Simplifying XML Schema: Single-Type Approximations of Regular Tree Languages[*]

Wouter Gelade[†]
Hasselt University and Transnational University of Limburg
Hasselt, Belgium
wouter.gelade@uhasselt.be

Tomasz Idziaszek
University of Warsaw
Warsaw, Poland
idziaszek@mimuw.edu.pl

Wim Martens[‡]
Technical University of Dortmund
Dortmund, Germany
wim.martens@udo.edu

Frank Neven
Hasselt University and Transnational University of Limburg
Hasselt, Belgium
frank.neven@uhasselt.be

ABSTRACT

XML Schema Definitions (XSDs) can be adequately abstracted by the single-type regular tree languages. It is well-known, that these form a strict subclass of the robust class of regular unranked tree languages. Sadly, in this respect, XSDs are not closed under the basic operations of union and set difference, complicating important tasks in schema integration and evolution. The purpose of this paper is to investigate how the union and difference of two XSDs can be approximated within the framework of single-type regular tree languages. We consider both optimal lower and upper approximations. We also address the more general question of how to approximate an arbitrary regular tree language by an XSD and consider the complexity of associated decision problems.

[*]We acknowledge the financial support of FWO-G.0821.09N and the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

[†]Research assistant of the Fund for Scientific Research - Flanders (Belgium).

[‡]Supported by a grant of the North-Rhine Westfalian Academy of Sciences and Arts, and the Stiftung Mercator Essen.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design; H.2.3 [**Database Management**]: Languages—*Data description languages (DDL)*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages

General Terms

Algorithms, Design, Theory

Keywords

XML, XML Schema, approximation, complexity

1. INTRODUCTION

Despite the existence of viable alternatives [9], XML Schema is momentarily the only industrially accepted and widely supported schema language for XML. Although the presence of a schema accompanying an XML repository has many advantages in terms of XML processing and (meta)data integration, it has already been observed several times that in practice XSDs are faulty or simply missing [2, 5, 19]. Even though the exact causes of the absence of schemas and the high percentage of errors in XSDs are difficult to pinpoint, the high complexity of XML Schema undoubtedly plays an important role.

In [4], we therefore initiated a research program to simplify the use of XML Schema. While the latter paper focused on the handling of non-deterministic content models (forbidden by the Unique Particle Attribution (UPA) constraint), the present paper concentrates on the Element Declaration Consistent (EDC) constraint which imposes restrictions on the use of the typing mechanism in XSDs. The most immediate advantage of EDC is that it facilitates a simple one-pass top-down validation algorithm. On the negative side, the constraint breaks the equivalence of XML Schema with the robust class of unranked regular tree languages and, more specifically, it prevents the closure of XSDs under two of the Boolean operations: union and set difference. The latter defect greatly complicates common tasks in XML Schema integration and evolution where the union and difference operators play a fundamental role (cf. [3]). Indeed, merging two (or more) XSDs becomes a non-trivial task when the target schema can no longer be represented by an XSD. The same holds true for refactoring a large schema into several components. To this end, we investigate in this paper how to compute optimal approximations of the union and difference of XSDs. More general, we look into optimal approximations of arbitrary unranked regular tree languages, thereby laying the foundation of a translation from Relax NG to XML Schema.

Approximations come in two distinct flavours. Depending on the application at hand, we are either interested in a maximal lower or a minimal upper approximation. For instance, in a typical data integration scenario, where the union of two XSDs, X and Y, needs to be represented by an XSD S, we want to allow all XML data described by X and Y but at the same time minimize the amount of errors, that is, XML documents outside $X \cup Y$. In such a setting S needs to be a minimal upper approximation of $X \cup Y$. Maximal lower approximations can, for instance, be motivated by the following kind of data exchange scenario. When a Web service describes its interface by means of a schema X in Relax NG, a corresponding XSD S needs to be made available for general use. To ensure a correct handling of requests, S should only define XML documents present in X. That is, S should be a maximal lower approximation of X.

Contributions. We show that, for every regular unranked tree langugage X, there is a unique minimal upper XSD-approximation S. The latter approximation can be computed in exponential time when X is represented as an extended DTD (EDTD). Furthermore, S can have exponentially more types than X and in general this blow-up cannot be avoided. In strong contrast, the union and difference of two XSDs can be uniquely approximated in polynomial time. Deciding whether a given single-type EDTD is a minimal upper XSD-approximation of a EDTD is shown to be complete for PSPACE.

Maximal lower XSD-approximations do not behave as nicely as their upper counterparts. Indeed, even for the union of two XSDs X and Y we show that there can be infinitely many maximal lower XSD-approximations. We therefore focus on XSD-approximation which extend either X or Y. We show such approximations to be unique and to be computable in polynomial time. We show that for the special case of non-recursive unranked regular tree languages there always exists a maximal lower approximation and that it is decidable whether a given XSD is a maximal lower XSD-approximation. It is unclear whether the same results hold for arbitrary regular languages.

Using the minimization algorithm from [16], we can also minimize the output XSDs of our approximation algorithms. Since minimizing an XSD can be done in polynomial time, this extra step would cost polynomial time in the size of our output XSDs. In that sense, we can always deliver optimal representations of optimal approximations.

Related Work. Murata et al. established a taxonomy of XML Schema languages in terms of tree languages [17]. More precisely, they classified DTDs as the local tree languages, XSDs as the single-type tree languages (ST-REG) and Relax NG as the unranked tree languages. Furthermore, they obtained a one-pass top-down validation algorithm for ST-REG and stated (without proof) that ST-REG is not closed under union and set difference. Martens et al [15] characterized ST-REG as the subclass of the regular tree languages closed under ancestor-guarded subtree exchange, from which the failure of closure of ST-REG under union and difference easily follows. In the same paper, the authors showed that it is EXPTIME-complete to decide whether a given regular tree language can be represented by an equivalent single-type one.

To the best of our knowledge, optimal single-type approximations of regular tree languages have not been investigated.

Outline. Section 2 introduces the necessary definitions. In Section 3, we discuss minimal upper XSD-approximations, while we address maximal lower XSD-approximations in Section 4. Section 5 discusses how our results change when NFAs and (deterministic) regular expressions are used as content models. We conclude in Section 6.

2. DEFINITIONS

For a finite set S, we denote by $|S|$ its cardinality.

2.1 Strings, trees, and contexts

By Σ we always denote a finite alphabet. As usual, a *(non-deterministic) finite automaton (NFA)* over alphabet Σ is a tuple $N = (Q, \Sigma, \delta, I, F)$, where Q is its finite set of states, Σ is the alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, I is the set of initial states, and F is the set of final states. The automaton N is *state-labeled* when, for every state q, all transitions to q carry the same label. That is for each $q \in Q$, $\{a \in \Sigma \mid q \in \delta(q', a) \text{ for some } q' \in Q\}$ is either empty or a singleton. In the latter case, we denote this unique alphabet symbol by $\text{label}(q)$. The automaton N is *deterministic*, or a *DFA*, if I is a singleton and the cardinality of each set $\delta(q, a)$ is at most one. By $N(w)$, we denote the set of states that N can end up in when reading $w \in \Sigma^*$ started in some state $q \in I$. The *regular expressions* (RE) r over Σ are of the form

$$r ::= \emptyset \mid \varepsilon \mid a \mid rr \mid r + r \mid (r)? \mid (r)^+ \mid (r)^*,$$

where ε denotes the empty string and a ranges over symbols in the alphabet Σ. Sometimes we also use the symbol $\cdot$ for regular expression concatenation to improve readability. As usual, we write $L(r)$ for the language defined by regular expression r and $L(N)$ for the language defined by finite automaton N.

The set of Σ-*trees*, denoted by $\mathcal{T}_\Sigma$, is inductively defined as follows: (1) every $a \in \Sigma$ is a Σ-tree; and (2) if $a \in \Sigma$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$ for $n \geq 1$ then $a(t_1, \ldots, t_n)$ is a Σ-tree. There is no a priori bound on the number of children of a node in a Σ-tree; such trees are therefore *unranked*. In the following, when we say tree we always mean a Σ-tree. A *tree language* is a set of trees.

For every tree t, the *set of nodes* of t, denoted by $\text{Dom}(t)$, is the set defined as follows: if $t = a(t_1, \ldots, t_n)$ with $a \in \Sigma$, $n \geq 0$, and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$, then $\text{Dom}(t) = \{\varepsilon\} \cup \{iu \colon 1 \leq i \leq n, u \in \text{Dom}(t_i)\}$. Thus, ε represents the root while ui represents the i-th child of u. For a node $v \in \text{Dom}(t)$, we denote the Σ-label of v by $\text{lab}^t(v)$. When v has n children, we denote by $\text{ch-str}^t(v)$ the child-string of v, i.e., the string $\text{lab}^t(v1) \cdots \text{lab}^t(vn)$. Denote by $t_1[v \leftarrow t_2]$ the tree obtained from a tree t_1 by replacing the subtree rooted at node v of

t_1 by t_2; hence, in $t_1[v \leftarrow t_2]$, the label of v is the root label of t_2. By $\text{subtree}^t(v)$ we denote the subtree of t rooted at v.

A *context* is a tree with a "hole" marker $\bullet$. More specifically, a context C is a tree over the alphabet $\Sigma \cup (\Sigma \times \{\bullet\})$ in which all nodes are labeled with Σ-symbols, except for one leaf that is labeled with $(a, \bullet)$ for some $a \in \Sigma$. Given a context C with a hole marker at node u and a tree $t' = a(t_1, \ldots, t_n)$, we denote by $C[t']$ the Σ-tree $C[u \leftarrow t']$. If C' is another context with root label a or $(a, \bullet)$, we denote by $C[C']$ the context $C[u \leftarrow C']$. We say that we *apply* the context C to tree t' (respectively, context C'). Notice that we can only apply a context C to a tree t' (respectively, context C') if the root of t' (respectively, C') bears the same Σ-label as the distinguished leaf in C.

2.2 XML schema languages

We abstract XML Document Type Definitions (DTDs) as follows:

Definition 2.1. A *DTD* is a tuple (Σ, d, S_d), where Σ is a finite alphabet, d is a function that maps Σ-symbols to regular string languages over Σ, and $S_d \subseteq \Sigma$ is the set of start symbols. For notational convenience we sometimes denote (Σ, d, S_d) by d.

A tree t *satisfies* d if its root is labeled by an element of S_d and, for every node v with label a, the child-string $\text{ch-str}^t(v)$ is in the language defined by $d(a)$. By $L(d)$ we denote the language of trees satisfying d.

The *size* of a DTD is $|\Sigma| + |S_d| + |d|$ where $|d|$ refers to the size of the representations of the regular string languages. Unless specified otherwise, we represent all such regular string languages by DFAs.[1] Hence, $|d|$ is the sum of the sizes of all DFAs representing languages $d(a)$ for $a \in \Sigma$.

To boost its expressiveness, the XML Schema specification extends DTDs with a typing mechanism, abstracted in the form of extended DTDs as follows [17, 18]:

Definition 2.2. An *extended DTD* (EDTD) is a tuple $D = (\Sigma, \Delta, d, S_d, \mu)$, where Δ is a finite set of *types*, (Δ, d, S_d) is a DTD and μ is a mapping from Δ to Σ.

A tree t *satisfies* D if $t = \mu(t')$ for some $t' \in L(d)$. Again, we denote by $L(D)$ the language of trees satisfying D.

Extended DTDs are well-known to define the class of *unranked regular tree languages (UREG)* [7, 18]. The size of an EDTD is $|\Sigma|$ plus the size of its underlying DTD.

Proviso 2.3. *In this paper, we assume that all EDTDs are* reduced. *Formally an EDTD $(\Sigma, \Delta, d, S_d, \mu)$ is* reduced *if, for each type $\tau \in \Delta$, there exists a tree $t' \in L(d)$ and a node u such that $\text{lab}^{t'}(u) = \tau$. It is widely known that an equivalent reduced EDTD can be computed from a given EDTD in polynomial time (see, e.g., [1, 14]).*

As the *Element Declarations Consistent* rule severely constrains the use of the typing mechanism [10], extended DTDs do not constitute a satisfactory abstraction of XSDs. Therefore, XSDs are commonly abstracted as single-type EDTDs [17, 15, 13]:

[1]In Section 5, we discuss how our results change when (deterministic) regular expressions and NFAs are used. Note also that XML Schema restricts regular expressions to be deterministic, a strict subclass of DFAs. In fact, any deterministic regular expression can be translated in quadratic time to a corresponding DFA.

Definition 2.4. A *single-type EDTD* (*stEDTD* in short) is an EDTD $(\Sigma, \Delta, d, S_d, \mu)$ with the property that no two types τ and τ' exist with $\mu(\tau) = \mu(\tau')$ such that τ and τ' occur *(i)* both in S_d or *(ii)* both in the same regular expression.

We refer to ST-REG as the class of tree languages definable by single-type EDTDs. The *type-size* of a language T in ST-REG is $\min\{|\Delta| \mid L(D) = T \text{ and } D = (\Sigma, \Delta, d, S_d, \mu)\}$, i.e., the smallest number of types among all stEDTDs defining T.

Martens et al. provided several alternative characterizations of single-type EDTDs [15, 13]. One of these is a simple extension of DTDs, which we define next. We denote by $\text{anc-str}^t(v)$ the sequence of labels on the path from the root to v including both the root and v itself.

Definition 2.5. A *DFA-based DTD* is a pair (A, d), where $A = (Q, \Sigma, \delta, \{q_{\text{init}}\})$ is a state-labeled DFA with initial state q_{init} and without final states, and d is a function mapping $Q \setminus \{q_{\text{init}}\}$ to regular languages over Σ.

A tree t *satisfies* D if, for every node u, $A(\text{anc-str}^t(u)) = \{q\}$ implies $\text{ch-str}^t(u)$ is in the language defined by $d(q)$.

Proposition 2.6 ([13]). *DFA-based DTDs are expressively equivalent to single-type EDTDs and one can translate between DFA-based DTDs and single-type EDTDs in linear time.*

We next recall a fundamental characterization of single-type EDTDs in terms of a subtree-exchange property, graphically illustrated in Figure 1.

Definition 2.7. A tree language T is *closed under ancestor-guarded subtree exchange* if the following property holds. Whenever for two trees $t_1, t_2 \in T$ with nodes v_1, v_2 respectively, $\text{anc-str}^{t_1}(v_1) = \text{anc-str}^{t_2}(v_2)$ then

$$t_1[v_1 \leftarrow \text{subtree}^{t_2}(v_2)] \in T.$$

Theorem 2.8 ([15]). *A regular tree language T is definable by a single-type EDTD if and only if it is closed under ancestor-guarded subtree exchange.*

2.3 XSD-Approximations

Here we define the notions of lower and upper XSD-approximations which constitute the central theme of this work.

Definition 2.9. An *upper XSD-approximation* of a tree language T is a language T' definable by a single-type EDTD that contains T. An upper XSD-approximation is *minimal* if there is no other upper XSD-approximation X of T such that $T \subseteq X \subset T'$.

A *lower XSD-approximation* of a tree language T is a language T' definable by a single-type EDTD that is contained in T. A lower XSD-approximation is *maximal* if there is no other lower XSD-approximation X of T such that $T' \subset X \subseteq T$.

2.4 Complexity-Theoretic Results

We recall a complexity-theoretic result about EDTDs which we use in the remainder of the paper. The following theorem follows from a well-known result by Seidl [20], and the close correspondence between EDTDs and tree automata discussed by Papakonstantinou and Vianu [18].

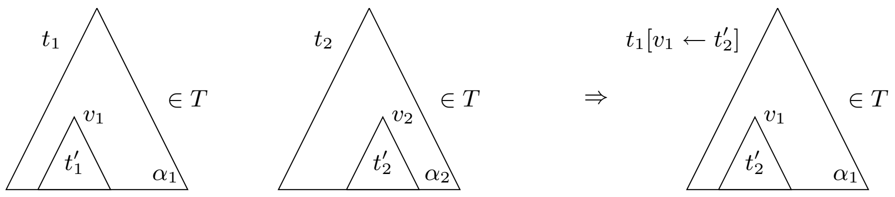


Figure 1: Ancestor-guarded subtree exchange (anc-str$^{t_1}(v_1)$ = anc-str$^{t_2}(v_2)$).

Theorem 2.10 ([18, 20]). *The universality problem for EDTDs, i.e., deciding whether $\mathcal{T}_\Sigma \subseteq L(D)$ for an EDTD D, is* EXPTIME*-complete.*

Notice that, since $\mathcal{T}_\Sigma$ is definable by a DTD, also the inclusion problem $L(D_1) \subseteq L(D_2)$ is EXPTIME-complete if D_2 is an EDTD and D_1 is either a DTD or stEDTD.

2.5 Single-type closure and derivation trees

Definition 2.11. We denote by closure(T) the smallest tree language closed under ancestor-guarded subtree exchange which contains T. We will write closure(t_1, t_2) if $T = \{t_1, t_2\}$.

By Lemma 2.12 this notion is well-defined.

Lemma 2.12. *Let $(X_i)_{i\in I}$ be an arbitrary family of tree languages where each X_i is closed under ancestor-guarded subtree exchange. Then the intersection $\bigcap_{i\in I} X_i$ is also closed under ancestor-guarded subtree exchange.*

Proof. Let $X = \bigcap_{i\in I} X_i$. Let t_1, t_2 be two trees from X with nodes v_1, v_2 resp., and anc-str$^{t_1}(v_1)$ = anc-str$^{t_2}(v_2)$. For each $i \in I$ we have $t_1, t_2 \in X_i$ and thus $t = t_1[v_1 \leftarrow \text{subtree}^{t_2}(v_2)] \in X_i$. Therefore $t \in X$, and thus X is closed under ancestor-guarded subtree exchange. □

Definition 2.13. Let X be a tree language and t a tree from closure(X). A *derivation tree* of t with respect to X is a binary tree ϑ labeled with trees from closure(X) such that:

- The root of ϑ is labeled with t: $\text{lab}^\vartheta(\varepsilon) = t$.
- For each leaf $v \in \text{Dom}(\vartheta)$ we have $\text{lab}^\vartheta(v) \in X$.
- For each internal node $v \in \text{Dom}(\vartheta)$ and $i \in \{0,1\}$, let $t_i = \text{lab}^\vartheta(vi)$. Then there are nodes $u_i \in \text{Dom}(t_i)$ such that anc-str$^{t_0}(u_0)$ = anc-str$^{t_1}(u_1)$ and $\text{lab}^\vartheta(v) = t_0[u_0 \leftarrow \text{subtree}^{t_1}(u_1)]$.

Lemma 2.14. *Let X be a tree language. For any tree t, $t \in$ closure(X) if and only if t has a derivation tree with respect to X.*

3. UPPER XSD-APPROXIMATIONS

In this section, we consider upper XSD-approximations of EDTDs. In general, constructing an optimal upper XSD-approximation of an EDTD requires exponential time. However, given two single-type EDTDs D_1 and D_2, we can construct optimal upper XSD-approximations for languages $L(D_1) \cup L(D_2)$, $L(D_1) \cap L(D_2)$, and $\mathcal{T}_\Sigma \setminus L(D_1)$ in polynomial time.

3.1 EDTDs

We show that for every regular tree language there exists an unique upper XSD-approximation. In particular, the latter approximation can be obtained by determinizing the type automaton corresponding to the given EDTD. The overall construction can be computed in exponential time and results in an approximation of exponential type-size which in general cannot be avoided.

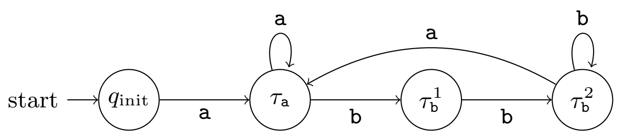
Definition 3.1. The *type automaton* of an EDTD $D = (\Sigma, \Delta, d, S_d, \mu)$ is a state-labeled NFA $N = (Q, \Sigma, \delta, \{q_{\text{init}}\})$ without final states such that $Q = \Delta \uplus \{q_{\text{init}}\}$ and for each $q \in Q$

- if $q = q_{\text{init}}$, then $\delta(q, a) = \{\tau \mid \mu(\tau) = a \text{ and } \tau \in S_d\}$, and
- otherwise, $\delta(q, a) = \{\tau \mid \mu(\tau) = a$ and τ occurs in (some string in) $d(q)\}$.

Example 3.2. Consider the following EDTD $D = (\Sigma, \Delta, d, S_d, \mu)$, with $\Delta = \{\tau_{\mathtt{a}}, \tau^1_{\mathtt{b}}, \tau^2_{\mathtt{b}}\}$, $S_d = \{\tau_{\mathtt{a}}\}$ and $\mu(\tau_{\mathtt{a}}) = \mathtt{a}$, $\mu(\tau^1_{\mathtt{b}}) = \mu(\tau^2_{\mathtt{b}}) = \mathtt{b}$:

$$\begin{aligned} \tau_{\mathtt{a}} &\to \tau_{\mathtt{a}} + \tau^1_{\mathtt{b}} \\ \tau^1_{\mathtt{b}} &\to \tau^2_{\mathtt{b}} + \varepsilon \\ \tau^2_{\mathtt{b}} &\to \tau_{\mathtt{a}} + \tau^2_{\mathtt{b}} + \varepsilon \end{aligned}$$

Then, this is the type automaton of D:


$$\begin{tikzcd}
\text{start} \arrow[r] & q_{\text{init}} \arrow[r, "\mathtt{a}"'] & \tau_{\mathtt{a}} \arrow[loop above, "\mathtt{a}"] \arrow[r, "\mathtt{b}"'] & \tau^1_{\mathtt{b}} \arrow[r, "\mathtt{b}"'] & \tau^2_{\mathtt{b}} \arrow[loop above, "\mathtt{b}"] \arrow[ll, bend right, "\mathtt{a}"']
\end{tikzcd}$$

We make the following observations:

Observation 3.3. *(1) Given an EDTD, its type automaton can be constructed in linear time.*

(2) For each EDTD, the state q_{init} of its type automaton has no incoming transitions.

(3) The type automaton of an EDTD D is a DFA if and only if D is a single-type EDTD.

Here we give a general construction for the upper approximation of a given EDTD $D = (\Sigma, \Delta, d, S_d, \mu)$. Let $N = (Q_N, \Sigma, \delta_N, \{q_{\text{init}}\})$ be the type automaton of D, and let $A_N = (Q, \Sigma, \delta, \{\{q_{\text{init}}\}\})$ be the DFA obtained from N by performing the standard subset construction. That is, $Q \in 2^{Q_N}$ is the smallest set such that $\{q_{\text{init}}\} \in Q$ and whenever $S \in Q$ then for every $a \in \Sigma$, $\bigcup_{q\in S} \delta_N(q, a) \in Q$. By construction and Observation 3.3(2), each non-initial state consists of a set of types S of D with $\mu(\tau) = \mu(\tau')$ for all $\tau, \tau' \in S$. Then define the DFA-based DTD (A_N, d') with

$$d'(S) := \bigcup_{\tau \in S} \mu(d(\tau)) \qquad \text{for every } S \in Q.$$

Here, μ is canonically extended to languages.

Theorem 3.4 will show that (A_N, d') is in fact a minimal upper XSD-approximation of D.

Theorem 3.4. *The minimal upper XSD-approximation of an EDTD is unique and can be computed in exponential time. There is a family of EDTDs $(D_n)_{n \geq 2}$, such that the size of every D_n is $O(n)$ but the type-size of the minimal upper XSD-approximation is $\Omega(2^n)$.*

We conclude this section by discussing the complexity of testing whether a given single-type EDTD is the minimal upper XSD-approximation of an EDTD. The proof makes use of the following lemma which is interesting in its own right as it contrasts with the EXPTIME-completeness of testing equivalence of an EDTD and a single-type EDTD (Theorem 2.10). Recall from Section 2 that EDTDs use DFAs and not NFAs to represent their regular string languages, which is crucial for the following lemma.

Lemma 3.5. *Let D_1 be an EDTD and let D_2 be a single-type EDTD. Testing whether $L(D_1) \subseteq L(D_2)$ is in* PTIME.

Using the previous lemma and an on-the-fly construction of the minimal upper XSD-approximation we get the following theorem.

Theorem 3.6. *Deciding whether a single-type EDTD is a minimal upper XSD-approximation of a given EDTD is* PSPACE*-complete.*

3.2 Unions of XSDs

We next address the optimal upper XSD-approximation for the union of two XSDs.

Theorem 3.7. *Let D_1 and D_2 be two single-type EDTDs. The minimal upper XSD-approximation of $L(D_1) \cup L(D_2)$ is unique and can be computed in time $O(|D_1||D_2|)$. There is a family of pairs of single-type EDTDs $(D_1^n, D_2^n)_{n \geq 1}$, such that the size of every D_1^n and D_2^n is $O(n)$ but the type-size of the minimal upper XSD-approximation for $L(D_1^n) \cup L(D_2^n)$ is $\Omega(n^2)$.*

Proof. Let D be an EDTD for the language $L(D_1) \cup L(D_2)$. The type automaton of D is the product[2] of the type automata of D_1 and D_2. The determinization process of Section 3.1 can in this case be performed in time $O(|D_1||D_2|)$. Therefore, the type-size of the minimal upper XSD-approximation D' for $L(D_1) \cup L(D_2)$ is $O(|D_1||D_2|)$. Furthermore, since each DFA in D' is the union of at most one DFA in D_1 and one in D_2, the size of D' is also $O(|D_1||D_2|)$. It follows from the proof of Theorem 3.4 that this is the unique minimal upper XSD-approximation. We omit The proof of the $\Omega(n^2)$ lower bound. $\square$

3.3 Intersection of XSDs

Proposition 3.8. *Let D_1 and D_2 be single type EDTDs. Their intersection $L(D_1) \cap L(D_2)$ is definable by a single-type EDTD.*

[2]For more details on the standard product construction of automata, see, e.g., [12].

Proof. This follows from Lemma 2.12, from the fact that regular languages are closed under intersection, and from Theorem 2.8. $\square$

Therefore, the minimal upper XSD-approximation will in fact be equal to the intersection.

Theorem 3.9. *Let D_1 and D_2 be two single-type EDTDs. The minimal upper XSD-approximation of $L(D_1) \cap L(D_2)$ is unique, accepts precisely $L(D_1) \cap L(D_2)$ and can be computed in time $O(|D_1||D_2|)$. There is a family of pairs of single-type EDTDs $(D_1^n, D_2^n)_{n \geq 1}$, such that the size of every D_1^n and D_2^n is $O(n)$ but the type-size of the minimal upper XSD-approximation for $L(D_1^n) \cap L(D_2^n)$ is $\Omega(n^2)$.*

Proof. The construction for the intersection of D_1 and D_2 is analogous to the construction in the proof of Theorem 3.7, with the difference that now we need to construct the *intersection* of the two internal DFAs. (I.e., for $d'(S)$, we need to construct $\bigcap_{\tau \in S} \mu(d(\tau))$.) However, since the standard product construction of DFAs can also construct the intersection, this construction is also possible in time $O(|D_1||D_2|)$. Correctness of this construction can be proved through the characterization in Proposition 2.6. We omit the proof of the $\Omega(n^2)$ lower bound. $\square$

3.4 Complements of XSDs

We next show that the complement of an XSD can be uniquely approximated within polynomial time.

Theorem 3.10. *Let D be a single-type EDTD. The minimal upper XSD-approximation for the complement of D is unique and can be computed in time polynomial in $|D|$.*

Proof. Let $D = (\Sigma, \Delta, d, S_d, \mu)$ and let (A, f) be the DFA-based DTD equivalent to D with $A = (\Delta, \Sigma, \delta, \{q_{\text{init}}\})$. According to the definition of a DFA-based DTD, a tree t is in $L(A, f)$ if and only if for every $v \in \text{Dom}(t)$ with $A(\text{anc-str}^t(v)) = \{\tau\}$, we have that $\text{ch-str}^t(v) \in L(f(\tau))$.

We will prove the theorem in two steps: first we will construct an EDTD D_c for the complement of D and then we will show that the minimal upper approximation of D_c can be constructed in polynomial time.

A tree t is in $\mathcal{T}_\Sigma \setminus L(A, f)$ if and only if there exists a $v \in \text{Dom}(t)$ with $A(\text{anc-str}^t(v)) = \{\tau\}$ such that $\text{ch-str}^t(v) \notin L(f(\tau))$. When given a tree t, the EDTD D_c guesses the path until such a node v and tests whether $\text{ch-str}^t(v) \notin L(f(\tau))$. Formally, for the definition of $D_c = (\Sigma, \Delta_c, d_c, S_{d_c}, \mu_c)$, we use two sets of types: Δ and Σ. We use Δ to guess the path to v and we use Σ as the set of types that accept every tree. More formally:

(1) $\Delta_c = \Delta \uplus \Sigma$;

(2) for every $\tau \in \Delta$, $\mu_c(\tau) = \mu(\tau)$ and, for every $a \in \Sigma$, $\mu_c(a) = a$;

(3) $S_{d_c} = S_d \uplus (\Sigma \setminus \mu(S_d))$;

(4) for every $\tau \in \Delta$,

$$d_c(\tau) = (\Sigma^* \setminus f(\tau)) + \Sigma^* \cdot \bigcup_{a \in \Sigma} \delta(\tau, a) \cdot \Sigma^*;$$

(5) for every $a \in \Sigma$, $d_c(a) = \Sigma^*$.

The EDTD D_c accepts $\mathcal{T}_\Sigma \setminus L(D)$ and $|D_c| = O(|\Sigma||D|)$. The factor $|\Sigma|$ in this complexity arises from rule (4) in which a product construction between a complement DFA of D and a DFA of size $O(|\Sigma|)$ must be performed.

To prove that the minimal upper approximation of $L(D_c)$ can be computed in polynomial time, we need to prove that determinizing the type automaton of D_c using the subset construction can be done in polynomial time. To this end, let us first investigate the type automaton N_c of D_c. This type automaton contains the type automaton A of D as a sub-automaton: rule (3) includes all the outgoing transitions from q_{init}, and rule number (4) includes all other transitions. The transitions that N_c has in addition, are the ones entering the states in Σ. These transitions arise from rules (3), (4), and (5). The Σ-states form a clique due to rule (5).

Due to the structure of N_c, the subset construction results in an automaton in which every state is a subset of $\{\tau, a\}$ for some $\tau \in \Delta, a \in \Sigma$. The reason is that, after reading a string, N_c can never arrive in two different states of type Δ or two different states of type Σ. Therefore, the subset determinization algorithm on N_c can be performed in time $|\Sigma||N_c|$. This shows that the minimal upper approximation of the complement of D can be computed in polynomial time in the size of D. $\square$

Since single-type EDTDs are closed under intersection, and we can construct the intersetion in polynomial time, we also have the following corollary.

Corollary 3.11. *Let D_1 and D_2 be single-type EDTDs. The minimal upper approximation of $L(D_1) \setminus L(D_2)$ can be computed in time polynomial in $|D_1| + |D_2|$.*

Proof. Let D be an EDTD for language $L(D_1) \setminus L(D_2)$. Since $L(D_1) \setminus L(D_2) = L(D_1) \cap (\mathcal{T}_\Sigma \setminus L(D_2))$, the type automaton of D is an intersection of type automata of D_1 and the complement of D_2. Construction and determinization of this intersection can be performed in polynomial time using the standard product construction. $\square$

4. LOWER XSD-APPROXIMATIONS

For lower approximations the picture is not so nice. First of all, there can be infinitely many maximal lower approximations for the union of two XSDs D_1 and D_2. Nevertheless, we show that there is a unique maximal lower approximation when it includes either all of D_1 or all of D_2. That is, there is a well-defined maximal part of D_1 (D_2) which can be added to D_2 (D_1) to form a maximal lower approximation of $D_1 \cup D_2$. Also the complement can not be uniquely approximated in general. We do not know whether for every EDTD there always exists at least one maximal lower approximation. We can answer this question positively for the class of bounded depth schemas. Finally, we discuss the complexity of deciding whether a given single-type EDTD is a maxinal lower approximation of a given EDTD.

4.1 A Modified Subtree Exchange Property

We first provide a modified version of the subtree exchange property for single-type EDTDs that will be helpful in this section. Let N be a state-labeled NFA. For a node v in a tree t, we call the set of types $N(\text{anc-str}^t(v))$ the *ancestor-type* of v in t w.r.t. N and we denote it by $\text{anc-type}^t_N(v)$. When N is clear from the context, we sometimes also write $\text{anc-type}^t(v)$.

Definition 4.1. Let N be an NFA. A set T is *closed under ancestor-type-guarded subtree exchange w.r.t. N* if the following holds. Whenever for two trees $t_1, t_2 \in T$ with nodes v_1, v_2, resp., $\text{anc-type}^{t_1}_N(v_1) = \text{anc-type}^{t_2}_N(v_2)$ then $t[v_1 \leftarrow \text{subtree}^{t_2}(v_2)] \in T$. We say that a set T is *closed under ancestor-type-guarded subtree exchange w.r.t. D* if it is closed under ancestor-type-guarded subtree exchange w.r.t. the type automaton of D.

Notice that $\text{anc-type}^{t_1}_N(v_1) = \text{anc-type}^{t_2}_N(v_2)$ implies that $\text{lab}^{t_1}(v_1) = \text{lab}^{t_2}(v_2)$, because the automaton N is always a state-labeled NFA.

Theorem 4.2. *A regular tree language which is defined by an EDTD D is definable by a single-type EDTD if and only if it is closed under ancestor-type-guarded subtree exchange w.r.t. D.*

Proof. If T is definable by a single-type EDTD, then we can construct an ancestor-guarded DTD for T by determinizing the type automaton N of D, as explained in Section 3.1. Therefore, T is closed under ancestor-type-guarded subtree exchange. If T is closed under ancestor-type-guarded subtree exchange, then it is also closed under ancestor-guarded subtree exchange and therefore definable by a single-type EDTD. $\square$

4.2 Unions of XSDs

4.2.1 Infinitely many optimal approximations

The next theorem underlines that lower XSD-approximations do not behave as nicely as their upper counterparts.

Theorem 4.3. *Let D_1 and D_2 be two single-type EDTDs. In general, the maximal lower XSD-approximation for the sum $L(D_1) \cup L(D_2)$ is not unique. The set of maximal lower XSD-approximations can be infinite.*

Proof. In the following, we use $\mathtt{a}^k(t)$ to abbreviate the tree $\mathtt{a}(\mathtt{a}\cdots(\mathtt{a}(t)))$ consisting of k $\mathtt{a}$'s and followed by a subtree t.

Take the following single-type EDTDs (which are even DTDs) with $\mathtt{a}$ as the root:

$$D_1: \begin{array}{l} \mathtt{a} \to \mathtt{a} + \mathtt{b} \\ \mathtt{b} \to \varepsilon \end{array} \qquad D_2: \mathtt{a} \to \mathtt{a} + \mathtt{aa} + \varepsilon$$

For every $n \geq 1$ the following single-type EDTD X_n is a maximal lower XSD-approximation of $L(D_1) \cup L(D_2)$:

$$\begin{array}{ll} \tau^i_{\mathtt{a}} \to \tau^{i+1}_{\mathtt{a}} + \tau_{\mathtt{b}} + \varepsilon & \text{for } 0 \leq i < n-1 \\ \tau^{n-1}_{\mathtt{a}} \to \tau^n_{\mathtt{a}} + \tau^n_{\mathtt{a}}\tau^n_{\mathtt{a}} + \tau_{\mathtt{b}} + \varepsilon & \\ \tau^n_{\mathtt{a}} \to \tau^n_{\mathtt{a}} + \tau^n_{\mathtt{a}}\tau^n_{\mathtt{a}} + \varepsilon & \\ \tau_{\mathtt{b}} \to \varepsilon & \end{array}$$

Here, $\mu(\tau^i_{\mathtt{a}}) = \mathtt{a}$ for every $i \in \{0, \ldots, n\}$ and $\mu(\tau_{\mathtt{b}}) = \mathtt{b}$. These languages are pairwise different, since a unary tree $t_m = \mathtt{a}^m\mathtt{b}$ is in $L(X_n)$ if and only if $n \geq m$.

Let t be an arbitrary tree from $(L(D_1) \cup L(D_2)) \setminus L(X_n)$. We prove that $\text{closure}(L(X_n) \cup \{t\}) \not\subseteq L(D_1) \cup L(D_2)$. If $t \in L(D_1) \setminus L(X_n)$ then it is a tree t_m with $m > n$. Then for a tree $\mathtt{a}^n(\mathtt{a},\mathtt{a}) \in L(X_n)$ we have that $\text{closure}(t, \mathtt{a}^n(\mathtt{a},\mathtt{a}))$ contains a tree $\mathtt{a}^n(\mathtt{a}^{m-n}\mathtt{b},\mathtt{a}) \notin L(D_1) \cup L(D_2)$ (just apply ancestor-guarded subtree exchange on nodes 1^n in $\text{Dom}(t)$ and $\text{Dom}(\mathtt{a}^n(\mathtt{a},\mathtt{a}))$).

If $t \in L(D_2) \setminus L(X_n)$ then in the first $n-1$ levels there is a node with two children, thus $t = \mathtt{a}^m(t', t'')$ for some $m < n$ and $t', t'' \in L(D_2)$. Then again $\text{closure}(t, t_n)$ contains a tree $\mathtt{a}^m(\mathtt{a}^{n-m}\mathtt{b}, t'') \notin L(D_1) \cup L(D_2)$ (apply ancestor-guarded subtree exchange on nodes 1^m in $\text{Dom}(t)$ and $\text{Dom}(t_n)$). $\square$

4.2.2 Uniquely extending D_1 or D_2

In this section, we show that one can compute a maximal lower XSD-approximation of $L(D_1) \cup L(D_2)$ which includes $L(D_1)$ and that such a maximal approximation containing $L(D_1)$ is unique. That is, we are looking for the maximal set $Y \subseteq L(D_2)$ such that $L(D_1) \cup Y$ is a maximal lower XSD-approximation of $L(D_1) \cup L(D_2)$. This set Y needs to come from the set of non-violating trees, as defined next:

Definition 4.4. Let D_1 and D_2 be single-type EDTDs. The set of *non-violating trees* from $L(D_2)$ with respect to D_1 is defined as

$$\text{nv}(D_2, D_1) := \{t \in L(D_2) \mid \forall t_1 \in L(D_1) \\ \text{closure}(t_1, t) \subseteq L(D_1) \cup L(D_2)\}.$$

That is, $\text{nv}(D_2, D_1)$ contains all individual trees t for which $\text{closure}(D_1 \cup \{t\})$ remains within the union of D_1 and D_2. If we want to find a set $Y \subseteq L(D_2)$ such that $L(D_1) \cup Y$ is a maximal lower XSD-approximation of $L(D_1) \cup L(D_2)$, then clearly $Y \subseteq \text{nv}(D_2, D_1)$, otherwise $L(D_1) \cup Y \not\subseteq L(D_1) \cup L(D_2)$. We show that, in fact, if $Y = \text{nv}(D_2, D_1)$, then $L(D_1) \cup Y$ is definable by a single-type EDTD. From the above, it then follows that $L(D_1) \cup Y$ is a maximal lower XSD-approximationof $L(D_1) \cup L(D_2)$. Therefore, the remainder of this section is devoted to proving that $L(D_1) \cup Y$ is definable by a single-type EDTD.

Let $D_i = (\Sigma, \Delta_i, d_i, S_{d_i}, \mu_i)$ for $i \in \{1, 2\}$. Moreover let $A_i = (\Delta_i \uplus q_I, \Sigma, \delta_i, q_I)$ be the type automaton for D_i.

Let $t \in L(D_2)$ and $t_1 \in L(D_1)$ be two trees. Clearly $\text{closure}(t_1, t) \subseteq L(D)$, where $D = (\Sigma, \Delta, d, S_d, \mu)$ is a single-type EDTD such that $L(D) = \text{closure}(L(D_1) \cup L(D_2))$. Thus from Theorem 4.2 we have that $\text{closure}(t_1, t)$ is closed under ancestor-type-guarded subtree exchange w.r.t. D. From the construction in Theorem 3.7, the type set for D is $\Delta = (\Delta_1 \cup \{\bot\}) \times (\Delta_2 \cup \{\bot\})$.

Therefore a tree $t \in L(D_2)$ belongs to $\text{nv}(D_2, D_1)$ if and only if for every $t_1 \in L(D_1)$ and all nodes $v \in \text{Dom}(t)$, $v_1 \in \text{Dom}(t_1)$, such that $\text{anc-type}^t(v) = \text{anc-type}^{t_1}(v_1)$, we have that

(a) $t[v \leftarrow \text{subtree}^{t_1}(v_1)] \in L(D_1) \cup L(D_2)$, and

(b) $t_1[v_1 \leftarrow \text{subtree}^t(v)] \in L(D_1) \cup L(D_2)$.

This is one characterization of all trees t belonging to $\text{nv}(D_2, D_1)$. However, we need another one which does not explicitly mention t_1.

Thereto, for $i \in \{1, 2\}$ and $\tau = (\tau_1, \tau_2) \in \Delta$, we define the following sets:

$$S_i(\tau) := \{\text{subtree}^t(v) \mid t \in L(D_i), \text{anc-type}^t(v) = \tau\},$$
$$C_i(\tau) := \{\text{context}^t(v) \mid t \in L(D_i), \text{anc-type}^t(v) = \tau\}.$$

We call a type $\tau \in \Delta$ an s-type if it satisfies the condition $S_1(\tau) \setminus S_2(\tau) \neq \emptyset$. We call this type a c-type if it satisfies the condition $C_1(\tau) \setminus C_2(\tau) \neq \emptyset$. Of course, a type can be both an s-type and a c-type.

With these definitions we can state that a tree $t \in L(D_2)$ belongs to $\text{nv}(D_2, D_1)$ if and only if, for every node $v \in \text{Dom}(t)$ and $\tau = \text{anc-type}^t(v)$,

(a') if τ is an s-type, then $\text{context}^t(v) \in C_1(\tau)$,

(b') if τ is a c-type, then $\text{subtree}^t(v) \in S_1(\tau)$.

We prove that (a) is satisfied if and only if (a') is. For the if part, let $t_1 \in L(D_1)$ and $v_1 \in \text{Dom}(t_1)$ such that $\text{anc-type}^{t_1}(v_1) = \tau$. If $t_1' = \text{subtree}^{t_1}(v_1) \in S_2(\tau)$, then clearly $t[v \leftarrow t_1'] \in L(D_2)$. On the other hand, if $t_1' \in S_1(\tau) \setminus S_2(\tau)$, then τ is an s-type. Therefore applying (a') we get that $\text{context}^t(v) \in C_1(\tau)$ and $t[v \leftarrow t_1'] \in L(D_1)$.

For the only if part, τ is an s-type and thus there exists a tree $t_1 \in L(D_1)$ and $v_1 \in \text{Dom}(t_1)$ such that $\text{anc-type}^{t_1}(v_1) = \tau$ and $t_1' = \text{subtree}^{t_1}(v_1) \in S_1(\tau) \setminus S_2(\tau)$. Therefore applying (a) we get that $t'' = t[v \leftarrow t_1'] \in L(D_1) \cup L(D_2)$. From the definition of t_1' it must be that $t'' \in L(D_1)$, and thus $\text{context}^t(v) = \text{context}^{t''}(v) \in C_1(\tau)$.

Similarly one can prove equivalence of (b) and (b').

Now we define a single-type EDTD $D' = (\Sigma, \Delta, d', S_{d'}, \mu)$ such that $L(D') = \text{nv}(D_2, D_1)$. Intuitively, D' will check locally whether conditions (a') and (b') are satisfied. For example, if $\tau = (\tau_1, \tau_2)$ is a c-type, then in order to satisfy $\text{subtree}^t(v) \in S_1(\tau)$ we have to check whether $\text{ch-str}^t(v) \in \mu_1(d_1(\tau_1))$. From Lemma 4.5 it will follow that together these local checks test whether (a') and (b') hold.

For a type $\tau_2 \in \Delta_2$, we define

$$\text{slab}(\tau_2) := \{a \in \Sigma \mid \delta_2(\tau_2, a) \text{ is an } s\text{-type}\}.$$

For every $\tau = (\tau_1, \tau_2) \in \Delta$, we define d' such that

$$\mu(d'(\tau)) = \begin{cases} \mu_2(d_2(\tau_2)) \cap \mu_1(d_1(\tau_1)) & \text{if } \tau \text{ is a } c\text{-type} \\ \big(\mu_2(d_2(\tau_2)) \cap (\Sigma \setminus \text{slab}(\tau_2))^*\big) & \\ \quad \cup \big(\mu_2(d_2(\tau_2)) \cap \mu_1(d_1(\tau_1)) & \\ \cap (\Sigma^* \cdot \text{slab}(\tau_2) \cdot \Sigma^*)\big) & \text{if } \tau \text{ is not a } c\text{-type} \end{cases}$$

That is, when τ is a c-type, $\mu(d'(\tau))$ contains exactly the intersection of $\mu_1(d_1(\tau_1))$ and $\mu_2(d_2(\tau_2))$. When τ is not a c-type, it contains the strings in $\mu_2(d_2(\tau_2))$ for which none of the symbols lead to an s-type, and the strings in $\mu_2(d_2(\tau_2)) \cap \mu_1(d_1(\tau_1))$, for which one of the elements leads to an s-type.

Moreover, in $d'(\tau)$, the type associated to any alphabet symbol a, i.e., the type τ' such that $\mu(\tau') = a$, is $\tau' = \big(\delta_1(\tau_1, a)), \delta_2(\tau_2, a))\big)$.

To show that $L(D') = \text{nv}(D_2, D_1)$, we need the following lemma.

Lemma 4.5. *Let* $t \in L(D')$, $v, u \in \text{Dom}(t)$ *and* $\tau_v = \text{anc-type}^t(v)$, $\tau_u = \text{anc-type}^t(u)$. *Then,*

(a) if τ_v is an s-type and u is the parent of v, then τ_u is an s-type;

(b) if τ_v is an s-type and u is a sibling of v, then τ_u is a c-type; and,

(c) if τ_v is a c-type and u is a child of v, then τ_u is a c-type.

We show that any tree $t \in L(D')$ satisfies (a') and (b') and thus $L(D') \subseteq \text{nv}(D_2, D_1)$. Thereto, let $t \in L(D')$, $v \in \text{Dom}(t)$ and $\tau = (\tau_1, \tau_2) = \text{anc-type}^t(v)$. From the definition of $d'(\tau)$, if τ is a c-type or v has a child which type is an s-type, then $\mu(d'(\tau)) \subseteq \mu_1(d_1(\tau_1))$.

To show that (b') holds, suppose that τ is a c-type. Then applying Lemma 4.5(c) recursively we get that, for every descendant u of v, with the type $\tau_u = (\tau_{u,1}, \tau_{u,2}) = \text{anc-type}^t(u)$, τ_u is a c-type. Hence, by construction of d', $\mu(d'(\tau_u)) \subseteq \mu_1(d_1(\tau_{u,1}))$. It follows that $\text{subtree}^t(v) \in S_1(\tau)$.

For (a'), assume that τ is an s-type. By Lemma 4.5(a) and (b), for every $u \in \text{Dom}(\text{context}^t(v))$, the type $\tau_u =$

anc-type$^t(u)$ is either an s-type or a c-type. More specifically, for all such nodes u not on the path from the root to v, τ_u is a c-type. Thus, by construction of d', $\mu(d'(\tau_u)) \subseteq \mu_1(d_1(\tau_{u,1}))$. For all nodes u on the path from the root to v, τ_u is an s-type. As any such node thus has a child which has an s-type, again by construction of d', $\mu(d'(\tau_u)) \subseteq \mu_1(d_1(\tau_{u,1}))$. Hence, context$^t(v) \in C_1(\tau)$.

Therefore, t satisfies conditions (a') and (b') and thus $L(D') \subseteq \mathrm{nv}(D_2, D_1)$. On the other hand, it can be shown that every tree which satisfies (a') and (b') belongs to $L(D')$ and thus $\mathrm{nv}(D_2, D_1) \subseteq L(D')$. Hence, $L(D') = \mathrm{nv}(D_2, D_1)$.

Lemma 4.6. *Let D_1 and D_2 be two single-type EDTDs. Then, $\mathrm{nv}(D_2, D_1)$ is definable by a single-type EDTD. Moreover, it is computable in time polynomial in $|D_1| + |D_2|$.*

Proof. We can calculate the set of s-types and the set of c-types in polynomial time. As also the content models in D' can be constructed in polynomial time, the single-type EDTD D' which defines $\mathrm{nv}(D_2, D_1)$ can be computed in polynomial time. □

Lemma 4.7. *Let D_1 and D_2 be two single-type EDTDs. The language $L(D_1) \cup \mathrm{nv}(D_2, D_1)$ is definable by a single-type EDTD.*

Proof. Let $E = \mathrm{nv}(D_2, D_1)$. From Lemma 4.6 E is regular, thus $L(D_1) \cup E$ is also regular.

We prove that $L(D_1) \cup E$ is closed under ancestor-guarded subtree exchange. Assuming otherwise, there exist trees $t_1, t_2 \in L(D_1) \cup E$ and $t_B \in \mathrm{closure}(t_1, t_2)$ such that $t_B \notin L(D_1) \cup E$. From Lemma 4.6, both $L(D_1)$ and E are closed under ancestor-guarded subtree exchange. Thus we only have to consider the case where $t_1 \in L(D_1)$ and $t_2 \in E$.

From the definition of E, $t_B \in L(D_2) \setminus E$ and there exist trees $t_A \in L(D_1)$ and $t \in \mathrm{closure}(t_A, t_B)$ such that $t \notin L(D_1) \cup L(D_2)$.

Therefore at least one of $t(t_A, t_B(t_1, t_2))$, $t(t_A, t_B(t_2, t_1))$, $t(t_B(t_1, t_2), t_A))$ or $t(t_B(t_2, t_1), t_A))$ is a derivation tree of $t \notin L(D_1) \cup L(D_2)$ with respect to $L(D_1) \cup \mathrm{nv}(D_2, D_1)$. It can be proved that such a tree cannot exist. □

Theorem 4.8. *Let D_1 and D_2 be single-type EDTDs. The language $L(D_1) \cup \mathrm{nv}(D_2, D_1)$ is a maximal lower XSD-approximation of $L(D_1) \cup L(D_2)$. It is a unique maximal lower XSD-approximation which includes $L(D_1)$.*

Proof. From Lemma 4.7, $L(D_1) \cup \mathrm{nv}(D_2, D_1)$ is a lower XSD-approximation of $L(D_1) \cup L(D_2)$. It is maximal and unique from the definition of non-violating set. (Uniqueness will also follows from Corollary 4.10.) □

We note that $L(D_1) \cup \mathrm{nv}(D_2, D_1)$ can be computed in polynomial time.

4.2.3 Relation with D_1 and D_2.

Previously, we have shown that when we fix D_1 there is a uniquely determined maximal regular subset $Y \subseteq L(D_2)$ such that $L(D_1) \cup Y$ is closed under ancestor-guarded subtree exchange. It remains open whether for every regular subset $X \subseteq L(D_1)$ there is a unique maximal regular subset $Y \subseteq L(D_2)$ such that $X \cup Y$ is closed under ancestor-guarded subtree exchange. We show that a maximal lower XSD-approximation is uniquely defined by its intersection with D_1 (and dually, it is uniquely defined by its intersection with D_2).

We will use the following lemma.

Lemma 4.9. *Let X, Y_1 and Y_2 be tree languages. If $X \cup Y_1$ and $X \cup Y_2$ are closed under ancestor-guarded subtree exchange, then $X \cup \mathrm{closure}(Y_1 \cup Y_2)$ is also closed under ancestor-guarded subtree exchange.*

Corollary 4.10. *Let A and B be two maximal lower XSD-approximations. If $A \cap D_1 = B \cap D_1$ then $A \cap D_2 = B \cap D_2$.*

Proof. Apply Lemma 4.9 to sets $X = A \cap D_1$, $Y_1 = A \cap D_2$ and $Y_2 = B \cap D_2$. Then we get that $A \cap D_1 \cup \mathrm{closure}(A \cap D_2 \cup B \cap D_2)$ is definable by a single-type EDTD and since $\mathrm{closure}(A \cap D_2 \cup B \cap D_2) \subseteq D_2$, it is a lower XSD-approximation. However it is a proper superset of A, unless $A \cap D_2 = B \cap D_2$. □

4.3 Complements of XSDs

Just as in the case of unions of XSDs, maximal lower XSD-approximations are not unique for complements of XSDs.

Theorem 4.11. *Let D be a DTD and let D_c be the EDTD for D's complement. In general, there does not exist a unique maximal lower XSD-approximation of $L(D_c)$, even over unary alphabets. The set of maximal lower XSD-approximations can be infinite.*

4.4 EDTDs

4.4.1 Existence of Maximal Lower XSD-Approximations

We say that a tree language T is *height-bounded* if there is a $k \in \mathbb{N}$ such that every tree from T has height at most k. We show that there exists a maximal lower XSD-approximation for every height-bounded regular tree language.

We introduce some terminology for the proof below. Let $(\mathcal{X}, \leq)$ be a *partially ordered set* (or, *poset*). A *chain* $\mathcal{C}$ is a set of elements from $\mathcal{X}$ such that for all $X, Y \in \mathcal{C}$, either $X \leq Y$ or $Y \leq X$.

A *forest* is an ordered sequence of trees (possibly empty). For a tree t and a node $v \in \mathrm{Dom}(t)$ such that subtree$^t(v) = a(t_1, \ldots, t_n)$, we denote by subforest$^t(v)$ the forest $t_1, \ldots, t_n$.

A *monoid forest automaton* [6] $\mathcal{A} = ((Q, +, q_0), \Sigma, \delta, F)$ is a deterministic automaton where $(Q, +, q_0)$ is a finite monoid[3] (a set of states with an operation for composition of states), $\delta : \Sigma \times Q \to Q$ is the transition function and $F \subseteq Q$ is a set of final states. The automaton assigns to every forest t a value $\mathcal{A}(t) \in Q$ which is defined as follows: (i) if t is empty, then $\mathcal{A}(t) = q_0$, (ii) if $t = a(s)$ for some forest s, then $\mathcal{A}(t) = \delta(a, \mathcal{A}(s))$, and (iii) if $t = t_1, \ldots, t_n$ for some trees $t_1, \ldots, t_n$, then $\mathcal{A}(t) = \mathcal{A}(t_1) + \ldots + \mathcal{A}(t_n)$. A forest is accepted by $\mathcal{A}$ if $\mathcal{A}(t) \in F$.

Theorem 4.12. *Let T be a height-bounded regular tree language. For every lower XSD-approximation X of T, there is a maximal lower XSD-approximation M of T with $X \subseteq M$.*

Proof. Let $(\mathcal{X}, \subseteq)$ be a poset of all lower XSD-approximations of T which include X. Obviously, $X \in \mathcal{X}$. Now let us take a non-empty chain $\mathcal{C}$ from the poset and define $X_{\mathcal{C}}$ as the union of all tree languages from $\mathcal{C}$. We show that $X_{\mathcal{C}}$ is

[3]Recall that a monoid is a set equiped with an associative composition operator and an identity element.

closed under ancestor-guarded subtree exchange. Indeed, for any two trees $t_1, t_2 \in X_\mathcal{C}$ there are two languages $X_1, X_2 \in \mathcal{C}$ such that $t_1 \in X_1$ and $t_2 \in X_2$. Since $\mathcal{C}$ is a chain we have either $X_1 \subseteq X_2$ or $X_2 \subseteq X_1$. W.l.o.g. we assume the latter, thus $t_1, t_2 \in X_1$, and since X_1 is a lower XSD-approximation we have $\text{closure}(t_1, t_2) \subseteq X_1 \subseteq X_\mathcal{C}$.

Hence, $X_\mathcal{C} \in \mathcal{X}$ and thus $X_\mathcal{C}$ is an upper bound of the chain $\mathcal{C}$. Therefore we can apply the Kuratowski-Zorn lemma [8] to the poset, from which it follows that there is at least one maximal element M in $(\mathcal{X}, \subseteq)$.

Therefore, there is a maximal set M which satisfies $X \subseteq M \subseteq T$ and which is closed under ancestor-guarded subtree exchange. We will show that M is a regular tree language.

Let us generalize the notion of single-type EDTDs to non-regular languages. In a *generalized* single-type EDTD we allow d to map symbols to non-regular string languages. Since M is closed under ancestor-guarded subtree exchange, we can define it by a generalized single-type EDTD $D = (\Sigma, \Delta, d, S_d, \mu)$. Let A be the type automaton for D. Since M is height-bounded, we can take such D that for every $\tau \in \Delta$ there is exactly one string w with $A(w) = \tau$. Let $\mathcal{A} = ((Q, +, q_0), \Sigma, \delta_\mathcal{A}, F)$ be a monoid forest automaton for T.

Let us assume that M is not regular. The height-bounded language M is not regular if and only if there is at least one $\tau \in \Delta$ for which $d(\tau)$ is not regular. Let us fix such a $\tau_\star$.

For every $a \in \Sigma$, let τ_a be a type which appears in $d(\tau_\star)$ and $\mu(\tau_a) = a$ (τ_a is undefined if there is no such type). Moreover, let

$$\begin{aligned}
L_a &= \{\text{subtree}^t(v) \mid t \in M, v \in \text{Dom}(t), \text{anc-type}^t(v) = \tau_a\},\\
Q_a &= \{q \in Q \mid \exists t \in L_a, \mathcal{A}(t) = q\},\\
Q_F &= \{q \in Q \mid \exists t \in M, v \in \text{Dom}(t), \text{anc-type}^t(v) = \tau_\star,\\
&\qquad\qquad \mathcal{A}(\text{subforest}^t(v)) = q\}.
\end{aligned}$$

Now we build a word automaton $\mathcal{B} = (2^Q, \Sigma, \delta_\mathcal{B}, \{q_0\}, 2^{Q_F})$ with transition function

$$\delta_\mathcal{B}(S, a) = \{q_1 + q_2 \mid q_1 \in S, q_2 \in Q_a\}.$$

Finally, we introduce $D' = (\Sigma, \Delta, d', S_d, \mu)$ with $d'(\tau) = d(\tau)$ for any $\tau \neq \tau_\star$, $d'(\tau_\star)$ contains only types from $\{\tau_a \mid a \in \Sigma\}$ and $\mu(d'(\tau_\star)) = L(\mathcal{B})$. It is clear that $L(D')$ is closed under ancestor-guarded subtree exchange and $M \subset L(D')$. We show that $L(D') \subseteq T$.

Let $t \in L(D')$ and let $v_1, \dots, v_k \in \text{Dom}(t)$ be nodes with $\text{anc-type}^t(v_i) = \tau_\star$. Let $f_i = \text{subforest}^t(v_i)$. Since $\mathcal{A}(f_i) \in Q_F$, we can find another forest f_i' such that

$$\mathcal{A}(f_i) = \mathcal{A}(f_i') \qquad (1)$$

and the tree t', obtained by replacing every f_i with f_i', belongs to M. Therefore, $t' \in T$ and from (1) $t \in T$.

Applying the above procedure until no type τ, with non-regular $d(\tau)$, can be found results in a regular set M' with $M \subset M' \subseteq T$. This contradicts the maximality of M and thus M is itself regular. □

4.4.2 Testing Maximal Lower XSD-Approximations

Let S be a single-type EDTD that is a lower approximation of an EDTD D. It is a maximal lower approximation if and only if

$$\text{there is no } t \in L(D) - L(S), \text{ with } \text{closure}(L(S) \cup \{t\}) \subseteq L(D).$$

Let T be a regular tree language and N be an NFA. The *type-closure* of T w.r.t. N, denoted by $\text{type-closure}^N(T)$ is the smallest language which contains T and is closed under ancestor-type-guarded subtree exchange w.r.t. N. Due to Theorem 4.2, S is a maximal lower approximation if and only if

$$\text{there is no } t \in L(D) - L(S), \text{ with } \text{type-closure}^N(L(S) \cup \{t\}) \subseteq L(D).$$

In the above statement, N is the type automaton of an EDTD for $\text{closure}(L(S) \cup \{t\})$. One approach for an algorithm to decide whether S is a maximal lower approximation could therefore be to guess an N and t such that the above property holds. However, we do not know a size bound on both N or t.

Here, we can solve one aspect of this problem: once we know N, the size of t is no longer problematic. However, the size of N is dependent of t and therefore, can also be arbitrarily large. For this reason, we need to restrict to height-bounded tree languages. If $L(D)$ and $L(S)$ are height-bounded by k, then we can bound the number of states of a deterministic type automaton for $\text{closure}(L(S) \cup \{t\})$ with $k \times \Sigma \times |S|$ states. The reason is that t contains at most $k|\Sigma|$ different ancestor-strings w. Since $\text{closure}(L(S) \cup \{t\})$ is closed under ancestor-guarded subtree exchange, each such ancestor-string w must arrive in the same state in the type automaton.

More formally, let N_k be the minimal DFA for the language $\cup_{0 \leq \ell \leq k} \Sigma^\ell$. Notice that, for languages height-bounded by k, closure under ancestor-guarded subtree exchange is exactly the same as closure under type-guarded subtree exchange by N_k. Therefore, for height-bounded languages by k, S is a maximal lower approximaion if and only if

$$\text{there is no } t \in L(D) - L(S), \text{ with } \text{type-closure}^{N_k}(L(S) \cup \{t\}) \subseteq L(D).$$

Our plan is to construct a *tree automaton*[4] for the language $\{t \in L(D) - L(S) \mid \text{type-closure}^{N_k}(L(S) \cup \{t\}) \subseteq L(D)\}$. This tree automaton accepts the empty language if and only if S is a maximal lower approximation. Constructing such a tree automaton, however, is not trivial. The main technical difficulty lies in the following Lemma:

LEMMA 4.13. *We can construct a tree automaton for $\{t \in L(D) - L(S) \mid \text{type-closure}^{N_k}(\{t\} \cup L(S)) \subseteq L(D)\}$ in time double exponential in $|D| + |S| + |N_k|$.*

Since emptiness testing is in PTIME for tree automata, we obtain the following Theorem:

THEOREM 4.14. *Deciding whether a single-type EDTD S is a maximal lower XSD-approximation of an EDTD D is in 2EXPTIME, if both S and D define height-bounded tree languages.*

[4]A tree automaton is an automata-theoretic model corresponding to EDTDs.

5. CONTENT MODELS

In the previous sections, we always represented content models in schemas by DFAs. We next discuss what changes when using regular expressions or NFAs.

For NFAs all remains the same, except for the following: Lemma 3.5 becomes PSPACE-complete, since already inclusion testing for NFAs is PSPACE-complete. The size of the optimal upper approximation of the complement of an XSD can become exponentially large (Theorem 3.10), since complementing an NFA causes an exponential blow-up.

For regular expressions things are similar to NFAs. Again, Lemma 3.5 becomes PSPACE-complete. Since the smallest expression for the intersection of two regular expressions can be exponential, and since complementing a regular expression can cause a double-exponential blow-up [11], we have an (optimal) exponential upper bound for Theorem 3.7 and an optimal double exponential upper bound for Theorem 3.10.

For deterministic regular expression the complexity of all decision problems remains the same as there is an efficient translation to DFAs. Unfortunately, we lose uniqueness. As is shown in [4], in general, there exists no best approximation for an arbitrary regular language by a deterministic regular expression. However, heuristics are available to transfer a DFA to a concise deterministic regular expressions which is an upper approximation of the given DFA [4]. So the present methods for computing upper approximations given in Section 3 followed by a translation of DFAs to deterministic regular expressions using the methods of [4] provides an algorithm for approximating real world XSDs.

Furthermore, the complexity of minimizing stEDTDs also depends on the formalism for the content models. In particular, for NFAs or DREs, deciding minimality of an single type EDTD is already PSPACE-complete.

6. CONCLUSION

We showed that the case of optimal upper approximations behaves very well: there always exists a unique one and for union and difference the latter is even tractable. In combination with the methods of [4], the present work provides usable algorithms for computing upper XSD-approximations. Optimal lower approximations, in strong contrast, are much less understood. The most important open problem is undoubtedly the question whether there is an optimal lower approximation for every regular tree language.

7. REFERENCES

[1] J. Albert, D. Giammerresi, and D. Wood. Normal form algorithms for extended context free grammars. *Theoretical Computer Science*, 267(1–2):35–47, 2001.

[2] D. Barbosa, L. Mignet, and P. Veltri. Studying the XML Web: Gathering statistics from an XML sample. *World Wide Web*, 8(4):413–438, 2005.

[3] P. A. Bernstein. Applying model management to classical meta data problems. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[4] G. J. Bex, W. Gelade, W. Martens, and F. Neven. Simplifying XML Schema: effortless handling of nondeterministic regular expressions. In *SIGMOD*, pages 731–744, 2009.

[5] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML Schema Definitions from XML data. In *International Conference on Very Large Data Bases (VLDB)*, pages 998–1009, 2007.

[6] Mikolaj Bojanczyk. Forest expressions. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 146–160. Springer, 2007.

[7] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1, april 3, 2001. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.

[8] K. Ciesielski. *Set Theory for the Working Mathematician*. Cambridge University Press, 1997.

[9] J. Clark and M. Murata. Relax NG specification. http://www.relaxng.org/spec-20011203.html, December 2001.

[10] S. Gao, C. M. Sperberg-McQueen, H.S. Thompson, N. Mendelsohn, D. Beech, and M. Maloney. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. W3C, April 2009.

[11] W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 325–336, 2008.

[12] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[13] W. Martens, F. Neven, and T. Schwentick. Simple off the shelf abstractions for XML Schema. *Sigmod RECORD*, 36(3):15–22, 2007.

[14] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *Siam Journal on Computing*, 39(4):1486–1530, 2009.

[15] W. Martens, F. Neven, T. Schwentick, and G.J. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

[16] W. Martens and J. Niehren. On the minimization of xml schemas and tree automata for unranked trees. *Journal of Computer and System Sciences*, 73(4):550–583, 2007.

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

[18] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *International Symposium on Principles of Database Systems (PODS)*, pages 35–46, 2000.

[19] A. Sahuguet. Everything you ever wanted to know about DTDs, but were afraid to ask. In *International Workshop on the Web and Databases (WebDB)*, pages 69–74, 2000.

[20] H. Seidl. Deciding equivalence of finite tree automata. *Siam Journal on Computing*, 19(3):424–437, 1990.