it can be verified that

Theorem 14. *L''' is the result of applying from one up to k constrained operations from $\mathcal{D} \cup \mathcal{I}$ on L.*

Taking the union $L \cup L'''$ gives us the new expanded schema language.

Finally, regarding the complexity we have that

Theorem 15. *The colored L can be computed in $O(\delta^{(m+n)k})$ time, where δ is an upper bound on the size of rule automata.*

Proof Sketch. The claim follows from the fact that each coloring VPT is applied k times, and we have $n + m$ such VPTs. □

We note that in practice, the numbers k, m, and n would typically be small. On the other hand, after having a colored L, obtaining L''' is polynomial.

9 Concluding Remarks

In this paper we proposed modeling the schema evolution for XML by using the language operations of deletion and insertion on VPLs. We showed that the VPLs are well-behaved under these operations and presented constructions for computing the result of the operations. Then, we introduced constrained operations which are arguably more useful in practice. In order to compute the results of constrained operations we developed special techniques (such as VPA coloring) achieving a compatible application of a set of different operations. Based on our techniques, the schema evolution operators can be applied in parallel without harmful interaction.

References

1. Alur, R., and Madhusudan, P. Visibly Pushdown Languages. In *Proc. 36th ACM Symp. on Theory of Computing* (Chicago, Illinois, 13–15 June 2004), pp. 202–211.
2. Alur, R. Marrying Words and Trees. In *Proc. 26th ACM Symp. on Principles of Database Systems* (Beijing, China, 11–13 June 2007), pp. 233–242.
3. Boobna, U., and de Rougemont, M. Correctors for XML Data. In *Proc. 2nd International XML Database Symposium* (Toronto, Canada, 29–30 August 2004), pp. 97–111.
4. Clark, J., and M. Murata, M. RELAX NG Specification. OASIS, December 2001.
5. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., and Tommasi, M. Tree Automata Techniques and Applications. Available on: http://www.grappa.univ-lille3.fr/tata, October 12, 2007.
6. Daley, M., Ibarra, H., O, Kari, K. Closure and decidability properties of some language classes with respect to ciliate bio-operations. *Theor. Comput. Sci.* 306(1–3): 2003, pp. 19–38.
7. Flesca, S., Furfaro, F., Greco, S., and Zumpano, E. Querying and Repairing Inconsistent XML Data. In *Proc. 6th International Conference on Web Information Systems Engineering* (New York, USA, 20–22 November 2005), pp. 175–188.
8. Gauwin, O., Caron, A. C., Niehren, J., and Tison, S. Complexity of Earliest Query Answering with Streaming Tree Automata. *PLAN-X.* San Francisco, Jan 2008 .
9. Grahne, G., and Thomo, A. Approximate Reasoning in Semistructured Data. In *Proc. of the 8th International Workshop on Knowledge Representation meets Databases* (Rome, Italy, 15 September 2001).

10. Grahne, G., and Thomo, A. Query Answering and Containment for Regular Path Queries under Distortions. In *Proc. of 3rd International Symposium on Foundations of Information and Knowledge Systems* (Wilhelmminenburg Castle, Austria, 17–20 February 2004), pp. 98–115.
11. Grahne, G., and Thomo, A. Regular Path Queries under Approximate Semantics. *Ann. Math. Artif. Intell.* 46(1-2): 2006, pp. 165–190.
12. Kari, L. On Insertion and Deletion in Formal Languages. 1991
13. Kumar, V., Madhusudan, P. and Viswanathan, M. Visibly Pushdown Automata for Streaming XML. In *Proc. of Int. Conf. on World Wide Web* (Alberta, Canada, 8–12 May, 2007), pp. 1053–1062.
14. M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.* 5(4): 2005, pp. 660–704.
15. Martens, W., Neven, F., Schwentick, T., Bex, G., J. Expressiveness and complexity of XML Schema. *ACM Trans. Database Syst.* 31(3): 2006, pp. 770–813.
16. Raskin, J.F., and Servais, F. Visibly Pushdown Transducers. In *Proc. 35th International Colloquium on Automata, Languages and Programming* (Reykjavik, Iceland, 7–11 July, 2008), pp. 386–397.
17. Schwentick, T. Automata for XML - A survey. *J. Comput. Syst. Sci.* 73(3): 2007, pp. 289–315.
18. Segoufin, L., and Vianu, V. Validating Streaming XML Documents. In *Proc. 21st ACM Symp. on Principles of Database Systems* (Madison, Wisconsin, 3–5 June 2002), pp. 53–64.
19. Sperberg-McQueen, C., M., and Thomson, H. XML Schema 1.0. http://www.w3.org/XML/Schema, 2005.
20. Staworko, S. Personal Communication, 2008.
21. Staworko, S., and Chomicki, J. Validity-Sensitive Querying of XML Databases. In *Proc. of 2nd International Workshop on Database Technologies for Handling XML Information on the Web, EDBT Workshops* (Munich, Germany, 26–31 March 2006), pp. 164–177.
22. Staworko, S., Filiot, E., and Chomicki, J. Querying Regular Sets of XML Documents. In *Logic in Databases* (Rome, Italy, 19–20 May 2008).
23. Thomo, A., Venkatesh, S., Ye, Y., Y. Visibly Pushdown Transducers for Approximate Validation of Streaming XML. In *Proc. 5th International Symposium on Foundations of Information and Knowledge Systems* (Pisa, Italy, 11–15 February 2008), pp. 219–238.
24. Thomo, A., Venkatesh, S. Rewriting of Visibly Pushdown Languages for XML Data Integration. In *Proc. 17th ACM Conference on Information and Knowledge Management* (Napa Valley, CA, 26–30 October 2008), pp. 521–530.