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

Tomasz Idziaszek[a], Wim Martens[b,*,1], Frank Neven[c], Jan Paredaens[d]

[a] *University of Warsaw, Warsaw, Poland*
[b] *Technical University of Dortmund, Dortmund, Germany*
[c] *Hasselt University and Transnational University of Limburg, Hasselt, Belgium*
[d] *University of Antwerp, Belgium*

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.

Key words: 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, 20]. 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

☆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.

*Corresponding author

Email addresses: idziaszek@mimuw.edu.pl (Tomasz Idziaszek), wim.martens@udo.edu (Wim Martens), frank.neven@uhasselt.be (Frank Neven), jan.paredaens@ua.ac.be (Jan Paredaens)

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

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 [17], 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 [18]. 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 [16] 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

2.1. Strings, Trees, and Contexts

For a finite set S, we denote by $|S|$ its cardinality. 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 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 a regular expression r and $L(N)$ for the language defined by a 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 Σ-tree. A *tree language* is a set of trees.

For every tree t, the *set of nodes* of t, denoted by 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 $L(d)$ 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.[2] 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 [18, 19]:

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.

[2]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.

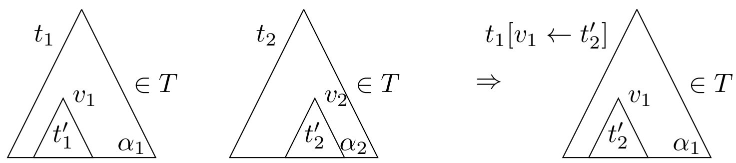


Figure 1: Ancestor-guarded subtree exchange (anc-str$^{t_1}(v_1)$ = anc-str$^{t_2}(v_2)$).

Extended DTDs are well-known to define the class of *unranked regular tree languages (UREG)* [7, 19]. 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, 15]).*

As the *Element Declarations Consistent* rule severely constrains the use of the typing mechanism [11], extended DTDs do not constitute a satisfactory abstraction of XSDs. Therefore, XSDs are commonly abstracted as single-type EDTDs [18, 16, 14]:

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_1 and τ_2 exist with $\mu(\tau_1) = \mu(\tau_2)$ such that *(i)* $\tau_1, \tau_2 \in S_d$; or, *(ii)* there is a type τ such that $w_1\tau_1 v_1 \in d(\tau)$ and $w_2\tau_2 v_2 \in d(\tau)$ for some strings w_1, v_1, w_2, and v_1.

In particular, condition (*ii*) states that τ_1 and τ_2 can occur in the same content-model albeit not necessarily in the same string.

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 [16, 14]. One of these is a simple extension of DTDs, which we define next. We denote by 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 $D = (A, d)$, where $A = (Q, \Sigma, \delta, \{q_{\text{init}}\}, \emptyset)$ is a state-labeled DFA with initial state q_{init} and without final states, and d is a function from $Q \setminus \{q_{\text{init}}\}$ to regular languages over Σ.

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

Proposition 2.6 ([14])**.** *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, anc-str$^{t_1}(v_1)$ = anc-str$^{t_2}(v_2)$ then

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

Theorem 2.8 ([16])**.** *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 [21], and the close correspondence between EDTDs and tree automata discussed by Papakonstantinou and Vianu [19].

Theorem 2.10 ([19, 21])**.** *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

Next, we introduce some definitions to capture single-typedness in terms of closure under ancestor-guarded subtree exchange.

Definition 2.11. Let T be a tree language. We denote by $\mathrm{closure}(T)$ the smallest tree language closed under ancestor-guarded subtree exchange which contains T. We will write $\mathrm{closure}(t_1, t_2)$ 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 $\text{anc-str}^{t_1}(v_1) = \text{anc-str}^{t_2}(v_2)$. For each $i \in I$ we have $t_1, t_2 \in X_i$ and thus $t = t_1[v_1 \leftarrow \mathrm{subtree}^{t_2}(v_2)] \in X_i$. Therefore $t \in X$, and thus X is closed under ancestor-guarded subtree exchange. □

When $t \in \mathrm{closure}(X)$ then t can be obtained from trees in X by using the subtree exchange property. The next definition is a formalization to this end.

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

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

Lemma 2.14. *Let X be a tree language and t a tree. Then $t \in \mathrm{closure}(X)$ if and only if t has a derivation tree with respect to X.*

Proof. It is immediate that whenever t has a derivation tree ϑ with respect to X, then $t \in \mathrm{closure}(X)$. Indeed, all leaf nodes of ϑ are labeled with trees of X, and all internal nodes are labeled by trees obtained by applying ancestor-guarded subtree exchange to their children. Hence, all trees occurring in ϑ are in $\mathrm{closure}(X)$. As the root of ϑ is labeled with t, $t \in \mathrm{closure}(X)$.

For the converse direction, let T_i be the set of the trees from $\mathrm{closure}(X)$ which have a derivation tree of height i. Clearly $T_0 = X$. If ϑ is a derivation tree for t, then $t(\vartheta, \vartheta)$ is too, thus $T_i \subseteq T_{i+1}$. We show that $T = \bigcup_{i\in\mathbb{N}} T_i$ is closed under ancestor-guarded subtree exchange. Indeed, for every $t_1, t_2 \in T$ there exist n_1, n_2 such that $t_1 \in T_{n_1}$ and $t_2 \in T_{n_2}$. Hence, any tree t obtained by applying ancestor-guarded subtree exchange to t_1 and t_2 is in $T_{\max(n_1,n_2)+1} \subseteq T$. Hence, T is closed under ancestor-guarded subtree exchange. As $\mathrm{closure}(X)$ is the smallest set closed under ancestor-guarded subtree exchange which contains X, $\mathrm{closure}(X) \subseteq T$. □

The next lemma will only be used in Section 4.2.2.

Lemma 2.15. *Let X be a tree language and t a tree in* $\mathrm{closure}(X)$*. If ϑ is a derivation tree of t with respect to X and $\vartheta = t(\vartheta_A, t_B(\vartheta_1, \vartheta_2))$ for some subtrees $\vartheta_1, \vartheta_2, \vartheta_A$ then*

(a) $t(\vartheta_A, \vartheta_1)$,

(b) $t(\vartheta_A, \vartheta_2)$, or,

(c) both $t(t_C(\vartheta_A, \vartheta_1), \vartheta_2)$ and $t(t_C(\vartheta_A, \vartheta_1), t_B(\vartheta_1, \vartheta_2))$ for some subtree t_C

are also derivation trees of t with respect to X.

With the same premises but $\vartheta = t(t_B(\vartheta_1, \vartheta_2), \vartheta_A)$ we have that

(a) both $t(t_D(\vartheta_1, \vartheta_A), \vartheta_2)$ and $t(t_D(\vartheta_1, \vartheta_A), t_B(\vartheta_1, \vartheta_2))$ for some subtree t_D, or

(b) both $t(\vartheta_1, t_E(\vartheta_2, \vartheta_A))$ and $t(t_B(\vartheta_1, \vartheta_2), t_E(\vartheta_2, \vartheta_A))$ for some subtree t_E, or,

(c) $t(\vartheta_1, \vartheta_A)$

is also a derivation tree of t with respect to X.

Proof. Let t_A, t_1 and t_2 be the root labels of subtrees ϑ_A, ϑ_1 and ϑ_2, respectively. From the definition of derivation tree for $i \in \{A, B, 1, 2\}$ there exist nodes $v_i \in \mathrm{Dom}(t_i)$ such that $\text{anc-str}^{t_1}(v_1) = \text{anc-str}^{t_2}(v_2)$, $\text{anc-str}^{t_A}(v_A) = \text{anc-str}^{t_B}(v_B)$, $t_B = t_1[v_1 \leftarrow \mathrm{subtree}^{t_2}(v_2)]$ and $t = t_A[v_A \leftarrow \mathrm{subtree}^{t_B}(v_B)]$. From definition of t_B it is clear that $v_1 \in \mathrm{Dom}(t_B)$. We consider three cases with respect to the position of nodes v_1 and v_B in t_B: (a) subtrees rooted at v_1 and v_B are disjoint, (b) v_B is in the subtree rooted at v_1 and (c) v_1 is in the subtree rooted at v_B.

It is easy to see that in cases (a) and (c) we have $v_B \in \mathrm{Dom}(t_1)$, and in case (b) there is $v'_B \in \mathrm{Dom}(t_2)$ such that $v'_B = v_2 w$, $v_B = v_1 w$ for some $w \in \Sigma^*$.

Let's first consider case (I) when $\vartheta = t(t_A, t_B(t_1, t_2))$. For (a) we have $t = t_A[v_A \leftarrow \mathrm{subtree}^{t_1}(v_B)]$. For (b) we have $t = t_A[v_A \leftarrow \mathrm{subtree}^{t_2}(v'_B)]$. For (c) we have $t_C = t_A[v_A \leftarrow \mathrm{subtree}^{t_1}(v_B)]$ and $t = t_C[v'_1 \leftarrow \mathrm{subtree}^{t_2}(v_2)] = t_C[v'_1 \leftarrow \mathrm{subtree}^{t_B}(v_1)]$ where $v'_1 \in \mathrm{Dom}(t_C)$ and $v'_1 = v_A w$, $v_1 = v_B w$ for some $w \in \Sigma^*$.

Now let's consider case (II) when $\vartheta = t(t_B(t_1, t_2), t_A)$. For (a) we have $t_D = t_1[v_B \leftarrow \mathrm{subtree}^{t_A}(v_A)]$ and $t = t_D[v_1 \leftarrow \mathrm{subtree}^{t_2}(v_2)] = t_D[v_1 \leftarrow \mathrm{subtree}^{t_B}(v_1)]$. For (b) we have $t_E = t_2[v'_B \leftarrow \mathrm{subtree}^{t_A}(v_A)]$ and $t = t_1[v_1 \leftarrow \mathrm{subtree}^{t_E}(v_2)] = t_B[v_1 \leftarrow \mathrm{subtree}^{t_E}(v_2)]$. For (c) we have $t = t_1[v_B \leftarrow \mathrm{subtree}^{t_A}(v_A)]$. □

3. Upper XSD-Approximations

In this section, we consider upper XSD-approximations of EDTDs. In general, constructing a minimal upper XSD-approximation of an EDTD requires exponential time. However, given two single-type EDTDs D_1 and D_2, we can construct minimal 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 a unique minimal 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.

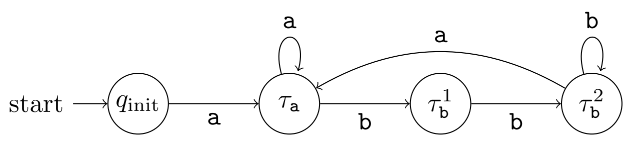
Definition 3.1. The *type automaton* of an EDTD $D = (\Sigma, \Delta, d, S_d, \mu)$ is a state-labeled NFA $N = (Q, \Sigma, \delta, \{q_{\text{init}}\})$ without final states such that $Q = \Delta \uplus \{q_{\text{init}}\}$ and for each $q \in Q$

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

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

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

Then, this is the type automaton of D:



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 minimal 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 $\Omega(2^n)$.*

Proof. We first show that, given EDTD $D = (\Sigma, \Delta, d, S_d, \mu)$, determinizing its type automaton results in a DFA-based DTD $D' = (A, d')$ which is the unique minimal upper XSD-approximation of D. Thereto, we show that (1) $L(D) \subseteq L(D')$, and (2) $L(D') \subseteq \text{closure}(L(D))$. The first condition says that D' is indeed an upper XSD-approximation of D. Any upper XSD-approximation of D must contain $L(D)$ and must be closed under ancestor-guarded subtree exchange. Since closure$(L(D))$ is the smallest set which satisfies these requirements, thus the second condition says that D' is indeed the smallest upper XSD-approximation, therefore the minimal upper XSD-approximation.

We first show that $L(D) \subseteq L(D')$. Thereto, let t be a tree in $L(D)$. Hence, there exists a tree t' such that $t' \in L(d)$ and $\mu(t') = t$. That is, for every node v of t', $\text{ch-str}^{t'}(v)$ is in $d(\text{lab}^{t'}(v))$. Let v be a node of t, and $S = A(\text{anc-str}^t(v))$. Then, by construction of D', $\text{lab}^{t'}(v) \in S$. Indeed, S contains all types which a node with the ancestor string $\text{anc-str}^t(v)$ can have, and $\text{lab}^{t'}(v)$ must clearly be one of them. But then, as $d'(S) = \bigcup_{\tau \in S} \mu(d(\tau))$, and we know that $\text{ch-str}^{t'}(v)$ is in $d(\tau)$, it follows that $\text{ch-str}^t(v)$ is in $d'(S)$. As this holds for all nodes of t, $t \in L(D')$.

We next show that $L(D') \subseteq \text{closure}(D)$. Thereto, let t be a tree in $L(D')$. We will show that t is also in closure(D), by explicitly constructing t using only trees in $L(D)$ and the subtree exchange property. We do this as follows: we iterate over the nodes of t in a breadth first manner, such that when we reach a node v, we have constructed a tree t_v, such that (1) $t_v \in \text{closure}(D)$ and (2) the parts of t and t_v up to v (in the breadth first order) are isomorphic. That is, the tree consisting of all nodes of t before and including v, and all their children, is isomorphic to the same initial part of t_v. Note that when v is the last node of t in the breadth first traversal, condition (2) ensures that $t_v = t$, and hence, by condition (1) $t \in \text{closure}(D)$.

In order to construct this sequence of trees t_v, we first assign a type τ_v to every node v of t. As t is in $L(D')$, for any node v of t, we can define $S_v = A(\text{anc-str}^t(v))$, and have $\text{ch-str}^t(v) \in d'(S_v)$. By definition of D', there is (at least) one τ in S_v such that $\text{ch-str}^t(v)$ is in $\mu(d(\tau))$. Denote this τ by τ_v.

We next construct the sequence of trees t_v for all nodes v of t in breadth first order. When v is the root node of t, let t_v be a tree such that D can accept t_v by assigning the type τ_v to the root node and such that $\text{ch-str}^{t_v}(v)$ equals $\text{ch-str}^t(v)$. As D is reduced, such a t_v must exist, and t_v satisfies both condition (1) and (2).

Next, let t_u be the already obtained tree, and v be the next node in breadth first order. Let t' be a tree containing a node v' such that $\text{anc-str}^{t'}(v')$ equals $\text{anc-str}^t(v)$, $\text{ch-str}^{t'}(v')$ equals $\text{ch-str}^t(v)$, and D accepts t' by assigning type τ_v to v'. By construction of D' and the fact that D is reduced, such a tree t' must exist. Then, t_v is constructed by exchanging the subtree rooted at v in t_u by the subtree rooted at v' in t'. Hence, t_v is in closure(D), and satisfies condition (2) as well. It follows that t is in closure(D), as desired.

Clearly (A, d') can be constructed in time exponential in D, by Proposition 2.6 the overall construction is in exponential time.

To show that this exponential size increase can not be avoided, it suffices to consider unary trees, i.e., trees in which every node has at most one child. Such a unary tree can be viewed as a word whose first position is the root node, and whose last position is the leaf. On unary trees, EDTDs and single-type EDTDs intuitively correspond to NFAs and DFAs, respectively, and thus the exponential size increase in translating from NFAs to DFAs carries over to EDTDs and single-type EDTDs.

More formally, for any n, consider the language $L_n = (a + b)^* a(a + b)^n$. It is well known that any DFA accepting this language must be of size exponential in n. Let D_n be an EDTD accepting only unary trees which correspond to words in L_n. That is, $L(D_n)$ contains only unary trees which have the property that the unique node at distance n of the leaf node is a, and all other nodes can be either a or b. Clearly, D_n is of size linear in n.

Let $D'_n = (A, d)$ be a DFA-based DTD such that $L(D_n) = L(D'_n)$. It suffices to show that D'_n is of size exponential in n, as one can translate in polynomial time between single-type EDTDs and DFA-based DTDs. Thereto, let A_n be obtained from A by making all states q of A, for which $\varepsilon \in L(d(q))$, final; and removing all transitions (q, σ, q') for which $\sigma \notin L(d(q))$. Then, $L(A_n) = L_n$ and thus A_n, and by extension A and D'_n, must be of size at least exponential in n. As $L(D'_n) = L(D_n)$, D'_n is the minimal upper XSD-approximation of D_n, and thus an exponential size increase can not be avoided in constructing such an upper XSD-approximation for general EDTDs. □

We conclude this subsection 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.

Proof. We provide a PTIME algorithm for the complement of the problem. Since PTIME is closed under complement, this proves the lemma.

Let $D_2 = (\Sigma, \Delta_2, d_2, S_{d_2}, \mu_2)$ and A_2 be the (deterministic) type automaton of D_2. A tree t is *not* in the language defined by the single-type EDTD D_2 if and only if there exists a node $u \in \mathrm{Dom}(t)$ such that $\text{ch-str}^t(u) \notin L(d_2(\tau))$, where $A_2(\text{anc-str}^t(u)) = \{\tau\}$. The intuition of the PTIME procedure is that we can check all paths up to such a node u such that this path can occur in a tree in $L(D_1)$. There are only polynomially many paths to check.

Since D_1 is reduced (Proviso 2.3), every string that can be handled by the type automaton A_1 of D_1 can occur as an ancestor-path of a tree in $L(D_1)$. More formally, for a string w, there exists a tree $t \in L(D_1)$ and a node u in t with $\text{anc-str}^t(u) = w$ if and only if $A_1(w) \neq \emptyset$.

Our PTIME algorithm consists of the following steps:

(1) Compute, using a dynamic programming algorithm, the set of pairs $P := \{(A_1(w), A_2(w)) \mid w \in \Sigma^*\}$.

(2) Test whether there exists a pair $(\tau_1, \tau_2) \in P$ for which $\mu_1(d_1(\tau_1)) \not\subseteq \mu_2(d_2(\tau_2))$.

Step (1) only requires polynomial time because there are only a quadratic number of such pairs. Step (2) is in PTIME since inclusion testing between DFAs is in PTIME. □

We recall the following result:

Theorem 3.6 ([22]). *The complexity of the language inclusion problem $L(X) \subseteq L(Y)$ is* PSPACE*-complete when X and Y are given as regular expressions or NFAs.*

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

Theorem 3.7. *Deciding whether a single-type EDTD is a minimal upper XSD-approximation of a given EDTD is* PSPACE*-complete.*

Proof. For the upper bound, let $D_1 = (\Sigma, \Delta_1, d_1, S_{d_1}, \mu_1)$ be a single-type EDTD and D be an EDTD. First, we test whether $L(D) \subseteq L(D_1)$, which can be done in PSPACE, according to Lemma 3.5. If $L(D) \not\subseteq L(D_1)$, we reject. Let D_2 be the minimal upper XSD-approximation of D according to Theorem 3.4. We claim that

(1) D_1 is the minimal upper XSD-approximation of D if and only if $L(D_1) \subseteq L(D_2)$;

(2) we can test whether $L(D_1) \subseteq L(D_2)$ in PSPACE, that is, without actually fully constructing D_2.

For (1), let $D_2 = (\Sigma, \Delta_2, d_2, S_{d_2}, \mu_2)$. Of course, D_1 is a minimal upper XSD-approximation for D if and only if $L(D_1) = L(D_2)$. But since $L(D_1)$ is a regular language that contains $L(D)$ and is closed under ancestor-guarded subtree exchange, and $L(D_2)$ is a *minimal* language with the same properties, we have that $L(D_1) = L(D_2)$ if and only if $L(D_1) \subseteq L(D_2)$.

For (2), notice that naively testing whether $L(D_1) \subseteq L(D_2)$ would cost double-exponential time, since D_2 is exponentially large. However, according to the proof of Theorem 4.10 in [15], testing the inclusion $L(D_1) \subseteq L(D_2)$ reduces to (1) computing a correspondence relation $R \subseteq \Delta_1 \times \Delta_2$ between their types and, (2) for each pair $(\tau_1, \tau_2) \in R$, testing the inclusion $\mu_1(d_1(\tau_1)) \subseteq \mu_2(d_2(\tau_2))$. In other words, we have that $L(D_1) \not\subseteq L(D_2)$ if and only if

$$\text{there is a } (\tau_1, \tau_2) \in R \text{ such that } \mu_1(d_1(\tau_1)) \not\subseteq \mu_2(d_2(\tau_2)).$$

We show that the latter can be tested in PSPACE. Since PSPACE is closed under complement, this would prove the theorem.

Let A_1 and A_2 be the (deterministic) type automata of D_1 and D_2, respectively. The relation R in the proof of Theorem 4.10 in [15] contains precisely the pairs $(\tau_1, \tau_2) \in \Delta_1 \times \Delta_2$ for which there exists a string w such that $A_1(w) = \{\tau_1\}$ and $A_2(w) = \{\tau_2\}$. Our PSPACE procedure consists of the following steps:

(a) Guess w and keep track of $(A_1(w), A_2(w))$ (without constructing A_2 itself).

(b) Test whether $\mu_1(d_1(\tau_1)) \not\subseteq \mu_2(d_2(\tau_2))$.

For step (a), we can guess w one symbol at a time and, by construction of D_2, the pair $(A_1(w), A_2(w))$ is equal to $(A_1(w), A(w))$, where A is the type automaton of D. We can do this by only keeping the last symbol of w in memory, a state of A_1, and a set of states of A, which only takes polynomial space.

Finally, let $(\tau, \{\tau_1, \ldots, \tau_k\})$ be the pair from R we obtain after having guessed w completely. We can now test step (2) by directly testing whether $\mu_1(d_1(\tau)) \not\subseteq \mu(d(\tau_1)) + \cdots + \mu(d(\tau_k))$. Since both $\mu_1(d_1(\tau))$ and $\mu(d(\tau_1)) + \cdots + \mu(d(\tau_k))$ can be represented by polynomial size NFAs or regular expressions, this test is also possible in PSPACE (Theorem 3.6).

The PSPACE lower bound for this theorem can be obtained from the fact that testing $L(A) \subseteq L(A_1) \cup \cdots \cup L(A_n)$ for DFAs $A, A_1, \ldots, A_n$ is PSPACE-complete. This is the complement of the well known intersection emptiness problem. □

3.2. Unions of XSDs

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

Theorem 3.8. *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 $L(D_1) \cup L(D_2)$ obtained by computing the cross-product of D_1 and D_2. The type automaton of D is the product[3] 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.

For the second part of the theorem fix $n \geq 1$ and consider the following single-type EDTD D_1 with $S_d = \{\tau_{\mathtt{a}}^0, \tau_{\mathtt{b}}^0\}$:

$$\begin{array}{lll} \tau_{\mathtt{a}}^i & \to & \tau_{\mathtt{a}}^{i+1} + \tau_{\mathtt{b}}^{i+1} + \varepsilon \quad \text{(for all } 0 \leq i < n-1) \\ \tau_{\mathtt{b}}^i & \to & \tau_{\mathtt{a}}^i + \tau_{\mathtt{b}}^i + \varepsilon \quad \text{(for all } 0 \leq i < n) \\ \tau_{\mathtt{a}}^{n-1} & \to & \tau_{\mathtt{b}}^n + \varepsilon \\ \tau_{\mathtt{b}}^n & \to & \tau_{\mathtt{b}}^n + \varepsilon \end{array}$$

The language $L(D_1)$ consists of unary trees which contains at most n nodes labeled with $\mathtt{a}$ (for $c \in \{\mathtt{a}, \mathtt{b}\}$ the type τ_c^i represents trees with root label c, which have at most $n - i$ nodes labeled with $\mathtt{a}$). By changing the roles of $\mathtt{a}$ and $\mathtt{b}$, we can define D_2 such that $L(D_2)$ consists of unary trees which contain at most n nodes labeled with $\mathtt{b}$. The type-size of both D_1 and D_2 is $O(n)$.

Clearly $L(D_1) \cup L(D_2)$ consists of unary trees which contains at most n nodes labeled with $\mathtt{a}$ or at most n nodes labeled with $\mathtt{b}$. We show that type-size of D' is $\Omega(n^2)$.

Let N' be the type automaton for D'. Let $\tau_{k,\ell} = N'(\mathtt{a}^k\mathtt{b}^\ell)$ for $1 \leq k, \ell \leq n$. Consider now types for $(k, \ell) \neq (k', \ell')$ and let us assume that $\tau_{k,\ell} = \tau_{k',\ell'}$. W.l.o.g. we can assume that $k > k'$. Both unary trees $t = \mathtt{a}^k\mathtt{b}^{2n}\mathtt{a}^{n-k}$ and $t' = \mathtt{a}^{k'}\mathtt{b}^{2n}\mathtt{a}^{n-k'}$ are in $L(D)$. Therefore applying ancestor-type-guarded subtree exchange to node $v = 1^{k+\ell-1}$ in $\mathrm{Dom}(t)$ and node $v' = 1^{k'+\ell'-1}$ in $\mathrm{Dom}(t')$ we get that a tree

[3] For more details on the standard product construction of automata, see, e.g., [13].

$t'' = t[v \leftarrow \text{subtree}^{t'}(v')]$ also belongs to $L(D)$ which is impossible since $t'' = \mathtt{a}^k\mathtt{b}^{\ell+2n-\ell'}\mathtt{a}^{n-k'}$ contains more than n nodes labeled with $\mathtt{b}$ and $n-k'+k > n$ nodes labeled with $\mathtt{a}$. Therefore all types $\tau_{k,\ell}$ for $1 \leq k, \ell \leq n$ are pairwise different. $\square$

3.3. Intersections of XSDs

We start with the following immediate observation.

Proposition 3.9. *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.*

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 3.10. *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, defines 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 3.8, with the difference that now we need to construct the *intersection* of the two internal DFAs. That is, 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.

The second part of the theorem is similar to the lower bound proof of Theorem 3.8. If we have a large $n \geq 1$, we can take the single-type EDTDs D_1 and D_2 accepting unary trees of the form a^{p_1} and a^{p_2}, where $p_1 \neq p_2$ are the two smallest prime numbers larger than n. Then, the type size of the single-type EDTD for $L(D_1) \cap L(D_2)$ is $\Omega(p_1 p_2)$, which proves the 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 3.11. *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 towards 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 intersection in polynomial time, we also have the following corollary.

Corollary 3.12. *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. We give an example of this in Theorem 4.3. Nevertheless, we show that there is a unique maximal lower approximation that includes all of D_1 and there is a unique maximal lower approximation that includes all of D_2. That is, there is a well-defined maximal part of D_1 (symmetrically, maximal part of D_2) which can be added to D_2 (respectively to D_1) to form a maximal lower approximation of $D_1 \cup D_2$. Also the complement cannot be uniquely approximated in general. We do not know whether for every EDTD there always exists at least one maximal lower approximation. That is, we do not know whether it is possible to have an infinite sequence of lower approximations that converges to the language of the EDTD but never reaches a fixpoint. For the class of bounded depth schemas, we show that there is at least one maximal lower approximation. Finally, we discuss the complexity of deciding whether a given single-type EDTD is a maximal 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 tree language 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. an EDTD D* if it is closed under ancestor-type-guarded subtree exchange w.r.t. the type automaton of D.

Notice that $\text{anc-type}_N^{t_1}(v_1) = \text{anc-type}_N^{t_2}(v_2)$ 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. □

4.2. Unions of XSDs

4.2.1. Infinitely Many Maximal 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 a's and followed by a subtree t.

Take the following single-type EDTDs (which are even DTDs) with a as the root:

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

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

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

Here, $\mu(\tau_{\mathtt{a}}^i) = \mathtt{a}$ 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 $\text{Dom}(t_n)$). □

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, $\mathrm{nv}(D_2, D_1)$ contains all individual trees t for which 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 $Y \subseteq \mathrm{nv}(D_2, D_1)$, otherwise $L(D_1) \cup Y \not\subseteq L(D_1) \cup L(D_2)$. We show that, in fact, if $Y = \mathrm{nv}(D_2, D_1)$, 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 Section 4.2.2 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 closure$(t_1, t) \subseteq L(D)$, where $D = (\Sigma, \Delta, d, S_d, \mu)$ is a single-type EDTD such that $L(D) = \mathrm{closure}(L(D_1) \cup L(D_2))$. Thus from Theorem 4.2 we have that closure(t_1, t) is closed under ancestor-type-guarded subtree exchange w.r.t. D. From the construction in Theorem 3.8, 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 $\mathrm{nv}(D_2, D_1)$ if and only if for every $t_1 \in L(D_1)$ and all nodes $v \in \mathrm{Dom}(t)$, $v_1 \in \mathrm{Dom}(t_1)$, such that anc-type$^t(v) =$ anc-type$^{t_1}(v_1)$, we have that

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

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

This is one characterization of all trees t belonging to $\mathrm{nv}(D_2, D_1)$. 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:

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

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 $\mathrm{nv}(D_2, D_1)$ if and only if, for every node $v \in \mathrm{Dom}(t)$ and $\tau = \text{anc-type}^t(v)$,

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

(b') if τ is a *c*-type, then 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 $v_1 \in \mathrm{Dom}(t_1)$ such that anc-type$^{t_1}(v_1) = \tau$. If $t'_1 = \mathrm{subtree}^{t_1}(v_1) \in S_2(\tau)$, 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 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 $v_1 \in \mathrm{Dom}(t_1)$ such that anc-type$^{t_1}(v_1) = \tau$ and $t'_1 = \mathrm{subtree}^{t_1}(v_1) \in S_1(\tau) \setminus S_2(\tau)$. 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 context$^t(v) =$ 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 $L(D') = \mathrm{nv}(D_2, D_1)$. 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 subtree$^t(v) \in S_1(\tau)$ we have to check whether 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

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

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

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

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

Lemma 4.5. *Let $t \in L(D')$, $v, u \in \mathrm{Dom}(t)$ 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.

Proof. Case (a): take a tree $t_\star \in S_1(\tau_v) \setminus S_2(\tau_v)$. From the definition of $S_1(\tau_v)$ there exist $t' \in L(D_1)$ and $v' \in \mathrm{Dom}(t')$ such that $\text{anc-type}^{t'}(v') = \tau_v$ and $\text{subtree}^{t'}(v') = t_\star$. Thanks to the definition of d' w.l.o.g. we can assume that $\text{anc-str}^{t'}(v') = \text{anc-str}^t(v)$ and $\text{ch-str}^{t'}(u') = \text{ch-str}^t(u)$ where u' is the parent of u. Therefore we have that $\text{subtree}^{t'}(u') \in S_1(\tau_u) \setminus S_2(\tau_u)$.

Case (b): take a tree $t_\star \in S_1(\tau_v) \setminus S_2(\tau_v)$. There exist $t' \in L(D_1)$ and $v' \in \mathrm{Dom}(t')$ such that $\text{anc-type}^{t'}(v') = \tau_v$ and $\text{subtree}^{t'}(v') = t_\star$. Since v has a sibling, it has also a parent w. W.l.o.g. we can assume that $\text{anc-str}^{t'}(v') = \text{anc-str}^t(v)$ and $\text{ch-str}^{t'}(w') = \text{ch-str}^t(w)$, where w' is a parent of w. Let $u' = w'a$ for $a \in \Sigma$ such that $u = wa$. We have that $\text{context}^{t'}(u') \in C_1(\tau_u) \setminus C_2(\tau_u)$.

Case (c): take a context $C_\star \in C_1(\tau_v) \setminus C_2(\tau_v)$. There exist $t' \in L(D_1)$ and $v' \in \mathrm{Dom}(t')$ such that $\text{anc-type}^{t'}(v') = \tau_v$ and $\text{context}^{t'}(v') = C_\star$. W.l.o.g. we can assume that $\text{ch-str}^{t'}(v') = \text{ch-str}^t(v)$. Let $u' = v'a$ for $u = va$. We have that $\text{context}^{t'}(u') \in C_1(\tau_u) \setminus C_2(\tau_u)$. ☐

We show that any tree $t \in L(D')$ satisfies (a') and (b') and thus $L(D') \subseteq \mathrm{nv}(D_2, D_1)$. Thereto, let $t \in L(D')$, $v \in \mathrm{Dom}(t)$ 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 $u \in \mathrm{Dom}(\text{context}^t(v))$, the type $\tau_u = \text{anc-type}^t(u)$ 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, $\text{context}^t(v) \in C_1(\tau)$.

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 $L(D_1) \cup \mathrm{nv}(D_2, D_1)$. We show that such a tree cannot exist. Applying Lemma 2.15 to these four derivation trees we get that there exists another derivation tree which is one of the following:

(a) $t(t_A, t_1)$,

(b) $t(t_1, t_A)$,

(c) $t(t_A, t_2)$,

(d) $t(t_2, t_A)$,

(e) $t(t_C(t_A, t_1), t_2)$,

(f) $t(t_D(t_1, t_A), t_2)$,

(g) $t(t_2, t_E(t_1, t_A))$,

(h) both $t(t_C(t_A, t_2), t_1)$ and $t(t_C(t_A, t_2), t_B(t_2, t_1))$,

(i) both $t(t_D(t_2, t_A), t_1)$ and $t(t_D(t_2, t_A), t_B(t_2, t_1))$,

(j) both $t(t_1, t_E(t_2, t_A))$ and $t(t_B(t_1, t_2), t_E(t_2, t_A))$.

Cases (a) and (b) contradict the fact that $L(D_1)$ is closed under ancestor-guarded subtree exchange. Cases (c) and (d) contradict the definition of E. For cases (e)–(g) we have that $t_C, t_D, t_E \in L(D_1)$ which leads to contradiction with the definition of E.

Finally, for case (h) we have that $t_C \in L(D_1) \cup L(D_2)$. If $t_C \in L(D_1)$, we use the first derivation tree and obtain a contradiction as $L(D_1)$ is closed under ancestor-guarded subtree exchange. If $t_C \in L(D_2)$, we use the second derivation tree and obtain a contradiction as $L(D_2)$ is closed under ancestor-guarded subtree exchange. Cases (i) and (j) are analogous. ☐

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 $L(D_1)$ (and dually, it is uniquely defined by its intersection with $L(D_2)$).

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.*

Proof. Let $t \in \mathrm{closure}(X \cup Y_1 \cup Y_2)$. We show that $t \in X \cup \mathrm{closure}(Y_1 \cup Y_2)$. As $X \cup \mathrm{closure}(Y_1 \cup Y_2) \subseteq \mathrm{closure}(X \cup Y_1 \cup Y_2)$, it then follows that $X \cup \mathrm{closure}(Y_1 \cup Y_2) = \mathrm{closure}(X \cup Y_1 \cup Y_2)$ and thus is closed under ancestor-guarded subtree exchange. Let ϑ be a derivation tree of t with respect to $X \cup Y_1 \cup Y_2$. We transform ϑ into a derivation tree of t which either consists of one node labeled by an element of X, and hence $t \in X$, or whose leaf nodes are all labeled by elements in $Y_1 \cup Y_2$, and hence $t \in \text{closure}(Y_1 \cup Y_2)$.

Therefore, as long as ϑ is not of the above form, i.e., has at least two leaves and one of them is from X, we will perform a transformation on ϑ. If the unique leaf node v with $\text{lab}^{\vartheta}(v) \in X$ has a sibling which is also a leaf, then, as both $X \cup Y_1$ and $X \cup Y_2$ are closed under ancestor-guarded subtree exchange, their parent's label is in $X \cup Y_1 \cup Y_2$. Therefore we can remove these two nodes from ϑ and the resulting tree will still be a valid derivation tree of t. On the other hand, if the sibling of v is not a leaf, we can apply one of the transformations from Lemma 2.15 which will result either in a smaller ϑ or in a derivation tree ϑ in which v is one level deeper (i.e., one node farther from the root).

In every transformation step, ϑ either decreases in size or at least one node labeled by a tree of X is one level deeper in the tree. Hence, we can only do a finite number of such transformations. The result is a valid derivation tree of t, which either consists of one node or is a derivation tree with respect to $Y_1 \cup Y_2$, as desired. $\square$

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

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

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 the complement of $L(D)$. 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.*

Proof. Let D be the DTD over alphabet $\{\mathtt{a}\}$ consisting of the single rule $\mathtt{a} \to \mathtt{a} + \varepsilon$. Therefore, a tree is in $L(D_c)$ if and only if at least one node has at least two children.

We argue that, for each $n \geq 1$ the following single-type EDTD X_n is a maximal lower XSD-approximation for $L(D_c)$:

$$\begin{array}{lll} \tau_{\mathtt{a}}^{i} & \to & (\tau_{\mathtt{a}}^{i+1})^{+} \qquad \text{(for all } 0 \leq i < n-1) \\ \tau_{\mathtt{a}}^{n-1} & \to & \tau_{\mathtt{a}}^{n}(\tau_{\mathtt{a}}^{n})^{+} \\ \tau_{\mathtt{a}}^{n} & \to & (\tau_{\mathtt{a}}^{n+1})^{*} \\ \tau_{\mathtt{a}}^{n+1} & \to & (\tau_{\mathtt{a}}^{n+1})^{*} \end{array}$$

Here $\mu(\tau_{\mathtt{a}}^{i}) = \mathtt{a}$ for each $i \in \{0, \ldots, n+1\}$. These languages are pairwise different, since a tree $t_n = \mathtt{a}^n(\mathtt{a}, \mathtt{a})$ is in $L(X_m)$ if and only if $n = m$.

Let t be an arbitrary tree from $L(D_c) \setminus L(X_n)$. We prove that $\text{closure}(L(X_n) \cup \{t\}) \not\subseteq L(D_c)$. First we consider the case when there is a leaf in the first n levels of tree t, i.e. there is a leaf $v \in \text{Dom}(t)$ with $\text{anc-str}^t(v) = \mathtt{a}^m$ and $1 \leq m \leq n$. Then we can show that $\text{closure}(t, t_n)$ contains a tree $\mathtt{a}^m$ from $L(D)$ (just apply ancestor-guarded subtree exchange on node v in $\text{Dom}(t)$ and on node 1^{m-1} in $\text{Dom}(t_n)$).

Another possibility is that in t every leaf v is deeper than in the first n levels. Then the tree must violate the conditions of X_n in some node on n-th level, i.e., there is a node v with $\text{anc-str}^t(v) = \mathtt{a}^n$ and v has exactly one child. Then again $\text{closure}(t, t_n)$ contains a tree $t' = \mathtt{a}^n(\text{subtree}^t(v1))$ (ancestor-guarded subtree exchange on node v in $\text{Dom}(t)$ and node 1^{n-1} in $\text{Dom}(t_n)$). Finally $\text{closure}(t', t_n)$ contains a tree $\mathtt{a}^{n+1}$ (ancestor-guarded subtree exchange on nodes 1^n in $\text{Dom}(t')$ and $\text{Dom}(t_n)$). $\square$

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 $\mathrm{subtree}^t(v) = a(t_1, \ldots, t_n)$, we denote by $\mathrm{subforest}^t(v)$ 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[4] (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 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 $\mathrm{closure}(t_1, t_2) \subseteq X_1 \subseteq X_{\mathcal{C}}$.

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

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

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 $L(D') \subseteq T$.

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

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

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

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

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

Let T be a regular tree language and N be an NFA. The *type-closure* of T with respect to 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

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

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 approximation if and only if

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

Our plan is to construct a *tree automaton*[5] 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* $\{t \in L(D) - L(S) \mid \textit{type-closure}^{N_k}(\{t\} \cup L(S)) \subseteq L(D)\}$ *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.*

The rest of the chapter is devoted to the proof of the Lemma 4.13. First we need to introduce some more definitions and lemmas.

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

Definition 4.15. A *non-deterministic tree automaton (NTA)* is a tuple $B = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is a function $\delta : Q \times \Sigma \to 2^{Q^*}$ such that $\delta(q, a)$ is a regular string language over Q for every $a \in \Sigma$ and $q \in Q$.

A *run* of B on a tree t is a labeling $\lambda : \mathrm{Dom}(t) \to Q$ such that, for every $v \in \mathrm{Dom}(t)$ with n children, $\lambda(v1) \cdots \lambda(vn) \in \delta(\lambda(v), \mathrm{lab}^t(v))$. Note that when v has no children, then the criterion reduces to $\varepsilon \in \delta(\lambda(v), \mathrm{lab}^t(v))$. A run is *accepting* iff the root is labeled with an accepting state, that is, $\lambda(\varepsilon) \in F$. A tree is accepted if there is an accepting run. It is well-known that tree automata are expressively equivalent to EDTDs and that there are quadratic time translations between tree automata and equivalent EDTDs.

The following Lemma is for *binary* trees. We say that a tree is binary if each node has either zero or two children. An EDTD $(\Sigma, \Delta, d, S_d, \mu)$ is *(blockwise) bottom-up deterministic* if, for every pair of rules $d(\tau_1^a) = L_1$ and $d(\tau_2^a) = L_2$ with $\mu(\tau_1^a) = \mu(\tau_2^a) = a$ we have that $L_1 \cap L_2 = \emptyset$. It is well-known that EDTDs can be determinized using a subset construction, similar to tree automata.

Lemma 4.16. *Let N be a state-labeled DFA and D a bottom-up deterministic EDTD for binary trees. There exists a (non-deterministic) tree automaton for $\{t \in L(D) \mid \text{type-closure}^N(\{t\}) \subseteq L(D)\}$ of size exponential in $|D| + |N|$.*

Proof. Let $D = (\Sigma, \Delta, d, S_d, \mu)$. For a type $\tau \in \Delta$, we denote by D^τ the bottom-up deterministic EDTD $(\Sigma, \Delta, d, \{\tau\}, \mu)$, i.e., the EDTD D with start symbol τ. W.l.o.g., we assume that D is complete, i.e., for each tree t', there is a type τ in Δ such that $t' \in L(D^\tau)$. Since D is bottom-up deterministic, this τ is unique for each t'. For a tree t', we denote by $D(t')$ the unique type $\tau \in \Delta$ such that $t' \in D^\tau$. We extend this notation to sets, i.e., for $T \subseteq \mathcal{T}_\Sigma$, $D(T)$ is the set $\{\tau \mid \exists t' \in T \text{ such that } t' \in D^\tau\}$.

We also assume that N is complete, i.e., that N has a run on each Σ-string. For a state q of N, we also denote by N^q the DFA obtained from N by making q its initial state. Let $N = (Q_N, \Sigma, \delta_N, I_N)$. In this proof, we denote by subtree(u) the parent of node u. If u is the root, then anc-str^t(subtree(u)) is the empty string.

We construct a bottom-up non-deterministic tree automaton A_{tc} for

$$\{t \in L(D) \mid \text{type-closure}^N(\{t\}) \subseteq L(D)\}.$$

The intuition is that the state set of A_{tc} includes the subsets $\{\tau_1, \ldots, \tau_n\}$ of Δ. When reading a tree t and A_{tc} visits a node $u \in \mathrm{Dom}(t)$ in such a state $\{\tau_1, \ldots, \tau_n\}$, we have that

$$\begin{aligned}&\forall \tau \in \Delta,\ \tau \in \{\tau_1, \ldots, \tau_n\} \text{ iff}\\ &\qquad \exists t' \in \text{type-closure}^{N^q}(\{\text{subtree}^t(u)\}) \text{ where } q = N(\text{anc-str}^t(\text{subtree}(u))) \text{ and } \tau = D(t'). \quad (2)\end{aligned}$$

Thus, if we visit the root of t in a state $\{\tau_1, \ldots, \tau_n\}$ in which each $\tau_i \in S_d$, then type-closure$^N(\{t\}) \subseteq L(D)$.

Formally, let $A_{\mathrm{tc}} = (Q_A, \Sigma, \delta_A, F_A)$ be a binary tree automaton. Hence, the transition rules of A_{tc} are either leaf transitions of the form $a \to R$ for $a \in \Sigma$ and $R \in Q_A$, or internal transitions of the form $a(R_1, R_2) \to R$ for $a \in \Sigma$ and $R_1, R_2, R \in Q_A$. In a state $R \in Q_A$, we want to store at least the sets mentioned above, to which we refer from now on as Types(R):

(R1): Types$(R) = \{\tau_1, \ldots, \tau_n\} \subseteq \Delta$, where Types$(R)$ satisfies equation (2) above.

If we can correctly make bottom-up transitions using these sets, we are done. For the leaf transitions, this would be easy: we just have $a \to \{D(a)\}$ as closing a single-node tree under subtree exchange does not add any trees. Now, consider the internal transitions. Here, assume that we have a transition with the left-hand side $a(R_1, R_2)$. We need to compute Types(R) such that $a(R_1, R_2) \to R$ is a valid transition, so we want to maintain the invariant of equation (2). In this respect, consider a node u in the input tree t and let subtree$^t(u) = a(t_1, t_2)$. Recall that, in t, the two children of u are $u1$ and $u2$. In the remainder of the proof, we want to exhibit a fixpoint computation that computers Types(R) correctly, i.e., such that equation (2) holds.

In order to compute Types(R) correctly for the transition $a(R_1, R_2) \to R$, the sets Types(R_1) and Types(R_2) do not provide enough information. We discuss next what information needs to be stored in R_1 and R_2 and explain later in the proof why the information is needed and how it can be computed. We will need to store four further kinds of sets, which are listed under (R2)–(R5) below. Two of these can already be defined directly. For a node v in a tree t', we call the state $N(\text{anc-str}^{t'}(v))$ the *ancestor-type* of v in t' and we denote it by $\text{anc-type}^{t'}(v)$. We also store the following sets in a state R:

(R2): A state $\text{anc-type}(R) \in Q_N$ for the ancestor-type of the current node u in t.

(R3): A set of pairs $\text{subtrees}(R) \subseteq Q_N \times \Delta$ which contains a pair (q, τ) if and only if there is a descendant uv of u such that $q = \text{anc-type}^t(\text{subtree}(uv))$ and $\tau \in D(\text{type-closure}^{N^q}(\text{subtree}^t(uv)))$.

For the other sets, we first need more formal background. Recall that a context is a tree with a hole marker in one leaf. A *fork* is a binary tree with three nodes, in which the two leaf nodes have hole markers. For example, $a((b, \bullet), (c, \bullet))$ is a fork.

To a context C we associate a function $f_C : \Delta \to \Delta$ which behaves as follows. For each type τ_1, $f_C(\tau_1) := D(C[t_1])$, where t_1 is a tree with $D(t_1) = \tau_1$. Similarly, we associate a function $f_F : \Delta \times \Delta \to \Delta$ to each fork $F = a((b, \bullet), (c, \bullet))$. For each pair of types (τ_1, τ_2), $f_F(\tau_1, \tau_2) = \tau$ if there are two trees t_1 and t_2 with $D(t_1) = \tau_1$ and $D(t_2) = \tau_2$, $\text{lab}^{t_1}(\varepsilon) = b$ and $\text{lab}^{t_2}(\varepsilon) = c$ and $\tau = D(a(t_1, t_2))$. Since D is bottom-up deterministic, these functions are well-defined.

Each pair of nodes u, uv of t induces a context C^u_{uv} which is rooted at u and has the hole marker at uv. Formally, if $\text{lab}^t(uv) = a$, then C^u_{uv} is the subtree of $t[uv \leftarrow (\bullet, a)]$ rooted at u. Similarly, the *fork induced by a (non-leaf) node $u \in t$* is $a((b, \bullet), (c, \bullet))$, where $a = \text{lab}^t(u)$, $b = \text{lab}^t(u1)$, and $c = \text{lab}^t(u2)$. In the following, we refer to $\text{subtree}^t(u)$ also as *the subtree of t induced by u*. Let F be a fork induced by node u with label a. A 6-tuple $(q_1, q_2, q_3, \tau_1, \tau_2, \tau_3)$ is induced by F when $\text{anc-type}^t(u) = q_1$, $\text{anc-type}^t(u1) = q_2$, $\text{anc-type}^t(u2) = q_3$, and there is a pair of trees t_1, t_2 with $D(t_1) = \tau_2$ and $D(t_2) = \tau_3$ such that $D(a(t_1, t_2)) = \tau_1$. We can now state the remaining information that needs to be stored in a state R of A_{tc}:

(R4): A set $\text{contexts}(R) \subseteq Q^2 \times \Delta^2$ which contains the quadruple $(q_1, q_2, \tau_1, \tau_2)$ if and only if there is a tree t' in $\text{type-closure}^{N^q}(\text{subtree}^t(u))$, where the state $q = N(\text{anc-str}^t(\text{subtree}(u)))$ and there are nodes v, vw in t' such that $N^q(\text{anc-str}^{t'}(\text{subtree}(v))) = q_1$, $N^q(\text{anc-str}^{t'}(\text{subtree}(vw))) = q_2$, and $f_{C^v_{vw}}(\tau_2) = \tau_1$, where $f_{C^v_{vw}}$ is the function associated to the context C^v_{vw}.

(R5): The set $\text{forks}(R) \subseteq Q^3 \times \Delta^3$ of all tuples induced by forks induced by nodes in $\text{subtree}^t(u)$.

In the next part of the proof, we explain why these components are needed and how they can be computed for the construction of the tree automaton. The overall approach is the following. If we want to correctly compute R for a transition $a(R_1, R_2) \to R$, we perform a fixpoint computation according to the rules we exhibit below. This fixpoint computation uses the information we store in R_1 and R_2 and iteratively adds sets to Types(R). Finally, we must also show how all the auxiliary information (anc-type(R), subtrees(R), contexts(R), and forks(R)) can be computed.

Before we explain the fixpoint computation for Types(R) we argue why anc-type(R) can already be assumed to be known for each state R. It is easy to construct a tree automaton that reads a tree in a top-down manner and that assigns, in an accepting run, the state $\text{anc-type}^t(u)$ to each node u. (This tree automaton just simulates N on each path.) As a bottom-up automaton, this is a non-deterministic automaton which is linearly large in N. In the current construction, we can simulate this automaton in parallel to the rest of the automaton.

We perform the fixpoint computation for Types(R) according to several rules. Rule 1 simply propagates information from Types(R_1) and Types(R_2) to Types(R):

Rule 1: If $\tau_1 \in \text{Types}(R_1)$, $\tau_2 \in \text{Types}(R_2)$, then $\tau_{prop} := \{\tau \mid \tau_1\tau_2 \in d(\tau)\}$ is in Types(R).

With Rule 1 we can compute all $\tau_{prop} = D(t') \in \text{Types}(R)$ such that

$$t' \in \left\{a(t_1, t_2) \mid \forall i \in \{1, 2\}, t_i \in \text{type-closure}^{N_{q_i}}(\{\text{subtree}^t(ui)\})\right\}, \quad \text{where } q_i = \text{anc-type}^t(\text{subtree}(ui)).$$

A second step would be to include all sets of types in $\text{Types}(R)$ that are reachable by performing subtree exchange between u and descendants uv of u in t with the same ancestor-type. In other words, we want to add all $\tau_{\text{sa}} = D(t')$ to $\text{Types}(R)$ such that

$$t' \in \{\text{type-closure}^{N_{q_i}}(\text{subtree}^t(uv)) \mid v \neq \varepsilon, q_i = \text{anc-type}^t(\text{subtree}(uv)) \text{ and } \text{anc-type}^t(uv) = \text{anc-type}^t(u)\}.$$

In order to compute τ_{sa} we need to store ancestor-types in states as well. Therefore, a state R in Q_A also needs to contain $\text{anc-type}(R)$, i.e., (R2), and $\text{subtrees}(R)$, i.e., (R3). Given these two sets, we can compute τ_{sa} with the following rules:

Rule 2a: If there exists an $i \in \{1, 2\}$ with $\text{anc-type}(R) = \text{anc-type}(R_i)$ then $\text{Types}(R_i) \subseteq \text{Types}(R)$.

Rule 2b: If there exists an $i \in \{1, 2\}$ with $(\text{anc-type}(R), \tau_{bu}) \in \text{subtrees}(R_i)$, then $\tau_{bu} \in \text{Types}(R)$.

As argued above, we can already assume $\text{anc-type}(R)$ to be known, so Rules 2a and 2b can indeed be used to compute all τ_{sa} correctly.

Finally, we want to compute all $\Delta_{\text{full}} = D(t')$ in $\text{Types}(R)$ such that

$$t' \in \text{type-closure}^{N^q}(\{\text{subtree}^t(u)\}), \quad \text{where } q = \text{anc-type}^t(\text{subtree}(u)). \tag{3}$$

Here, subtrees from t' with the same ancestor-type can be arbitrarily exchanged. In order to keep track of the possible effects of subtree exchange on the reachable types in D, we need to remember the effects of contexts in t' on the reachable types in D. Therefore, we need to store $\text{contexts}(R)$, i.e., (R4).

However, it is not clear that $\text{contexts}(R)$ is enough. If t' is an arbitrary tree obeying equation (3) then this tree is a "patchwork" of parts from $\text{subtree}^t(u)$. These parts are, in general, subtrees, contexts, and *generalized contexts*. A *generalized context* is a tree C which can have arbitrarily many leaves labeled with $\Sigma \times \{\bullet\}$-symbols. Context application for generalized contexts is defined analogously as for contexts. Notice that, if C is a generalized context, then $C[t]$ is possibly again a (generalized) context. The following lemma states that each tree t' can be seen as a patchwork mentioned above:

Lemma 4.17. *Let t be an arbitrary tree. Each tree in type-closure$^N(\{t\})$ can be partitioned into (generalized) contexts and subtrees induced by nodes in t.*

This lemma is illustrated in Figure 2(a). The lemma claims that each tree in the closure can be split up into parts, similar to the tree in Figure 2(a). It can easily be proved by induction on the number of subtree exchanges performed.

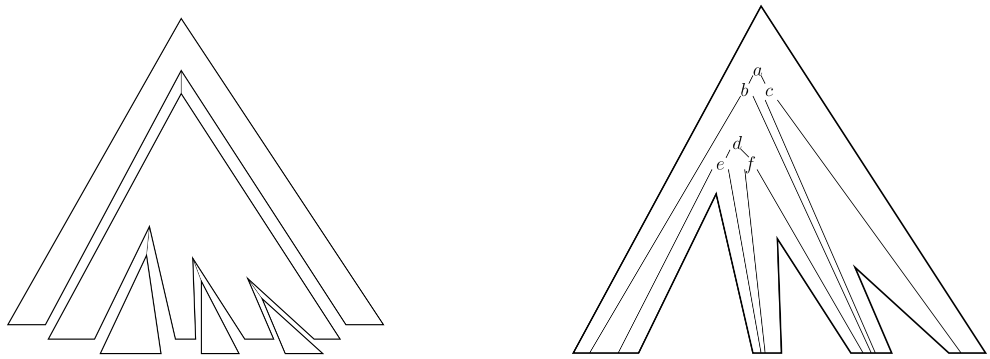
However, remembering all possible effects of generalized contexts on types is problematic for the tree automaton construction, because a generalized context induces a function from $\Delta^k \to \Delta$ for an arbitrary (non-fixed) k. If k is not fixed, the number of such functions is arbitrarily large and therefore we cannot store them in a finite state of the automaton. However, the next lemma states that remembering the effects of (1) contexts and (2) forks is sufficient to also be able to compute the effects of all generalized contexts. The next lemma follows from the definitions:

Lemma 4.18. *Each generalized context can be partitioned into contexts and forks.*

This lemma is illustrated in Figure 2(b), which contains a partitioning of a generalized context into contexts and forks.

By combining Lemmas 4.17 and 4.18, we obtain that remembering contexts and forks is sufficient:

Lemma 4.19. *Let t be an arbitrary tree. Each tree in type-closure$^N(\{t\})$ can be partitioned into contexts, forks, and subtrees induced by nodes in t.*



(a) Illustration of a possible partitioning of a tree into subtrees, contexts, and generalized contexts.

(b) Illustration of a possible partitioning of a generalized context into contexts, forks, and subtrees.

Figure 2: Illustrations for Lemmas 4.17 and 4.18.

This is the reason why we store forks(R), i.e., (R5). Notice that forks(R) can already be computed from $\text{lab}^t(u)$, $\text{lab}^t(u1)$, $\text{lab}^t(u2)$, forks(R_1), and forks(R_2). We can therefore assume forks(R) to be already known.

In the following rules, 3a simply propagates contexts, forks, and subtrees from R_1 and R_2 to R. Rule 3b adds the new fork defined by node u to R. Rules 3c–3e add the new contexts an subtrees obtained by taking contexts and subtrees from R_1 and R_2 and adding the new root u.

Rule 3a: For all $i \in \{1, 2\}$, $\text{contexts}(R_i) \subseteq \text{contexts}(R)$, $\text{forks}(R_i) \subseteq \text{forks}(R)$, and $\text{subtrees}(R_i) \subseteq \text{subtrees}(R)$.

Rule 3b: For all $\tau_1 \in \text{Types}(R_1)$, $\tau_2 \in \text{Types}(R_2)$, $\tau_3 \in \{\tau \mid \tau_1\tau_2 \in d(\tau)\}$, $(\text{anc-str}(R), \text{anc-str}(R_1), \text{anc-str}(R_2), \tau_3, \tau_1, \tau_2) \in \text{forks}(R)$.

Rule 3c: For all $(\text{anc-type}(R_1), \tau_1) \in \text{subtrees}(R_1)$, $(\text{anc-type}(R_2), \tau_2) \in \text{subtrees}(R_2)$, and $\tau_3 \in \{\tau \mid \tau_1\tau_2 \in d(\tau)\}$, we have $(\text{anc-type}(R), \tau_3) \in \text{subtrees}(R)$.

Rule 3d: For all $(\text{anc-type}(R_1), q_2, \tau_1, \tau_2) \in \text{contexts}(R_1)$, $(\text{anc-type}(R_2), \tau_3) \in \text{subtrees}(R_2)$, $\tau_4 \in \{\tau \mid \tau_1\tau_3 \in d(\tau)\}$, $(\text{anc-type}(R), q_2, \tau_4, \tau_2) \in \text{contexts}(R)$.

Rule 3e: For all $(\text{anc-type}(R_1), \tau_1) \in \text{subtrees}(R_1)$, $(\text{anc-type}(R_2), q_3, \tau_2, \tau_3) \in \text{contexts}(R_2)$, $\tau_4 \in \{\tau \mid \tau_1\tau_2 \in d(\tau)\}$, $(\text{anc-type}(R), q_3, \tau_4, \tau_3) \in \text{contexts}(R)$.

The following rules now close the already obtained information under ancestor-type-guarded subtree exchange.

Rule 4a: If $(q_1, q_2, \tau_1, \tau_2) \in \text{contexts}(R)$ and $(q_2, q_3, \tau_3, \tau_1) \in \text{contexts}(R)$ then $(q_1, q_3, \tau_3, \tau_2) \in \text{contexts}(R)$.

Rule 4b: If $(q_1, q_2, q_3, \tau_1, \tau_2, \tau_3) \in \text{forks}(R)$, $(q_4, \tau_4) \in \text{subtrees}(R)$, and $(q_2, q_4, \tau_2, \tau_4)$ and $(q_3, q_5, \tau_3, \tau_5) \in \text{contexts}(R)$, then $(q_1, q_5, \tau_1, \tau_5) \in \text{contexts}(R)$.

Rule 4c: If $(q_1, q_2, q_3, \tau_1, \tau_2, \tau_3) \in \text{forks}(R)$, $(q_5, \tau_5) \in \text{subtrees}(R)$, and $(q_2, q_4, \tau_2, \tau_4)$ and $(q_3, q_5, \tau_3, \tau_5) \in \text{contexts}(R)$, then $(q_1, q_4, \tau_1, \tau_4) \in \text{contexts}(R)$.

Rule 4d: If $(q_1, q_2, \tau_1, \tau_2) \in \text{contexts}(R)$ and $(q_2, \tau_2) \in \text{subtrees}(R)$, $(q_1, \tau_1) \in \text{subtrees}(R)$.

Rule 4e: If there are (q_1, τ_1) and $(q_2, \tau_2) \in \text{subtrees}(R)$ such that $q_1 = \text{anc-type}(R_1)$ and $q_2 = \text{anc-type}(R_2)$, then $\tau_{\text{full}} = \{\tau \mid \tau_1\tau_2 \in d(\tau)\}$ is in $\text{Types}(R)$.

Rule 4f: If $(\text{anc-type}(R), \tau_{\text{full}}) \in \text{subtrees}(R)$, then $\tau_{\text{full}} \in \text{Types}(R)$.

This concludes the description of the transition rules. The actual transition function is computed by applying a fixed point computation using these rules. □

Notice that we can adapt the construction in Lemma 4.16 easily to only allow subtree exchange for a subset of the alphabet. For a set of alphabet symbols $\Sigma' \subseteq \Sigma$, the set $\text{type-closure}^{N,\Sigma'}(T)$ is the smallest set such that

- $T \subseteq \text{type-closure}^{N,\Sigma'}(T)$, and
- if $t_1, t_2 \in \text{type-closure}^{N,\Sigma'}(T)$, then $t := t_1[u \leftarrow \text{subtree}^{t_2}(v)]$ is also in $\text{type-closure}^{N,\Sigma'}(T)$, where $N(\text{anc-str}^{t_1}(u)) = N(\text{anc-str}^{t_2}(v))$ and $\text{lab}^{t_1}(u) = \text{lab}^{t_2}(v) \in \Sigma'$.

In other words, $\text{type-closure}^{N,\Sigma'}(T)$ is obtained from T by performing ancestor-type guarded subtree exchange, *only on nodes labeled with Σ'-labels.* This gives the following corollary:

Corollary 4.20. *Let N be a state-labeled DFA and D be a bottom-up deterministic EDTD for binary trees. Let $\Sigma' \subseteq \Sigma$. There exists a tree automaton for $\{t \in L(D) \mid \text{type-closure}^{N,\Sigma'}(\{t\}) \subseteq L(D)\}$ of size double exponential in $|D| + |N|$.*

Lemma 4.21. *Let N be a state-labeled DFA and D and S be bottom-up deterministic EDTDs for binary trees such that $L(S) \subseteq L(D)$. Let $\Sigma' \subseteq \Sigma$. There exists a tree automaton for $\{t \in L(D) \mid \text{type-closure}^{N,\Sigma'}(\{t\} \cup L(S)) \subseteq L(D)\}$ of size exponential in $|D| + |S| + |N|$.*

Proof. If we want to compute whether

$$\text{type-closure}^{N,\Sigma'}(\{t\} \cup L(S)) \subseteq L(D)$$

instead of

$$\text{type-closure}^{N,\Sigma'}(\{t\}) \subseteq L(D),$$

we need to include the trees of $L(S)$ into the fixpoint computation in the proof of Corollary 4.20.

Intuitively, we want to precompute sets $\text{subtrees}(N, S, D)$, $\text{contexts}(N, S, D)$, and $\text{forks}(N, S, D)$ that we can add to each state R in the automaton of the proof of Lemma 4.16. So, the extra sets $\text{subtrees}(N, S, D)$, $\text{contexts}(N, S, D)$, and $\text{forks}(N, S, D)$ are the same for every state R in Lemma 4.16.

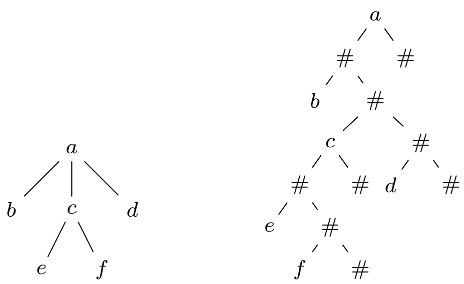
Formally, let $D = (\Sigma, \Delta_D, d_D, S_D, \mu_D)$ and $S = (\Sigma, \Delta_S, d_S, S_S, \mu_S)$. For every $t \in L(S)$ we denote by t_S (resp., t_D) the unique tree in d_S (resp., d_D) with $\mu_S(t_S) = t$ (resp., $\mu_D(t_D) = t$). We first define the sets $\text{subtrees}'(N, S, D)$, $\text{contexts}'(N, S, D)$, and $\text{forks}'(N, S, D)$ which also include types from S that will be projected out later. These sets are formally defined as follows:

$\text{subtrees}'(N, S, D)$**:** $\{(\tau_N, \tau_S, \tau_D) \mid \exists t \in L(S), u \in \text{Dom}(t)$ such that $N(\text{anc-str}^t(u)) = \{\tau_N\}, \text{lab}^{t_S}(u) = \tau_S,$ and $\text{lab}^{t_D}(u) = \tau_D\}$

$\text{contexts}'(N, S, D)$**:** $\{(\tau_N^1, \tau_N^2, \tau_S^1, \tau_S^2, \tau_D^1, \tau_D^2) \mid \exists t \in L(S),\ u,\ uv \in \text{Dom}(t)$ such that $N(\text{anc-str}^t(u)) = \{\tau_N^1\}$, $N(\text{anc-str}^t(uv)) = \{\tau_N^2\}$, $\text{lab}^{t_S}(u) = \tau_S^1$, $\text{lab}^{t_S}(uv) = \tau_S^2$ and, for the induced context C by u and uv we have that $f_C(\tau_D^1) = \tau_D^2\}$ (Induced contexts and f_C are defined in the proof of Lemma 4.16.)

$\text{forks}'(N, S, D)$**:** $\{(\tau_N^1, \tau_N^2, \tau_N^3, \tau_S^1, \tau_S^2, \tau_S^3, \tau_D^1, \tau_D^2, \tau_D^2) \mid \exists t \in L(S), u, u1, u2 \in \text{Dom}(t)$, such that $N(\text{anc-str}^t(u)) = \{\tau_N^1\}$, $N(\text{anc-str}^t(u1)) = \{\tau_N^2\}$, $N(\text{anc-str}^t(u2)) = \{\tau_N^3\}$, $\text{lab}^{t_S}(u) = \tau_S^1$, $\text{lab}^{t_S}(u1) = \tau_S^2$, $\text{lab}^{t_S}(u2) = \tau_S^2$ and, for the induced fork F by u we have that $f_F(\tau_D^2, \tau_D^3) = \tau_D^1\}$ (Induced forks and f_F are defined in the proof of Lemma 4.16.)

These sets can all be computed from N, S, and D in a similar fashion. We illustrate the computation for the sets $\text{subtrees}'(N, S, D)$ and $\text{contexts}'(N, S, D)$. First we construct the product $S \times D$ of S and D and we reduce it. (One can immediately construct a reduced product by a bottom-up construction similar to the standard tree automaton emptiness test, see, e.g., [10].) The type set of this product EDTD is precisely the subset of $\Delta_S \times \Delta_D$ with pairs (τ_S, τ_D) such that there's a tree t and a node $u \in \text{Dom}(t)$ for which



(a) (b)

Figure 3: A tree and its binary encoding.

$\mathrm{lab}^{t_S}(u) = \tau_S$ and $\mathrm{lab}^{t_D}(u) = \tau_D$. Then, we construct the reduced product of $S \times D$ with the EDTD that simulates N on each path, i.e., the EDTD E_N with DTD d_{E_N} defined as $d_{E_N}(\tau) = \cup_{a \in \Sigma} \tau_a$, where τ_a is the unique state of N such that $\delta(\tau, a) = \tau_a$. The set of triples of types in $E_N \times S \times D$ is precisely $\mathrm{subtrees}'(N, S, D)$.

The set $\mathrm{contexts}'(N, S, D)$ can now be computed as follows. First, we compute the reduced product $E_N \times S$. For each $\tau \in \Delta_D$, we consider all pairs (τ_N, τ_S) of types in $E_N \times S$. Then, for each such triple (τ_N, τ_S, τ), we perform a construction very similar to the standard tree automaton emptiness test, i.e., we add in a fixpoint computation, all tuples $(\tau_N^1, \tau_N^2, \tau_S^1, \tau_S^2, \tau_D^1, \tau)$ to $\mathrm{contexts}'(N, S, D)$ for which either

(a) $\tau_N^2 = \tau_N$, $\tau_S^2 = \tau_S$, there is an $a \in \Sigma$ such that $\mu_{E_N}(\tau_N^1) = \mu_S(\tau_S^1) = \mu_D(\tau_D^1) = a$, and $\tau_N^3 \tau_N^2 \in d_{E_N}(\tau_N^1)$, $\tau_S^3 \tau_S^2 \in d_S(\tau_S^1)$ and $\tau_D^3 \tau \in d_D(\tau_D^1)$ for some $(\tau_N^3, \tau_S^3, \tau_D^3) \in \mathrm{subtrees}'(N, S, D)$;

(b) $\tau_N^2 = \tau_N$, $\tau_S^2 = \tau_S$, there is an $a \in \Sigma$ such that $\mu_{E_N}(\tau_N^1) = \mu_S(\tau_S^1) = \mu_D(\tau_D^1) = a$, and $\tau_N^2 \tau_N^3 \in d_{E_N}(\tau_N^1)$, $\tau_S^2 \tau_S^3 \in d_S(\tau_S^1)$ and $\tau \tau_D^3 \in d_D(\tau_D^1)$ for some $(\tau_N^3, \tau_S^3, \tau_D^3) \in \mathrm{subtrees}'(N, S, D)$;

(c) there is a $(\tau_N^3, \tau_N^2, \tau_S^3, \tau_S^2, \tau_D^3, \tau)$ in $\mathrm{contexts}'(N, S, D)$, a $(\tau_N^4, \tau_S^4, \tau_D^4) \in \mathrm{subtrees}'(N, S, D)$ and an $a \in \Sigma$ such that $\mu_{E_N}(\tau_N^1) = \mu_S(\tau_S^1) = \mu_D(\tau_D^1) = a$ and $\tau_N^4 \tau_N^3 \in d_{E_N}(\tau_N^1)$, $\tau_S^4 \tau_S^3 \in d_S(\tau_S^1)$ and $\tau_D^4 \tau_D^3 \in d_D(\tau_D^1)$;

(d) there is a $(\tau_N^3, \tau_N^2, \tau_S^3, \tau_S^2, \tau_D^3, \tau)$ in $\mathrm{contexts}'(N, S, D)$, a $(\tau_N^4, \tau_S^4, \tau_D^4) \in \mathrm{subtrees}'(N, S, D)$ and an $a \in \Sigma$ such that $\mu_{E_N}(\tau_N^1) = \mu_S(\tau_S^1) = \mu_D(\tau_D^1) = a$ and $\tau_N^3 \tau_N^4 \in d_{E_N}(\tau_N^1)$, $\tau_S^3 \tau_S^4 \in d_S(\tau_S^1)$ and $\tau_D^3 \tau_D^4 \in d_D(\tau_D^1)$.

The set $\mathrm{forks}'(N, S, D)$ can be computed in a similar fashion.

Finally, the relations $\mathrm{subtrees}(N, S, D)$, $\mathrm{contexts}(N, S, D)$, and $\mathrm{forks}(N, S, D)$ are obtained from their primed variants by projecting out all types of $\tau_S, \tau_S^1, \tau_S^2$, and τ_S^3. ☐

Lemma 4.22. *Let N be a state labeled DFA and D an EDTD. There exists a tree automaton for $\{t \in L(D) \mid \textit{type-closure}^N(\{t\}) \subseteq L(D)\}$ of size double exponential in $|D| + |N|$.*

Proof. We obtain this lemma by calling Corollary 4.20 after going through a binary encoding for unranked trees. The concrete encoding is illustrated in Figure 3. This encoding is very similar to the well-known first-child next-sibling encoding, with the difference that a subtree in the current encoding precisely corresponds to a subtree of the original unranked tree. We need this in order to be able to translate the type closure of an unranked language to a type closure of the encoded binary language.

Similar to the first-child next-sibling encoding, we can translate non-deterministic tree automata (and EDTDs) for unranked regular tree languages to non-deterministic tree automata accepting the corresponding encoded regular tree language and vice versa.

Notice that, when Σ' is the alphabet for D, the alphabet for the corresponding EDTD D_{bin} for the encoded trees is $\Sigma' \uplus \{\#\}$. The lemma now follows by calling Corollary 4.20 with the following ingredients:

- The state-labeled DFA N in Corollary 4.20 is obtained from the current N by adding self-loops labeled $\#$ to every state and translating the resulting automaton in a state-labeled automaton.
- The alphabet Σ' in Corollary 4.20 is the same as Σ' here.
- The EDTD D in Corollary 4.20 is the determinized D_{bin}.

Notice that Corollary 4.20 holds for trees of arbitrary depth.

We obtain the tree automaton for our present lemma by transforming the binary tree automaton that results from Corollary 4.20 back to an unranked tree automaton. □

Proof of Lemma 4.13. Analogous to the proof of Lemma 4.22, but now we call Lemma 4.21 after going through the binary encoding for unranked trees. Notice that now, we need to encode and determinize S as well.

We need to intersect the resulting automaton with the complement of S in order to obtain the trees $t \in L(D) - L(S)$ instead of just $t \in L(D)$. Since the automaton resulting from Lemma 4.21, is doubly exponential large, the resulting automaton here is also of double exponential size. □

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.

Lemma 5.1. *Let D_1 be an EDTD(NFA) and let D_2 be a single-type EDTD(NFA). Testing whether $L(D_1) \subseteq L(D_2)$ is in* PSPACE.

Proof. We provide a PSPACE algorithm for the complement of the problem. Since PSPACE is closed under complement, this proves the lemma.

Let $D_2 = (\Sigma, \Delta_2, d_2, S_{d_2}, \mu_2)$ and A_2 be the (deterministic) type automaton of D_2. A tree t is *not* in the language defined by the single-type EDTD D_2 if and only if there exists a node $u \in \text{Dom}(t)$ such that $\text{ch-str}^t(u) \notin L(d_2(\tau))$, where $A_2(\text{anc-str}^t(u)) = \{\tau\}$. The intuition of our PSPACE procedure is to guess a path up to such a node u, such that this path can occur in a tree in $L(D_1)$.

Since D_1 is reduced (Proviso 2.3), every string that can be handled by the type automaton A_1 of D_1 can occur as an ancestor-path of a tree in $L(D_1)$. More formally, for a string w, there exists a tree $t \in L(D_1)$ and a node u in t with $\text{anc-str}^t(u) = w$ if and only if $A_1(w) \neq \emptyset$.

Our PSPACE algorithm consists of the following steps:

(1) Guess w one symbol at a time while maintaining $(A_1(w), A_2(w))$.

(2) Test whether there exists a $\tau_1 \in A_1(w)$ for which $\mu_1(d_1(\tau_1)) \not\subseteq \mu_2(d_2(\tau_2))$ for the unique $\tau_2 \in A_2(w)$.

Step (1) only requires polynomial space because we only need to remember the last symbol of w, the set $A_1(w)$ and the singleton $A_2(w)$. Step (2) is in PSPACE since inclusion testing between NFAs and regular expressions is in PSPACE (Theorem 3.6). □

The size of the optimal upper approximation of the complement of an XSD can become exponentially large (Theorem 3.11), 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 [12], we have an (optimal) exponential upper bound for Theorem 3.8 and an optimal double exponential upper bound for Theorem 3.11.

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.

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] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] L. Stockmeyer and A. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9. ACM, 1973.