Which XML Schemas Admit 1-Pass Preorder Typing?

Wim Martens[1], Frank Neven[1], and Thomas Schwentick[2]

[1] Limburgs Universitair Centrum
Universitaire Campus
B-3590 Diepenbeek, Belgium
{wim.martens,frank.neven}@luc.ac.be
[2] Philipps Universität Marburg
Fachbereich 12, Mathematik und Informatik
tick@informatik.uni-marburg.de

Abstract. It is shown that the class of regular tree languages admitting one-pass preorder typing is exactly the class defined by restrained competition tree grammars introduced by Murata et al. [14]. In a streaming context, the former is the largest class of XSDs where every element in a document can be typed when its opening tag is met. The main technical machinery consists of semantical characterizations of restrained competition grammars and their subclasses. In particular, they can be characterized in terms of the context of nodes, closure properties, allowed patterns and guarded DTDs. It is further shown that deciding whether a schema is restrained competition is tractable. Deciding whether a schema is equivalent to a restrained competition tree grammar, or one of its subclasses, is much more difficult: it is complete for EXPTIME. We show that our semantical characterizations allow for easy optimization and minimization algorithms. Finally, we relate the notion of one-pass preorder typing to the existing XML Schema standard.

1 Introduction

XML (eXtensible Markup Language) constitutes the basic format for data exchange on the Web [4]. For many applications, it is important to constrain the structure of documents by providing a schema specified in a schema language. The most common schemas are *Document Type Definitions (DTDs)*. A DTD is basically a set of rules of the form $a \to r$, where a is a tag name and r is a regular expression. A document is *valid* with respect to a DTD if each element labeled with a has a sequence of children whose tags match r. We view an XML document as a tree in the way indicated by Figure 1.

Unfortunately, DTDs are limited in various ways. A particular limitation is that the type of an element can only depend on its tag but not on its context. As an example, in Figure 1 it is not possible to assign different types to discount DVDs and non-discount DVDs while retaining the same tag.

XML Schema Definitions (XSDs) is the standard proposed by the World Wide Web consortium (W3C) to answer the shortcomings of DTDs [5]. In database theory, the latter are modeled by extended context-free grammars, the former by unranked regular tree languages [2]. Such regular tree languages can be represented by specialized DTDs (SDTDs) [16] allowing to assign types a^i to elements with tags a (cf. Definition 2). The rules are of the form $a^i \to r$ where r is a regular expression over types, i.e., the rules constrain, for each element type, the sequence of types of sub-elements. In our example, regular DVDs could get the type dvd^1, discount DVDs the type dvd^2 (cf. Section 2.2). A tree is then valid w.r.t. an SDTD if there is an assignment of types matching the rules of the grammar. The enlarged flexibility of SDTDs requires an additional algorithmic task: besides simply checking validity it will often be necessary to compute a matching assignment. We refer to this as *typing*.

The goal of the present paper is to identify the largest class of SDTDs which can be typed in a streaming fashion. In other words, when processing an XML document as a stream of opening and closing tags, the type of each element should be uniquely determined when the opening tag is met. We will refer to this as *1-pass preorder typing*. The latter can be an important first step in processing streaming XML data. On top of this information, e.g., subscription queries can be defined (e.g., *inform me if there are new discounted dvds*) and their evaluation can be optimized.

Note that a document is valid w.r.t. an SDTD if all elements can be correctly typed. Hence, 1-pass preorder typing implies 1-pass (preorder) validation, but not vice versa. Indeed, consider the SDTD consisting of the rules $a^0 \to b^1 + b^2$, $b^1 \to c$ and $b^2 \to d$, defining the finite tree language $\{a(b(c)), a(b(d))\}$. This language can easily be validated via an algorithm making a preorder traversal through the input tree, but does not admit preorder typing: the type of the b-element cannot be determined without looking at its child.

Murata, Lee and Mani [14] proposed[3] two restrictions of SDTDs, *single-type* and *restrained competition*, which guarantee 1-pass preorder typing. An SDTD is *single-type* if for each rule $a^i \to r$ and each tag b at most one type b^j occurs in r. It is *restrained competition* if there is no rule $a^i \to r$ for which there exist strings wb^ju and wb^kv in $L(r)$ with $j \neq k$. Clearly, both restrictions assure 1-pass preorder typing. However, from the definition of these restrictions it is not immediately clear whether they are the weakest possible to ensure 1-pass preorder typing. More importantly, a precise semantical characterization providing insight in fundamental properties of these classes remained open.

Contributions. It turns out that an SDTD admits 1-pass preorder typing if and only if its trimmed version (i.e., without useless symbols) is restrained competition. So, a regular tree language admits 1-pass preorder typing if and only if it can be described by a restrained competition SDTD. Therefore, restrained competition SDTDs might be a good basis for an XML schema language extending XSDs without losing the ability of efficient parsing. Interestingly, for

[3] Actually, they defined these classes in the slightly different framework of regular tree grammars. We use SDTDs here to simplify proofs. Nevertheless, w.r.t. defining tree languages, the two formalisms are equally expressive and one can be translated into the other efficiently in a straightforward manner.

this purpose no further restriction to one-unambiguous regular expressions [3] is necessary. We discuss this further in Section 7.

Starting from this, we study the classes of tree languages which can be described by restrained-competition SDTDs and single-type SDTDs, respectively. The next contribution is a set of semantical characterizations of these classes. The main parameter in these characterizations is the dependency of the type of a node on the context of the node in the document. In particular, we prove that a regular tree language can be defined by (1) a single-type SDTD if and only if the type of each node only depends on the sequence of tags on the path from the root to the node; and, (2) a restrained competition SDTD if and only if the type of each node only depends on the tags of the nodes on the path from the root to the node *and their left siblings*. The other characterizations are in terms of closure properties, allowed patterns and guarded DTDs.

Next, we turn to algorithmic issues. Two algorithmic problems immediately arise from the above. Given an SDTD **d**, (1) is **d** a DTD, single-type SDTD or restrained competition SDTD, and (2) is there a DTD, single-type SDTD or restrained competition SDTD **d**' describing the same tree language as **d**? The first question is trivial for DTDs and single-type SDTDs. We prove that it is in NLOGSPACE for restrained competition SDTDs. The second question turns out to be much harder: in all three cases it is complete for EXPTIME. Furthermore, the algorithm is constructive. That is, if **d** is in fact in the desired class, an equivalent DTD, single-type SDTD or restrained competition SDTD $\mathbf{d}'$ is constructed.

Our semantical characterizations lead to easy optimization and minimization algorithms. Whereas the inclusion problem is EXPTIME-complete for general SDTDs (even with one-unambiguous regular expressions [13]) it follows from our characterizations that these problems are in PSPACE for restrained competition SDTDs and even in PTIME if it is additionally required that the regular expressions are one-unambiguous. We show that, in contrast to general SDTDs (cf. Section 5.2), for every tree language definable by restrained competition grammars, there exists a unique minimal restrained competition grammar that describes it. Moreover, this minimal grammar can be computed in polynomial time.

We conclude with an observation on post-order typing. Although in general, arbitrary SDTDs do not admit 1-pass preorder typing, we show that for each regular tree language there is an SDTD which allows *1-pass postorder typing*, i.e., a parsing algorithm that determines a type of an element when it reaches its closing tag. That every SDTD allows 1-pass *validation* was already observed by Segoufin and Vianu [18].

Related work. Brüggemann-Klein, Murata, and Wood study unranked regular tree languages as a formal model for XML schema languages [2]. In particular, they prove that the latter model is equivalent to the morphic image of tree-local tree languages. Papakonstantinou and Vianu [16] formalize the latter as the more manageable specialized DTDs which are used in this paper. Murata et al. [14] provided a taxonomy of XML schema languages in terms of restrictions on grammars which are equivalent to specialized DTDs. In particular, they pro-

```
<store>
  <dvd>
    <title> "Amelie" </title> <price> 17 </price>
  </dvd>
  <dvd>
    <title> "Gothika" </title>
    <price> 15 </price> <discount> 4 </discount>
  </dvd>
</store>
```

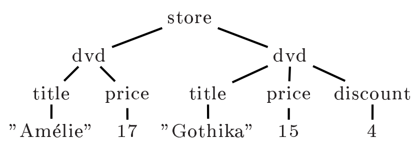

Fig. 1. An example of an XML document and its tree representation.

pose to formalize DTDs, XML Schema, and Relax NG [24] as local, single-type, and arbitrary regular tree grammars, respectively. They also introduce the notion of restrained competition and show that these are 1-pass preorder typeable but do not discuss optimality or give any semantical characterizations.

The organization of the paper is as follows. In Section 2 we define the various classes of SDTDs and the properties by which we characterize them. The actual characterizations are given in Section 3. In Section 4 the complexity of the basic decision problems is addressed. In Section 5, we discuss optimization and minimization algorithms. Section 6 shows that every regular tree language allows 1-pass postorder typing. We discuss our results in Section 7.

2 Definitions

2.1 Trees and Tree Languages

For our purposes, an XML document is basically a sequence of opening and closing tags, properly nested. As usual, we identify XML documents with their corresponding trees. The domain $\mathrm{Dom}(t)$ of a tree t is the set of its nodes, represented in a fixed way by sequences of numbers. The empty sequence ε represents the root. The n children of a node u are named $u1, \ldots, un$ in the order given by the document. Nodes carry labels from alphabet Σ of tags. We denote the label of v in t by $\mathrm{lab}^t(v)$. The set of all unranked Σ-trees is denoted by $\mathcal{T}_\Sigma$. A *tree language* is a set of trees. For a gentle introduction into trees, tree languages and tree automata we refer to [15].

2.2 XML Schema Languages

Definition 1. A *DTD* is a pair (d, s_d) where d is a function that maps Σ-symbols to regular expressions and $s_d \in \Sigma$ is the start symbol. We usually simply denote (d, s_d) by d. A tree t *is valid w.r.t.* d (or *satisfies* d) if its root is labeled by s_d and, for every node with label a, the sequence $a_1 \cdots a_n$ of labels of its children is in $L(d(a))$. By $L(d)$ we denote the set of trees that satisfy d.

A simple example of a DTD defining the inventory of a store is the following:

$$\text{store} \to \text{dvd}\ \text{dvd}^* \qquad \text{dvd} \to \text{title}\ \text{price}(\text{discount} + \varepsilon)$$

Definition 2 ([16, 17]). A *specialized DTD* (SDTD) is a 4-tuple $\mathbf{d} = (\Sigma, \Sigma', (d, s_d), \mu)$, where Σ' is an alphabet of *types*, (d, s_d) is a DTD over Σ' and μ is a mapping from Σ' to Σ. A tree t is *valid w.r.t.* $\mathbf{d}$ (or satisfies $\mathbf{d}$) if $t = \mu(t')$ for some $t' \in L(d)$ (where μ is extended to trees). Again, we denote the set of trees defined by $\mathbf{d}$, by $L(\mathbf{d})$. We denote by $(\mathbf{d}, a^i)$ the specialized DTD $\mathbf{d}$, where we replace the DTD (d, s_d) by (d, a^i).

The class of tree languages defined by SDTDs corresponds precisely to the regular (unranked) tree languages [2]. For ease of exposition, we always take $\Sigma' = \{a^i \mid 1 \leq i \leq k_a, a \in \Sigma, i \in \mathbb{N}\}$ for some $k_a \in \mathbb{N}$ and set $\mu(a^i) = a$. We refer to the label a^i of a node (or sometimes also to i) in t' as its *state* or *type*. We say that an SDTD $\mathbf{d}$ is *trimmed* if d has no unreachable rules and that there exists no $a^i \in \Sigma'$ for which $L((d, a^i)) = \emptyset$. Note that $L((d, a^i))$ contains trees over alphabet Σ', whereas $L((\mathbf{d}, a^i))$ contains Σ-trees. In the remainder of the paper, we assume that all SDTDs are trimmed. We note that trimming an SDTD is PTIME-complete. A simple example of an SDTD is the following:

$$\text{store} \to (\text{dvd}^1 + \text{dvd}^2)^*\text{dvd}^2(\text{dvd}^1 + \text{dvd}^2)^*$$
$$\text{dvd}^1 \to \text{title price} \qquad \text{dvd}^2 \to \text{title price discount}$$

Here, dvd^1 defines ordinary DVDs while dvd^2 defines DVDs on sale. The rule for store specifies that there should be at least one DVD on discount.

Murata et al. [14] argue that the expressiveness of SDTDs corresponds to the XML schema language Relax NG, while the single-type SDTDs defined next correspond to XML Schema.

Definition 3. A *single-type SDTD* (SDTD$^{\text{st}}$) is an SDTD $(\Sigma, \Sigma', d, \mu)$ in which in no regular expression $d(a)$ two types b^i and b^j with $i \neq j$ occur.

The above defined SDTD is not single type as both dvd^1 and dvd^2 occur in the rule for store. An example of a single-type SDTD is given next:

$$\text{store} \to \text{regulars discounts}$$
$$\text{regulars} \to (\text{dvd}^1)^* \qquad \text{discounts} \to \text{dvd}^2\ (\text{dvd}^2)^*$$
$$\text{dvd}^1 \to \text{title price} \qquad \text{dvd}^2 \to \text{title price discount}$$

Although there are still two element definitions dvd^1 and dvd^2, they can only occur in a different context. The next class was defined in [14] because it still allows 1-pass preorder typing.

Definition 4. A regular expression r *restrains competition* if there are no strings wa^iv and wa^jv' in $L(r)$ with $i \neq j$. An SDTD is *restrained competition* (SDTD$^{\text{rc}}$) iff all regular expressions occurring in rules restrain competition.

An example of a restrained competition SDTD that is not single-type is given next:

$$\text{store} \to (\text{dvd}^1)^*\ \text{discounts}\ (\text{dvd}^2)^*$$
$$\text{discounts} \to \varepsilon \qquad \text{dvd}^1 \to \text{title price} \qquad \text{dvd}^2 \to \text{title price discount}$$

The classes of tree languages defined by the grammars introduced above are included as follows: DTD $\subsetneq$ SDTD$^{\text{st}}$ $\subsetneq$ SDTD$^{\text{rc}}$ $\subsetneq$ SDTD [14].

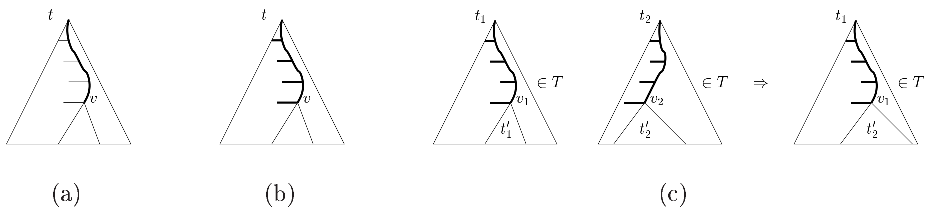

Fig. 2. Illustration of notions introduced in Section 2.3. Figures 2(a) and 2(b) illustrate the ancestor-string (anc-str) and ancestor-sibling string (anc-sib-str) of v. Figure 2(c) illustrates the notion of ancestor-sibling-guarded subtree exchange.

2.3 Ancestor- and Ancestor-Sibling-Patterns

Finally, we define the notions that will be used in our semantical characterizations. Let t be a tree and v be a node. By $\text{ch-str}^t(v)$ we denote the string formed by the children of v, i.e., $\text{lab}^t(v1)\cdots\text{lab}^t(vn)$ if v has n children. Usually we omit the superscript t. By $\text{anc-str}^t(v)$ we denote the string formed by the labels on the path from the root to v, i.e., $\text{lab}^t(\varepsilon)\text{lab}^t(i_1)\text{lab}^t(i_1i_2)\cdots\text{lab}^t(i_1i_2\cdots i_k)$ where $v = i_1i_2\cdots i_k$. By $\text{l-sib-str}^t(v)$ we denote the string formed by the labels of the left siblings of v, i.e., $\text{lab}^t(u1)\cdots\text{lab}^t(uk)$ where $v = uk$. By $\text{anc-sib-str}^t(v)$ we denote the string $\text{l-sib-str}^t(\varepsilon)\#\text{l-sib-str}^t(i_1)\#\cdots\#\text{l-sib-str}^t(i_1i_2\cdots i_k)$ formed by concatenating the left-sibling strings of all ancestors starting from the root. We assume that $\# \notin \Sigma$. Note that the final symbol of $\text{anc-str}^t(v)$ and $\text{anc-sib-str}^t(v)$ is always the label of v.

Definition 5. We say that a specialized SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ *has ancestor-based types* if there is a (partial) function $f : (\Sigma \cup \{\#\})^* \to \Sigma'$ such that, for each tree $t \in L(\mathbf{d})$ the following holds: (1) there is a unique tree $t' \in L(d)$ with $\mu(t') = t$; and (2) for each node $v \in \text{Dom}(t)$, the label of v in t' is $f(\text{anc-str}^t(v))$. We say $\mathbf{d}$ *has ancestor-sibling based types* if the same holds with $\text{anc-str}^t(v)$ replaced by $\text{anc-sib-str}^t(v)$.

By $t_1[u \leftarrow t_2]$ we denote the tree obtained from a tree t_1 by replacing the subtree rooted at $u \in \text{Dom}(t_1)$ by t_2. By $\text{subtree}^t(u)$ we denote the subtree of t rooted at u.

Definition 6. We say that a tree language T is *closed under ancestor-guarded subtree exchange* if the following holds. Whenever for two trees $t_1, t_2 \in T$ with nodes $u_1 \in \text{Dom}(t_1)$ and $u_2 \in \text{Dom}(t_2)$ it holds that $\text{anc-str}^{t_1}(u_1) = \text{anc-str}^{t_2}(u_2)$ implies $t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)] \in T$. We call it *closed under ancestor-sibling-guarded subtree exchange* if the same property holds with $\text{anc-sib-str}^{t_1}(u_1) = \text{anc-sib-str}^{t_2}(u_2)$ as precondition of the implication. Figure 2 illustrates the just defined notions.

Definition 7. An *ancestor-guarded DTD* $\mathbf{d}$ is a pair (d, s_d) where $s_d \in \Sigma$ is the start symbol as in a DTD. But in contrast to a DTD, d is a finite set of triples (r, a, s), where $a \in \Sigma$ and r and s are regular expressions. If there are triples (r, a, s) and (r', a, s') in d then $L(r)$ and $L(r')$ are disjoint. A tree t satisfies $\mathbf{d}$ if for every node $v \in \text{Dom}(t)$ the following holds. If anc-str(v) matches r and lab$(v) = a$ there must be a triple (r, a, s) in d and ch-str(v) must match s.

An *ancestor-sibling-guarded DTD* is defined in the same way with the difference that r has to be matched by anc-sib-str(v).

Definition 8. Let $P_{\text{anc}}(t) = \{\text{anc-str}(v)\#\text{ch-str}(v) \mid v \in t\}$ and $P_{\text{anc-sib}}(t) = \{\text{anc-sib-str}(v)\#\text{ch-str}(v) \mid v \in t\}$. Let T be a set of trees. We say that T *can be characterized by ancestor-based patterns*, if there is a regular string language L such that, for every tree t, we have that $t \in T$ if and only if $P_{\text{anc}}(t) \subseteq L$. We say T *can be characterized by ancestor-sibling-based patterns* if the same holds with $P_{\text{anc}}(t)$ replaced by $P_{\text{anc-sib}}(t)$.

3 Semantic Characterizations of Single-Type and Restrained Competition SDTDs

In this section, we first show that an SDTD is restrained competition if and only if it allows for 1-pass preorder typing. Afterwards, as an intermediate step, we characterize the regular tree languages definable by single-type SDTDs. Finally, we characterize the class of tree languages which can be described by restrained competition SDTDs.

3.1 Schemas with 1-Pass Preorder Typing

It follows from Theorem 12 that in restrained competition SDTDs the type of a node only depends on its ancestor-sibling string. However, in an SDTD which admits 1-pass preorder typing the type of a node might depend on all parts of the tree which occur before the node. We formalize this notion via SDTDs with preceding based types. Nevertheless, it will turn out that these two notions are identical.

For a tree t and a node v we denote by preceding$^t(v)$ the tree resulting from t by removing everything below v, all right siblings of v's ancestors and of v, and their respective subtrees (cf. Figure 3). We define the term *preceding-based types* in analogy to Definition 5 with preceding$^t(v)$ in place of anc-str$^t(v)$.

Expressed in a different way, the type of an element only depends on the prefix of the XML document ending with its opening tag.

Theorem 9. *A trimmed SDTD* $\mathbf{d}$ *has preceding based types if and only if it is restrained competition.*

Proof sketch. The "if"-part of the statement is obvious. We sketch the "only if". Actually, it is easy to show that every trimmed SDTD $\mathbf{d}$ with ancestor-sibling based types is restrained competition. Otherwise, a counterexample could

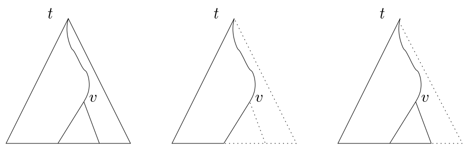

Fig. 3. From left to right: a tree t, preceding$^t(v)$ and preceding-subtree$^t(v)$.

be constructed in a straightforward manner (cf. Theorem 12). It can also be shown by contraposition that each SDTD with preceding based types already has ancestor-sibling based types. □

Hence, we immediately obtain the following:

Corollary 10. *Restrained competition SDTDs are exactly those SDTDs which admit 1-pass preorder typing.*

3.2 Ancestor Based Schemas

In this subsection, we characterize single-type SDTDs in terms of the ancestor axis. In the following theorem we assume that all the trees in language T have the same root label.

Theorem 11. *For a regular tree language T the following are equivalent:*

(a) T is definable by a single-type SDTD;
(b) T is definable by an SDTD with ancestor-based types;
(c) T is closed under ancestor-guarded subtree exchange;
(d) T can be characterized by ancestor-based patterns; and,
(e) T is definable by an ancestor-guarded DTD.

Proof. We show the following sequence of implications. (a) ⇒ (e) ⇒ (d) ⇒ (b) ⇒ (c) ⇒ (a). We only give the necessary constructions.

(a) ⇒ (e): Let T be defined by a single-type SDTD $\mathbf{d} = (\Sigma, \Sigma', (d, s_d), \mu)$ with $\perp \notin \Sigma'$. Let A be a DFA over Σ with state set $Q = \Sigma' \cup \{\perp\}$ and let $\delta(a^i, b)$ equal the unique b^j occurring in $d(a^i)$ if such a symbol exists, otherwise $\perp$. Note that the single-type property ensures that A is deterministic.

Let $\mathbf{d}' = (d', s_d)$ be the guarded DTD with all triples $(r_{a,i}, a, \mu(d(a^i)))$, where $r_{a,i}$ is a regular expression describing the set $\{w \mid \delta^*(s_d, w) = a^i\}$ of strings which bring A into state a^i. Of course, the languages $L(r_{a,1}), \ldots, L(r_{a,k_a})$ are all disjoint where $\{a^1, \ldots, a^{k_a}\}$ are the symbols mapped to a by μ.

(e) ⇒ (d): Let T be defined by the ancestor-guarded DTD $\mathbf{d} = (d, s_d)$. Then T can be characterized by the set $L = \{ua\#v \mid ua \in L(r), v \in L(s), (r, a, s) \in d\}$.

(d) ⇒ (b): Let T be characterized by ancestor-based patterns using the language L. Let $A = (\Sigma, Q, \delta, s, F)$ be a DFA for L. Let $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ be defined as follows. Σ' is the set of all pairs (a, q), where $a \in \Sigma$ and $q \in Q$. We

let $d((a,q))$ be a regular expression describing all strings $(b_1,q_1)\cdots(b_n,q_n)$, for which A accepts $\#b_1\cdots b_n$ when started from state q and $q_i=\delta(q,b_i)$, for every $i\le n$.

(b) ⇒ (c): Let T be defined by a SDTD $\mathbf{d}=(\Sigma,\Sigma',d,\mu)$ with ancestor-based types. Let t_1,t_2 be in T and let u_1 and u_2 be nodes in t_1 and t_2, respectively, with $\text{anc-str}^{t_1}(u_1)=\text{anc-str}^{t_2}(u_2)$. Let t_1' and t_2' be the unique trees in $L(d)$ with $\mu(t_1')=t_1$ and $\mu(t_2')=t_2$. As the labels of u_1 in t_1' and the label of u_2 in t_2' are determined by $\text{anc-str}^{t_1}(u_1)=\text{anc-str}^{t_2}(u_2)$, they are the same. Hence, by replacing the subtree rooted at u_1 in t_1' with the subtree rooted at u_2 in t_2' we get a tree $t'\in L(d)$. Therefore, $\mu(t')=t_1[u_1\leftarrow \text{subtree}^{t_2}(u_2)]$ is in T, as required.

(c) ⇒ (a): The idea of the proof is as follows. In a sense, we close a given SDTD $\mathbf{d}$ for T with respect to the single-type property. Assume, e.g., that the regular expression $d(a^i)$ contains two different types b^j and b^k. Then, we replace all occurrences of b^j and b^k by a new type $b^{\{j,k\}}$ obtaining a single-type expression with respect to b. Of course, we now need a new rule with $b^{\{j,k\}}$ on the left-hand side. This rule should capture the union of $d(b^j)$ and $d(b^k)$. By applying this step inductively, we arrive at an SDTD $\mathbf{d_1}$ which is single-type but uses types of the form b^S, for $S\subseteq\{1,\dots,k_b\}$ where $\{1,\dots,k_b\}$ are the types of b in Σ'. In a second step we prove that $L(\mathbf{d_1})=T$ unless T fails to fulfill (c).

Let T be a tree language defined by an SDTD $\mathbf{d}=(\Sigma,\Sigma',d,\mu)$. Let the alphabet Σ_1' consist of all symbols a^S, where $S\subseteq\{1,\dots,k_a\}$. We extend this notation to sets $C\subseteq\Sigma'$ in a natural way. We write a^C for the type a^S with $S=\{i\mid a^i\in C\}$. For example, for $C=\{a^1,a^2,b^1,b^3\}$, a^C is the type $a^{\{1,2\}}$. For a regular expression r over Σ' and $C\subseteq\Sigma'$ let r^C denote the expression which is obtained from r by replacing every symbol a^i by a^C.

We define the SDTD $\mathbf{d_1}=(\Sigma,\Sigma_1',d_1,\mu_1)$ as follows. For each symbol a^S, $\mu_1(a^S)=a$, and $d_1(a^S)=\bigcup_{i\in S}d(a^i)^{C(a^S)}$, where $C(a^S)$ is the set of all b^j in $\bigcup_{i\in S}d(a^i)$. For instance, for $S=\{1,2\}$, $d(a^1)=a^1b^1(a^2+b^1)$ and $d(a^2)=(a^3+b^3)a^1$, $d_1(a^S)$ equals the expression $(a^{\{1,2,3\}}b^{\{1,3\}}(a^{\{1,2,3\}}+b^{\{1,3\}}))+((a^{\{1,2,3\}}+b^{\{1,3\}})a^{\{1,2,3\}})$.

Note that in $d_1(a^S)$, for each symbol $b\in\Sigma$, there is at most one symbol of the form $b^{S'}$, hence $\mathbf{d_1}$ is a single-type SDTD. It can be shown that, if $L(\mathbf{d})\neq L(\mathbf{d_1})$, the language T is not closed under ancestor-guarded subtree exchange. By contraposition we get that (c) implies (a). □

It should be noted that an analogous characterization can be easily obtained for DTDs by replacing *ancestor* by *parent*. The equivalence between (c) and (a) is then already obtained in [16].

3.3 Ancestor-Sibling Based Schemas

Finally, we consider restrained competition SDTDs and show that their tree languages can be characterized in terms of the ancestor and left-sibling axis. We again assume that all the trees in language T have the same root label.

Theorem 12. *For a regular tree language T the following are equivalent:*

(a) T is definable by a restrained competition SDTD;
(b) T is definable by an SDTD with ancestor-sibling-based types;
(c) T is closed under ancestor-sibling-guarded subtree exchange;
(d) T can be characterized by ancestor-sibling-based patterns; and
(e) T is definable by an ancestor-sibling-guarded DTD.

Proof. Again we show (a) $\Rightarrow$ (e) $\Rightarrow$ (d) $\Rightarrow$ (b) $\Rightarrow$ (c) $\Rightarrow$ (a).
(e) $\Rightarrow$ (d), (d) $\Rightarrow$ (b), (b) $\Rightarrow$ (c): These proofs are almost word for word the same as for Theorem 11. Only *ancestor* has to be replaced by *ancestor-sibling*.

(c) $\Rightarrow$ (a): The proof is similar as but a bit more involved than the corresponding proof in Theorem 11. Let T be a tree language defined by a SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$.

Let, for each state a^i of $\mathbf{d}$, $A_{a,i} = (Q_{a,i}, \Sigma', \delta_{a,i}, s_{a,i}, F_{a,i})$ be an NFA for $L(d(a^i))$. W.l.o.g. we assume that the sets $Q_{a,i}$ are pairwise disjoint and that for every state in each $A_{a,i}$ a final state is reachable.

Let Σ'_1 be defined as in the proof of Theorem 11. We define, for each $a^S \in \Sigma'_1$ a DFA $A_{a,S} = (Q_{a,S}, \Sigma'_1, \delta_{a,S}, s_{a,S}, F_{a,S})$ as follows.

– $Q_{a,S} = \{q^\perp\} \cup \bigcup_{i \in S} 2^{Q_{a,i}}$;
– $s_{a,S} = \bigcup_{i \in S} \{s_{a,i}\}$;
– $F_{a,S} = \{B \in Q_{a,S} \mid B \cap F_{a,i} \neq \emptyset, i \in S\}$;
– In order to define $\delta_{a,S}$, let $B \in Q_{a,S}$ and $b \in \Sigma$. We set $S' := \{j \mid \delta_{a,i}(p, b^j) \neq \emptyset, i \in S, j \leq k_b, p \in B\}$ and $\delta_{a,S}(B, b^{S'}) := \bigcup_{i,p,j} \delta_i(p, b^j)$, where the latter union is over all $i \in S$, $p \in B$ and $j \leq k_b$. For all other sets S'', we set $\delta_{a,S}(B, b^{S''}) := q^\perp$.

Intuitively, $A_{a,S}$ can be seen as obtained in two steps from $\mathbf{d}$. First, we take the product of the power set automata of the $A_{a,i}$, $i \in S$. Then, for each symbol b, for each state of this intermediate automaton, all outgoing edges with label of the form b^j are combined into one transition which ends in the (component wise) union of the all possible target states. The transition is labeled by b to the union of all outgoing b-labels.

We now define the SDTD $\mathbf{d}_1 = (\Sigma, \Sigma'_1, d_1, \mu_1)$, where, for each a and S, $d_1(a^S)$ is a regular expression corresponding to $A_{a,S}$.

Note that each $d_1(a^S)$ has restrained competition. Indeed, as $A_{a,S}$ is deterministic, for each string w, $A_{a,S}$ enters a unique state. Furthermore, for each $b \in \Sigma$ there is only one outgoing transition of the form $b^{S'}$ that can lead to acceptance.

(a) $\Rightarrow$ (e): Let T be defined by a restrained competition DTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$. For each symbol a^i in Σ', let $A_{a,i} = (Q_{a,i}, \Sigma', \delta_{a,i}, s_{a,i}, F_{a,i})$ be a DFA for $d(a^i)$. We can modify $A_{a,i}$ such that it has exactly one state $q^\perp$ from which no accepting state is reachable and such that it has no unreachable states (possibly besides $q^\perp$). From the restrained competition property it immediately follows that in $A_{a,i}$, for each state q, if $\delta(q, b^j) = q_1$, $\delta(q, b^k) = q_2$, $q_1 \neq q_2$ and $j \neq k$ then q_1 or q_2 must be $q^\perp$. We require that the sets $Q_{a,i}$ are pairwise disjoint.

From these DFAs over the extended alphabet Σ' we construct a DFA $A = (Q_A, \Sigma, s_A, \delta_A, F_A)$ as follows. The set Q_A consists of all pairs (q, b), where $q \in$

$Q_{a,i}$, for some a^i, and $b \in \Sigma' \cup \{\#\}$. Intuitively, q is the current state of an automaton $A_{a,i}$ and b is the last extended symbol or type that has been identified. The initial state s_A of A is $(s_{a,i}, \#)$ for the initial symbol a^i of d. The transition function δ_A is defined as follows. For each $q \in Q_{a,i}$, $c \in \Sigma'$ and $b \in \Sigma$ we let $\delta_A((q, c), b) = (\delta_{a,i}(q, b^j), b^j)$, for the unique j with $\delta_{a,i}(q, b^j) \neq q^\perp$, if such a j exists. Otherwise, $\delta_A((q, c), b) = (q^\perp, \#)$. Furthermore, we let $\delta_A((q, b^j), \#) = (s_{b,j}, \#)$. We set $F_A = \{q \mid q \in F_{a,i}\}$.

Now we are ready to define the ancestor-sibling guarded DTD $\mathbf{d}'$. It consists of all triples (r, a, s), for which there is a state (q, a^i) of A, such that r describes the set of strings w with $\delta^*_A(s_A, w) = (q, a^i)$ and s is $\mu(d(a^i))$. □

4 Complexity of Basic Decision Problems

As the definition of a DTD and single-type SDTD is syntactical in nature, it can be immediately verified by an inspection of the rules whether an SDTD is in fact a DTD or a single-type SDTD.

Theorem 13. *It is decidable in* NLOGSPACE *for an SDTD* $\mathbf{d}$ *whether it is restrained competition.*

We study the complexity of determining whether a tree language, given by an SDTD, can be defined by a DTD, a single-type or a restrained competition SDTD, respectively.

Theorem 14. *Each of deciding whether an SDTD has an equivalent DTD, single-type SDTD or restrained competition SDTD is* EXPTIME*-complete.*

Proof sketch. In all three cases, the lower bound is obtained by a reduction from the universality problem for non-deterministic tree automata [19].

The exponential time upper bounds for the single-type and restrained competition cases can be obtained by performing the constructions in the proofs (c) $\Rightarrow$ (a) in Theorems 11 and 12. Both the construction of the SDTD and checking equivalence with the original one can be done in exponential time. For DTDs a similar construction is in polynomial time but the equivalence check still needs exponential time. □

5 Applications of the Semantical Characterizations

5.1 Inclusion and Equivalence of Schemas

Decision problems like testing for inclusion or equivalence of schema languages often occur in schema optimization or as basic building blocks of algorithms for typechecking or type inference [8, 11, 12, 16, 22]. In general these problems are PSPACE and EXPTIME-complete for DTDs and SDTDs, respectively [21, 19]. The XML specification, however, restricts regular expressions in DTDs to be deterministic [4] (sometimes also called 1-unambiguous [3]).

Theorem 15. *Given two restrained competition SDTDs $\mathbf{d_1}$ and $\mathbf{d_2}$, deciding whether (a) $L(\mathbf{d_1}) \subseteq L(\mathbf{d_2})$, and whether (b) $L(\mathbf{d_1}) = L(\mathbf{d_2})$ is* PSPACE*-complete in general, and* PTIME*-complete if $\mathbf{d}_1$ and $\mathbf{d}_2$ use deterministic regular expressions.*

This result strongly contrasts with our results in [13], where we show that even for very simple non-deterministic regular expressions these decision problems are intractable, and with the case of arbitrary SDTDs with deterministic regular expressions, for which inclusion and equivalence test are EXPTIME-complete.

5.2 Minimization of SDTDs

In strong contrast to ranked trees, there are unranked regular tree languages for which there is no unique minimal deterministic bottom-up tree automaton. Moreover, minimization can not be obtained by the standard translation to the ranked case. Using the characterizations of Section 3, we obtain that when content models are represented by DFAs rather than by regular expressions, every restrained competition SDTD can be minimized in polynomial time and this minimal SDTD is unique up to isomorphism.

Theorem 16. *Every restrained competition (single-type) SDTD can be minimized in* PTIME*. This minimal SDTD is unique up to isomorphism.*

6 Subtree Based Schemas

From what was presented so far an obvious question arises. What happens if we soften the requirement that the type of an element has to be determined when its *opening* tag is visited? What if instead it has to be computed when the *closing* tag is seen? It turns out that every regular tree language has a SDTD which allows such 1-pass *postorder* typing. Furthermore, the SDTDs used for this purpose can be defined as straightforward extensions of restrained competition SDTDs.

Definition 17. An SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ is *extended restrained competition* iff for every regular expression r occurring in a rule the following holds: whenever there are two strings wa^iv and wa^jv' in $L(r)$ with $i \neq j$, then $L((\mathbf{d}, a^i)) \cap L((\mathbf{d}, a^j))$ is empty.

For a tree t and a node v we denote by preceding-subtree$^t(v)$ the tree resulting from t by removing all right siblings of v and its ancestors together with the respective subtrees (cf. Figure 3).

Definition 18. We say that a specialized SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ *has preceding-subtree based types* if there is a (partial) function $f : \mathcal{T}_\Sigma \times \mathrm{Dom} \to \Sigma'$ such that, for each tree $t \in L(\mathbf{d})$ the following holds: (1) there is a unique tree $t' \in L(d)$ with $\mu(t') = t$, and (2) for each node $v \in \mathrm{Dom}(t)$, the label of v in t' is $f(\text{preceding-subtree}^t(v), v)$.

Stated in terms of XML documents, the type of an element depends on the prefix of the document which ends with the closing tag of the element. The following result shows that all regular tree languages admit 1-pass postorder typing. We assume that all the trees in language T have the same root label.

Theorem 19. *For a tree language T the following are equivalent:*

(a) T is definable by an extended restrained competition SDTD;
(b) T is definable by an SDTD with preceding-subtree-based types;
(c) T is regular.

Proof sketch. The directions (a) $\Rightarrow$ (c) and (b) $\Rightarrow$ (c) are trivial. The proof of the opposite directions uses the fact that regular languages can be validated by deterministic bottom-up automata. $\square$

In the SDTD used in the proof the type of each element actually only depends on its subtree. This should be compared with the previous characterizations where the type depended on the upper context. These issues are further discussed in Section 7.

Note that not every SDTD is extended restrained competition. The SDTD $\mathbf{d}$ defined by $r \to (a^1 + a^2)$, $a^1 \to b + c + \varepsilon$, and $a^2 \to c + d + \varepsilon$ is *not* extended restrained competition, as $\{\varepsilon, c\} \subseteq L((\mathbf{d}, a^1)) \cap L((\mathbf{d}, a^2))$.

We conclude by noting that extended restrained competition is a tractable notion.

Theorem 20. *It is decidable in* PTIME *for an SDTD* $\mathbf{d}$ *whether it is extended restrained competition.*

7 Conclusion

The results of this paper show that its initial question has a simple answer. The regular tree languages which admit 1-pass preorder typing are exactly those which can be described by a restrained competition SDTD.

From the proof of Theorem 12 (c) $\Rightarrow$ (a) it further follows that for each such language a very simple and efficient typing algorithm exists. It is basically a deterministic pushdown automaton with a stack the height of which is bounded by the depth of the document. For each opening tag it pushes one symbol, for each closing tag it pops one. Hence, it only needs a constant number of steps per input symbol. In particular, it works in linear time in the size of the document. It should be noted that such automata have been studied in [18] and [9] in the context of streaming XML documents. The subclass of the context-free languages accepted by such automata has recently been studied in [1].

Further, the paper shows that restrained competition SDTDs can be efficiently recognized (in NLOGSPACE but also in quadratic time) and that from an SDTD without the restrained competition property an equivalent one with the property can effectively (though not efficiently, in general) be constructed if it exists at all.

The 1-pass preorder typing constraint can be seen as a generalization of the determinism constraint on content models of DTDs (Appendix E in [4]) to XSDs. In the case of DTDs, the meaning of a tag is determined by the position in the matching regular expression. The determinism constraint then specifies that this meaning should be computed independent of the tags occurring to the right of the current tag. Similarly, in the context of XML Schema, the meaning of a tag corresponds to its type and should be computed independent of the remainder of the nodes.

Brüggemann-Klein and Wood gave a clean formalization for the concept of determinism needed for DTDs in terms of 1-unambiguous regular expressions [3]. Intuitively, a regular expression is 1-unambiguous if, when processing the input from left to right, it is always determined which symbol in the expression matches the next input symbol. Just as Brüggemann-Klein and Wood contributed to the formal underpinnings of DTDs, our characterization contributes to the foundation of XML Schema by providing a complete notion for 1-pass preorder typeable schemas.

How do these results relate to existing standards? The XML Schema specification requires XSDs to be single-type (end of Section 4.5 in [6] and the Element Declarations Consistent constraint in Section 3.8.6 in [7]) and regular expressions (after dropping the superscripts describing the types) to be deterministic or 1-unambiguous [3] (cf. Section 3.8.6 of [7], Unique Particle Attribution). Although such schemas are always restrained competition, it is easy to prove that they do not capture the complete class of 1-pass preorder typeable schemas. Indeed, from a 1-ambiguous regular language a restrained competition expression can be easily constructed by giving to each symbol the same superscript. The results in the present paper, therefore, indicate that replacing the Element Declarations Consistent and Unique Particle Attribution constraints by the single requirement that regular expressions are restrained competition allows for a larger expressive power without (essential) loss in efficiency. Indeed, for both classes, validation and typing is possible in linear time, allowed schemas can still be recognized in quadratic time and an allowed schema can be constructed in exponential time, if one exists [3]. The latter would also eliminate the heavily debated restriction to 1-unambiguous regular expressions (cf., e.g., pg 98 of [23] and [10, 20]).

On the negative side, both 1-unambiguous expressions and restrained competition expressions lack a comprehensive syntactical counterpart. Whether such an equivalent syntactical restriction exists remains open. It would also be interesting to find syntactic restrictions which imply an efficient construction of an equivalent restrained competition SDTD.

We already mentioned that Murata, Lee, and Mani showed that $\text{DTD} \subsetneq \text{SDTD}^{\text{st}} \subsetneq \text{SDTD}^{\text{rc}} \subsetneq \text{SDTD}$. They exhibited concrete tree languages that are in one class but not in the other. Our semantical characterizations provide a toolbox to show inexpressibility for arbitrary tree languages. For instance, using the closure of restrained-competition SDTDs under ancestor-guarded subtree exchange, it is immediate that SDTD^{rc} cannot define the set of all Boolean tree-shaped circuits evaluating to true.

Acknowledgments

We thank Geert Jan Bex, Christoph Koch, Nicole Schweikardt, Luc Segoufin and Stijn Vansummeren for helpful discussions.

References

1. R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC 2004*, pages 202-211, 2004.
2. 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.
3. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.
4. World Wide Web Consortium. Extensible Markup Language (XML). http://www.w3.org/XML.
5. World Wide Web Consortium. XML Schema. http://www.w3.org/XML/Schema.
6. World Wide Web Consortium. XML Schema Part 0: Primer. http://www.w3.org/TR/xmlschema-0/.
7. World Wide Web Consortium. XML Schema Part 1: Structures. http://www.w3.org/TR/xmlschema-1/.
8. H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Transactions on Internet Technology (TOIT)*, 3(2):117–148, 2003.
9. C. Koch and S. Scherzinger. Attribute grammars for scalable query processing on XML streams. In *DBPL*, pages 233–256, 2003.
10. M. Mani. Keeping chess alive - Do we need 1-unambiguous content models? In *Extreme Markup Languages*, Montreal, Canada, 2001.
11. W. Martens and F. Neven. Typechecking top-down uniform unranked tree transducers. In *ICDT 2003*, pages 64–78, 2003.
12. W. Martens and F. Neven. Frontiers of tractability for typechecking simple XML transformations. In *PODS 2004*, pages 23–34, 2004.
13. W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions. In *MFCS 2004*, pages 889–900, 2004.
14. M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages*, Montreal, Canada, 2001.
15. F. Neven. Automata, logic, and XML. In *CSL 2002*, pages 2–26. Springer, 2002.
16. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS 2000*, pages 35–46. ACM Press, 2000.
17. Y. Papakonstantinou and V. Vianu. Incremental validation of XML documents. In *ICDT 2003*, pages 47–63. Springer, 2003.
18. L. Segoufin and V. Vianu. Validating streaming XML documents. In *PODS 2002*, pages 53–64. ACM Press, 2002.
19. H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.
20. C. M. Sperberg-McQueen. XML Schema 1.0: A language for document grammars. In *XML 2003 - Conference Proceedings*, 2003.
21. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC 1973*, pages 1–9, 1973.
22. D. Suciu. Typechecking for semistructured data. In *DBPL 2001*, 2001.
23. E. van der Vlist. *XML Schema*. O'Reilly, 2002.
24. E. van der Vlist. *Relax NG*. O'Reilly, 2003.

A Appendix: Full Proofs

For the convenience of the referees we give the full proofs of those theorems which only have proof sketches in the main text. The point where the proof in the main text ended is indicated by $\boxed{\cdots}$. We also restate the respective theorems.

Before that we first describe some notions related to trees and tree automata.

As an abstraction of XML-documents, we define *documents* over a set Σ of tags and a set Γ of basic symbols as follows. For $w \in \Gamma$, $\langle a\rangle w\langle /a\rangle$ is a document. If $a \in \Sigma$ and $x_1, \ldots, x_k$ are documents then $\langle a\rangle x_1 \cdots x_k\langle /a\rangle$ is also a document. We refer to the string enclosed by matching tags as an *element*. Figure 1 shows an example of a document. It also indicates how documents can be represented as trees.

Of course, elements in XML documents can also contain references to nodes. But as XML schema languages usually do not constrain these and can only specify the format of data values occurring at leaves (e.g., a string should be telephone number), it is safe to view schemas as simply defining documents over an empty alphabet Γ, i.e., where the only basic string is the empty word ε.

Formally, we associate an *unranked Σ-tree* $t = t(x)$ with a document x as follows.

(i) if $x = \langle a\rangle\varepsilon\langle /a\rangle$, for some $a \in \Sigma$, then the *set of nodes of t*, denoted by $\mathrm{Dom}(t)$, is $\{\varepsilon\}$. The label $\mathrm{lab}^t(\varepsilon)$ is a;

(ii) if $x = \langle a\rangle x_1 \cdots x_n\langle /a\rangle$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n}\{iu \mid u \in \mathrm{Dom}(t(x_i))\}$, $\mathrm{lab}^t(\varepsilon) = a$, and for each $iu \in \mathrm{Dom}(t)$, $\mathrm{lab}^t(iu) = \mathrm{lab}^{t(x_i)}(u)$.

We call a node ui a *child* of u and u the *parent* of ui. If $w = uv$, for some v then u is an *ancestor* of w. A node ui, $i < j$, is a *left sibling* of uj.

The set of all unranked Σ-trees is denoted by $\mathcal{T}_\Sigma$. A *tree language* is a set of trees. In the remainder of this paper we will identify documents with their corresponding trees.

In the remainder of the appendix, we denote by $(\mathbf{d}, a^i)$ the specialized DTD $\mathbf{d} = (\Sigma, \Sigma', (d, r), \mu)$, where we replace the DTD (d, r) by (d, a^i).

The robust notion of regular string and ranked tree languages, can easily be generalized to the unranked counterparts. The latter class is usually defined in terms of non-deterministic tree automata and posses similar closure properties [2].

Definition 21. A *nondeterministic 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. The set of all accepted trees is denoted by $L(B)$. The class of tree languages accepted by NTAs is called the *unranked regular tree languages.*

An NTA is *bottom-up deterministic* iff $\delta(q,a) \cap \delta(q',a) = \emptyset$ for all $q \neq q'$. For every unranked regular tree language there is a bottom-up deterministic NTA which accepts it.

Semantic Characterizations of Single-Type and Restrained Competition SDTDs

Proof of Theorem 9. *A trimmed SDTD* $\mathbf{d}$ *has preceding based types if and only if it is restrained competition.*

The "if"-part of the statement is obvious. We show the "only if". Actually, it is easy to show that every trimmed SDTD $\mathbf{d}$ with ancestor-sibling based types is restrained competition. Otherwise, a counterexample could be constructed in a straightforward manner. Hence, it only remains to show that each SDTD with preceding based types already has ancestor-sibling based types.

$\boxed{\cdots}$ Let $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ be an SDTD which has preceding based types. Towards a contradiction we assume that $\mathbf{d}$ has types which are *not* ancestor-sibling based. Hence, there are trees $t_1, t_2 \in L(\mathbf{d})$ with nodes $v_1 \in t_1$ and $v_2 \in t_2$ such that $\text{anc-sib-str}^{t_1}(v_1) = \text{anc-sib-str}^{t_2}(v_2)$ but v_1 has a different label in t_1' than v_2 in t_2'. We call t_1, t_2, v_1, v_2 a *counterexample.* Let t_1, t_2, v_1, v_2 be a counterexample for which the length of $\text{anc-sib-str}^{t_1}(v_1)$ is minimal.

Let $u_1, \ldots, u_n$ be the nodes that are siblings of ancestors of v_1 in the order in which they appear in a depth-first left-to-right walk on t_1. Let $w_1, \ldots, w_n$ be the corresponding nodes in t_2. Because the counterexample is minimal, for each $i \leq n$, the label of u_i in t_1' is the same as the label of w_i in t_2'. Let s be the tree resulting from t_1 by replacing, for every i, the subtree rooted at u_i by the subtree rooted at w_i in t_2.

Let the labels in s' be defined as in t_2' for the nodes that come into s by replacements and as in t_1' for the others. Obviously, $s' \in L(d)$. But as $\text{preceding}^{s}(v_1) = \text{preceding}^{t_2}(v_1)$, v_1 must have the same label in s' as in t_2'. As it also has the same label in t_1' as in s' it follows that the labels in t_1' and t_2' are the same which leads to the desired contradiction. This completes the proof of the theorem. $\square$

Proof of Theorem 11. *For a regular tree language T the following are equivalent:*

(a) T is definable by a single-type SDTD;
(b) T is defined by an SDTD with ancestor-based types;
(c) T is closed under ancestor-guarded subtree exchange;
(d) T can be characterized by ancestor-based patterns; and,
(e) T is definable by an ancestor-guarded DTD.

We show the following sequence of implications.

$$\text{(a)} \Rightarrow \text{(e)} \Rightarrow \text{(d)} \Rightarrow \text{(b)} \Rightarrow \text{(c)} \Rightarrow \text{(a)}.$$

$\boxed{\text{(a)} \Rightarrow \text{(e)}}$: Let T be defined by a single-type SDTD $\mathbf{d} = (\Sigma, \Sigma', (d, s_d), \mu)$ with $\perp \notin \Sigma'$. Let A be a DFA over Σ with state set $Q = \Sigma' \cup \{\perp\}$ and let $\delta(a^i, b)$ equal the unique b^j occurring in $d(a^i)$ if such a symbol exists, otherwise $\perp$. Note that the single-type property ensures that A is deterministic.

Now we define a guarded DTD $\mathbf{d}' = (d', s_d)$ by putting all triples $(r_{a,i}, a, \mu(d(a^i)))$ into d', where $r_{a,i}$ is a regular expression which describes the set $\{w \mid \delta^*(s_d, w) = a^i\}$ of strings which bring A into state a^i. Of course, the languages $L(r_{a,1}), \ldots, L(r_{a,k_a})$ are all disjoint where $\{a^1, \ldots, a^{k_a}\}$ are the symbols mapped to a by μ. $\boxed{\cdots}$ It remains to show that $\mathbf{d}'$ defines the same set of trees as $\mathbf{d}$. Let t be in $L(\mathbf{d})$. Hence, there is t' in $L(d)$ with $\mu(t') = t$. It is easily shown by induction that, for each node v of t', $\text{lab}^{t'}(v) = \delta^*(s_d, \text{anc-str}^t(v))$. Hence, for each node v labeled with a^i, the triple of $\mathbf{d}'$ responsible for v is $(r_{a,i}, a, \mu(d(a^i)))$ and can therefore be applied. The proof of the opposite inclusion is similar.

$\boxed{\text{(e)} \Rightarrow \text{(d)}}$: Let T be defined by the ancestor-guarded DTD $\mathbf{d} = (d, s_d)$. Let L be the set

$$\{ua\#v \mid ua \in L(r), v \in L(s), (r, a, s) \in d\}.$$

$\boxed{\cdots}$ By definition, for every tree $t \in T$ it holds that $P_{\text{anc}}(t) \subseteq L$. For the other direction, let t be a tree which is not in T. Hence, there is a node w in t with some label a such that either there is no triple $(r, a, s) \in d$ with $\text{anc-str}(w) \in L(r)$ or for every such triple $\text{ch-str}(w) \notin L(s)$. This implies that $\text{anc-str}\#\text{ch-str}(w) \notin L$. Therefore, a tree t is in T if and only if $P_{\text{anc}}(t) \subseteq L$ which shows (d).

$\boxed{\text{(d)} \Rightarrow \text{(b)}}$: Let T be characterized by ancestor-based patterns using the language L. Let $A = (\Sigma, Q, \delta, s, F)$ be a DFA for L. Let $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ be defined as follows. Σ' is the set of all pairs (a, q), where $a \in \Sigma$ and $q \in Q$. We let $d((a, q))$ be a regular expression which describes the set of all strings $(b_1, q_1) \cdots (b_n, q_n)$, for which A accepts $\#b_1 \cdots b_n$ when started from state q and $q_i = \delta(q, b_i)$, for every $i \leq n$. $\boxed{\cdots}$ Obviously, $\mathbf{d}$ defines T. Furthermore, for each node v in a tree $t \in T$, (a, q) is uniquely determined and only depends on $\text{anc-str}(v)$. Hence, $\mathbf{d}$ has ancestor-based types.

$\boxed{\text{(b)} \Rightarrow \text{(c)}}$: Let T be defined by an SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ with ancestor-based types. Let t_1, t_2 be in T and let u_1 and u_2 be nodes in t_1 and t_2, respectively, with $\text{anc-str}^{t_1}(u_1) = \text{anc-str}^{t_2}(u_2)$. Let t_1' and t_2' be the unique trees in $L(d)$ with $\mu(t_1') = t_1$ and $\mu(t_2') = t_2$. As the labels of u_1 in t_1' and the label of u_2 in t_2' are determined by $\text{anc-str}^{t_1}(u_1) = \text{anc-str}^{t_2}(u_2)$, they are the same. Hence, by replacing the subtree rooted at u_1 in t_1' with the subtree rooted at u_2 in t_2' we get a tree $t' \in L(d)$. Therefore, $\mu(t') = t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)]$ is in T, as required.

$\boxed{\text{(c)} \Rightarrow \text{(a)}}$: The idea of the proof is as follows. In a sense, we close a given SDTD $\mathbf{d}$ for T with respect to the single-type property. Assume, e.g., that the regular expression $d(a^i)$ contains two different types b^j and b^k. Then, we replace all occurrences of b^j and b^k by a new type $b^{\{j,k\}}$ obtaining a single-type expression with respect to b. Of course, we now need a new rule with $b^{\{j,k\}}$ on the left-hand side. This rule should capture the union of $d(b^j)$ and $d(b^k)$. By applying this step inductively, we arrive at an SDTD $\mathbf{d_1}$ which is single-type but uses types of the form b^S, for $S \subseteq \{1, \ldots, k_b\}$ and $\{b^1, \ldots, b^{k_b}\}$ are the types of b in Σ'. In a second step we prove that $L(\mathbf{d_1}) = T$ unless T fails to fulfill (c).

Let T be a tree language defined by an SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$. Let the alphabet Σ'_1 consist of all symbols a^S, where $S \subseteq \{1, \ldots, k_a\}$ and $\{a_1, \ldots, a_{k_a}\}$ are the types of a in Σ'.

We extend this notation to sets $C \subseteq \Sigma'$ in a natural way. We write a^C for the type a^S with $S = \{i \mid a^i \in C\}$. For example, for $C = \{a^1, a^2, b^1, b^3\}$, a^C is the type $a^{\{1,2\}}$. For a regular expression r over Σ' and $C \subseteq \Sigma'$ let r^C denote the expression which is obtained from r by replacing every symbol a^i by a^C.

We define the SDTD $\mathbf{d_1} = (\Sigma, \Sigma'_1, d_1, \mu_1)$ as follows. For each symbol a^S, $\mu_1(a^S) = a$, and

$$d_1(a^S) = \bigcup_{i \in S} d(a^i)^{C(a^S)},$$

where $C(a^S)$ is the set of all b^j in $\bigcup\limits_{i \in S} d(a^i)$.

For instance, for $S = \{1,2\}$, $d(a^1) = a^1 b^1 (a^2 + b^1)$ and $d(a^2) = (a^3 + b^3)a^1$, $d_1(a^S)$ equals the expression $(a^{\{1,2,3\}} b^{\{1,3\}} (a^{\{1,2,3\}} + b^{\{1,3\}})) + ((a^{\{1,2,3\}} + b^{\{1,3\}})a^{\{1,2,3\}})$.

Note that in $d_1(a^S)$, for each symbol $b \in \Sigma$, there is at most one symbol of the form $b^{S'}$, hence $\mathbf{d_1}$ is a single-type SDTD. We show next that, if $L(\mathbf{d}) \neq L(\mathbf{d_1})$, the language T is not closed under ancestor-guarded subtree exchange. By contraposition we get that (c) implies (a).

$\boxed{\cdots}$ To this end, first observe that when moving from $\mathbf{d}$ to $\mathbf{d_1}$ no trees are lost. Indeed, let $t' \in L(d)$ be a witness for $t \in L(\mathbf{d})$. We get a tree $t'' \in L(d_1)$ with $\mu_1(t'') = t$ as follows. We assign to each node v a type from Σ'_1 in a top-down fashion. If the root node has type a^i in t' it gets type $a^{\{i\}}$. Let now v be a node with type a^i in t' and already assigned type a^S in t''. Then a child u of v with label b gets the type $b^{C(a^S)}$ in t''. Of course, the sequence of t''-types at the children of u matches $d_1(a^S)$ because the sequence of t'-types matches $d(a^i)$. Hence, we have $L(\mathbf{d}) \subseteq L(\mathbf{d_1})$.

Consider now a tree $t \in L(\mathbf{d_1}) - L(\mathbf{d})$ and its extension $t' \in L(d_1)$ with $\mu_1(t') = t$. Each node u has a type a^S in t', and we write $S(u)$ for S. For each u, ch-str(u) matches $d_1(a^{S(u)})$. More precisely, it matches $d(a^i)^{C(a^S)}$, for some $i \in S(u)$. Let g be a function, which fixes one such a^i, for each node u. On the other hand, as ch-str$(u) = v_1, \ldots, v_n$ matches $d(a^i)^{C(a^S)}$ we can assign to each node v_i a type $f(v_i) \in \Sigma'$ such that $f(v_1) \cdots f(v_n)$ is in $L(d(a^i))$. Note that f is defined for each node besides the root. Furthermore, if the type of a node v in t' is a^S then $f(v)$ is of the form a^i, for some $i \in S$. We call a node v *critical*, if $f(v) \neq g(v)$. Note that t must contain at least one critical node, because otherwise f and g would witness that $t \in L(\mathbf{d})$. By $c(t,f,g)$ we denote $\sum_{v \text{ critical}} \text{depth}(v)$.

Now let $t \in L(\mathbf{d_1}) - L(\mathbf{d})$, $t' \in L(d_1)$ and f and g be fixed such that $c(t,f,g)$ is as small as possible. Let v be a critical node in t such that there is no other critical node below v and let $\text{lab}^t(v) = b^S$. Let $b^j = g(v)$, $b^k = f(v)$ and let t_v be the subtree of t rooted at v, so $t_v \in L((\mathbf{d}, b^j))$. Let t_0 be an arbitrary tree in $L((\mathbf{d}, b^k))$ and let t_1 denote the tree resulting from t by replacing t_v with t_0. As $j,k \in S$, it is easy to see that $t_1 \in L(\mathbf{d_1})$ with f_1 and g_1, that can be obtained by extending f and g such that no node below v in t_1 is critical and $g_1(v) = f_1(v) = b^k$. Hence, $c(t_1, f_1, g_1) < c(t,f,g)$ and therefore $t_1 \in L(\mathbf{d})$.

We construct another tree t_2 from t as follows. Let u denote the parent of v. Let $a^{S'}$ be the label of u in t'. By our construction, there must be an a^ℓ, $\ell \in S'$, such that b^j occurs in $d(a^\ell)$. Hence, there is a string $w = w_1 \cdots w_k \in L(d(a^\ell))$ such that $w_i = b^j$, for some i. For each symbol w_m, we pick a tree $t_{w_m} \in L((d, w_m))$, in particular let $t_{w_i} = t_v$. Let t_2 result from t by plugging in the trees $t_{w_1}, \ldots, t_{w_k}$ below u (and deleting all nodes that had been below u in t). The node corresponding to w_i is called x.

Let f_2 and g_2 be defined as f and g, respectively, for all nodes in t_2 which are not in the subtree rooted at u and let $g_2(u) = a^\ell$. Clearly, $t_2 \in L(\mathbf{d_1})$. As $\text{subtree}^{t_2}(u) \in L((\mathbf{d}, a^\ell))$, the functions f_2 and g_2 can be chosen such that no node below u is critical. Hence, in t_2 the node v is no longer critical (because it was deleted) but the node u might have become critical. But, as the depth of u is smaller than the depth of v, $c(t_2, f_2, g_2) < c(t,f,g)$, therefore $t_2 \in L(\mathbf{d})$.

Note that t can be obtained from t_1 by replacing t_0 by t_x. Hence, as $\text{anc-str}^{t_2}(x) = \text{anc-str}^{t_1}(v)$, closure of T under ancestor-guarded subtree exchange would imply $t \in L(\mathbf{d})$, the desired contradiction. □

Proof of Theorem 12. *For a regular tree language T the following are equivalent:*

(a) T is definable by a restrained competition SDTD;
(b) T is defined by an SDTD with ancestor-sibling-based types;
(c) T is closed under ancestor-sibling-guarded subtree exchange;
(d) T can be characterized by ancestor-sibling-based patterns; and
(e) T is definable by an ancestor-sibling-guarded DTD.

Again we show (a) ⇒ (e) ⇒ (d) ⇒ (b) ⇒ (c) ⇒ (a).

$\boxed{\text{(e)} \Rightarrow \text{(d), (d)} \Rightarrow \text{(b), (b)} \Rightarrow \text{(c)}}$: These proofs are almost word for word the same as for Theorem 11. Only *ancestor* has to be replaced by *ancestor-sibling.* In the proof (d) ⇒ (b), the states q_i must be defined so that $q_i = \delta^*(q, b_1 \cdots b_i)$ for every $i \leq n$.

$\boxed{\text{(c)} \Rightarrow \text{(a)}}$: The proof is similar as but a bit more involved than the corresponding proof in Theorem 11. Let T be a tree language defined by an SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$.

Let, for each type a^i of $\mathbf{d}$, $A_{a,i} = (Q_{a,i}, \Sigma', \delta_{a,i}, s_{a,i}, F_{a,i})$ be an NFA for $L(d(a^i))$. W.l.o.g. we assume that the sets $Q_{a,i}$ are pairwise disjoint.

Let Σ'_1 be defined as in the proof of Theorem 11. We define, for each $a^S \in \Sigma'_1$ a DFA $A_{a,S} = (Q_{a,S}, \Sigma'_1, \delta_{a,S}, s_{a,S}, F_{a,S})$ as follows.

- $Q_{a,S} = \{q^\perp\} \cup \bigcup_{i \in S} 2^{Q_{a,i}}$;
- $s_{a,S} = \{s_{a,i} \mid i \in S\}$;
- $F_{a,S} = \{B \in Q_{a,S} \mid B \cap F_{a,i} \neq \emptyset, i \in S\}$;
- In order to define $\delta_{a,S}$, let $B \in Q_{a,S}$ and $b \in \Sigma$. We set

  $$S' := \{j \mid \delta_{a,i}(p, b^j) \neq \emptyset, i \in S, j \leq k_b, p \in B\}$$

  and

  $$\delta_{a,S}(B, b^{S'}) := \bigcup_{i,p,j} \delta_i(p, b^j),$$

  where the latter union is over all $i \in S$, $p \in B$ and $j \leq k_b$. For all other sets S'', we set $\delta_{a,S}(B, b^{S''}) := q^\perp$.

Intuitively, $A_{a,S}$ can be seen as obtained in two steps from $\mathbf{d}$. First, we take the product of the power set automata of the $A_{a,i}$, $i \in S$. Then, for each symbol b, for each state of this intermediate automaton, all outgoing edges with label of the form b^j are combined into one transition which ends in the (component-wise) union of the all possible target states. The transition is labeled by b to the union of all outgoing b-labels.

We now define the SDTD $\mathbf{d_1} = (\Sigma, \Sigma'_1, d_1, \mu_1)$, where, for each a and S, $d_1(a^S)$ is a regular expression corresponding to $A_{a,S}$.

Note that each $d_1(a^S)$ has restrained competition. Indeed, as $A_{a,S}$ is deterministic, for each string w, $A_{a,S}$ enters a unique state. Furthermore, for each $b \in \Sigma$ there is only one outgoing transition of the form $b^{S'}$ that can lead to acceptance.

$\boxed{\cdots}$ In analogy to the corresponding proof for Theorem 11 it is sufficient to show that, if $L(\mathbf{d}) \neq L(\mathbf{d_1})$, the language T is not closed under ancestor-sibling-guarded subtree exchange. Again, $L(\mathbf{d}) \subseteq L(\mathbf{d_1})$. Therefore, for each $t \in L(\mathbf{d_1}) - L(\mathbf{d})$ and its extension $t' \in L(d_1)$, we can define mappings g and f in correspondence to the proof of Theorem 11. Let f and g be functions which assign to each node of t a type from Σ' such that, for each node u for which $g(u) = a^i$, with type a^S in t' and children $v_1, \ldots, v_n$ it holds that, $\delta^*_{a,S}(s_{a,S}, f(v_1) \cdots f(v_n)) \cap F_{a,i} \neq \emptyset$. Furthermore, if a node v has type a^S in t', and $g(v) = a^j$ and $f(v) = a^k$, then $\{j, k\} \subseteq S$. Note that the construction of $A_{a,S}$ guarantees the existence of such functions.

Again, we call a node v *critical* if $f(v) \neq g(v)$, and we write $c(t, f, g)$ for $\sum_{v \text{ critical}} \text{depth}(v)$.

Now let again $t \in L(\mathbf{d_1}) - L(\mathbf{d})$, $t' \in L(d_1)$ and f and g be fixed such that $c(t, f, g)$ is as small as possible. Let v be a critical node in t such that there is no other critical node in the subtree t_v rooted at v and such that it is the leftmost critical child of its parent node u.

Let a^S be the label of u in t', let $v_1, \ldots, v_n$ be the children of u from left to right and let m be such that $v = v_m$. Let $B \in Q_{a,S}$ be the state of $A_{a,S}$ after reading $\text{lab}^{t'}(v_1) \cdots \text{lab}^{t'}(v_{m-1})$.

Let $b^j = g(v)$, $b^k = f(v)$ and let t_0 be an arbitrary tree in $L((\mathbf{d}, b^k))$. Let t_1 denote the tree resulting from t by replacing t_v with t_0. It is easy to see that $t_1 \in L(\mathbf{d_1})$ with $g_1(v) = f_1(v) = b^k$ and that g_1 and f_1 can be obtained by extending f and g such that no node below v is critical. Hence, $c(t_1, f_1, g_1) < c(t, f, g)$ and therefore $t_1 \in L(\mathbf{d})$.

Let t_2 be constructed from t as follows. Recall that a^S is the label of u in t'. By our construction, there must be an a^ℓ, $\ell \in S$ and a string $z_{m+1} \cdots z_n$ so that $f(v_1) \cdots f(v_{m-1}) b^j z_{m+1} \cdots z_n \in L(d(a^\ell))$, because $b^j \in S'$, where S' is unique so that $\delta_{a,S}(B, b^{S'}) \neq q^\perp$. For each symbol $f(v_i)$ for $i < m$, let s_i be a tree in $L((\mathbf{d}, f(v_i)))$, and for each $i > m$ we take a tree s_i in $L((\mathbf{d}, z_i))$. Let t_2 result from t by deleting all nodes below u and plugging in the trees $s_1, \ldots, s_{m-1}, t_v, s_{m+1}, \ldots, s_n$ below u.

Let f_2 and g_2 be defined as f and g respectively, for all nodes in t_2 which are not in the subtree rooted at u, and let $g_2(u) = a^\ell$. Clearly, $t_2 \in L(\mathbf{d}_1)$. Below u, the functions f_2 and g_2 can be chosen such that no node below u is critical. Analogously as in Theorem 11, we have that $c(t_2, f_2, g_2) < c(t, f, g)$ and therefore $t_2 \in L(\mathbf{d})$.

But for the m-th child x of u in t_2 it holds that $\text{anc-sib-str}^{t_2}(x) = \text{anc-sib-str}^{t}(v)$. Therefore t results from $t_1 \in T$ by replacing the subtree t_0 rooted at v with the subtree t_v rooted at x in t_2. Hence, if T was closed under ancestor-sibling-guarded subtree exchange, t would be in T too, a contradiction.

$\boxed{\text{(a)} \Rightarrow \text{(e)}}$: Let T be defined by a restrained competition DTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$. For each symbol a^i in Σ' let $A_{a,i} = (Q_{a,i}, \Sigma', \delta_{a,i}, s_{a,i}, F_{a,i})$ be a DFA for $d(a^i)$. We can modify $A_{a,i}$ such that it has exactly one state $q^\perp$ from which no accepting state is reachable and such that it has no unreachable states (possibly besides $q^\perp$). From the restrained competition property it immediately follows that in $A_{a,i}$, for each state q, if $\delta(q, a^i) = q_1$, $\delta(q, a^j) = q_2$, $q_1 \neq q_2$ and $i \neq j$ then q_1 or q_2 must be $q^\perp$. We require that the sets $Q_{a,i}$ are pairwise disjoint.

From these DFAs over the extended alphabet Σ' we construct a DFA $A = (Q_A, \Sigma, s_A, \delta_A, F_A)$ as follows. The set Q_A consists of all pairs (q, b), where $q \in Q_{a,i}$, for some a^i, and $b \in \Sigma' \cup \{\#\}$. Intuitively, q is the current state of an automaton $A_{a,i}$ and b is the last extended symbol or type that has been identified. The initial state s_A of A is $(s_{a,i}, \#)$ for the initial symbol a^i of d. The transition function δ_A is defined as follows. For each $q \in Q_{a,i}$, $c \in \Sigma'$ and $b \in \Sigma$ we let $\delta_A((q, c), b) = (\delta_{a,i}(q, b^j), b^j)$, for the unique j with $\delta_{a,i}(q, b^j) \neq q^\perp$, if such a j exists. Otherwise, $\delta_A((q, c), b) = (q^\perp, \#)$. Furthermore, we let $\delta_A((q, b^j), \#) = (s_{b,j}, \#)$. We set $F_A = \{q \mid q \in F_{a,i}\}$.

Now we are ready to define the ancestor-sibling guarded DTD $\mathbf{d}'$. It consists of all triples (r, a, s), for which there is a state (q, a^i) of A, such that r describes the set of strings w with $\delta_A^*(s_A, w) = (q, a^i)$ and s is $\mu(d(a^i))$.

$\boxed{\cdots}$ It only remains to show that $\mathbf{d}'$ and $\mathbf{d}$ describe the same tree language. By the construction it is obvious that every tree in $L(\mathbf{d})$ is also in $L(\mathbf{d}')$: indeed, for a tree $t \in L(\mathbf{d})$ and a node v, the automaton enters a state (q, a^i) after reading the symbol a corresponding to v if and only if v gets the label a^i in the unique labeling with respect to $\mathbf{d}$. Hence, ch-str(v) is in $\mu(d(a^i))$.

Now let $t \in L(\mathbf{d}')$ and let v be a node of t. If anc-sib-str(v) matches r in (r, a, s) then, by construction, v can only be labeled by a^i if a labeling of t with respect to $\mathbf{d}$ exists. But then, as s is $\mu(d(a^i))$, ch-str(v) is in $\mu(d(a^i))$. As this holds for all nodes v, we can conclude that t matches $\mathbf{d}$. □

Complexity of Basic Decision Problems

Proof of Theorem 13. *It is decidable in* NLOGSPACE *for an SDTD* $\mathbf{d}$ *whether it is restrained competition.*

$\boxed{\cdots}$ We need to check that every regular expression r occurring in a rule restrains competition. We present a nondeterministic logspace algorithm which accepts a regular expression if it does *not* restrain competition. As NLOGSPACE is closed under complement, the theorem follows.

Let $N_r = (\Sigma', Q, \delta, q_0, F)$ be an NFA equivalent to r. The algorithm works as follows. Let R denote the set $\{q \mid \exists v \in \Sigma'^* \text{ such that } \delta^*(q, v) \cap F \neq \emptyset\}$ of states from which a final state can be reached.

1. it first guesses two states (q_1, q_2) of N_r;
2. it verifies that there is a string u such that $\{q_1, q_2\} \subseteq \delta^*(q_0, u)$;
3. it verifies that there are a, i, j such that $\delta(q_1, a^i) \cap R \neq \emptyset$ and $\delta(q_2, a^j) \cap R \neq \emptyset$;
4. it accepts if all these verifications work out.

Obviously, this algorithm accepts r, if and only if there are strings u, v, w such that ua^iv and ua^jw are in $L(r)$ as required. Furthermore, all steps can be done in logarithmic space, as neither the NFA A nor the set R has to be computed in advance. Indeed, it can be checked in logarithmic space that, for given p, q, b^j, whether $q \in R$ and whether $q \in \delta(p, b^j)$. □

Let NTA(REG) denote the class of NTAs where the regular languages encoding the transition function are represented by regular expressions.

Lemma 22. *Let $a \in \Sigma$ and let A be an NTA(REG), only having one accept state, such that whenever $t \in L(A)$ then the root of t is labeled a. Then an SDTD $\mathbf{d}$ can be computed in* PTIME *such that $L(A) = L(\mathbf{d})$.*

Proof. Let $A = (Q, \Sigma = \{a_1, \ldots, a_n\}, \delta, F = \{q_F\})$ be an NTA(REG). Then define $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ as follows: $\Sigma' = \{b^q \mid b \in \Sigma, q \in Q\}$, $\mu(b^q) = b$ for every $b \in \Sigma$, $s_d = a^{q_F}$, and d consists of the rules $d(b^q) = r_{b,q}$ where $r_{b,q}$ is the regular expression obtained from $\delta(b, q)$ by replacing every occurrence of a state p by $(a_1^p + \cdots + a_n^p)$. As every $t \in L(d)$ induces an accepting run of A on $\mu(t)$, it is immediate that A and $\mathbf{d}$ are equivalent. □

Proof of Theorem 14. *Each of deciding whether an SDTD has an equivalent DTD, single-type SDTD or restrained competition SDTD is* EXPTIME*-complete.*

In all three cases, we make use of a reduction from the universality problem for NTAs, which is known to be hard for EXPTIME [19].

$\boxed{\cdots}$ The latter even holds for NTA(REG) where automata only have one final state. Therefore, let A be an NTA(REG) over alphabet $\Sigma = \{a, b\}$. By Lemma 22, an equivalent SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ can be constructed in PTIME.

We now modify $\mathbf{d}$ into an SDTD $\mathbf{d}_1$ over the alphabet $\Delta = \{a, b, \alpha, \beta, \text{root}\}$ which accepts all trees t such that t is of the form $\text{root}(\sigma(t'))$ where σ is α or β, $t' \in \mathcal{T}_\Sigma$, and the tree obtained from t' by deleting the right-most leaf is accepted by A.

Let $\mathbf{d}_2$ be the SDTD accepting all trees t of the form $\text{root}(\sigma(t'))$ where the right-most leaf is a (respectively, b) when σ is α (respectively, β).

Finally, define $\mathbf{d}_3$ as the SDTD accepting $L(\mathbf{d}_1) \cup L(\mathbf{d}_2)$. Set $S := L(\mathbf{d}_3)$.

We show the following

(a) if $L(A) = \mathcal{T}_\Sigma$ then S is defined by a DTD; and,
(b) if $L(A) \neq \mathcal{T}_\Sigma$ then S is not defined by a restrained competition SDTD.

Of course (a) and (b) together imply the statement of the theorem.

(a) First note that when $L(A) = \mathcal{T}_\Sigma$, then $L(\mathbf{d}_2) \subseteq L(\mathbf{d}_1)$ and S equals $\{\text{root}(\sigma(t)) \mid \sigma \in \{\alpha, \beta\}, t \in \mathcal{T}_\Sigma\}$. The latter can clearly be defined by a DTD.

(b) Let $L(A) \neq \mathcal{T}_\Sigma$ and let t be a tree not in $L(A)$. Let t_a and t_b be the trees obtained from t by adding an a and b respectively, to the right of the right-most leaf. Then $t'_a := \text{root}(\alpha(t_a)) \in S$ while $t'_b := \text{root}(\alpha(t_b)) \notin S$. Let t''_b be the tree obtained from t'_b by adding an a-leaf as right-most child of α, i.e. $t''_b := \text{root}(\alpha(t_b a))$. By definition of B, $t''_b \in S$. Let vn be the right-most leaf of t'_a and let v be its parent. Then note that $\text{anc-sib-str}^{t'_a}(v) = \text{anc-sib-str}^{t''_b}(v)$. So, by Theorem 12, $t'_a[v \leftarrow \text{subtree}^{t''_b}(v)]$ is in S when S is defined by a restrained competition SDTD. Hence, (b) follows.

The exponential time upper bounds are shown as follows.

$\boxed{\cdots}$

– In the case of single-type SDTDs we proceed as follows. Let $\mathbf{d}$ be a given SDTD. We first construct the SDTD $\mathbf{d_2}$ as described in the proof of Theorem 11 (c) $\Rightarrow$ (a). This can be done in exponential time and $\mathbf{d_2}$ might be of exponential size in $\mathbf{d}$. Then it has to be checked whether they are equivalent. Fortunately, as always $L(\mathbf{d}) \subseteq L(\mathbf{d_2})$, we only have to check whether $L(\mathbf{d_2}) - L(\mathbf{d})$ is empty. This involves the complementation of the tree automaton for $\mathbf{d}$ resulting in a tree automaton of possibly exponential size and in the test whether the automata for $L(\mathbf{d_2})$ and the complement of $L(\mathbf{d})$ have a non-empty intersection. The latter is polynomial in the size of the automata. Hence, we altogether get an exponential time algorithm.
– Testing whether a SDTD has an equivalent restrained competition SDTD can be done along the same lines, this time based on the proof of Theorem 12 (c) $\Rightarrow$ (a). Note that, the size of the automata $A_{a,S}$ is at most exponential in the size of $\mathbf{d}$.

– Finally, we describe how it can be tested whether a given SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ has an equivalent DTD. Let, for each $a^i \in \Sigma'$, $r_{a,i}$ be the regular expression obtained from $d(a^i)$ by replacing every symbol b^j by b. We define a DTD (d_1, s_d) with alphabet Σ simply by taking the rules $a \to \bigcup_i r_{a,i}$, for every $a \in \Sigma$. It remains to show that $\mathbf{d}$ has an equivalent DTD if and only if $L(\mathbf{d}) = L(d_1)$.
Analogously as in Theorem 11((c)⇒(a)), we have that $L(\mathbf{d}) \subseteq L(d_1)$. Towards a contradiction, suppose that $\mathbf{d}$ has an equivalent DTD and that $t \in L(d_1) - L(\mathbf{d})$. According to Lemma 2.10 in [16], $L(\mathbf{d})$ is closed under parent-guarded subtree exchange. As $t \notin L(\mathbf{d})$ there exists a node u in t such that $\text{subtree}^t(u) \notin L((\mathbf{d}, a^i))$ for any $a^i \in \Sigma'$, but for every child $u1, \ldots, un$ of u, we have that $\text{subtree}^t(uj) \in L((\mathbf{d}, b_j^{i_j}))$ for some $b_j^{i_j} \in \Sigma'$. By definition of d_1, for every $b_j^{i_j}$, there exists an a^k such that $b_j^{i_j}$ occurs in $d(a^k)$. So, for every uj there exists a tree $t_j \in L(\mathbf{d})$ with a $v \in \text{Dom}(t)$ such that $\text{lab}^{t_j}(v) = b_j$, the parent of v is labeled a, and $\text{subtree}^{t_j}(v) = \text{subtree}^t(u)$. But this means that t can be constructed from $t_1, \ldots, t_n$ by parent-guarded subtree exchange, which is a contradiction as $t \notin L(\mathbf{d})$.

□

Applications of the Characterizations

Inclusion and Equivalence of Schemas

Proof of Theorem 15. *Given two restrained competition SDTDs* $\mathbf{d_1}$ *and* $\mathbf{d_2}$*, deciding whether*

(a) $L(\mathbf{d_1}) \subseteq L(\mathbf{d_2})$*, and whether*
(b) $L(\mathbf{d_1}) = L(\mathbf{d_2})$

is PSPACE*-complete in general, and* PTIME*-complete when* $\mathbf{d}_1$ *and* $\mathbf{d}_2$ *use deterministic regular expressions.*

$\boxed{\cdots}$ The theorem rather directly follows from the pattern based characterizations of the different subclasses.

For the upper bounds, (b) follows from (a), hence we only show (a).

It follows from Theorem 12 that for a tree language T defined by a restrained competition SDTD it holds that a tree t is in T if and only if $P_{\text{anc-sib}}(t)$ is in $P_{\text{anc-sib}}(T) := \{P_{\text{anc-sib}}(s) \mid s \in T\}$. Hence, $L(\mathbf{d_1}) \subseteq L(\mathbf{d_2})$ if and only if $P_{\text{anc-sib}}(L(\mathbf{d_1})) \subseteq P_{\text{anc-sib}}(L(\mathbf{d_2}))$.

The statement of the theorem now follows from the fact that, for each restrained competition SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$, an NFA A for $P_{\text{anc-sib}}(L(\mathbf{d}))$ can be computed in polynomial time and equivalence of NFAs can also be checked in polynomial space. Also, A is deterministic if $\mathbf{d}$ uses deterministic regular expressions.

The lower bounds are easy reductions of the inclusion and equivalence problems of regular expressions, which are PSPACE-complete, and from the emptiness problem of a language defined by a DTD, which is PTIME-complete.

For the upper bounds, it remains to show the construction of $A = (Q_A, \Sigma \cup \{\#\}, \delta_A, s_A, F_A)$.

Let for each $a^i \in \Sigma'$, $A_{a,i} = (Q_{a,i}, \Sigma', \delta_{a,i}, s_{a,i}, F_{a_i})$ be an NFA that defines $d(a^i)$ and has a unique state $q^\perp$ from which no final state is reachable. We adapt $A_{a,i}$ so that it uses alphabet Σ, but remembers the types of the symbols that it reads, i.e., we define $A'_{a,i} = (Q'_{a,i}, \Sigma, \delta'_{a,i}, s'_{a,i}, F'_{a_i})$ where $Q'_{a,i} = Q_{a,i} \times \Sigma'$. For each $b \in \Sigma$ and $c \in \Sigma'$ we define $\delta'_{a,i}((q,c),b) = \{(p, b^j) \mid p \in \delta_{a,i}(q, b^j)\}$. Note that, as $\mathbf{d}$ is restrained competition, $\delta'_{a,i}((q,c),b)$ contains no $(p_1, b^{j_1}), (p_2, b^{j_2})$ for $j_1 \neq j_2$. The start state and the set of final states are defined in the obvious way. W.l.o.g. we assume that all $Q'_{a,i}$ are pairwise disjoint. The NFA $A'_{a,i}$ can be constructed in PTIME.

We now formally define A. The state set Q_A is the union of all $Q'_{a,i}$. Its start state is s'_r, where r is the start symbol of d and the set of accept states is the union of all $F'_{a,i}$. It remains to define the transition function. For every $b \in \Sigma$, $\delta_A((q_{a,i}, c^j), b) = \delta_{a,i}((q_{a,i}, c^j), b)$, where $q_{a,i} \in Q'_{a,i}$. Finally, $\delta_A((q_{a,i}, c^j), \#) = s'_{c,j}$. It is easy to see that the size of A is no larger than the sum of the sizes of all $A'_{a,i}$. This concludes the proof. $\square$

Minimization of SDTDs

We prove Theorem 16. In order to construct a unique minimal single-type or restrained competition grammar, we use DFAs in DTDs instead of regular expressions. We define what it means for an SDTD to be single-type or restrained competition in this context.

Definition 23. A DFA D with alphabet Σ' is *single-type* if $L(D)$ contains no strings wb^iv and $w'b^jv'$ for $i \neq j$. A DFA D *restrains competition* if $L(D)$ contains no strings wb^iv and wb^jv' for $i \neq j$. A SDTD is *single-type*, resp. *restrained competition* if all DFAs in its DTD are *single-type*, resp. *restrained competition.*

An SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ is *bottom-up deterministic* when for each $a^i, a^j \in \Sigma', i \neq j, L(d(a^i)) \cap L(d(a^j)) = \emptyset$. The *size* of an SDTD $\mathbf{d}$ is $|\Sigma'| + \sum_{a^i \in \Sigma'} |d(a^i)|$, where we denote by $|d(a^i)|$ the number of states of the DFA representing $d(a^i)$.

We show how to construct a minimal SDTD$^{\mathrm{st}}$ from a given SDTD$^{\mathrm{st}}$. We note that the minimization algorithm and uniqueness proof is entirely analogous for restrained competition SDTDs. We merely need to replace *ancestor-based types* in the proof of Lemma 25 by *ancestor-sibling-based types.* Let $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ be an SDTD$^{\mathrm{st}}$. We recall that $L((\mathbf{d}, a^i))$ is the language defined by $\mathbf{d}$, where the start symbol of d is replaced by a^i. The SDTD$^{\mathrm{st}}$ $\mathbf{d}_{\min}$ with $L(\mathbf{d}) = L(\mathbf{d}_{\min})$ is be constructed as follows:

1. *Trim* $\mathbf{d}$, that is, remove all unreachable rules from d, and remove all $a^i \in \Sigma'$ for which $L((d, a^i)) = \emptyset$, and their corresponding rules.

2. Test, for each a^i and a^j in Σ', $i \neq j$, whether $L((\mathbf{d}, a^i)) = L((\mathbf{d}, a^j))$. According to Theorem 15 this is in PTIME. If $L((\mathbf{d}, a^i)) = L((\mathbf{d}, a^j))$, then replace all occurrences of a^j in d by a^i, remove the rule in d that corresponds to a^j, and remove a^j from Σ'.
3. For each $a^i \in \Sigma'$, minimize the DFA representing $d(a^i)$.

Let $\mathbf{d}_{\min} = (\Sigma, \Sigma'_{\min}, d_{\min}, \mu_{\min})$ be the SDTD$^{\text{st}}$ obtained by the above algorithm. It remains to show that $\mathbf{d}_{\min}$ is the minimal SDTD$^{\text{st}}$ for $L(\mathbf{d})$. More formally, we show that

(a) $L(\mathbf{d}_{\min}) = L(\mathbf{d})$; and that
(b) every minimal SDTD$^{\text{st}}$ $\mathbf{d}'$ for $L(\mathbf{d}_{\min})$ is isomorphic to $\mathbf{d}_{\min}$.

The following lemma is easy to show.

Lemma 24. *The SDTD $\mathbf{d}_{min}$ can be computed in* PTIME.

Obviously (a) holds. We proceed with showing (b).

Lemma 25. *Let $\mathbf{d}_1 = (\Sigma, \Sigma'_1, d_1, \mu_1)$ and $\mathbf{d}_2 = (\Sigma, \Sigma'_2, d_2, \mu_2)$ be trimmed, equivalent single-type SDTDs. If there exist trees $t'_1 \in L(d_1)$, $t'_2 \in L(d_2)$ with $\mu_1(t'_1) = \mu_2(t'_2) = t$, and a node u such that $lab^{t'_1}(u) = a^i$ and $lab^{t'_2}(u) = a^j$, then $L((\mathbf{d}_1, a^i)) = L((\mathbf{d}_2, a^j))$.*

Proof. If $|L((\mathbf{d}_1, a^i))| = |L((\mathbf{d}_2, a^j))| = 1$, the proof is trivial. We show that $L((\mathbf{d}_1, a^i)) \subseteq L((\mathbf{d}_2, a^j))$. The other inclusion is analogous. Let $t'_1 \in L(d_1)$, $t'_2 \in L(d_2)$ with that $\mu_1(t'_1) = \mu_2(t'_2) = t$, and u be a node such that $\text{lab}^{t'_1}(u) = a^i$ and $\text{lab}^{t'_2}(u) = a^j$. Towards a contradiction, assume that there exists a $\tau_1 \in L((\mathbf{d}_1, a^i)) - L((\mathbf{d}_2, a^j))$.

Let τ'_1 be the unique typed tree in $L((d_1, a^i))$ with $\mu_1(\tau'_1) = \tau_1$. As $\mathbf{d}_1$ is trimmed, there exists a tree T'_1 in $L(d_1)$, such that τ'_1 is a subtree of T'_1 at some node v. As $\text{lab}^{t'_1}(u) = a^i$ and $\text{lab}^{T'_1}(v) = a^i$, the tree $t[u \leftarrow \tau_1]$ is also in $L(\mathbf{d}_1)$. As $L(\mathbf{d}_1) = L(\mathbf{d}_2)$, $t[u \leftarrow \tau_1] \in L(\mathbf{d}_2)$. As $\mathbf{d}_2$ has ancestor-based types and u has the same ancestor string in t as in $t[u \leftarrow \tau_1]$, u gets the same type a^j in the unique labeling. Therefore, $\tau_1 \in L((\mathbf{d}_2, a^j))$, which leads to the desired contradiction. □

The next lemma says that every minimal SDTD$^{\text{st}}$ has as many types as $\mathbf{d}_{\min}$.

Lemma 26. *Let $\mathbf{d}' = (\Sigma, \Sigma', d, \mu)$ be a minimal SDTDst for $L(\mathbf{d}_{min})$, where $\mathbf{d}_{min} = (\Sigma, \Sigma'_{min}, d_{min}, \mu_{min})$. Then for every $a \in \Sigma$ we have $|\{a^i \in \Sigma' \mid \mu(a^i) = a\}| = |\{a^i \in \Sigma'_{min} \mid \mu_{min}(a^i) = a\}|$.*

Proof. We first show that $|\{a^i \in \Sigma'\}|$ cannot be larger than $|\{a^i \in \Sigma'_{\min}\}|$. Towards a contradiction, assume that $|\{a^i \in \Sigma'\}| > |\{a^i \in \Sigma'_{\min}\}|$. For every $a^i \in \Sigma'$, let t_i be an arbitrary tree so that a^i is a label in the unique $t'_{i,\mathbf{d}'}$ for which $\mu(t'_{i,\mathbf{d}'}) = t_i$. Also, let $t'_{i,\mathbf{d}_{\min}}$ be the unique tree for which $\mu_{\min}(t'_{i,\mathbf{d}_{\min}}) = t_i$. According to the Pigeonhole Principle, there must be two trees $t'_{j,\mathbf{d}'}$ and $t'_{k,\mathbf{d}'}$ so that an a^j-labeled node u in $t'_{j,\mathbf{d}'}$ and an a^k-labeled node v in $t'_{k,\mathbf{d}'}$ are labeled by the same a^ℓ in both $t'_{j,\mathbf{d}_{\min}}$ and $t'_{k,\mathbf{d}_{\min}}$.

From Lemma 25, it now follows that $L((\mathbf{d}', a^j)) = L((\mathbf{d}_{\min}, a^i)) = L((\mathbf{d}', a^k))$. Therefore, renaming all a^k to a^j in $\mathbf{d}'$ results in an equivalent, strictly smaller SDTD$^{\text{st}}$ than $\mathbf{d}'$. Contradiction.

The other direction can be proved analogously, with the roles of $\mathbf{d}'$ and $\mathbf{d}_{\min}$ interchanged. Now the contradiction is that $\mathbf{d}_{\min}$ cannot be the output of the minimization algorithm, as there still exist a^j and a^k in $\mathbf{d}_{\min}$ so that $L((\mathbf{d}_{\min}, a^j)) = L((\mathbf{d}_{\min}, a^k))$. □

We now know that every minimal SDTD$^{\text{st}}$ for $L((\mathbf{d}_{\min}))$ has the same number of types for each alphabet symbol. We now argue that there exists a bijection I between Σ' and $\Sigma'_{\min}$ so that $I(a^i)$ is the unique $a^j \in \Sigma'_{\min}$ for which $L((\mathbf{d}', a^i)) = L((\mathbf{d}_{\min}, a^j))$. In other words, we only need to show that for every $a^i \in \Sigma'$, there exists an $a^j \in \Sigma'_{\min}$ so that $L((\mathbf{d}', a^i)) = L((\mathbf{d}_{\min}, a^j))$. But this immediately follows from Lemma 25. It now follows that for each $a^i \in \Sigma'_{\min}$, we have that $L(d_{\min}(a^i)) = \mathbf{I}^{-1}(L(d(I(a^i))))$ (where we denoted by $\mathbf{I}$ the obvious extension of I to string languages). As minimal DFAs for a given regular language are unique up to isomorphisms, we have the following lemma:

Lemma 27. *Every minimal SDTDst $\mathbf{d}'$ for $L(\mathbf{d}_{min})$ is isomorphic to $\mathbf{d}_{min}$.*

Theorem 16 now follows from Lemma 24 and Lemma 27.

Subtree Based Schemas

Proof of Theorem 19. *For a tree language T the following are equivalent:*

(a) T is definable by an extended restrained competition SDTD;
(b) T is definable by an SDTD with preceding-subtree-based types;
(c) T is regular.

The directions (a) ⇒ (c) and (b) ⇒ (c) are trivial. The proof of the opposite directions uses the fact that regular languages can be validated by deterministic bottom-up automata. (c) ⇒ (a) and (c) ⇒ (b):
Let T be the tree language defined by a bottom-up deterministic tree automaton $B = (Q, \Sigma, \delta, F)$. We can assume that transition functions are represented by regular expressions. We construct an SDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ such that $L(\mathbf{d}) = L(B)$ exactly as in Lemma 22. It is immediate that a tree $t \in L(\mathbf{d}, a^q)$ iff $\delta^*(t) = q$, where $\text{lab}^t(\varepsilon) = a$. Here, δ^* is the canonical extension of δ to trees. As B is deterministic, $L((\mathbf{d}, a^q)) \cap L((\mathbf{d}, a^{q'})) = \emptyset$ for all $a \in \Sigma$ and $q \neq q' \in Q$. Hence, $\mathbf{d}$ is extended restrained competition. By observing that there is only one accepting run for every tree and defining $f(\text{preceding-subtree}^t(u), u) = \delta^*(\text{subtree}^t(u))$, it follows that $\mathbf{d}$ has preceding-subtree-based types. □

Proof of Theorem 20. *It is decidable in* PTIME *for an SDTD* $\mathbf{d}$ *whether it is extended restrained competition.*

Let $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ be an SDTD.

Let E be the set $\{(a^i, a^j) \mid L((\mathbf{d}, a^i)) \cap L((\mathbf{d}, a^j)) \neq \emptyset\}$. This set can be computed in polynomial time by checking whether the non-deterministic tree automata for $L((\mathbf{d}, a^i))$ and $L((\mathbf{d}, a^j))$ have a non-empty intersection [11]. Here, $(\mathbf{d}, a^i)$ denotes the SDTD $\mathbf{d}$ with start symbol a^i.

The algorithm now basically proceeds as in the proof of Theorem 13. In step 3. it additionally has to check that $(a^i, a^j) \in E$. $\square$