Rewriting of Visibly Pushdown Languages for XML Data Integration

Alex Thomo
University of Victoria
British Columbia, Canada
thomo@cs.uvic.ca

S. Venkatesh
University of Victoria
British Columbia, Canada
venkat@cs.uvic.ca

ABSTRACT

In this paper, we focus on XML data integration by studying rewritings of XML target schemas in terms of source schemas. Rewriting is very important in data integration systems where the system is asked to find and assemble XML documents from the data sources and produce documents which satisfy a target schema.
As schema representation, we consider Visibly Pushdown Automata (VPAs) which accept Visibly Pushdown Languages (VPLs). The latter have been shown to coincide with the family of (word-encoded) regular tree languages which are the basis of formalisms for specifying XML schemas. Furthermore, practical semi-formal XML schema specifications (defined by simple pattern conditions on XML) compile into VPAs which are exponentially more concise than other representations based on tree automata.

Notably, VPLs enjoy a "well-behavedness" which facilitates us in addressing rewriting problems for XML data integration. Based on VPAs, we positively solve these problems, and present detailed complexity analyses.

1. INTRODUCTION

The Extensible Markup Language (XML) is the ubiquitous standard for representing data and documents on the Web and is used in a variety of domains ranging from collaborative commerce to medical databases. One of the most important applications of XML is data integration, where XML is used to structure or wrap data from multiple provider sources. Such sources contain diverse information and they present to the outside world a schema for the data they make available. A crucial problem in this setting is to be able to determine relevant sources and then combine them producing data which satisfy a given target-schema. This is exactly the focus of this paper.

Schemas for XML. In this paper, we will represent XML schemas by Visibly Pushdown Automata (VPAs) introduced in [2]. 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. Furthemore, VPLs have been shown to coincide with the class of (word-encoded) regular tree languages.

When it comes to (formal) XML schema specifications, the most popular ones are Document Type Definition (DTD), XML Schema ([16]) and Relax NG ([4]). Notably, all these schema formalisms can be captured by Extended Document Type Definitions (EDTDs) (cf. [13, 12, 14, 5]). It is well known that the tree languages specified by EDTDs coincide with (unranked) regular tree languages (cf. [5]). Recent work [11] has also shown that EDTDs can be directly compiled into VPAs.

For all the above reasons, working with VPAs makes our methods as general as possible with respect to the current XML schemas. Also, decision problems for VPAs have the same complexity as those for unranked tree automata, which have been the classical tool for representing EDTD-based schemas for XML.

On the other hand, when finally applying the automata on XML documents, by using VPAs, we do not have the overhead of building and storing the tree representation of documents, which is a price to pay when using tree automata. This is because VPAs are word automata (as opposed to tree ones), and XML documents are initially words (strings) before being possibly transformed into trees.

A stronger 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. Fleshing out the example of [1], to express that we want properly nested XML documents which contain elements $a_1, \ldots, a_n$ (in this order) we can specify the word language $L(\Sigma^*\langle a_1\rangle\Sigma^*\langle/a_1\rangle\Sigma^* \ \ldots \ \Sigma^*\langle a_n\rangle\Sigma^*\langle/a_n\rangle\Sigma^*) \cap PN$, where PN is the language of all properly nested words on Σ. Notably, this specification compiles into a deterministic VPA of linear size, but standard deterministic bottom-up tree automata for this specification must be of size exponential in n. Such approaches for specifying wanted documents are not uncommon or contrived examples in favor of VPAs. Rather, they are very practical and popular among users accustomed with regular expressions, and as such, embodied in important software such as the prominent .NET platform of Microsoft (cf. [6]).

Rewritings. In this paper, we study the following two families of target-rewriting/source-composition problem for

XML information integration:

> Given source schemas $S_1, \ldots, S_n$, and target schema T (all being VPLs), *rewrite* T in terms of $S_1, \ldots, S_n$ such that, the *source composition* induced by the rewriting is either
>
> 1. *relevant*, having a non empty intersection with T, or
> 2. *safe*, being completely contained in T.

To illustrate, suppose that we have three XML sources:

S_1 containing documents about books, with elements such as *title*, *author*, *pid* (publisher ID) etc.,

S_2 containing documents about books (as above) and also journals, with elements such as *journal-name*, *editor*, *pid* etc.,

S_3 containing documents about publishers, with elements such as *pid*, *publisher-name*, *address* etc.

Now, suppose that we also have a target schema specification T asking for (full) documents about books and their publishers. Let s_1, s_2 and s_3 be symbols in some "outer alphabet" representing the XML sources. Informally, these symbols are the source "id's" or "names." Then, $(s_1 + s_2)s_3$ would be a relevant rewriting, while $s_1 s_3$ would be a safe rewriting of T. After formally defining VPAs in Section 2, we give a detailed and more enhanced version of this example in Section 3.

Using a relevant rewriting for merging documents from different sources might create documents which might or might not fit a target schema. For example, using $(s_1+s_2)s_3$ might create not only book-publisher documents, but also journal-publisher documents, and clearly, the latter do not fit the target schema. On the other hand, using a safe rewriting, always creates documents which fit the target schema. In contrast, documents obtained by a relevant rewriting need an additional check for their validity, and this adds a data-complexity dimension in the data-integration problem under consideration.

Also, in order to perform document merges, one needs to specify join elements which will make the merge possible. For example, for documents from source S_1 and S_3, one needs to specify that the merge should be based on the equality of the *pid* contents. In this paper, we will assume that such join conditions are given for any meaningful pair of sources, and thus focus exclusively on the rewriting (reformulating) the target schema in terms of source schemas.

Orthogonally with being relevant or safe, the rewritings can be "complete" or "partial." A rewriting is complete when its words do not have "uncovered" XML data (text) placeholders, otherwise it is partial. For example the above rewritings, $(s_1 + s_2)s_3$ and $s_1 s_3$, are complete. On the other hand, if source S_3 is not available, then the best one can do is to compute partial rewritings $(s_1 + s_2)\rho$ and $s_1\sigma$, where ρ and σ represent languages (on the base alphabet) optimally chosen to satisfy the relevancy and safety, respectively, of the rewritings. By using partial rewritings we can obtain documents having data in some parts and dataless "skeletons" in some other parts. Depending on the availability of sources, the system can compute complete or partial rewritings.

Notably, for regular languages, the analogous problems have been positively solved. We mention here [10] and [3] which provide two different algorithms for computing the analog of safe rewriting for regular languages. On the other hand, [7] gives an algorithm for computing the analog of relevant rewriting for regular languages. Also, partial rewritings of regular languages have been studied in [8] and [9].

On the negative side, rewriting problems are unsolvable for context free languages (CFLs). This can be easily shown by reduction from the undecidable problems of non-emptiness of intersection for CFLs (for relevant rewritings) and inclusion for CFLs (for safe rewritings).

In this paper, we positively solve the rewriting problems for VPLs of properly nested words. We believe that our results are important not only from a database perspective, but also from a formal language one as they enrich the body of positive results for VPLs. Specifically, our contributions in this paper are as follows. Firstly, we discuss and formally define relevant and safe complete rewritings for XML, and present algorithms for computing these rewritings. Secondly, we study optimal partial rewritings presenting key properties and algorithms to compute them. Finally, we give detailed complexity analyses by showing upper and lower bounds for the rewriting problems that we consider.

Organization. The rest of the paper is organized as follows. In Section 2, we overview VPAs and VPLs. In Section 3, we illustrate and formally define rewritings. In Section 4 and 5, we study complete and partial rewritings, respectively. Section 6 concludes the paper.

2. VISIBLY PUSHDOWN AUTOMATA

VPAs were formally introduced in [2] 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) $\mathcal{A}$ 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]

[1]When referring to arbitrary elements of Σ_r, we will use

3. Γ is a finite stack alphabet, and $\perp \notin \Gamma$ is a special symbol for the bottom of the stack.

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:
$\tau_c \subseteq Q \times \Sigma_c \times Q \times \Gamma$, $\tau_r \subseteq Q \times \Sigma_r \times \Gamma \times Q$, $\tau_l \subseteq Q \times \Sigma_l \times Q$ and $\tau_\epsilon \subseteq Q \times \{\epsilon\} \times Q$

A run of a VPA $\mathcal{A}$, on a word $w = x_1 x_2 \dots x_k \in \Sigma^*$, is a sequence $\rho = (q_0, \sigma_0), (q_1, \sigma_1), \dots, (q_k, \sigma_k)$ where $q_i \in Q$ and $\sigma_i \in (\Gamma \backslash \{\perp\})^* \perp$, $\sigma_0 = \perp$ and for every $1 \leq i \leq k$, the following holds:

- If x_i is a call symbol, then for some $\gamma \in \Gamma$, $(q_{i-1}, x_i, q_i, \gamma) \in \tau$ and $\sigma_i = \gamma \sigma_{i-1}$.
- If x_i is a return symbol, then for some $\gamma \in \Gamma \backslash \{\perp\}$, $(q_i, x_i, \gamma, q_{i+1}) \in \tau$ and $\sigma_{i-1} = \gamma \sigma_i$.
- If x_i is a local symbol, then $(q_{i-1}, x_i, q_i) \in \tau$ and $\sigma_i = \sigma_{i-1}$.

A run $\rho = (q_0, \sigma_0), \dots, (q_k, \sigma_k)$ is accepting if $q_k \in F$ and $\sigma_k = \perp$. 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 $\mathcal{A}$ is denoted by $L(\mathcal{A})$. We remark 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.

We assume that schema VPAs do not contain local symbols[2] and that the data (text) can go wherever a call symbol meets a return symbol in the words accepted by such VPAs. Thus, if a word $w = ab\bar{b}\bar{a}$ is accepted by a schema VPA, then an XML document corresponding to w would be for example abCIKM$\bar{b}\bar{a}$, where C, I, K and M are local symbols. We note that, all our results can be easily modified to handle the case of schema VPAs containing local symbols (or wildcards for local symbols) as well.

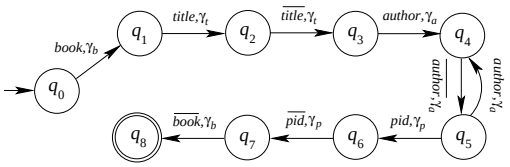

Figure 1: VPA $\mathcal{A}_1$.

EXAMPLE 1. *Suppose that we want to build a VPA accepting XML documents describing books. Such documents will have a* title *element, one or more* author *elements, and a publisher ID (pid) element. A VPA accepting well-formed documents of this structure is* $\mathcal{A}_1 = (Q, (\Sigma, f), \Gamma, \tau, q_0, F)$, *where*

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\},\\
\Sigma &= \Sigma_c \cup \Sigma_r\\
&= \{book, title, author, pid\} \cup \{\overline{book}, \overline{title}, \overline{author}, \overline{pid}\},\\
\Gamma &= \{\gamma_b, \gamma_t, \gamma_a, \gamma_p\},\\
F &= \{q_8\},\\
\tau &= \{(q_0, book, q_1, \gamma_b), (q_1, title, q_2, \gamma_t),\\
&\quad (q_3, author, q_4, \gamma_a), (q_5, pid, q_6, \gamma_p)\} \cup\\
&\quad \{(q_2, \overline{title}, \gamma_t, q_3), (q_4, \overline{author}, \gamma_a, q_5),\\
&\quad (q_6, \overline{pid}, \gamma_p, q_7), (q_7, \overline{book}, \gamma_b, q_8)\}.
\end{aligned}
$$

We show this VPA in Fig. 1. □

For simplicity, in the rest of the paper, we sometimes omit the state labels from the example VPAs.

3. REWRITINGS

We first introduce the rewritings for XML by means of an example and discuss in detail their properties.

Suppose that the VPA of Example 1 represents the schema of a source S_1 of XML documents. Also, suppose that we have two other sources as well, S_2 and S_3, with schema represented by VPA $\mathcal{A}_2$ and $\mathcal{A}_3$ given in Fig. 2. Source S_2 contains (XML) documents about books and journals, while S_3 contains documents about publishers.

Let $\Omega = \{s_1, s_2, s_3\}$ be an alphabet of *source names*; s_1, s_2 and s_3 are the names of the first, second and third source, respectively.

Now, let a target schema T be represented by VPA $\mathcal{A}$ shown in Fig. 3 [top]. The transitions labeled by $\#, \gamma_\#$ are wildcard transitions. This target schema asks for XML documents containing full information about books. Specifically, it asks for information not only about their title and authors, but also about their publishers.

It can be easily seen that we can compose sources S_1 or S_2 with S_3 to get documents that *might* fit the target schema T. Namely, a possible *rewriting* of T using S_1, S_2 and S_3 can be captured by the VPA in Fig. 3 [bottom-left]. This rewriting suggests joining together documents from S_1 with documents from S_3, or documents from S_2 with documents from S_3. While the former documents indeed fit the target schema, the latter ones "possibly" fit the target schema. This is because source S_2 provides not only documents about books but also documents about journals, and clearly, the target schema does not ask for journal data. Another rewriting is given in Fig. 3 [bottom-right]. This rewriting is *safe* since it only suggests joining together documents from S_1 with documents from S_3.

Observe that, in a rewriting, the source schemas do not need to completely "cover" all the parts of the target schema (for instance bp and $\overline{bp}$ are "uncovered" in our example). The role of such symbols is to add extra structure to the information supplied by the sources.

$\bar{a}, \bar{b}, \dots$ in order to emphasize that these elements correspond to $a, b, \dots$ elements of Σ_c.

[2]On the other hand, the VPAs for capturing rewritings will contain local symbols representing source names.

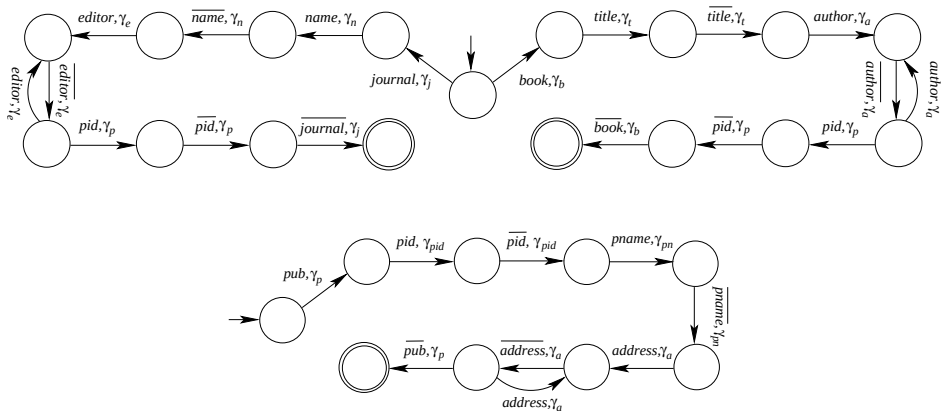

Figure 2: VPAs $\mathcal{A}_2$ [top] and $\mathcal{A}_3$ [bottom].

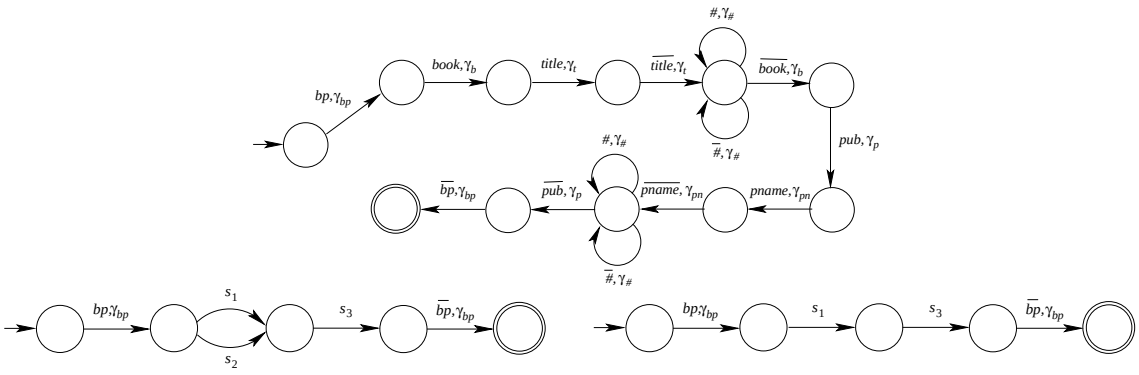

Figure 3: VPAs $\mathcal{A}$ [top], $\mathcal{A}'$ [bottom-left] and $\mathcal{A}''$ [bottom-right].

However, in a *complete* rewriting, we do not allow words with uncovered data placeholders. Specifically, we do not want to have in a complete rewriting words where an opening tag (call symbol) is immediately followed by a closing tag (return symbol), e.g. $a\overline{a}$. In such a case, the word is not able to pull data from the sources to cover that part of the target schema.

Depending on the availability of data sources, we can compute instead *partial* rewritings which extract the most possible out of the available sources. For example, if source S_3 is not available, we construct the partial rewritings given in Fig.4.

If we substitute the source VPLs for their names in a rewriting, we get the "expansion language" induced by the rewriting. Intuitively, this is the language of all the documents that the rewriting is able to possibly generate from the documents in the sources. The main property of the rewritings captured by VPAs $\mathcal{A}'$ and $\mathcal{A}'_p$ in Fig. 3 [bottom-left] and Fig.4 [top], respectively, is that their expansion languages have a non-empty intersection with the target language. On the other hand, the expansion languages of the rewritings captured by VPAs $\mathcal{A}''$ and $\mathcal{A}''_p$ in Fig. 3 [bottom-right] and Fig.4 [bottom], respectively, is that they are completely contained in the target language.

Finally, observe that in both rewritings, the source names are local symbols in the rewriting. This is because the source languages are VPLs of properly nested words.

3.1 Formalization

Here, we formalize the complete rewritings that we described above by example. For better readability we postpone the formalization of partial rewriting to Section 5. Let

1. T and $S_1, \ldots, S_n$ be the target and source languages, respectively,
2. $\Delta = \Delta_c \cup \Delta_r$ be the underlying XML alphabet of these languages,
3. $\Omega = \{s_1, \ldots, s_n\}$ be the alphabet (namely the set) of source names, and
4. Exp be a substitution defined on symbols, words and languages as follows:
   - $\text{Exp}(a) = a$ and $\text{Exp}(s_i) = S_i$, for $a \in \Delta$ and $s_i \in \Omega$, respectively,
   - $\text{Exp}(w) = w$, for $w \in \Delta^*$,
   - $\text{Exp}(w) = \{u_1\}S_{i_1}\{u_2\}\ldots\{u_p\}S_{i_p}\{u_{p+1}\}$, for $w = u_1 s_{i_1} u_2 \ldots u_p s_{i_p} u_{p+1}$, where $u_h \in \Delta^*$ and $s_{i_h} \in \Omega$, for $1 \leq h \leq p$, and
   - $\text{Exp}(L) = \bigcup_{w \in L} \text{Exp}(w)$, for $L \subseteq (\Delta \cup \Omega)^*$.

Now, we define the (complete) rewritings as follows.

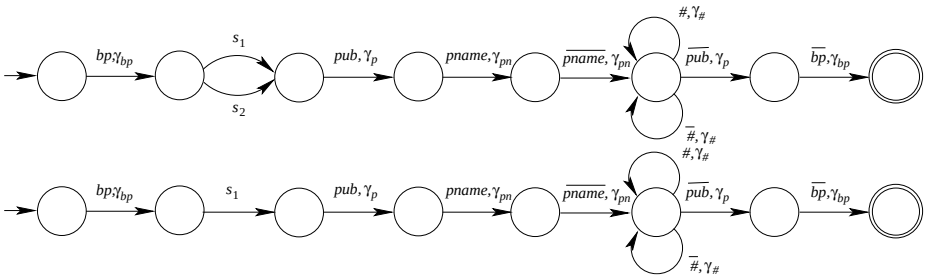

Figure 4: VPAs $\mathcal{A}'_p$ [top] and $\mathcal{A}''_p$ [bottom].

DEFINITION 1. *The* maximal relevant rewriting T' *is the set of* all *words* w *on* $\Delta_c \cup \Omega \cup \Delta_r$ *such that*

1. $\text{EXP}(w) \cap T \neq \emptyset$
2. w *does not have an* $a\bar{a}$ *subword for any* $a \in \Delta_c$.

DEFINITION 2. *The* maximal safe rewriting T'' *is the set of* all *words* w *on* $\Delta_c \cup \Omega \cup \Delta_r$ *such that*

1. $\text{EXP}(w) \subseteq T$, *and*
2. w *does not have an* $a\bar{a}$ *subword for any* $a \in \Delta_c$.

The second condition in the above definitions asks for the "completeness" of rewritings. As we show in the next section, both T' and T'' are VPLs with Ω as their alphabet of local symbols. Clearly, $T'' \subseteq T'$.

4. COMPUTING THE COMPLETE REWRITINGS

Let $\mathcal{A}$ and $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be (nondeterministic) VPAs for the target and source schema languages, T and $S_1, \ldots, S_n$, respectively. As these languages represent XML structural schemas, they do not have local symbols, but only call and return symbols.

4.1 Relevant Rewriting

Let specifically, $\mathcal{A} = (Q, (\Delta_c \cup \Delta_r, f), \Gamma, \tau, q_0, F)$, where $Q = \{q_0, q_1, \ldots, q_m\}$.

We denote by $\mathcal{A}_{ij}$, for $0 \leq i, j \leq m$, the VPA obtained by $\mathcal{A}$ making states q_i and q_j initial and final, respectively. Formally, $\mathcal{A}_{ij} = (Q, (\Delta_c \cup \Delta_r, f), \Gamma, \tau, q_i, \{q_j\})$. From VPA $\mathcal{A}$ we construct automaton

$$\mathcal{A}' = (Q, (\Delta_c \cup \Omega \cup \Delta_r, f), \Gamma, \tau', q_0, F),$$

where $\tau' = \tau \cup \{(q_i, s_k, q_j) : S_k \cap L(\mathcal{A}_{ij}) \neq \emptyset \text{ for } 0 \leq k \leq n\}$.

As S_k and $L(\mathcal{A}_{ij})$ are VPLs, the non-emptiness of their intersection can be decided in polynomial time (see [2]). VPA $\mathcal{A}'$, in contrast to $\mathcal{A}$ and $\mathcal{A}_1, \ldots, \mathcal{A}_n$, does have local symbols. They are the source names, i.e. the Ω elements. For VPA $\mathcal{A}'$, we show that

THEOREM 1. $\mathcal{A}'$ *accepts* all *and* only *the words* w *on* $\Delta_c \cup \Omega \cup \Delta_r$ *such that* $\text{EXP}(w) \cap T \neq \emptyset$.

Before presenting the proof, we illustrate the construction of $\mathcal{A}'$ with the following example.

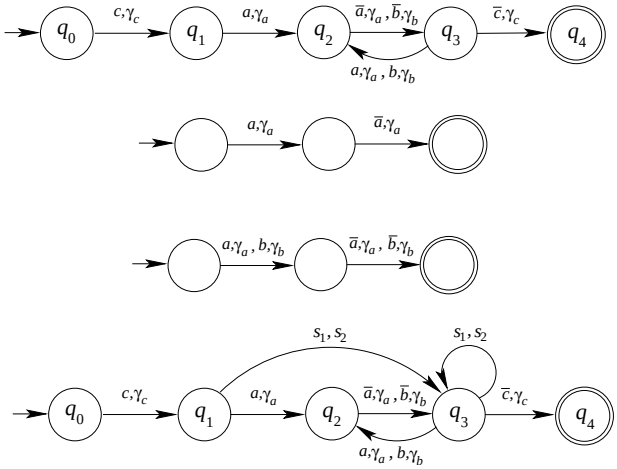

Figure 5: VPAs $\mathcal{A}$ [first], $\mathcal{A}_1$ [second], $\mathcal{A}_2$ [third] and $\mathcal{A}'$ [fourth].

EXAMPLE 2. *Suppose that we have the target schema captured by the VPA in Fig. 5 [first] and the source schemas captured by the VPAs in the same figure [second] and [third]. Then the constructed VPA* $\mathcal{A}'$ *is the one given in the bottom of the figure.* □

Based on Theorem 1 and Definition 2, for the maximal relevant rewriting T', we have that

COROLLARY 1. $T' = L(\mathcal{A}') \cap M^c$, *where* $M = (\Delta_c \cup \Omega \cup \Delta_r)^* \cdot \{a\bar{a} : a \in \Delta_c\} \cdot (\Delta_c \cup \Omega \cup \Delta_r)^*$.

Now, we give the proof of Theorem 1.

Proof of Theorem 1.

"*all*" Let $w \in (\Delta_c \cup \Omega \cup \Delta_r)^*$ such that $\text{EXP}(w) \cap T \neq \emptyset$. Suppose that w has some Ω symbol in it; otherwise the claim follows trivially. As such, $w = u_1 s_{i_1} u_2 \ldots u_p s_{i_p} u_{p+1}$, for some positive integer p and $u_i \in (\Delta_c \cup \Delta_r)^*$, for $1 \leq i \leq p+1$. Now, there exist words $v_{i_1}, \ldots, v_{i_p}$ in $S_{i_1}, \ldots, S_{i_p}$, respectively, such that $u_1 v_{i_1} u_2 \ldots u_p v_{i_p} u_{p+1}$ is accepted by $\mathcal{A}$.

Let $q_0, q_{i_1}, q_{j_1} \ldots, q_{i_p}, q_{j_p}, q_f$ be a subsequence of states that $\mathcal{A}$ goes on for accepting $u_1 v_{i_1} u_2 \ldots u_p v_{i_p} u_{p+1}$. Specifically,

$\mathcal{A}$ starts in q_0 (with an empty stack), is in q_{i_1} after reading u_1, then goes to q_{j_1} after reading v_{i_1}, and so on. Since, words v_{i_h}, for $1 \leq h \leq p$, are properly nested, and $\mathcal{A}$ is a VPA, $\mathcal{A}$ will have the same stack when being in both q_{i_h} and q_{j_h}.

Hence, if $\mathcal{A}$ starts consuming word v_{i_h} from state q_{i_h} on an empty stack (behaving as $\mathcal{A}_{i_h,j_h}$), it will eventually reach state q_{j_h} with an empty stack as well. This means that $v_{i_h} \in L(\mathcal{A}_{i_h,j_h})$. The latter implies that $S_{i_h} \cap L(\mathcal{A}_{i_h,j_h}) \neq \emptyset$, which is the condition for having an s_{i_h}-transition from q_{i_h} to q_{j_h}. All this holds for $1 \leq h \leq p$. Thus, we have that $w = u_1 s_{i_1} u_2 \dots u_p s_{i_p} u_{p+1}$ is a word accepted by $\mathcal{A}'$, and this proves our claim.

"*only*" Let $w \in L(\mathcal{A}')$. As such, w will take $\mathcal{A}'$ from initial state q_0 to some final state q_f starting with an empty stack and ending again with an empty stack. Suppose that w has some Ω symbol in it, otherwise its "expansion" has trivially a non-empty intersection with T.

As before, word w can be written as $u_1 s_{i_1} u_2 \dots u_p s_{i_p} u_{p+1}$, for some positive integer p and $u_i \in (\Delta_c \cup \Delta_r)^*$, for $1 \leq i \leq p+1$. Let $q_0, q_{i_1}, q_{j_1} \dots, q_{i_p}, q_{j_p}, q_f$ be a subsequence of states that $\mathcal{A}'$ goes for accepting w. Specifically, $\mathcal{A}'$ starts in q_0 (with an empty stack), is in q_{i_1} after reading u_1, then hops to q_{j_1} after reading s_{i_1}, and so on. Recall that $s_{i_1}, \dots, s_{i_p}$ are local symbols for $\mathcal{A}'$, and thus, the stack remains "as is" upon reading them.

Now, by the construction of $\mathcal{A}'$ there exist words $v_{i_1}, \dots, v_{i_p}$ in $S_{i_1}, \dots, S_{i_p}$, respectively, such that v_{i_h}, for $1 \leq h \leq p$, takes automaton $\mathcal{A}$ from q_{i_h} to q_{j_h} starting and ending with the same stack. From this and from the fact that $\mathcal{A}'$ accepts $u_1 s_{i_1} u_2 \dots u_p s_{i_p} u_{p+1}$ through $q_0, q_{i_1}, q_{j_1} \dots, q_{i_p}, q_{j_p}, q_f$, we have that $\mathcal{A}$ accepts $u_1 v_{i_1} u_2 \dots u_p v_{i_p} u_{p+1}$ going through the same subsequence of states. As $u_1 v_{i_1} u_2 \dots u_p v_{i_p} u_{p+1}$ is in $\textsc{Exp}(w)$, we have that $\textsc{Exp}(w) \cap T \neq \emptyset$. $\square$

Next, we show the following theorem for the complexity of computing the maximal relevant rewriting (T') of a target VPL T in terms of source VPLs $S_1, \dots, S_n$.

THEOREM 2. *Computing T' can be done in polynomial time.*

Proof. To construct VPA $\mathcal{A}'$ takes polynomial time. Then, we need to take the intersection with the complement of the mask $M = (\Delta_c \cup \Omega \cup \Delta_r)^* \cdot \{a\bar{a} : a \in \Delta_c\} \cdot (\Delta_c \cup \Omega \cup \Delta_r)^*$.

One can build an NFA for M which has $O(|\Delta_c|)$ states. Thus, computing the complement of M can be done in exponential time in the size of Δ_c. However, since $L(\mathcal{A}')$ is a VPL of properly nested words, T' can be computed instead by intersecting $L(\mathcal{A}')$ with the complement of

$$M' = (\Delta_c \cup \Omega \cup \Delta_r)^* \cdot \{a\bar{b} : a \in \Delta_c \text{ and } \bar{b} \in \Delta_r\} \cdot (\Delta_c \cup \Omega \cup \Delta_r)^*.$$

We can always construct an NFA recognizing M' which has only three states, regardless of the alphabet size. So, we can construct a DFA for M' which has not more than 8 states, regardless of the alphabet size. By changing the final states to non-final and vice versa, we obtain a DFA for M'^c. As this DFA has not more than 8 states, we can compute $L(\mathcal{A}') \cap M'^c$ in polynomial time. $\square$

4.2 Safe Rewriting

For computing the safe rewriting, we first determinize VPA $\mathcal{A}$ obtaining a deterministic VPA accepting the same language. For notational simplicity let us denote by the same "notational signature" the deterministic variant of $\mathcal{A}$, i.e. $(Q, (\Delta_c \cup \Delta_r, f), \Gamma, \tau, q_0, F)$. This is done for the ease of notations only, and one should not confuse this automaton with the one in the previous subsection.

As in [2], for deterministic VPA $\mathcal{A}$ we have that

- for every $a \in \Delta_c$ there is at most one $(q, a, q', \_)$ transition, and
- for every $\bar{a} \in \Delta_r$ and $\gamma \in \Gamma$ there is at most one $(q, \bar{a}, q', \gamma)$ transition.

As VPA $\mathcal{A}$ captures an XML schema, it does not have transitions with local symbols. However, in general when a VPA does have such transitions, there is a third condition for a VPA to be deterministic:

- for every local symbol a there is at most one (q, a, q') transition.

We mention this third condition as the VPAs we construct for the safe rewritings are deterministic and they have transitions with local symbols.

In addition to the above conditions (listed in [2]), here we also require that deterministic VPAs do not get stuck in any state for any symbol when reading a properly nested word. The latter can be achieved by adding a "garbage state."

Specifically, let q_g be a new state which we add to $\mathcal{A} = (Q, (\Delta_c \cup \Delta_r, f), \Gamma, \tau, q_0, F)$. Also, let $\Gamma' = \{\gamma_a : a \in \Delta_c\}$ and $\Gamma' \cap \Gamma \neq \emptyset$. Then we add the following transitions to τ:

- (q, a, q_g, γ_a) for each $q \in Q$ and $a \in \Delta_c$ such that there is no $(q, a, \_, \_)$ transition in τ,
- $(q, \bar{a}, \gamma, q_g)$ for each $q \in Q$, $\bar{a} \in \Delta_r$ and $\gamma \in \Gamma$ such that there is no $(q, \bar{a}, \gamma, \_)$ transition in τ,
- (q_g, a, q_g, γ_a) for each $a \in \Delta_c$, and
- (q_g, a, q_g, γ) for each $a \in \Delta_c$ and $\gamma \in \Gamma \cup \Gamma'$.

Formally, we have

FACT 1. *Given any properly nested word w, VPA $\mathcal{A}$, starting from any state with an empty stack, is able to (deterministically) fully consume w and finish with with an empty stack as well.*

Now, on this deterministic VPA $\mathcal{A}$ we perform the construction of the previous subsection and obtain $\mathcal{A}'$.

Language-wise, an $\mathcal{A}'$ built on a deterministic $\mathcal{A}$ is the same as an $\mathcal{A}'$ built on a non-deterministic $\mathcal{A}$. Also, $\mathcal{A}'$ is non-deterministic no matter whether the base VPA $\mathcal{A}$ is deterministic or not. This is because the $S_{\_}$ languages have in general more than one word, and thus, from a given state, $\mathcal{A}'$ can jump to more than one state with $s_{\_}$-labeled transitions. However, the structure of an $\mathcal{A}'$ built on a deterministic $\mathcal{A}$ is different from that of an $\mathcal{A}'$ built on a non-deterministic $\mathcal{A}$.

This is because a deterministic $\mathcal{A}$ (with a garbage state as above) is able to fully consume a given properly nested word and does that in only one way. From this and the construction of $\mathcal{A}'$, if a properly nested word $w \in (\Delta_c \cup \Omega \cup \Delta_r)^*$ is able to take $\mathcal{A}'$ from the initial state to a *non-final* state (starting and ending with an empty stack), then there exists $u \in \textsc{Exp}(w)$, such that u is not accepted by $\mathcal{A}$. [On the other hand, if $\mathcal{A}'$ is built on a non-deterministic $\mathcal{A}$ this could happen even if u is accepted by $\mathcal{A}$. This happens when word u is able to take $\mathcal{A}$, from the initial state, not only to a final state, but also to some non-final one. Clearly, this cannot happen with a deterministic VPA.]

On the contrary, if word $w \in (\Delta_c \cup \Omega \cup \Delta_r)^*$ takes $\mathcal{A}'$ from the initial state to final states only, then $\textsc{Exp}(w) \subseteq L(\mathcal{A}) = T$.

Conversely, let word $w \in (\Delta_c \cup \Omega \cup \Delta_r)^*$ have the property that $\textsc{Exp}(w) \subseteq L(\mathcal{A}) = T$. From this and the fact that the $S_{\_}$ languages are properly nested, we have that w is propely nested. As $\mathcal{A}'$, by construction, does not get stuck on properly nested words, it is able to consume w, starting from the initial state with an empty stack and ending in a number of states again with empty stack. We claim that all these ending states are final. Suppose not; i.e. there exists one of those states, say q, which is not final. By the construction of $\mathcal{A}'$, we have that there exists word $u \in \textsc{Exp}(w)$ that can take base automaton $\mathcal{A}$ from state q_0 to q. Since base automaton $\mathcal{A}$ is deterministic, there is only one way to consume u and this implies that u cannot take $\mathcal{A}$ from q_0 to some other (than q) state which would "hopefully" be final. Thus, u is not accepted by $\mathcal{A}$, i.e. $u \notin T$, and this is a contradiction.

Hence, for VPA $\mathcal{A}'$ constructed as above [on a deterministic base VPA $\mathcal{A}$ that does not get stuck on properly nested words], and a word $w \in (\Delta_c \cup \Omega \cup \Delta_r)^*$, we have that

THEOREM 3. *$w \in T''$ if and only if upon reading w, $\mathcal{A}'$ reaches only final states.*

From the above theorem, in order to compute the safe rewriting we need to extract *all* the words which take $\mathcal{A}'$ from its initial state to only final states. For this, we use on $\mathcal{A}'$ the determinization procedure of [2]. This procedure is applied "as is" on $\mathcal{A}'$ to obtain a deterministic variant of it. The procedure, among other information, also encodes in a state, say p, of the newly created VPA $\mathcal{A}''$, the subset of states, say Q_p, that the VPA $\mathcal{A}'$ reaches upon reading the words that take $\mathcal{A}''$ from its initial state to p. The condition of [2] for designating a state as final in the output VPA is that the corresponding subset of states (from the input VPA) has some final state in it. Specifically, state p, according to [2], would be designated as final if $Q_p \cap F \neq \emptyset$, where F is the set of final states in $\mathcal{A}'$. Now instead of this condition, we will require that a state p in $\mathcal{A}''$ is final only if its $Q_p \subseteq F$. One can verify that for VPA $\mathcal{A}''$ obtained in this way, we have that

THEOREM 4. *$\mathcal{A}''$ accepts* all *and* only *the words in $(\Delta_c \cup \Omega \cup \Delta_r)^*$ which can take $\mathcal{A}'$ from its initial state to final states only.*

From this and Theorem 3, we have that

COROLLARY 2. *$\mathcal{A}''$ accepts* all *and* only *the words w on $\Delta_c \cup \Omega \cup \Delta_r$ such that $\textsc{Exp}(w) \subseteq T$.*

From this and Definition 2, we finally have that

COROLLARY 3. *$T'' = L(\mathcal{A}'') \cap M^c$, where $M = (\Delta_c \cup \Omega \cup \Delta_r)^* \cdot \{a\bar{a} : a \in \Delta_c\} \cdot (\Delta_c \cup \Omega \cup \Delta_r)^*$.*

Next, we show the following theorem which establishes an upper bound for computing the maximal safe rewriting T''.

THEOREM 5. *Computing T'' can be done in doubly exponential time.*

Proof. Let us refer to the construction of this section for computing T''. Computing a deterministic VPA $\mathcal{A}$ needs exponential time (see [2]). Computing VPA $\mathcal{A}''$ needs exponential time in the size of $\mathcal{A}$, and this size might be exponential. Intersecting with mask M is absorbed by the complexity of the previous steps. Thus, in total, we need doubly exponential time in the size of a non-deterministic VPA for the target language. □

A matching lower bound can be derived from the 2EXPTIME lower bound for the analogous problem for regular languages [3].

REMARK 1. *In general, regular languages and VPLs of* properly nested words *are incomparable families of languages. However, we can encode any regular language L on some alphabet Σ as a VPL L' of properly nested words over alphabet $\Sigma \cup \Sigma'$, where $\Sigma' = \{\bar{a} : a \in \Sigma\}$, by taking $L' = \{a_1\bar{a}_1 \ldots a_p\bar{a}_p : a_1 \ldots a_p \in L\}$. Thus, any lower bound for problems on regular languages carries over VPLs of properly nested words.*

Thus, we formally state that

THEOREM 6. *Computing T'' is 2EXPTIME-hard.*

5. PARTIAL REWRITINGS

Here we study partial rewritings. Specifically, some words in the target language could only "partially be rewritten" in terms of the sources. Nevertheless, such words are able to pull data from the sources, albeit partial. For example, suppose that the target language has a word $a\bar{a}b\bar{b}c\bar{c}d\bar{d}$, but we only have available sources $S_1 = \{a\bar{a}\}$, $S_2 = \{b\bar{b}\}$ and $S_3 = \{d\bar{d}\}$. Then, the best we can do is to partially rewrite the target word as $s_1 s_2 c\bar{c} s_3$.

We can observe that $s_1 s_2 c\bar{c} s_3$ is an "optimal partial rewriting word" in that we cannot rewrite $a\bar{a}b\bar{b}c\bar{c}d\bar{d}$ any further using the given sources. On the other hand, $s_1 b\bar{b}c\bar{c}s_3$ is not optimal as it can be further rewritten into $s_1 s_2 c\bar{c} s_3$.

DEFINITION 3. *A word $w \in (\Delta_c \cup \Omega \cup \Delta_r)^*$, such that $\text{EXP}(w) \cap T \neq \emptyset$, is* type-one non-optimal *if it has some subword which belongs to S_i, for some $i \in [1, n]$.*

Intuitively, such a word $w = w_1 w_2 w_3$, where $w_2 \in S_i$, can be further rewritten in terms of the sources. Specifically, it can be rewritten as $w_1 s_i w_3$. This is because

$$\text{EXP}(w_1 s_i w_3) = \text{EXP}(w_1) \cdot S_i \cdot \text{EXP}(w_3) \supseteq$$

$$\text{EXP}(w_1) \cdot \{w_2\} \cdot \text{EXP}(w_3) = \text{EXP}(w),$$

and thus $\text{EXP}(w_1 s_i w_3) \cap T \neq \emptyset$ (since $\text{EXP}(w) \cap T \neq \emptyset$).

All the words that do not satisfy Definition 3 are called *type-one optimal*.

Certainly, when it comes to partial rewritings, we need to compute all the optimal rewriting words and filter out those that are non optimal. Formally, for target and source VPLs, T and $S_1, \ldots, S_n$, respectively, we define

DEFINITION 4. *The* maximal relevant partial rewriting (T'_p) *is the set of* all *type-one optimal words on $\Delta_c \cup \Omega \cup \Delta_r$.*

Now, we define

DEFINITION 5. *A word $w \in (\Delta_c \cup \Omega \cup \Delta_r)^*$, such that $\text{EXP}(w) \subseteq T$, is* type-two non-optimal *iff there exist w_1, w_2 and w_3, such that $w = w_1 w_2 w_3$, $w_1, w_3 \in (\Delta_c \cup \Omega \cup \Delta_r)^*$, $w_2 \in S_i$ for some $i \in [1, n]$ and $\text{EXP}(w_1) \cdot S_i \cdot \text{EXP}(w_3) \subseteq T$.*

Intuitively, such a word $w = w_1 w_2 w_3$, where $w_2 \in S_i$, can be further rewritten in terms of the sources. Specifically, it can be rewritten as $w_1 s_i w_3$. This is because

$$\text{EXP}(w_1 s_i w_3) = \text{EXP}(w_1) \cdot S_i \cdot \text{EXP}(w_3) \subseteq T.$$

All the words that do not satisfy Definition 5 are called *type-two optimal*.

DEFINITION 6. *The* maximal safe partial rewriting (T''_p) *is the set of* all *type-two optimal words on $\Delta_c \cup \Omega \cup \Delta_r$.*

We show the following relationships between the rewritings introduced so far.

THEOREM 7. *(a) $T' \subseteq T'_p$, (b) $T'' \subseteq T''_p$ and (c) $T''_p \subseteq T'_p$.*

Proof.

(a) Let $w \in T'$. Then $\text{EXP}(w) \cap T \neq \emptyset$. Now suppose that w is not in T'_p. Then, w can be written as $w_1 w_2 w_3$, such that $w_2 \in S_i$, for some $i \in [1, n]$. Since S_i is a VPL of properly nested words, w_2 is properly nested as well. As such, w_2 contains at some point a call symbol immediately followed by a return symbol. Thus, $w \notin T'$, and this is a contradiction.

(b) Similar to (a).

(c) Direct from definitions. □

5.1 Maximal Relevant Partial Rewriting

We can compute T'_p by first constructing VPA $\mathcal{A}'$ as in Subsection 4.1 and then intersecting with the *complement* of mask

$$M_p = (\Delta_c \cup \Omega \cup \Delta_r)^* \cdot (S_1 \cup \ldots \cup S_n) \cdot (\Delta_c \cup \Omega \cup \Delta_r)^*.$$

We capture this construction with the following statement.

THEOREM 8. $T'_p = L(\mathcal{A}') \cap (M_p)^c$.

Thus, regarding an upper bound for the maximal relevant partial rewriting we have

THEOREM 9. *Computing T'_p can be done in exponential time.*

Proof. Computing VPA $\mathcal{A}'$ is polynomial. Intersecting with the complement of mask M_p is exponential. □

Regarding the lower bound, we show that computing T'_p is PSPACE-hard. For this, we start by the following theorem.

THEOREM 10. *Deciding the emptiness of $T \cap T'_p$ is PSPACE-hard.*

Proof. We show this via a reduction from the following PSPACE-complete problem given in [8].

> Given an alphabet Σ and regular languages K and L, deciding the emptiness of $K \cap (\Sigma^* \cdot L \cdot \Sigma^*)^c$ is PSPACE-complete.

As we showed in Remark 1 (in Section 4.2), although regular languages and VPLs of *properly nested words* are incomparable families of languages, we can easily carry over any lower bound for problems on regular languages to analogous problems for VPLs of properly nested words. In particular, if K

and L, in the above problem, are VPLs of properly nested words on $\Sigma \cup \Sigma'$, then the problem becomes PSPACE-hard. Specifically, we have

> Given an alphabet $\Sigma \cup \Sigma'$ and VPLs K and L of properly nested words, deciding the emptiness of $K \cap [(\Sigma \cup \Sigma')^* \cdot L \cdot (\Sigma \cup \Sigma')^*]^c$ is PSPACE-hard.

Now, our reduction is as follows. Take $\Delta_c = \Sigma$, $\Delta_r = \Sigma'$, target $T = K$ and single source $S = L$. By the construction of VPA $\mathcal{A}'$ (see Subsection 4.1), $T \subseteq L(\mathcal{A}')$. Based on Theorem 8, we have that

$$T'_p = L(\mathcal{A}') \cap [(\Delta_c \cup \{s\} \cup \Delta_r)^* \cdot S \cdot (\Delta_c \cup \{s\} \cup \Delta_r)^*]^c.$$

Thus, $T \cap T'_p = T \cap [(\Delta_c \cup \{s\} \cup \Delta_r)^* \cdot S \cdot (\Delta_c \cup \{s\} \cup \Delta_r)^*]^c = T \cap [(\Delta_c \cup \Delta_r)^* \cdot S \cdot (\Delta_c \cup \Delta_r)^*]^c$.
The second equality follows from the fact that $T \subseteq (\Delta_c \cup \Delta_r)^*$. Finally, it is clear that,
$K \cap [(\Sigma \cup \Sigma')^* \cdot L \cdot (\Sigma \cup \Sigma')^*]^c$ is empty if and only if $T \cap T'_p$ is empty. □

From the above theorem, we can conclude that computing the maximal partial relevant rewriting is PSPACE-hard. This is because, given T'_p, we can test for the emptiness of $T \cap T'_p$ in time polynomial in the size of T'_p (the size of T is absorbed by the size of T'_p). Thus, our algorithm for computing T'_p is optimal, under the assumption that PSPACE $\not\subseteq$ SUB-EXPTIME.

5.2 Maximal Safe Partial Rewriting

On the other hand, computing the maximal partial safe rewriting T''_p is more complicated. We first compute VPA $\mathcal{A}''$ as in Subsection 4.2. Recall that this VPA accepts all (and only) the words w in $(\Delta_c \cup \Omega \cup \Delta_r)^*$ such that $\textsc{Exp}(w) \subseteq T$. From this and the definition of T''_p, it follows that $T''_p \subseteq L(\mathcal{A}'')$. Thus, we need to find a mask to filter out unwanted words, which are those that still have subwords that can be rewritten in terms of the sources. It might seem that the above mask M_p would do the job. However this is not true as the following example illustrates.

Example 3. *Let $T = \{ea\bar{a}\bar{e}b\bar{b}d\bar{d}\}$ and $S_1 = \{a\bar{a}\}$, $S_2 = \{b\bar{b}, c\bar{c}\}$ and $S_3 = \{d\bar{d}\}$. Then, $\mathcal{A}''$ will accept the language*

$$\{ea\bar{a}\bar{e}b\bar{b}d\bar{d}, es_1\bar{e}b\bar{b}d\bar{d}, ea\bar{a}\bar{e}b\bar{b}s_3, es_1\bar{e}b\bar{b}s_3\}$$

and by intersecting with the complement of

$$(\Delta_c \cup \Omega \cup \Delta_r)^* \cdot (S_1 \cup S_2 \cup S_3) \cdot (\Delta_c \cup \Omega \cup \Delta_r)^*$$

we only get the empty set. What we want is to filter out the (type-two) non-optimal words $ea\bar{a}\bar{e}b\bar{b}d\bar{d}$, $es_1\bar{e}b\bar{b}d\bar{d}$, $ea\bar{a}\bar{e}b\bar{b}s_3$ and compute $T''_p = \{es_1\bar{e}b\bar{b}s_3\}$. □

In order to build the aforementioned mask, we need to be able to detect words in $L(\mathcal{A}'')$ that can be further rewritten.

Now, let $\Omega' = \{s'_1, \ldots, s'_n\}$ be another alphabet of source names, for which we require $\Omega' \cap \Omega = \emptyset$. We build VPA $\mathcal{A}'''$ on $\mathcal{A}''$ in exactly the same way as we built $\mathcal{A}''$ on $\mathcal{A}$, but using Ω' as the set of view names. Recall that $\mathcal{A}''$ is already deterministic (see Subsection 4.2), so we do not need to determinize it first.

In order to state the key property of $\mathcal{A}'''$, let us first define substitution $\textsc{Exp}'$ on $\Delta_c \cup \Omega \cup \Omega' \cup \Delta_r$ as $\textsc{Exp}'(s'_i) = S_i$ and $\textsc{Exp}'(s_i) = s_i$, for $1 \leq i \leq n$, and $\textsc{Exp}'(a) = a$, for $a \in \Delta_c \cup \Delta_r$. Also, we extend $\textsc{Exp}'$ to words and languages similarly as in Section 3.

As in Theorem 2, we can show that

> VPA $\mathcal{A}'''$ accepts *all* and *only* the words w on $\Delta_c \cup \Omega \cup \Omega' \cup \Delta_r$ such that $\textsc{Exp}'(w) \subseteq L(\mathcal{A}'')$.

Now, consider language

$$N = L(\mathcal{A}''') \cap (\Delta_c \cup \Omega \cup \Omega' \cup \Delta_r)^* \cdot \Omega' \cdot (\Delta_c \cup \Omega \cup \Omega' \cup \Delta_r)^*.$$

This language contains all the words w in $L(\mathcal{A}''')$ that have some symbol from Ω'. By the construction of $\mathcal{A}'''$, we have that for such words w, $\textsc{Exp}'(w) \subseteq L(\mathcal{A}'')$. Formally, we show that

Theorem 11. *$\textsc{Exp}'(N)$ is the set of all (type-two) non-optimal words in $L(\mathcal{A}'')$.*

Proof. Let $w \in \textsc{Exp}'(N)$. As such, there exists word $v \in N$ whose expansion includes w, i.e. $w \in \textsc{Exp}'(v)$. Since $v \in N$, $v = v_1 s'_i v_2 \in L(\mathcal{A}''')$ and $\textsc{Exp}'(v_1) \cdot S_i \cdot \textsc{Exp}'(v_2) \subseteq L(\mathcal{A}'')$, for some $i \in [1, n]$, and $v_1, v_2 \in (\Delta_c \cup \Omega \cup \Omega' \cup \Delta_r)^*$. From this, we have that $w \in \textsc{Exp}'(v) = \textsc{Exp}'(v_1) \cdot S_i \cdot \textsc{Exp}'(v_2)$. As $\textsc{Exp}'(v_1), \textsc{Exp}'(v_2) \subseteq (\Delta_c \cup \Omega \cup \Delta_r)^*$, w is non-optimal.

Conversely, let $w \in (\Delta_c \cup \Omega \cup \Delta_r)^*$ be a non-optimal word. As such, there exist $w_1, w_2, w_3 \in (\Delta_c \cup \Omega \cup \Delta_r)^*$ such that $w = w_1 w_2 w_3$, $w_2 \in S_i$ and $\textsc{Exp}(w_1) \cdot S_i \cdot \textsc{Exp}(w_3) \subseteq T$, for some $i \in [1, n]$. Since $L(\mathcal{A}'')$ is the set of *all* words in $(\Delta_c \cup \Omega \cup \Delta_r)^*$ whose expansion is contained in T, we have that $\{w_1\} \cdot S_i \cdot \{w_3\} \in L(\mathcal{A}'')$. From the construction of $\mathcal{A}'''$, $w_1 s'_i w_3 \in L(\mathcal{A}''')$, and furthermore, $w_1 s'_i w_3 \in N$. Finally, our claim follows since $w \in \textsc{Exp}'(w_1 s'_i w_3)$. □

From all the above we conclude that

Corollary 4. $T''_p = L(\mathcal{A}'') \cap (\textsc{Exp}'(N))^c$.

Using the above construction we get a quad-exponential time upper bound. For this observe that computing VPA $\mathcal{A}'''$ is triple exponential. Computing N by intersecting with $(\Delta_c \cup \Omega \cup \Omega' \cup \Delta_r)^* \cdot \Omega' \cdot (\Delta_c \cup \Omega \cup \Omega' \cup \Delta_r)^*$ is polynomial in the size of $\mathcal{A}'''$. Obtaining $\textsc{Exp}'(N)$ is polynomial in the size of $\mathcal{A}'''$. Computing $(\textsc{Exp}'(N))^c$ and finally producing T''_p is exponential in the size of $\mathcal{A}'''$, for a total of quad-exponential time.

However, with the following theorem we show how to obtain a triple exponential time upper bound by computing $\mathcal{A}'''$ more efficiently.

THEOREM 12. *Computing T''_p can be done in triply exponential time.*

Proof. We can obtain $\mathcal{A}'''$ in only doubly exponential time, thus having in total triply-exponential time instead. For this, from VPA $\mathcal{A}''$, we construct in polynomial time VPA $\mathcal{B}$ which is the same as $\mathcal{A}''$, but also has for each transition $(\_, s_i, \_)$, where $1 \leq i \leq n$, an (additional) transition $(\_, s'_i, \_)$. We show that

LEMMA 1. $L(\mathcal{B}) = L(\mathcal{A}''')$.

Proof. Recall that VPA $\mathcal{A}'''$ accepts *all* and *only* the words w on $\Delta_c \cup \Omega \cup \Omega' \cup \Delta_r$ such that $\text{EXP}'(w) \subseteq L(\mathcal{A}'')$.

"$\subseteq$" Let $w \in L(\mathcal{B})$ and suppose that w has some Ω or Ω' symbol in it; otherwise the claim follows immediately. As such, $w = u_1 \hat{s}_{i_1} u_2 \ldots u_p \hat{s}_{i_p} u_{p+1}$, for some positive integer p, and where $u_h \in (\Delta_c \cup \Delta_r)^*$, for $1 \leq h \leq p+1$, and $\hat{s}_{i_h} \in \{s_{i_h}, s'_{i_h}\}$. By the construction of $\mathcal{B}$, we can see that if we replace all the $\hat{s}_\_$ symbols in w by their corresponding $S_\_$ languages, we get only words in T. Differently said, $\text{EXP}(\text{EXP}'(w)) \subseteq T$. This means that $\text{EXP}'(w) \subseteq L(\mathcal{A}'')$ [Recall that $L(\mathcal{A}'')$ is the set of all words in $(\Delta_c \cup \Omega \cup \Delta_r)^*$ with expansion completely contained in T]. The condition that $\text{EXP}'(w) \subseteq L(\mathcal{A}'')$ is exactly what is required from a word to be in $L(\mathcal{A}''')$. Thus, $w \in L(\mathcal{A}''')$.

"$\supseteq$" Now suppose that $w = u_1 \hat{s}_{i_1} u_2 \ldots u_p \hat{s}_{i_p} u_{p+1}$ (written as above) is in $L(\mathcal{A}''')$. This means that $\text{EXP}'(w) \subseteq L(\mathcal{A}'')$. In other words, by replacing all the $s'_\_$ symbols in w by their corresponding source languages, we get words that are included in $L(\mathcal{A}'')$. Further replacing the $s_\_$ symbols, we get words that are included in T. In short,
$\text{EXP}(\text{EXP}'(u_1 \hat{s}_{i_1} u_2 \ldots u_p \hat{s}_{i_p} u_{p+1})) =$
$\text{EXP}(u_1 s_{i_1} u_2 \ldots u_p s_{i_p} u_{p+1})) \subseteq T$, and this implies that $u_1 s_{i_1} u_2 \ldots u_p s_{i_p} u_{p+1} \in L(\mathcal{A}'')$.

By the construction, if $\mathcal{B}$ accepts a word $v = u_1 s_{i_1} u_2 \ldots u_p s_{i_p} u_{p+1}$ it will also accept all the other words obtained from v by changing an arbitrary number of $s_\_$ symbols to their $s'_\_$ counterparts. Clearly, word w is among these and hence in $L(\mathcal{B})$. (End of Lemma 1) □

The above lemma concludes the proof of this theorem. □

Regarding the lower bound, we show that computing the maximal safe partial rewriting T'_p is 2EXPTIME-hard. For this, we start by the following theorem.

THEOREM 13. *Computing $T''_p \cap M^c$, where $M = (\Delta_c \cup \Omega \cup \Delta_r)^* \cdot \{a\bar{a} : a \in \Delta_c\} \cdot (\Delta_c \cup \Omega \cup \Delta_r)^*$, is 2EXPTIME-hard.*

Proof. Based on Theorem 7 (and its proof), we have that $T'' \subseteq T''_p$ and $T''_p \cap M^c = T''$. Also, reasoning similarly as in Theorem 2, we have that $T''_p \cap M^c = T''_p \cap M'^c$, where $M' = (\Delta_c \cup \Omega \cup \Delta_r)^* \cdot \{a\bar{b} : a \in \Delta_c \text{ and } \bar{b} \in \Delta_r\} \cdot (\Delta_c \cup \Omega \cup \Delta_r)^*$. Recall that the DFA for M'^c has not more than 8 states.

So, we can compute T'' by computing $T''_p \cap M'^c$, which can be done in polynomial time in the size of T''_p. Now, our claim follows because computing T'' is 2EXPTIME-hard (see Theorem 6). □

From the above theorem, we can conclude that computing the maximal safe partial rewriting is 2EXPTIME-hard.

6. CONCLUSIONS

We formally defined relevant and safe rewritings for VPLs of properly nested words, both in complete and partial settings. In summary our results are as follows.

- Computing the maximal relevant rewriting can be done in PTIME.
- Computing the maximal safe rewriting can be done in 2EXPTIME and this is a tight bound.
- Computing the maximal relevant partial rewriting can be done in EXPTIME. This is shown to be PSPACE-hard.
- Computing the maximal safe partial rewriting can be done in 3EXPTIME. This is shown to be 2EXPTIME-hard.

As VPLs of properly nested words capture all the popular (formal and semi-formal) XML schema languages, we believe that our results about rewritings of VPLs serve as an important step in approaching XML data integration.

7. REFERENCES

[1] 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.

[2] 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.

[3] CALVANESE, D., DE GIACOMO, G., LENZERINI, M., AND VARDI, Y. M. Rewriting of Regular Expressions and Regular Path Queries. *J. Comput. Syst. Sci.* 64(3): 2002, pp. 443–465.

[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] FRIEDL, F., E. Mastering Regular Expressions. O'Reilly, Sebastopol, CA., 2006.

[7] GRAHNE, G., AND THOMO, A. An Optimization Technique for Answering Regular Path Queries. In *Proc. 3d Workshop on Web and Databases* (Dallas, Texas, 18–19 May 2000), pp. 215–225.

[8] GRAHNE, G., AND THOMO, A. Algebraic Rewritings for Optimizing Regular Path Queries. In *Proc. 8th International Conference on Database Theory* (London, UK, 4–6 January 2001), pp. 301–315.

[9] Grahne, G., and Thomo, A. New Rewritings and Optimizations for Regular Path Queries. In *Proc. 9th International Conference on Database Theory* (Siena, Italy, 8–10 January 2003), pp. 242–258.

[10] Hashiguchi, K. Representation Theorems on Regular Languages. *J. Comput. Syst. Sci.* 27(1): 1983, pp. 101–115.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] Sperberg-McQueen, C., M., and Thomson, H. XML Schema 1.0. http://www.w3.org/XML/Schema, 2005.