Visibly Pushdown Transducers for Approximate Validation of Streaming XML

Alex Thomo, S. Venkatesh, and Ying Ying Ye

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

Abstract. Visibly Pushdown Languages (VPLs), recognized by Visibly Pushdown Automata (VPAs), are a nicely behaved family of context-free languages. It has been shown that VPAs are equivalent to Extended Document Type Definitions (EDTDs), and thus, they provide means for elegantly solving various problems on XML. Especially, it has been shown that VPAs are the apt device for streaming XML.
One of the important problems about XML that can be addressed using VPAs is the validation problem in which we need to decide whether an XML document conforms to the specification given by an EDTD. In this paper, we are interested in solving the approximate version of this problem, which is to decide whether an XML document can be modified by a tolerable number of edit operations to yield a valid one with respect to a given EDTD.
For this, we define Visibly Pushdown Transducers (VPTs) that give us the framework for solving this problem under two different semantics for edit operations on XML. While the first semantics is a generalization of edit operations on strings, the second semantics is new and motivated by the special nature of XML documents. Usings VPTs, we give streaming algorithms that solve the problem under both the semantics. These algorithms use storage space that only depends on the size of the EDTD and the number of tolerable errors. Furthermore, they can check approximate validity of an incoming XML document in a single pass over the document, using auxilliary stack space that is proportional to the depth of the XML document.

1 Introduction

The Extensible Markup Language (XML) is the lingua franca for data and document exchange on the Web and used in a variety of applications ranging from collaborative commerce to medical databases. One of the most important problems on XML is the validation of documents against a schema specification typically given by one of the popular schema languages, Document Type Definition (DTD), XML Schema ([19]) or Relax NG ([8]). In many applications for data and document exchange, the data is streaming in large quantities and an on-line-one-pass processing and validation of XML using limited memory is required.

Due to its importance, the XML validation problem has received a lot of attention (cf. [18, 22, 4, 6, 5, 21]). One of the most recent developments on the problem is the use of Visibly Pushdown Automata (VPAs) for validating streaming XML ([14]). In this work, it was shown that VPAs precisely capture the languages of XML documents induced by Extended Document Type Definitions (EDTDs), introduced in [17][1].

EDTDs are essentially extended context free grammars enriched with types and can model all three popular schema formalisms mentioned above: DTD, XML Schema and Relax NG (see [17, 16]). After constructing a VPA for a given EDTD, the XML validation problem reduces to the one of accepting or rejecting an XML formatted word with the constructed VPA. We note here that the correspondence of VPAs to EDTDs is with respect to the word-encoded derivation trees of EDTDs rather than the set of words they generate. We remark that, when one uses an EDTD as a specification for XML documents, it is its language of derivation trees that is relevant; namely, for an XML document to be valid, when viewed as a tree, it has to correspond to a derivation tree (after applying typing) of the given EDTD.

VPAs, which as mentioned, precisely capture XML specifications given by EDTDs, are in essence pushdown automata. Their 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. For example, VPLs are closed under intersection and complement, and the containment problem is decidable.

In this paper, we introduce Visibly Pushdown Transducers (VPTs), which preserve the VPL family under their transductions. That is, given a VPL L and a VPT T, the transduction of L through T is again a VPL. We give constructions to obtain transductions for VPLs, as well as to perform useful operations on VPTs. Notably, VPTs give us a framework for solving the approximate validation of XML, where the approximation is in the sense that an XML document might not conform in its current form to a given EDTD but will do so after a few edit operations. Formally, in this paper, we study the K-validation problem for streaming XML:

> Given an EDTD and a positive integer K, preprocess and store the EDTD succinctly so that queries of the form "Does an XML document fit the EDTD specification after at most K edit operations?" can be answered efficiently.

Here, in line with the streaming XML model presented in [18], we distinguish two phases, which we explicitly call: the *preprocessing phase* and the *querying phase*. The preprocessing phase can be done offline and it has to be such as to

[1] [17] calls this formalism *specialized DTD* as types specialize tags. Similarly with [16], we use the term *extended DTD* to convey that this formalism is more powerful than DTD.

facilitate the next phase of querying which in turn has to be done online and in a single pass on the streaming XML.

In this paper, we study the approximate validation problem under two different semantics for edit operations on XML. The first semantics generalizes the standard edit operations on strings to XML. These edit operations are the substitution of a symbol by another, deletion of a symbol, and insertion of a string. In the context of XML, we generalize them by the substitution, deletion and insertion of pairs of matching open and close tags.

In the *preprocessing phase* of our algorithm for the first semantics, we construct VPTs for each of the three edit operations, and then superimpose them to produce a combined VPT for all the operations. Then, we transduce the given VPA through this VPT to get a new VPA. The preprocessing phase will store this VPA as its final output. The size of the final VPA is $O(KM)$ where M is the size of the given VPA (or EDTD). Thus, our algorithm for the first semantics uses storage space that only depends polynomially on the size of the EDTD and the error parameter.

In the *querying phase*, we receive as input a streaming XML document and check if this document is accepted or not by the VPA constructed in the preprocessing phase. Our querying scheme checks membership in a single pass over the XML document using time which depends only linearly on the size of the document, and auxilliary space proportional to the depth (and not size) of the document.

In the second part of the paper, we introduce another semantics for the edit operations on XML. Under this second semantics, whenever we decide to perform an edit operation with respect to an element, we apply the operation on all the occurrences of the element. To see the usefulness of this semantics, consider the following XML document

```
<collection>
   <book> Book-One </book> ... <book> Book-One-Thousand </book>
</collection>
```

in which there are 1000 *book* elements. It is clear that if we change *book* to *livre* we need to apply this all over the board for a total of 1000 times. Nevertheless, in this example, this should semantically count as one change, not as 1000. Our second semantics does exactly that; the "same fate" happens to all the occurrences of an element and this has a cost of one. On the other hand, under the first semantics, one would need 1000 edit operations to change all the *book* elements to *livre*. Depending on the application, the user can select the first or the second semantics.

Similarly to the first semantics, the preprocessing phase of our algorithm for the new semantics constructs VPTs for each of the three edit operations and glues these VPTs together through superimposition and union to construct a combined VPT for all the operations under the second semantics. We store the transduction of the given VPA through the VPT as the final output of this phase. The edit VPT for the second semantics has a size of $R^{O(K)}$, where R is the size

of the underlying alphabet, and thus, the transduction of the given VPA can have a size proportional to $R^{O(K)}M$. We believe that this exponential penalty in K is an artifact of the requirement that the queries be answered using only one pass through the XML document.

As in the case of the first semantics, the querying scheme is in fact membership testing which is done in a single pass over the XML document using time proportional to its size and auxilliary stack space proportional to the depth of the XML document.

We would like to remark that the three parameters - number of tolerable errors, size of the EDTD and the depth of the XML document are typically small in practice. Hence, our algorithms are viable for large and streaming XML.

The rest of the paper is organized as follows. In Section 2, we discuss related work. Section 3 reviews VPAs. In Section 4, we introduce VPTs, their transductions and operations on them. In Section 5 and 6, we present VPTs for edit operations under the first and second semantics respectively. Finally, Section 8 concludes the paper.

2 Related Work

The XML validation problem has received a lot of attention in the last years (cf. [18, 21, 14, 7, 20, 9, 4, 6, 5]).

The first three, [18, 21, 14], study the exact validation of XML in a streaming context. In [14], it was argued that VPAs are the apt device for the validation of streaming XML. Also, in the same work, it was shown that VPAs precisely correspond to EDTDs. In fact, this result could also be established based on [17] and [3]. Namely, [17] shows that the tree languages specified by EDTDs coincide with the class of regular tree languages, while [3] shows that the latter coincide with the VPL class.

The next three works, [7, 20, 9], consider variants of approximate XML validation, but in a non-streaming setting.

[7] presents a randomized methodology for validating and repairing XML documents. The main difference from our work is that [7] considers edit distance with *moves* and the error is relative rather than absolute. This means that the bigger the document is the bigger the error is tolerated to be. We believe that there are practical cases when an absolute tolerable error must be specified as opposed to a relative one.[2]

The methodology of [7] can be adapted to work in a streaming context with constant space for deciding the validity. However, reparing needs to build first the XML tree and then perform two passes on the tree.

[2] For example, suppose that there is a schema for XML documents about (people) contact information. Now, following [7], if a contact XML file has a mailing address and a phone number then we would tolerate more structural errors than for some other contact file with only the mailing address. We believe that in this case, one should use an absolute number of tolerable errors in order to not bias the tolerance of validation towards the first file.

[20] presents an exact algorithm, for validating and repairing XML documents. Both [7] and [20] have a similar flavor in that both have a recursive nature traversing the XML tree top-down and bottom-up, thus making two passes on the document. In contrast, our (exact) algorithms do not build an XML tree, and perform only a single pass on the document considered as a word. Also, our algorithms can be easily adapted to succinctly produce all the possible repairs for an XML document.

Regarding [9], it focuses on validating and repairing XML documents under a set of integrity constraints. The general problem for [9] is undecidable, and thus, it restricts the edit operations to either deletions or insertions only. All [7, 20, 9] consider simple DTDs only, while we consider VPAs which computationally represent EDTDs which in turn can abstract DTD, XML Schema and Relax NG.

The other three works, [4, 6, 5], consider the incremental validation of XML, which is validating documents after updates are being applied on them. The challenge there is to not rescan the document from the scratch, but rather work on the relevant (updated) part of the document. Also, the validation sought is exact rather than approximate. Although these works consider operations that edit (update) documents, the studied problem is very different from the approximate validation of streaming XML.

In all [7, 20, 9, 4, 6, 5], the edit operations are variants of, or can be achieved by, our edit operations under the first semantics. On the other hand, edit operations under our second semantics, although quite useful in practice, to the best of our knowledge, have not been studied by any work.

Our treatment of approximate XML validation bears some similar flavor with [10–12]. However, these works deal with regular languages only and revolve around a different problem, which is finding paths in graph databases that approximately spell words in a given regular language.

3 Visibly Pushdown Automata

VPAs were introduced in [3] and are a special case of pushdown automata. Thier 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.[3]

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.
   - $\tau_c \subseteq Q \times \Sigma_c \times Q \times \Gamma$
   - $\tau_r \subseteq Q \times \Sigma_r \times \Gamma \times Q$
   - $\tau_l \subseteq Q \times \Sigma_l \times Q$
   - $\tau_\epsilon \subseteq Q \times \{\epsilon\} \times Q$

When reasoning about XML structure and validity, the local symbols are not important, and thus, for simplicity we will not mention local symbols in the rest of the paper. So, for the above definition, we can consider Σ being partitioned into Σ_c and Σ_r, and τ being partitioned into τ_c, τ_r and τ_ϵ only.

Any transition involves two states (not necessarily distinct). We call the first the *origin state* and the second the *destination state*.

Two transitions are called *consecutive* if the destination state of the first is the same as the origin state of the second. This definition applies regardless of whether the transitions involve a push or a pop.

A sequence of consecutive transitions is an *accepting run* if (a) the origin state of the first transition is q_0, (b) the destination state of the last transition is in F and (c) when starting with an empty stack ($\perp$) and following all the transitions in order, in the end, we get again an empty stack ($\perp$).

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 $L(A)$.

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

$$
\begin{aligned}
&Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\},\\
&\Sigma = \Sigma_c \cup \Sigma_r =\\
&\qquad \{collection, book, author, title\} \cup \{\overline{collection}, \overline{book}, \overline{author}, \overline{title}\},\\
&\qquad f \text{ maps the } \Sigma_c \text{ elements into their "bar"-ed counterparts in } \Sigma_r,\\
&\Gamma = \{\gamma_c, \gamma_b, \gamma_a, \gamma_t\} \cup \{\perp\},\\
&F = \{q_8\},
\end{aligned}
$$

[3] 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.

$$\tau = \{(q_0, collection, q_1, \gamma_c), (q_1, book, q_2, \gamma_b), (q_2, author, q_3, \gamma_a),$$
$$(q_3, \overline{author}, \gamma_a, q_4), (q_4, author, q_3, \gamma_a), (q_4, title, q_5, \gamma_t),$$
$$(q_5, \overline{title}, \gamma_t, q_6), (q_6, \overline{book}, \gamma_b, q_7), (q_7, \overline{collection}, \gamma_c, q_8), (q_7, \epsilon, q_1)\}.$$

We show this VPA in Fig. 1.

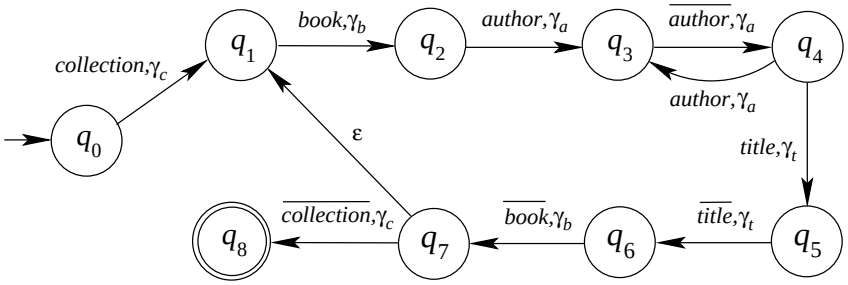

Fig. 1. Example of a VPA.

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

Now, the question is whether this can be done using a non-deterministic VPA A. Recall that the well-known procedure for deciding the membership of a word w to a general context-free language, is grammar-based and takes $|w|^3$ time, which is not appropriate for a streaming context.

On the other hand, as shown in [3], VPAs can be determinized, and thus allow for a linear one-pass scanning of a given word. However, there is an exponential penalty to pay for storing deterministic VPAs. Specifically, Theorem 2 in [3] shows that for a given non-deterministic VPA with M states there is an equivalent deterministic VPA with $O(2^{M^2})$ states and with stack alphabet of size $O(2^{M^2} \cdot |\Sigma_c|)$. Nevertheless, for processing a word w, we do not need the whole deterministic VPA, but rather only the single transition sequence spelling w in this automaton. Along the lines of Theorem 2 in [3], one can construct this path on the fly. The amount of space needed is $O(M^2)$, while the time for processing a symbol of w and finding for it the "next transition" in the sequence of transitions is $O((M^2 + M) \cdot |\Sigma_c|)$. In total, one needs only one pass on word w, for a time $O((M^2 + M) \cdot |\Sigma_c| \cdot |w|)$, which depends only linearly on the length of w.

4 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 $\perp$.
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
   – $\tau_c \subseteq (P \times I_c \times O_c \times P \times \Gamma) \cup (P \times \{\epsilon\} \times O_c \times P \times \Gamma) \cup (P \times I_c \times \{\epsilon\} \times P \times \Gamma)$
   – $\tau_r \subseteq (P \times I_r \times O_r \times \Gamma \times P) \cup (P \times \{\epsilon\} \times O_r \times \Gamma \times P) \cup (P \times I_r \times \{\epsilon\} \times \Gamma \times P)$
   – $\tau_\epsilon \subseteq P \times \{\epsilon\} \times \{\epsilon\} \times P$.

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.[4]

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 $T(L) = \bigcup_{u \in L} T(u)$.

If language L is a VPL, then we show that $T(L)$ is a VPL as well. To show this, let $A = (Q, (\Sigma^A, f^A), \Gamma^A, \tau^A, q_0, F^A)$ be a VPA accepting L, and $T = (P, (I, f^T), (O, g^T), \Gamma^T, \tau^T, p_0, F^T)$ be a VPT as above, where $I \supseteq \Sigma^A$ and f^T is an extension of f^A. Then, we present a construction to obtain a VPA B, whose accepting language is $T(L)$, showing thus that the image of L through T is again a VPL.

The construction is a Cartesian product of A and T and similar in spirit to the construction of [3] for showing the closure of VPLs under intersection.

Specifically, $B = (R, (\Sigma^B, g^B), \Gamma^B, \tau^B, r_0, F^B)$, where

1. $R = Q \times P$,
2. $\Sigma^B \subseteq O$, and g^B is a refinement of g^T
3. $\Gamma^B \subseteq (\Gamma^A \cup \dagger) \times \Gamma^T$, where $\dagger$ is a special symbol not in Γ^A and Γ^T.
4. $r_0 = (q_0, p_0)$,
5. $F^B = F^A \times F^T$,

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

6. $\tau^B = \tau_c^B \cup \tau_r^B$, where

$$\begin{aligned}
\tau_c^B = {} & \{(q,p), b, (q',p'), (\gamma^A, \gamma^T)) : (q, a, q', \gamma^A) \in \tau_c^A \text{ and } (p, a, b, p', \gamma^T) \in \tau_c^T\} \cup \\
& \{((q,p), \epsilon, (q',p')) : (q, a, q', \gamma^A) \in \tau_c^A, a \neq \epsilon \text{ and } (p, a, \epsilon, p', \gamma^T) \in \tau_c^T\} \cup \\
& \{((q,p), b, (q,p'), (\dagger, \gamma^T)) : q \in Q \text{ and } (p, \epsilon, b, p', \gamma^T) \in \tau_c^T\} \\
\tau_r^B = {} & \{((q,p), \bar{b}, (\gamma^A, \gamma^T), (q',p')) : (q, \bar{a}, \gamma^A, q') \in \tau_r^A \text{ and } (p, \bar{a}, \bar{b}, \gamma^T, p') \in \tau_r^T\} \cup \\
& \{((q,p), \epsilon, (q',p')) : (q, \bar{a}, \gamma^A, q') \in \tau_r^A, a \neq \epsilon \text{ and } (p, \bar{a}, \epsilon, \gamma^T, p') \in \tau_r^T\} \cup \\
& \{((q,p), \bar{b}, (\dagger, \gamma^T), (q,p')) : q \in Q \text{ and } (p, \epsilon, \bar{b}, \gamma^T, p') \in \tau_r^T\}
\end{aligned}$$

Clearly, B is a VPA, and we can show that

Theorem 1. *The language accepted by B is the image of L through T, i.e. $L(B) = T(L)$.*

Proof. A VPT T can be considered as two VPAs; the *input* VPA A_{T_I} and the output *output* VPA A_{T_O}. A_{T_I} and A_{T_O} can be obtained from T by ignoring the output and input parts, respectively, of the transitions of T. A_{T_I} and A_{T_O} have the same structure; each transition path in A_{T_I} has some corresponding transition path in A_{T_O} and vice versa.

Now, the construction of VPA B computes the Cartesian product of VPA A with VPA A_{T_I}, but instead of keeping the matched transitions, it replaces them by the corresponding transitions in A_{T_O}.

Thus, if B accepts a word w, it means that there exists a corresponding word u accepted by A and A_{T_I}, such that $w \in T(u)$. As $u \in L$, we have that $T(u) \subseteq T(L)$ and $w \in T(L)$.

On the other hand, for a word u in $T(L)$, there exists some accepting transition path in the Cartesian product of A with A_{T_I}. By the construction of B, this accepting path induces an accepting path in B as well. Let w be the word spelled out by such a path in B. We have that $w \in L(B)$, and this concludes our proof. □

Union of VPTs. In this paper, we will need to take the union of transducers. Formally given two VPTs $T_1 = (P_1, (I,f), (O,g), \Gamma_1, \tau_1, p_{01}, F_1)$ and $T_2 = (P_2, (I,f), (O,g), \Gamma_2, \tau_2, p_{02}, F_2)$, their union VPT is

$$T = (P_1 \cup P_2 \cup \{p_0\}, (I,f), (O,g), \Gamma_1 \cup \Gamma_2, \tau_0 \cup \tau_1 \cup \tau_2, p_0, F_1 \cup F_2),$$

where $p_0 \notin P_1 \cup P_2$, and $\tau_0 = \{(p_0, \epsilon, \epsilon, p_{01}), (p_0, \epsilon, \epsilon, p_{02})\}$.

Superimposition of VPTs. Given two VPTs, $T_1 = (P, (I,f), (O,g), \Gamma_1, \tau_1, p_0, F)$ and $T_2 = (P, (I,f), (O,g), \Gamma_2, \tau_2, p_0, F)$, which are the same except for the stack alphabet and transition relation, their superimposition VPT is

$$T = (P, (I,f), (O,g), \Gamma_1 \cup \Gamma_2, \tau_1 \cup \tau_2, p_0, F).$$

VPTs for edit operations. In the rest of the paper, we will work on buiding transducers for preprocessing a given VPA specification A, transducing it into a "wider" VPA B, which accepts all the words obtainable by applying at most K edit operations on the words accepted by A. After such a preprocessing phase, the querying phase amounts to accepting or rejecting the streaming XML document considering it as a word.

5 VPTs for Edit Operations under the First Semantics

Since XML documents are nested, when we edit one call element, we also need to edit the corresponding return element. Thus, we consider an (XML) edit operation to consist of two single-symbol operations.

We want to build a visibly pushdown transducer, which given an input word u produces as output all the words v obtainable by applying not more than a certain number (say K) of edit operations on u. We define the edit operations as substitutions, deletions, and insertions of call-return matches, and computationally represent them by using VPTs.

5.1 Substitution

A *call-return match substitution* replaces in an input word a call-return match $a, \bar{a}$ by another call-return match $b, \bar{b}$.

For example, consider the XML document given in Fig. 2 [left]. By substituting `<phone>`, `</phone>` by `<tel>`, `</tel>`, we obtain the document shown in Fig. 2 [right].

```
<contact>                    <contact>
  <address>                    <address>
    <str>...</str>               <str>...</str>
    <city>...</city>             <city>...</city>
  </address>                   </address>
  <phone>...</phone>           <tel>...</tel>
</contact>                   </contact>
```

Fig. 2. Illustration of substitution under the first semantics.

In the following, given a non-negative integer K, we build a VPT which for any word u produces as output the set of all the words w obtainable from u by applying at most K substitutions. We denote this transducer by $T_{\sigma}^{\leq K}$ and formally define it as a VPT with

- $Q = \{q_0, q_1, q_2, \ldots, q_{2K}\}$,
- $I = O = \Sigma$, $I_c = O_c = \Sigma_c$, $I_r = O_r = \Sigma_r$ and $f = g$,
- $\Gamma = \{\gamma_a : a \in \Sigma_c\} \cup \{\sigma_{ab} : a, b \in \Sigma_c, a \neq b, \} \cup \{\bot\}$,
- $F = \{q_{2i} : 0 \leq i \leq K\}$,
- $\tau = \tau_c \cup \tau_r$, where

$$\begin{aligned}
\tau_c = {} & \{(q_i, a, a, q_i, \gamma_a) : 0 \leq i \leq 2K \text{ and } a \in \Sigma_c\} \cup \\
& \{(q_i, a, b, q_{i+1}, \sigma_{ab}) : 0 \leq i \leq 2K-1, a, b \in \Sigma_c \text{ and } a \neq b\}, \\
\tau_r = {} & \{(q_i, \bar{a}, \bar{a}, \gamma_a, q_i) : 0 \leq i \leq 2K \text{ and } \bar{a} \in \Sigma_r\} \cup \\
& \{(q_i, \bar{a}, \bar{b}, \sigma_{ab}, q_{i+1}) : 1 \leq i \leq 2K-1, \bar{a}, \bar{b} \in \Sigma_r \text{ and } \bar{a} \neq \bar{b}\}.
\end{aligned}$$

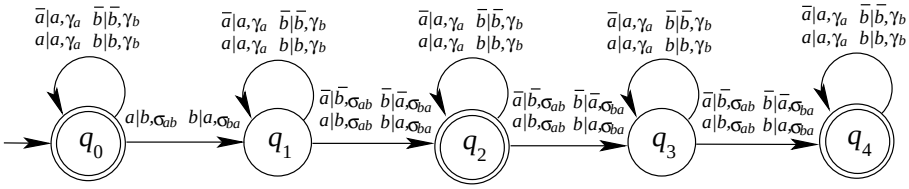

Fig. 3. VPT $T_{\bar{\sigma}}^{\leq 2}$.

For illustration, in Fig. 3, we show $T_{\bar{\sigma}}^{\leq 2}$, for alphabet $\{a, b\} \cup \{\bar{a}, \bar{b}\}$.

Intuitively, the transitions in the first set of τ_c and in the first set of τ_r leave the consumed call and return symbols unchanged.

Regarding the transitions in the second set of τ_c, they substitute a call symbol, say a, by another call symbol, say b. A substitution marking symbol σ_{ab} is pushed onto the stack. Symbol σ_{ab} in the stack is crucial in determining which occurrence of $\bar{a}$ has to be replaced by $\bar{b}$ using a transition in the second set of τ_r.

Finally, since we want to substitute 0, 1, ..., K symbols, we need $K + 1$ different final states for the $K + 1$ different cases.

5.2 Deletion

A *call-return match deletion* removes in an input word a call-return match $a, \bar{a}$. Deletion is a "structure flattening" operation.

For example, consider the XML document given in Fig. 4 [left]. By deleting `<address>`, `</address>`, we (partially) flatten the document to the one shown in Fig. 4 [right].

```
<contact>
  <address>
    <str>...</str>
    <city>...</city>
  </address>
  <phone>...</phone>
</contact>
```

```
<contact>
  <str>...</str>
  <city>...</city>
  <phone>...</phone>
</contact>
```

Fig. 4. Illustration of deletion under the first semantics.

In the following, given a non-negative integer K, we build a VPT which for any word u produces as output the set of all the words w obtainable from u by applying at most K deletions. We denote this transducer by $T_{\bar{\delta}}^{\leq K}$ and formally define it as a VPT with

- $Q = \{q_0, q_1, q_2, \ldots, q_{2K}\}$,

– $I = O = \Sigma$, $I_c = O_c = \Sigma_c$, $I_r = O_r = \Sigma_r$ and $f = g$,
– $\Gamma = \{\gamma_a : a \in \Sigma_c\} \cup \{\delta_a : a \in \Sigma_c\} \cup \{\bot\}$,
– $F = \{q_{2i} : 0 \leq i \leq K\}$,
– $\tau = \tau_c \cup \tau_r$, where

$$\begin{aligned}
\tau_c = \{&(q_i, a, a, q_i, \gamma_a) : 0 \leq i \leq 2K \text{ and } a \in \Sigma_c\} \cup \\
&\{(q_i, a, \epsilon, q_{i+1}, \delta_a) : 0 \leq i \leq 2K-1, \text{ and } a \in \Sigma_c\}, \\
\tau_r = \{&(q_i, \bar{a}, \bar{a}, \gamma_a, q_i) : 0 \leq i \leq 2K \text{ and } \bar{a} \in \Sigma_r\} \cup \\
&\{(q_i, \bar{a}, \epsilon, \delta_a, q_{i+1}) : 1 \leq i \leq 2K-1, \text{ and } \bar{a} \in \Sigma_r\}.
\end{aligned}$$

For illustration, in Fig. 5, we show $T_{\delta}^{\leq 2}$, for alphabet $\{a, b\} \cup \{\bar{a}, \bar{b}\}$.

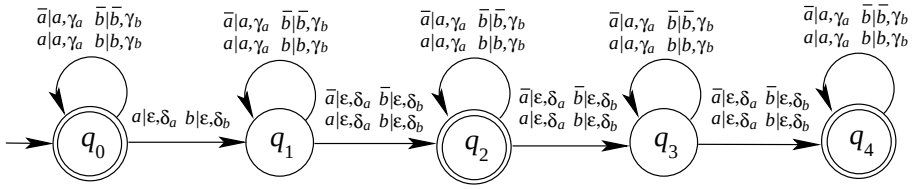

Fig. 5. VPT $T_{\delta}^{\leq 2}$.

Similarly with the substitution, the transitions in the first set of τ_c and in the first set of τ_r leave the consumed call and return symbols unchanged.

Regarding the transitions in the second set of τ_c, they delete a call symbol, say a. A deletion marking symbol δ_a is pushed onto the stack. Symbol δ_a in the stack is crucial in determining which occurrence of $\bar{a}$ has to be deleted by using a transition in the second set of τ_r.

Since we want to perform 0, 1, ..., K deletions, we need $K+1$ different final states for the $K+1$ different cases.

5.3 Insertion

A *call-return match insertion* inserts in an input word a call symbol a and a corresponding return symbol $\bar{a}$ while maintaining the well-formedness of the XML document. Thus, insertion is a "structure creation" operator.

For example, consider the XML document given in Fig. 6 [left]. By inserting `<address>`, `</address>`, surrounding the street and city elements, we obtain the document in Fig. 6 [right].

In the following, given a non-negative integer K, we build a VPT which for any word u produces as output the set of all the words w obtainable from u by applying at most K insertions. We denote this transducer by $T_{\eta}^{\leq K}$ and formally define it as a VPT with

– $Q = \{q_0, q_1, q_2, \ldots, q_{2K}\}$,

```
<contact>                          <contact>
  <str>...</str>                     <address>
  <city>...</city>                     <str>...</str>
  <phone>...</phone>                   <city>...</city>
</contact>                         </address>
                                   <phone>...</phone>
                                 </contact>
```

Fig. 6. Illustration of insertion under the first semantics.

- $I = O = \Sigma$, $I_c = O_c = \Sigma_c$, $I_r = O_r = \Sigma_r$ and $f = g$,
- $\Gamma = \{\gamma_a : a \in \Sigma_c\} \cup \{\eta_a : a \in \Sigma_c\} \cup \{\perp\}$,
- $F = \{q_{2i} : 0 \leq i \leq K\}$,
- $\tau = \tau_c \cup \tau_r$, where

$$\begin{aligned}
\tau_c = &\{(q_i, a, a, q_i, \gamma_a) : 0 \leq i \leq 2K \text{ and } a \in \Sigma_c\} \cup \\
&\{(q_i, \epsilon, a, q_{i+1}, \eta_a) : 0 \leq i \leq 2K-1, \text{ and } a \in \Sigma_c\}, \\
\tau_r = &\{(q_i, \bar{a}, \bar{a}, \gamma_a, q_i) : 0 \leq i \leq 2K \text{ and } \bar{a} \in \Sigma_r\} \cup \\
&\{(q_i, \epsilon, \bar{a}, \eta_a, q_{i+1}) : 1 \leq i \leq 2K-1, \text{ and } \bar{a} \in \Sigma_r\}.
\end{aligned}$$

For illustration, in Fig. 7, we show $T_\eta^{\leq 2}$, for alphabet $\{a, b\} \cup \{\bar{a}, \bar{b}\}$.

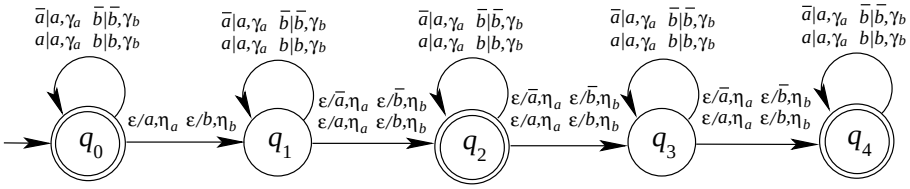

Fig. 7. VPT $T_\eta^{\leq 2}$.

Again, the transitions in the first set of τ_c, and in the first set of τ_r leave the consumed call and return (respectively) symbols unchanged.

Regarding the transitions in the second set of τ_c, they insert a call symbol, say a. An insertion marking symbol η_a is inserted on the stack. Symbol η_a in the stack is crucial in determining when to insert $\bar{a}$ by using a transition in the second set of τ_r.

5.4 A VPT for All Operations

Here, for a given a non-negative integer K, we want to construct a VPT which for any word u produces as output the set of all the words w obtainable from u

by applying at most K edit operations, which can be substitutions, deletions or insertions.

As can be observed above, sets Q, I, O and F are the same for all the transducers constructed so far. Notably, a VPT $T^{\leq K}$ for at most K edit operations can be simply obtained by superimposing $T_{\sigma}^{\leq K}$, $T_{\delta}^{\leq K}$ and $T_{\eta}^{\leq K}$.

Transducer $T^{\leq K}$ has $2K+1$ states and $O(KR^2)$ transitions, where R is the size of the underlying alphabet.

Edit Distance. Edit operations transform a word into other words. For two given words u and w we define the *distance between u and w* as the least number of edit operations needed to transform u into w. We denote this distance by $d(u, w)$. It is easy to see that the distance defined using the edit operations under the first semantics is metric.

Given, a VPL L and a non-negative integer K, we define

$$L^{(K)} = \{u : \exists w \in L \text{ and } d(u, w) \leq K\}.$$

Now, we can show that for the above transducer $T^{\leq K}$

Theorem 2. $T^{\leq K}(L) = L^{(K)}$.

Proof. **Basis step.** For $k = 0$, we have $L^{(k)} = L^{(0)} = L$. On the other hand, $T^{\leq 0}$ is nothing but just a single state transducer with only self-loop transitions which leave everything unchanged. Thus, $T^{\leq 0}(L) = L = L^{(0)}$.
Induction step. Suppose that $T^{\leq k}(L) = L^{(k)}$ is true for non-negative k. We want to show that $T^{\leq k+1}(L) = L^{(k+1)}$ is true as well.

Based on the construction of edit VPAs, we have that $T^{\leq k+1}$ is in fact $T^{\leq k}$ with two more additional states, q_{2k+1} and q_{2k+2}. These two states allow $T^{\leq k+1}$ to optionally perform one more operation, which can be substitution, deletion or insertion.

By the hypothesis, $T^{\leq k}(L) = L^{(k)} = \{w : \exists w' \in L \text{ and } d(w, w') \leq k\}$. Now, we have that, since $T^{\leq k+1}$ can perform one more operation,
$T^{\leq k+1}(L) = \{w : \exists w' \in L \text{ and } d(w, w') \leq k \text{ or } d(w, w') = k+1\} =$
$\{w : \exists w' \in L \text{ and } d(w, w') \leq k+1\}$. The latter is nothing but $L^{(k+1)}$, and this completes the proof. □

From all the above, and the construction for the language transduction of a VPT (in Section 4), we can show that

Theorem 3. *Under the first semantics, the total time for the preprocessing phase, and the space for storing the output of it, is $O(KR^2M)$.*

Proof. This claim follows from the fact that transducer $T^{\leq K}$ has $2K+1$ states and $O(KR^2)$ transitions, and the transduction of the schema VPA with M states is done through a Cartesian product, which will have in this case $O(KM)$ states and $O(KR^2M)$ transitions. The latter is thus an upper bound for the time and space needed to compute and store the transduction of the schema VPA. This is nothing but the time and space needed for the preprocessing phase. □

6 VPTs for Edit Operations under the Second Semantics

6.1 Substitution

Under the second semantics, a substitution replaces in an input word *all* the call-return matches of a call-return pair $(a, \bar{a})$ by call-return matches of another call-return pair $(b, \bar{b})$.

Let alphabet Σ be $\{a_1, \ldots, a_R\} \cup \{\bar{a}_1, \ldots, \bar{a}_R\}$. Clearly, we can have $R(R-1)$ pairs of different call symbols (e.g. (a_1, a_2), etc). We can now construct substitution transducers which are indexed by these pairs and perform accordingly the substitution indicated by their index pair. For example, the substitution transducer indexed by (a_1, a_2), denoted in short by $T^1_{\sigma:12}$, will substitute all call-return matches of $(a_1, \bar{a}_1)$ by call-return matches of $(a_2, \bar{a}_2)$ in any word provided as input. The supscript says that this transducer is of "order one," i.e. it substitutes only the call-return matches of one call-return pair. It is not difficult to construct such transducers. Formally, $T^1_{\sigma:ij}$ (for $i \neq j$) is defined as a VPT with a single state q_0 which is both initial and final, stack alphabet $\Gamma = \{\gamma_k : k \in \{1, \ldots, r\} \backslash \{i\}\} \cup \{\sigma_{ij}\} \cup \{\perp\}$, and transition relation

$$\tau_c = \{(q_0, a_k, a_k, q_0, \gamma_k) : k \neq i\} \cup \{(q_0, a_i, a_j, q_0, \sigma_{ij})\},$$
$$\tau_r = \{(q_0, \bar{a}_k, \bar{a}_k, \gamma_k, q_0) : k \neq i\} \cup \{(q_0, \bar{a}_i, \bar{a}_j, \sigma_{ij}, q_0)\}.$$

Similarly, we can construct transducers of "order two," which perform substitutions for the call-return matches of two call-return pairs. Such a transducer $T^2_{\sigma:ij,kl}$ (for $i \neq j, k$ and $k \neq l$) is defined again as a VPT with a single state q_0 which is both initial and final, stack alphabet $\Gamma = \{\gamma_m : m \in \{1, \ldots, r\} \backslash \{i, k\}\} \cup \{\sigma_{ij}, \sigma_{kl}\} \cup \{\perp\}$, and transition relation

$$\tau_c = \{(q_0, a_m, a_m, q_0, \gamma_m) : m \neq i, k\} \cup \{(q_0, a_i, a_j, q_0, \sigma_{ij}), (q_0, a_k, a_l, q_0, \sigma_{kl})\},$$
$$\tau_r = \{(q_0, \bar{a}_m, \bar{a}_m, \gamma_m, q_0) : m \neq i, k\} \cup \{(q_0, \bar{a}_i, \bar{a}_j, \sigma_{ij}, q_0), (q_0, \bar{a}_k, \bar{a}_l, q_0, \sigma_{kl})\}.$$

We can observe that, regardless of H, the number of transitions in these one-state transducers is $2R$. The transitions in each of the τ_c and τ_r sets are divided into: "leave unchanged" transitions and "modify symbol" transitions.

In general, we can construct substitution transducers of any order up to the size R of the alphabet. Let T^H_σ be the union of all substitution transducers of order H. Then, we construct transducer $T^{\leq K}_\sigma = \bigcup_{H=0}^{K} T^H_\sigma$ (considering also T^0_σ which leaves everything unchanged). Given a word u as input, $T^{\leq K}_\sigma$ produces as output the set of all words w obtainable from u by applying at most K substitutions under the second semantics.

Now the question is: For a given H, how many transducers of order H can be created? We show that

Theorem 4. *Given a non-negative integer $H \leq R$, the number of substitution transducers of order H is $C^H_R \cdot (R-1)^H$.*

Proof. For this, recall that a transducer of order H substitutes call-return matches of H call-return pairs. We have C^H_R choices for these pairs. In each choice, any chosen pair, say $(a_i, \bar{a}_i)$, can be substituted by any of the $(R-1)$ remaining pairs $(a_1, \bar{a}_1), \ldots, (a_{i-1}, \bar{a}_{i-1}), (a_{i+1}, \bar{a}_{i+1}), \ldots, (a_R, \bar{a}_R)$. □

6.2 Deletion

A deletion removes from an input word *all* the call-return matches of a call-return pair $(a, \bar{a})$. A deletion transducer of order H deletes all the call-return matches of H call-return pairs, say $(a_{i_1}, \bar{a}_{i_1}), \ldots, (a_{i_H}, \bar{a}_{i_H})$, in an input word. This transducer, denoted by $T^H_{\delta:i_1,\ldots,i_H}$, has a single state which is both initial and final, stack alphabet $\{\gamma_j : j \in \{1, \ldots, R\} \backslash \{i_1, \ldots, i_H\}\} \cup \{\delta_{i_1}, \ldots, \delta_{i_H}\} \cup \{\perp\}$ and transition relation

$$\tau_c = \{(q_0, a_j, a_j, q_0, \gamma_j) : j \neq i_1, \ldots, i_H\} \cup \{(q_0, a_{i_1}, \epsilon, q_0, \delta_{i_1}), \ldots, (q_0, a_{i_H}, \epsilon, q_0, \delta_{i_H})\},$$
$$\tau_r = \{(q_0, \bar{a}_j, \bar{a}_j, \gamma_j, q_0) : j \neq i_1, \ldots, i_H\} \cup \{(q_0, \bar{a}_{i_1}, \epsilon, \delta_{i_1}, q_0), \ldots, (q_0, \bar{a}_{i_H}, \epsilon, \delta_{i_H}, q_0)\}.$$

We can observe that, regardless of H, the number of transitions in these one-state transducers is $2R$. Also, reasoning similarly as for the substitution, we can show that

Theorem 5. *Given a non-negative integer $H \leq R$, the number of deletion transducers of order H is C^H_R.*

Proof. This follows from the fact that a deletion transducer of order H deletes call-return matches of H call-return pairs, and we have C^H_R choices for these pairs. □

In general, we can construct deletion transducers of any order up to the size R of the alphabet.

6.3 Insertion

An insertion operation under the second semantics non-deterministically inserts in an input word *any* number of a single call symbol a balancing those insertions by inserting in the right places the corresponding return symbol $\bar{a}$.

An insertion transducer of order H inserts call-return matches for H call-return pairs, say $(a_{i_1}, \bar{a}_{i_1}), \ldots, (a_{i_H}, \bar{a}_{i_H})$, in an input word. This transducer, denoted by $T^H_{\eta:i_1,\ldots,i_H}$, has a single state which is both initial and final, stack alphabet $\{\gamma_j : j \in \{1, \ldots, R\}\} \cup \{\eta_{i_1}, \ldots, \eta_{i_H}\} \cup \{\perp\}$ and transition relation

$$\tau_c = \{(q_0, a_j, a_j, q_0, \gamma_j) : j \in \{1, \ldots, R\}\} \cup \{(q_0, \epsilon, a_{i_1}, q_0, \eta_{i_1}), \ldots, (q_0, \epsilon, a_{i_H}, q_0, \eta_{i_H})\},$$
$$\tau_r = \{(q_0, \bar{a}_j, \bar{a}_j, \gamma_j, q_0) : j \in \{1, \ldots, R\}\} \cup \{(q_0, \epsilon, \bar{a}_{i_1}, \eta_{i_1}, q_0), \ldots, (q_0, \epsilon, \bar{a}_{i_H}, \eta_{i_H}, q_0)\}.$$

We can observe that, the number of transitions in these one-state transducers is $2(R + H)$. Also, as for the deletion, we have that

Theorem 6. *Given a non-negative integer $H \leq R$, the number of insertion transducers of order H is C^H_R.*

Proof. This follows from the fact that an insertion transducer of order H inserts call-return matches of H call-return pairs, and we have C^H_R choices for these pairs. □

In general, we can construct insertion transducers of any order up to the size R of the alphabet.

6.4 A VPT for All Operations

We now can easily create edit transducers of order H, by superimposing substitution, deletion and insertion transducers of orders H_1, H_2 and H_3, such that $H_1 + H_2 + H_3 = H$.

Formally, let $T^{H_1}_{\sigma:i_1 j_1,\ldots,i_{H_1} j_{H_1}}$, $T^{H_2}_{\delta:k_1,\ldots,k_{H_2}}$ and $T^{H_3}_{\eta:l_1,\ldots,l_{H_3}}$ be substitution, deletion and insertion transducers. If $\{i_1,\ldots,i_{H_1}\}\cap\{k_1,\ldots,k_{H_2}\} = \emptyset$, then we superimpose these three transducers to obtain a transducer of order $H = H_1+H_2+H_3$. The condition $\{i_1,\ldots,i_{H_1}\} \cap \{k_1,\ldots,k_{H_2}\} = \emptyset$ says that the call-return pairs we substitute must be different from those we delete. This is because under the second semantics of edit operations, all call-return matches of a call-return pair have "the same fate." If the transducer both substitutes and deletes the call-return matches of a call-return pair, then we will have a situation where some of these call-return matches have been substituted and some other ones have been deleted.

Of course, in a superimposition, there does not need to be a transducer for each kind of edit operation. For example, we can create a transducer of order H by superimposing a substitution transducer and a deletion transducer of orders H_1 and H_2 respectively, such that $H = H_1 + H_2$.

Now, based on the above as well as theorems 4, 5 and 6, we can state that

Theorem 7. *Given a non-negative integer $H \leq R$, the number of edit transducers of order H under the second semantics is $O\left(R^{2H} \cdot H^2\right)$.*

Proof. We can create an edit transducer of order $H = H_1+H_2+H_3$ by selecting for the superimposition one of $C_R^{H_1} \cdot (R-1)^{H_1}$, $C_R^{H_2}$ and $C_R^{H_3}$ substitution, deletion and insertion transducers respectively. Of course, one or two of H_1, H_2 and H_3 might be zero.

Clearly, $C_R^{H_1} \cdot (R-1)^{H_1}$, $C_R^{H_2}$ and $C_R^{H_3}$ are bounded by $R^{H_1} \cdot (R-1)^{H_1}$, R^{H_2} and R^{H_3} respectively. Thus, given H_1, H_2 and H_3 we have $R^{H_1} \cdot (R-1)^{H_1} \cdot R^{H_2} \cdot R^{H_3} = R^H \cdot (R-1)^{H_1}$, or $O(R^{2H})$ edit transducers.

Now, the claimed upper bound follows from the above and the fact that we have $O(H^2)$ possibilities of choosing H_1, H_2 and H_3 such that $H = H_1+H_2+H_3$. □

Let T^H be the union of all edit transducers of order H. Then, we construct transducer $T^{\leq K} = \bigcup_{H=0}^{K} T^H$. Given a word u as input, $T^{\leq K}$ produces as output the set of all words w obtainable from u by applying at most K edit operations under the second semantics.

Based on Theorem 7 and the construction given for the union of transducers in Section 4, we can state that

Theorem 8. *Transducer $T^{\leq K}$ has $O\left(R^{2K} \cdot K^3\right)$ states.*

Proof. By the construction for the union of transducers, $T^{\leq K} = \bigcup_{H=0}^{K} T^H$ has $O\left(\Sigma_{H=0}^{K} R^{2H} \cdot H^2\right)$ states. This is subsumed by $O\left(K \cdot R^{2K} \cdot K^2\right)$, which is $O\left(R^{2K} \cdot K^3\right)$, i.e. the upper bound in our claim. □

Finally, by the fact that, in a superimposition, each of the three transducers has $O(R)$ transitions, we can state that

Theorem 9. *Transducer $T^{\leq K}$ has $O\left(R^{2K+1} \cdot K^3\right)$ transitions.*

Proof. Direct from the constructions for the superimposition and union of VPTs. □

Edit Distance. Similarly as for the first semantics, given words u and w, we define the *distance between words u and w* as the least number of edit operations (under the second semantics) needed to transform u into w. Here as well, it is easy to see that this distance is metric.

Given, a VPL L and a non-negative integer K, we define $L^{(K)}$ as in Subsection 5.4, but considering instead the distance under the second semantics. Here we can show that, similarly with Theorem 2, $T^{\leq K}(L) = L^{(K)}$.

From the above and theorems 8 and 9, we can state that

Theorem 10. *Under the second semantics, the total time for the preprocessing phase, and the space for storing the output of it, is $O(R^{2K+1}K^3M)$.*

Proof. This claim follows from the fact that transducer $T^{\leq K}$ has $O\left(R^{2K} \cdot K^3\right)$ states, and the transduction of the schema VPA with M states is done through a Cartesian product, which will have in this case $O(R^{2K} \cdot K^3 \cdot M)$ states and $O(R \cdot R^{2K} \cdot K^3 \cdot M)$ transitions. The latter is thus an upper bound for the time and space needed to compute and store the transduction of the schema VPA. This is nothing but the time and space needed for the preprocessing phase under the second semantics. □

We believe that this exponential penalty in K is an artifact of the requirement that the queries be answered using only one pass through the XML document, while using auxilliary storage space only bounded by the depth of the document. If we were allowed to use auxilliary space polynomial in the size N of the document, we believe that a *cubic* in N algorithm similar in spirit to [1] could possibly be devised to use storage space only polynomial in K and M. However, such an algorithm is useful only in a non-streaming context and when the document size is not large. This is a topic for our future investigation.

We can also observe that the edit distance between any XML document and a VPA in the second semantics is at most $2R$, i.e. twice the size of the underlying alphabet. This is because we can first delete all the call-return matches of the document using at most R delete operations and then create a string in the language of the VPA using at most R insert operations.

7 Repairs

Now, suppose that we are also interested in obtaining the set $L_{w,K}$ of words accepted by the original schema VPA A that can be transformed to match an XML document w by applying on them at most K edit operations based on either semantics. The words in $L_{w,K}$ are the possible *repairs* of XML document w.

For computing $L_{w,K}$, we need to "enrich" the construction of the transduction to remember the lineage of its words. For this, instead of VPA B, we can construct a VPT T_B, which coincides with B when considering only the input of its transitions. On the other hand, the output of its transitions "remembers" the input of T's transitions that matched the A's transitions. Formally, the transition relation of T_B is $\tau^B = \tau_c^B \cup \tau_r^B$, where

$$\begin{aligned}
\tau_c^B = &\{(q,p), b, a, (q',p'), (\gamma^A, \gamma^T)) : (q, a, q', \gamma^A) \in \tau_c^A \text{ and } (p, a, b, p', \gamma^T) \in \tau_c^T\} \cup \\
&\{((q,p), \epsilon, a, (q',p')) : (q, a, q', \gamma^A) \in \tau_c^A, a \neq \epsilon \text{ and } (p, a, \epsilon, p', \gamma^T) \in \tau_c^T\} \cup \\
&\{((q,p), b, \epsilon, (q,p'), (\dagger, \gamma^T)) : q \in Q \text{ and } (p, \epsilon, b, p', \gamma^T) \in \tau_c^T\} \\
\tau_r^B = &\{((q,p), \bar{b}, \bar{a}, (\gamma^A, \gamma^T), (q',p')) : (q, \bar{a}, \gamma^A, q') \in \tau_r^A \text{ and } (p, \bar{a}, \bar{b}, \gamma^T, p') \in \tau_r^T\} \cup \\
&\{((q,p), \epsilon, \bar{a}, (q',p')) : (q, \bar{a}, \gamma^A, q') \in \tau_r^A, a \neq \epsilon \text{ and } (p, \bar{a}, \epsilon, \gamma^T, p') \in \tau_r^T\} \cup \\
&\{((q,p), \bar{b}, \epsilon, (\dagger, \gamma^T), (q,p')) : q \in Q \text{ and } (p, \epsilon, \bar{b}, \gamma^T, p') \in \tau_r^T\}.
\end{aligned}$$

Now, it can be easily seen that for a given document w, we have $L_{w,K} = T_B(w)$.

8 Concluding Remarks

In this work, we have investigated the problem of approximate XML validation, an important problem in XML processing. Useful contributions of this paper include the introduction of VPTs and their application to building two algorithms for checking approximate XML validity in the streaming model. We also believe that the new semantics introduced in this paper is interesting in the context of XML and merits further investigation.

References

1. Aho, V., A., Peterson, G., T. A Minimum Distance Error-Correcting Parser for Context-Free Languages. *SIAM J. Comput.* 1(4): 1972, pp. 305–312.
2. Alur, R., Kumar. V., Madhusudan, P., and Viswanathan, M. Congruences for Visibly Pushdown Languages. In *Proc.* 32*nd International Colloquium of Automata, Languages and Programming (ICALP)* (Lisboa, Portugal, 11–15 July 2005), pp. 1102–1114.
3. Alur, R., and Madhusudan, P. Visibly Pushdown Languages. In *Proc.* 36*th ACM Symp. on Theory of Computing* (Chicago, Illinois, 13–15 June 2004), pp. 202–211.
4. Balmin, A., Papakonstantinou, Y., and Vianu, V. Incremental Validation of XML Documents. *ACM Trans. Database Syst.* 29(4): 2004, pp. 710–754.
5. Barbosa, D., Leighton, G., and Smith, A. Efficient Incremental Validation of XML Documents After Composite Updates. In *Proc. of* 2*nd Int. XML Database Symp.* (Seoul Korea, 10–11 September 2006), pp. 107–121.
6. Barbosa, D., Mendelzon, A. O., Libkin, L., Mignet, L., and Arenas, M. Efficient Incremental Validation of XML Documents. In *Proc. of* 20*th Int. Conf. on Data Engineering* (Boston, USA, 30 March–2 April 2004), pp. 671–682.
7. Boobna, U., and de Rougemont, M. Correctors for XML Data. In *Proc.* 2*nd International XML Database Symposium* (Toronto, Canada, 29–30 August 2004), pp. 97–111.

8. Clark, J., and M. Murata, M. RELAX NG Specification. OASIS, December 2001.
9. 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.
10. 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).
11. 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.
12. Grahne, G., and Thomo, A. Regular Path Queries under Approximate Semantics. *Ann. Math. Artif. Intell.* 46(1-2): 2006, pp. 165–190.
13. Green, T. J., Gupta, A., Miklau, G., Onizuka, M., and Suciu, A. Processing XML Streams with Deterministic Automata and Stream Indexes. *ACM Trans. Database Syst.* 29(4): 2004, pp. 752–788.
14. 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.
15. Neven, F. Automata Theory for XML Researchers. *SIGMOD Record* 31(3): 2002 pp. 39–46.
16. 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.
17. Papakonstantinou, Y., and Vianu, V. DTD Inference for Views of XML Data. In *Proc. 19th ACM Symp. on Principles of Database Systems* (Dallas, Texas, 15–17 May 2000), pp. 35–46.
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., 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.
21. Segoufin, L., and Sirangelo, C. Constant-Memory Validation of Streaming XML Documents Against DTDs. In *Proc. 11th International Conference on Database Theory* (Barcelona, Spain, 10–12 January 2007), pp. 299–313.
22. Suciu, D. The XML Typechecking Problem. In *SIGMOD record* 31(1), 2002, pp. 89–96.