Theory Comput Syst (2013) 52:542–585
DOI 10.1007/s00224-012-9428-x

Generating, Sampling and Counting Subclasses of Regular Tree Languages

Timos Antonopoulos · Floris Geerts · Wim Martens · Frank Neven

Published online: 26 October 2012

Abstract To experimentally validate learning and approximation algorithms for XML Schema Definitions (XSDs), we need algorithms to generate uniformly at random a corpus of XSDs as well as a similarity measure to compare how close the generated XSD resembles the target schema. In this paper, we provide the formal foundation for such a testbed. We adopt similarity measures based on counting the number of common and different trees in the two languages, and we develop the necessary machinery for computing them. We use the formalism of extended DTDs (EDTDs) to represent the unranked regular tree languages. In particular, we obtain an efficient algorithm to count the number of trees up to a certain size in an unambiguous EDTD. The latter class of unambiguous EDTDs encompasses the more familiar classes of single-type, restrained competition and bottom-up deterministic EDTDs. The single-type EDTDs correspond precisely to the core of XML Schema, while the others are strictly more expressive. We also show how constraints on the shape of allowed trees

We acknowledge the financial support of 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.
Supported by grant number MA 4938/2–1 from the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).

T. Antonopoulos (✉) · F. Neven
Hasselt University and Transnational University of Limburg, Hasselt, Belgium
e-mail: timos.antonopoulos@uhasselt.be

F. Neven
e-mail: frank.neven@uhasselt.be

F. Geerts
University of Antwerp, Antwerp, Belgium
e-mail: floris.geerts@ua.ac.be

W. Martens
Universität Bayreuth, Bayreuth, Germany
e-mail: wim.martens@uni-bayreuth.de

can be incorporated. As we make use of a translation into a well-known formalism for combinatorial specifications, we get for free a sampling procedure to draw members of any unambiguous EDTD. When dropping the restriction to unambiguous EDTDs, i.e. taking the full class of EDTDs into account, we show that the counting problem becomes #P-complete and provide an approximation algorithm. Finally, we discuss uniform generation of single-type EDTDs, i.e., the formal abstraction of XSDs. To this end, we provide an algorithm to generate k-occurrence automata (k-OAs) uniformly at random and show how this leads to the uniform generation of single-type EDTDs.

Keywords XML schema languages · Counting · Complexity

1 Introduction

XML Schema is the accepted industry standard for the specification of schemas for collections of XML documents. At the same time, it is widely recognized that XML Schema is not a simple language. As it is very unlikely that the World Wide Web Consortium (W3C) will adopt a new schema standard any time soon, several initiatives have been taken to simplify XML Schema. For instance, algorithms have been developed to automatically infer XML Schema Definitions (XSDs) from XML data [8, 10, 11]. We later refer to this setting as the *learning scenario*. Another type of simplification is to let users design a schema in a different, but more user-friendly formalism and then offer the means to automatically convert this schema into an XSD. In general, the latter schema can not be equivalent but, hopefully, constitutes a best approximation in some well-defined way. The latter approach was taken in [19]. We later refer to this setting as the *approximation scenario*. In addition, algorithms to approximate non-deterministic content models by deterministic ones, hereby relieving the user from the Unique Particle Attribution constraint, are studied in [7].

Because it is not always possible to formally prove optimality of the above mentioned types of algorithms, their effectiveness is usually validated by an experimental study using real-world data, for instance using XSDs and corresponding XML corpora found on the web. Unfortunately, as real world data is often only sparsely available, ad-hoc methods are used to generate schemas and corresponding XML corpora. At the same time, a similarity measure is needed that quantifies how closely two unranked regular tree languages resemble each other, and which can be efficiently computed.

The aim of this paper is to provide the machinery to efficiently compute the similarity between two tree languages and to provide algorithms to generate a corpus of XSDs uniformly at random. As usual, we use the abstraction of XSDs as single-type EDTDs [27, 30]. In particular, we consider the following three problems:

(i) **Counting:** Given a tree language $\mathcal{L}$ and $n \in \mathbb{N}$ in unary notation, compute the number of trees in $\mathcal{L}$ of size n;
(ii) **Sampling:** Given a tree language $\mathcal{L}$ and $n \in \mathbb{N}$ in unary notation, generate uniformly at random a tree $t \in \mathcal{L}$ of size n;

(iii) **Generation:** Given a class of tree languages $\mathcal{C}$ and $n \in \mathbb{N}$ in unary notation, generate uniformly at random a member $\mathcal{L} \in \mathcal{C}$ of size n.

We next provide further motivation and describe our contributions for each of these three problems.

Counting and Sampling We start by discussing an approach towards a similarity measure for tree languages. To this end, let $\mathcal{S}$ and $\mathcal{T}$ be two tree languages. In the schema learning case described above, $\mathcal{T}$ can be the target language and $\mathcal{S}$ can be the schema inferred by the learning algorithm under consideration. Or, in the second scenario of schema approximation, $\mathcal{T}$ can be the schema designed by the user and $\mathcal{S}$ is an approximation of $\mathcal{T}$ in a certain (simple) subclass of tree languages. This raises the natural question of how closely $\mathcal{S}$ resembles $\mathcal{T}$. In this paper, we approach this problem by quantifying the number of common and different trees in $\mathcal{S}$ and $\mathcal{T}$. For instance, one possibility is to define the *similarity* of $\mathcal{S}$ and $\mathcal{T}$ as

$$\mathrm{sim}_{\leq n}(\mathcal{S}, \mathcal{T}) := \frac{\sum_{k=0}^{n} |(\mathcal{S} \cap \mathcal{T})^{=k}|}{\sum_{k=0}^{n} |(\mathcal{S} \cup \mathcal{T})^{=k}|},$$

where the set of trees of size k in a language $\mathcal{L}$ is denoted by $\mathcal{L}^{=k}$, and the cardinality of $\mathcal{L}^{=k}$ is denoted by $|\mathcal{L}^{=k}|$. This similarity measure coincides with a measure commonly used when comparing regular *string* languages [7, 8, 10]. Furthermore, this measure has a natural probabilistic interpretation: the similarity between $\mathcal{S}$ and $\mathcal{T}$ is defined as (an approximation) of the expected probability that a tree, chosen uniformly at random from $\mathcal{S} \cup \mathcal{T}$, belongs to $\mathcal{S} \cap \mathcal{T}$. The approximation is realized by restricting attention to trees up to a certain size n. The algorithmic challenge is to efficiently compute $|\mathcal{L}^{=k}|$ for a tree language $\mathcal{L}$.

For string languages, when $\mathcal{L}$ is represented by a deterministic finite automaton, the counting problem reduces to counting the number of accepting paths in a graph; an easy exercise in dynamic programming. However, when $\mathcal{L}$ is represented by an NFA the problem becomes #P-complete [24]. We establish a similar dichotomy for *tree languages*.

Three classes of unranked regular tree languages are of immediate interest to us: single-type, restrained competition, and bottom-up deterministic EDTDs. Whereas single-type EDTDs correspond to the core of XML Schema [27, 30], restrained competition EDTDs correspond to EDTDs that can be correctly typed in a one-pass preorder manner [27]. Both of these classes are deterministic in a top-down sense and are strict subclasses of the unranked regular tree languages. Moreover, the single-type EDTDs are known to be a strict subclass of the restrained-competition EDTDs [27]. The class of bottom-up deterministic EDTDs are deterministic in a bottom-up sense and correspond to the full class of unranked regular tree languages. We observe that while every restrained-competition EDTD is equivalent to a bottom-up deterministic EDTD, there is in general no efficient translation. Indeed, in some cases an exponential size increase can not be avoided.

In fact, we consider the class of *unambiguous* EDTDs in which any tree can have at most one valid typing. We observe that every single-type, restrained-competition

and bottom-up deterministic EDTD is in effect an unambiguous EDTD. As a consequence, it suffices to develop counting and sampling algorithms for unambiguous EDTDs only. Rather than providing an ad-hoc dynamic programming solution to count the number of trees of a certain size in an unambiguous EDTD, we exhibit a mapping from the class of unambiguous EDTDs into a (recursive) *combinatorial specification*. The latter is a formalism defined by Flajolet, Zimmermann and Van Cutsem [18] and provides an elegant way to derive counting and sampling algorithms. We show that in the case of unambiguous EDTDs, these algorithms are also efficient.

In addition, we show how to incorporate *shape constraints* into combinatorial specifications. These are numerical constraints on the depth and width of trees in relation to the total size of the tree. For instance, to avoid string-like trees, we can restrict the depth of a tree to be at most logarithmic in the total number of nodes. In this way, the computation of the similarity of two tree languages can be restricted to trees of a certain shape (which is not necessarily regular).

Finally, when going beyond unambiguous EDTDs, the counting problem becomes intractable. That is, for general EDTDs, we show that computing the number of trees of a certain size is #P-complete. However, we do provide a pseudo-polynomial approximation algorithm based on a similar result for context-free grammars [21].

Generation To assess the average behavior of an algorithm, one can test it on a substantial input set drawn uniformly at random. This approach makes sense when no or little real-world data is available and opens up the possibility to quantify the quality of the obtained results in terms of confidence intervals.

In this paper, we consider the problem of generating XSDs uniformly at random. That is, for each n, every non-isomorphic XSD of size n must be generated with the same probability. This definition is the same as for the random generation of deterministic finite automata [2, 5]. Furthermore, since XSDs can be modelled as top-down DFAs that map states to content models [25, 27], we can extend methods for DFA generation to XSDs.

Unfortunately, current DFA generation methods do not constrain the occurrence of alphabet symbols, a constraint important for XSDs. Indeed, it has been noted in [8] that content models in XSDs contain large alphabets but every alphabet symbol occurs only a small number of times. We have referred to such expressions with alphabet symbol occurrence up to k as k-OREs (k-occurrence regular expressions) and to their automata counterparts as k-OAs (k-occurrence automata). In this paper, we provide an algorithm to generate uniformly at random deterministic k-OAs and show how this leads to uniform XSD generation.

Outline In Sect. 2, we introduce the necessary definitions concerning automata, regular expressions and abstractions of XML schema languages. We study the problem of counting of general EDTDs and unambiguous EDTDs in Sects. 3 and 4, respectively. The sampling problem for EDTDs is considered in Sect. 5. The uniform generation problem for XSDs is discussed in Sect. 6. Finally, Sect. 7 contains related work and the paper is concluded in Sect. 8.

2 Preliminaries

We define regular expressions, automata and XML Schema languages. First, we fix some basic notation.

2.1 String Languages

Strings For any two integers $n, m \in \mathbb{N}$ where $n \leq m$, we denote by $[n, m]$ the set of all the integers j such that $n \leq j \leq m$. A *symbol* is an element of the alphabet Σ and a *string* w is a finite sequence of symbols $\sigma_1 \cdots \sigma_n$ for some $n \in \mathbb{N}$. We assume that the alphabet Σ is finite. We define the *length* of a string $w = \sigma_1 \cdots \sigma_n$, denoted by $|w|$, as n and we also refer to $|w|$ as the *size* of w. The empty string is denoted by ε and is the unique string of size 0. If w_1 and w_2 are two strings, we denote their concatenation by $w_1 \cdot w_2$ or simply by $w_1 w_2$. The set of all strings is denoted by Σ^* and a *string language* is a subset of Σ^*. If L_1 and L_2 are two string languages, then their concatenation is defined as the set $\{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$, and is denoted by $L_1 \cdot L_2$ or simply by $L_1 L_2$. For a string language L and for any $k \in \mathbb{N}$, we denote by $L^{=k}$ the set of strings in L that have length or size k.

Automata A *non-deterministic finite automaton* (NFA) A is a tuple $(\Sigma, Q, I, F, \delta)$, such that Q is a finite set of states, $I \subseteq Q$ is the set of initial states, F is the set of final states, and δ is the transition function of the automaton, defined as $\delta : Q \times \Sigma \to 2^Q$, mapping each pair of a state and symbol to a set of states. A *run* ρ of A on some string $w = a_1 \cdots a_n$ is a sequence of states $q_0, \ldots, q_n$, such that $q_0 \in I$ and, for each $i \in [1, n]$, $q_i \in \delta(q_{i-1}, a_i)$. Furthermore, when q_n is a member of F, we say that the run is *accepting*. The *string language accepted by* A is denoted by $\mathcal{L}(A)$ and is defined as the set of strings w for which there exists an accepting run of A on w. We define the *size* of an automaton A, denoted by $|A|$, as the size of its transition function $|\{(q, a, q') \in Q \times \Sigma \times Q \mid q' \in \delta(q, a)\}|$. An automaton A is *complete* if the transition function maps every state/symbol pair to a non-empty set of states. Finally, a non-deterministic finite automaton A is said to be *deterministic* (or A is a DFA) if I is a singleton set and the transition function maps each state/symbol-pair to a *singleton* set or the empty set.

Regular Expressions The set of *regular expressions* (REs) over Σ is defined recursively as follows. The empty string ε, the empty set $\emptyset$, and every symbol in Σ is a regular expression and if r_1 and r_2 are regular expressions, then so are $r_1 \cdot r_2$, $r_1 + r_2$, r^+ and r^*. The string language defined by a regular expression r, is denoted by $\mathcal{L}(r)$ and is defined as follows. If $r = \varepsilon$ then $\mathcal{L}(r) = \{\varepsilon\}$, if $r = \emptyset$ then $\mathcal{L}(r) = \emptyset$, and if $r = \sigma$ for some $\sigma \in \Sigma$, then $\mathcal{L}(r) = \{\sigma\}$. If $r = r_1 \cdot r_2$ then $\mathcal{L}(r) = \mathcal{L}(r_1)\mathcal{L}(r_2)$, if $r = r_1 + r_2$ then $\mathcal{L}(r) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$, and finally if $r = r_1^+$ then $\mathcal{L}(r) = \{w \mid w = w_1 \cdots w_n$ for some $n \geq 1$ and $\forall i \in [1, n], w_i \in \mathcal{L}(r_1)\}$. For any regular expression r, the regular expression r^* is equivalent to the regular expression $r^+ + \varepsilon$ and $r?$ is used to abbreviate $r + \varepsilon$. We assume w.l.o.g. that $\emptyset$ is not used as a subexpression in any other regular expression. We define the size of a regular expression r, denoted by $|r|$, as the number of symbols in Σ and operators occurring in it.

Formally, $|\emptyset| = |\varepsilon| = |\sigma| = 1$, $|r_1 r_2| = |r_1 + r_2| = |r_1| + |r_2| + 1$, $|r^+| = |r| + 1$. In addition, $|r^*| = |r^+ + \varepsilon|$ and $|r?| = |r + \varepsilon|$. For example, the size of $r = ab^*(b + a^+)$ is 11.

For any regular expression r, we denote by $\bar{r}$ the regular expression obtained from r by replacing, for each i and each $a \in \Sigma$, the i-th occurrence of a by a_i. For example, if $r = ab^*(b + a^+)$, then $\bar{r} = a_1 b_1^*(b_2 + a_2^+)$. The XSD and DTD specifications (to be defined later) restrict regular expressions to be deterministic. A regular expression r is *deterministic* or *1-unambiguous* if there are no strings $w \cdot a_i \cdot v$ and $w \cdot a_j \cdot v'$ in $\mathcal{L}(\bar{r})$ such that $i \neq j$ [15]. We recall that a deterministic regular expression can be translated into an equivalent DFA in quadratic time [13].

2.2 XML Schema Languages

Trees A set of strings S is *prefix closed* if for every string $s \in S$ and any prefix s_p of s, s_p is also in S. A *tree* t over an alphabet Σ is a tuple $(\mathsf{Nodes}, \mathsf{lab}, \Sigma)$ where Nodes, the set of nodes of t, is a finite prefix closed set of strings over the natural numbers, such that if $v \cdot i \in \mathsf{Nodes}$ then $v \cdot i' \in \mathsf{Nodes}$ for all $i' < i$, and $\mathsf{lab} : \mathsf{Nodes} \to \Sigma$ is a labeling function assigning symbols of Σ to each node in Nodes. The size of a tree equals its number of nodes. A node $v \in \mathsf{Nodes}$ is a leaf node if there is no $v' \in \mathsf{Nodes}$ different from v, such that v is a prefix of v'. The root of t is the empty string in Nodes. The children of a node v in t are all nodes $v' \in \mathsf{Nodes}$ such that $v' = v \cdot i$ for $i \in \mathbb{N}$. The subtree of a tree t rooted at a node v of t is the set of nodes with prefix v. For the tree consisting of a single leaf node v labeled with the symbol σ, we write $\sigma(\varepsilon)$, and for any node v labeled with σ and having subtrees $t_1, t_2, \ldots, t_n$ rooted at its children, we write $\sigma(t_1, t_2, \ldots, t_n)$, denoting the subtree of t rooted at v. For a tree t, and a node $v \in \mathsf{Nodes}$ with parent $v' \in \mathsf{Nodes}$, the height of the node v in t, denoted by $\mathsf{height}^t(v)$, is equal to $\mathsf{height}^t(v') + 1$, with the root of t having height 0. The height of a tree t is $\max_{v \in \mathsf{Nodes}}\{\mathsf{height}^t(v)\}$. The width of a node v in a tree t, denoted by $\mathsf{width}^t(v)$, is equal to the number of children of v, and the width of a tree t is $\max_{v \in \mathsf{Nodes}}\{\mathsf{width}^t(v)\}$.

The set of all trees over Σ is denoted by Trees_Σ and a tree language $\mathcal{T}$ over Σ is a subset of Trees_Σ. The set of trees over Σ that have exactly k nodes is denoted by $\mathsf{Trees}_\Sigma^{=k}$, for $k \in \mathbb{N}$. For a tree language $\mathcal{T}$, $\mathcal{T}^{=k}$ denotes the set of trees with k nodes, namely $\mathcal{T}^{=k} = \mathcal{T} \cap \mathsf{Trees}_\Sigma^{=k}$.

DTDs and Extended DTDs A DTD over some finite alphabet Σ is a tuple $D = (\Sigma, R, d, S_d)$ where R is a set of deterministic regular expressions over Σ, d is a function that maps symbols in Σ to expressions in R, and $S_d \subseteq \Sigma$ is the set of start symbols. We refer to the regular expressions in R as the *content models* of the DTD. A finite tree t is *valid with respect to* a DTD D or *satisfies* D, if its root is labeled by an element of S_d and, for every node labeled with some $a \in \Sigma$, the sequence $a_1 \cdots a_n$ of labels of its children, is in the language defined by $d(a)$.

A DTD-DFA (Σ, A, d, S_d) over some finite alphabet Σ is a DTD whose content models are represented by the DFAs in the finite set A, instead of regular expressions.

An extended DTD (EDTD) over a finite alphabet Σ is a tuple $D = (\Sigma, \Delta, R, d, S_d, \mu)$, where Δ is a finite set of types, (Δ, R, d, S_d) is a DTD and μ is a mapping

from Δ to Σ. A tree t is *valid with respect to* an EDTD D or *satisfies* D if $t = \mu(t')$ for some tree t' that satisfies the DTD (Δ, R, d, S_d), where μ is extended to trees. We call t' a witness to t.

An EDTD-DFA $(\Sigma, \Delta, A, d, S_d, \mu)$ over a finite alphabet Σ is an EDTDwhere (Δ, A, d, S_d) is a DTD-DFA.

The tree language consisting of trees that are valid with respect to a DTDor EDTD D is denoted by $\mathcal{L}(D)$. An EDTD D is *reduced* if, for every type τ, there exists a witness tree t' of some tree $t \in \mathcal{L}(D)$ such that the label τ occurs somewhere in t'. Any EDTD can be transformed to an equivalent reduced EDTD in polynomial time [1, 26]. In the following, we assume that all EDTDs are reduced.

Let D be an EDTD $(\Sigma, \Delta, R, d, S_d, \mu)$. Then, for any $\tau \in \Delta$, we denote by D_τ the EDTD $(\Sigma, \Delta, R, d, \{\tau\}, \mu)$. In particular the set of start symbols S_d of D is changed to $\{\tau\}$. We use the same notation for EDTD-DFAs.

Subclasses of EDTDs We recall the following subclasses of EDTDs: *single-type EDTDs*, *restrained competition EDTDs*, and *bottom-up deterministic EDTDs*. Intuitively, these classes have the following significance. Single-type EDTDs are the formal abstraction of XSDs [27] and are therefore central in this paper. The class of restrained competition EDTDs corresponds to the EDTDs that can be correctly typed in a one-pass preorder manner [27]. This means that, when visiting the children of a node from left to right it is clear which type is associated with each node without looking ahead at the nodes to the right. Restrained competition EDTDs form a strict superclass of the single-type EDTDs. Finally, bottom-up deterministic EDTDs are a class of EDTDs that are equally expressive as general EDTDs, i.e., they recognize all regular tree languages. They correspond to bottom-up deterministic tree automata [14].

More formally, let $D = (\Sigma, \Delta, R, d, S_d, \mu)$ be an EDTD.

- D is *single-type* if S_d does not contain two conflicting types and no regular expression in R contains two conflicting types. Here, two types $\tau \neq \tau'$ *conflict* if $\mu(\tau) = \mu(\tau')$.
- D is *restrained competition* if S_d does not contain two conflicting types and all regular expressions in R restrain competition. Here, a regular expression r over Δ *restrains competition* if there are no strings $w\tau v$ and $w\tau' v'$ in $\mathcal{L}(r)$ with $\tau \neq \tau'$ and $\mu(\tau) = \mu(\tau')$.
- D is *bottom-up deterministic*, if for any two distinct types $\tau_1, \tau_2 \in \Delta$, it holds that $\mathcal{L}(d(\tau_1)) \cap \mathcal{L}(d(\tau_2)) = \emptyset$.

These notions are defined analogously for EDTD-DFAs. The class of all single-type (resp., restrained competition, bottom-up deterministic) EDTDs is denoted by EDTD^{st} (resp., EDTD^{rc}, EDTD^{bud}).

Let $D = (\Sigma, \Delta, R, d, S_d, \mu)$ be an EDTD. We define the *size* of D, denoted by $|D|$, as the sum of sizes of the regular expressions $d(\tau) \in R$, for $\tau \in \Delta$. Similarly, the *size* of an EDTD-DFAis the sum of the sizes of the DFAs occurring in it.

We note that translating between EDTD^{st}s and EDTD^{bud}s gives rise to unavoidable exponential blow-ups. The following proposition holds for all formalisms used for representing content models of EDTDs in this paper.

Proposition 2.1 *There is a class $(D_n)_{n\in\mathbb{N}}$ of EDTDsts such that each D_n has size $O(n)$ and the smallest EDTD^{bud} for $\mathcal{L}(D_n)$ has size $2^{\Omega(n)}$. Likewise, there is a class $(D_n)_{n\in\mathbb{N}}$ of EDTD^{bud}s such that each D_n has size $O(n)$ and the smallest EDTDst and EDTD^{rc} for $\mathcal{L}(D_n)$ has size $2^{\Omega(n)}$.*

Proof The class $(D_n)_{n\in\mathbb{N}}$ of EDTDsts defines the unary trees that, when read as a string from root to leaf, obey the regular expression $(a+b)^n a(a+b)^*$. The fact that the minimal EDTD^{bud} for $\mathcal{L}(D_n)$ has size $2^{\Omega(n)}$ immediately follows from the fact that the smallest DFA for $(a+b)^*a(a+b)^n$ has size $2^{\Omega(n)}$ [29]. Informally, when reading a tree from leaf to root, the EDTD^{bud} has to remember, for the last n positions, all positions at which the symbol 'a' appeared. If the EDTD^{bud} uses less than $2^{\Omega(n)}$ types, it can be shown with a fooling argument that the EDTD^{bud} does not recognize the correct language. This proof is analogous to the proof that the smallest DFA for $(a+b)^*a(a+b)^n$ has size $2^{\Omega(n)}$. The direction from EDTD^{bud}s to EDTDsts is analogous. □

To conclude this section, we next provide two examples of EDTDs that will also be used in Sect. 4.

Example 2.2 Consider the EDTDst $D_1 = (\Sigma, \Delta, R, d, S_d, \mu)$ with $\Sigma = \{a\}$, $\Delta = \{\tau_o, \tau_e\}$, $d(\tau_e) = (\tau_o\tau_o)^*$, $d(\tau_o) = \tau_e(\tau_e\tau_e)^*$, $S_d = \{\tau_e\}$ and $\mu(\tau_o) = \mu(\tau_e) = a$. Then D_1 defines trees of even height where each node at even height has an even number of children and each node at odd height has an odd number of children. Here, the root has height 0. Let D_2 be the EDTD $(\Sigma, \Delta, R, d', S_d, \mu)$, where d' is such that $d'(\tau_o) = \tau_e(\tau_e\tau_e)^*$ and $d'(\tau_e) = (\tau_o\tau_o\tau_o\tau_o)^*$. Then, D_2 defines trees where a node at odd height has an odd number of children, but nodes at even height have 0 (mod 4) number of children.

2.3 Unambiguous EDTDs

We next define the class of *unambiguous* EDTDs and show that this class contains the single-type, restrained competition, and bottom-up deterministic EDTDs previously defined.

Definition 2.3 An EDTD or EDTD-DFA D is *unambiguous*, denoted by EDTD^{un}, if every tree $t \in \mathcal{L}(D)$ has a unique witness tree t' with $\mu(t') = t$.

Proposition 2.4 *Let $D = (\Sigma, \Delta, R, d, S_d, \mu)$ be an EDTD. If D is single-type, restrained competition, or bottom-up deterministic then D is unambiguous.*

Proof If D is single-type, then D is also restrained competition. Therefore, let us first assume that D is restrained competition. We prove that D is also unambiguous. Towards a contradiction, assume that D is not unambiguous. Then there are two distinct trees t_1, t_2 over Δ that are witnesses to some tree $t \in \mathcal{L}(D)$ and are such that $\mu(t_1) = \mu(t_2)$. Let v be a node in t such that the type of v in t_1, say τ_1, is different from the type of v in t_2, say τ_2. Moreover, let v be such that none of its ancestors or

left siblings have this property. Notice that v cannot be the root of t since by definition of restrained competition EDTDs, the set of initial types cannot contain conflicting types. Let v' be the parent of v in t, and let its type (which is the same both in t_1 and t_2) be τ'. Then the child-string of v' in t_1 is of the form $w\tau_1 u_1$ and the child-string of v' in t_2 is of the form $w\tau_2 u_2$ by the assumption that v is the leftmost child with the property above. Therefore, the regular language associated with τ' does not restrain competition, which is a contradiction.

Finally, assume that D is bottom-up deterministic. Towards a contradiction, assume D that is not unambiguous. Then there are two distinct trees t_1, t_2 over Δ that are witnesses to some tree $t \in \mathcal{L}(D)$ and are such that $\mu(t_1) = \mu(t_2)$. Notice that t_1, t_2 and t are the same trees with different type-labeling. Let v be a node in t such that the type τ_1 of v in t_1 is different from the type τ_2 of v in t_2, and such that none of v's descendants have this property. Since none of v's descendants have this property, v's child string is the same in t_1 and t_2. Denote this child string by $s \in \Delta^*$. However, this means that $s \in \mathcal{L}(d(\tau_1)) \cap \mathcal{L}(d(\tau_2))$, which contradicts that D is bottom-up deterministic. □

The following result readily follows from the standard product construction of automata (see, e.g., [19]). We add the observation that, if the input EDTDs are EDTD^{un}s, then the product EDTDs for the union and intersection are also EDTD^{un}s.

Proposition 2.5 *Let D_1 and D_2 be two EDTD-DFA^{un}s. Then we can construct, in quadratic time, an EDTD-DFA^{un} for $\mathcal{L}(D_1) \cup \mathcal{L}(D_2)$ and an EDTD-DFA^{un} for $\mathcal{L}(D_1) \cap \mathcal{L}(D_2)$.*

Finally, we recall that deciding whether a given EDTD is in one of the particular classes we use here is in polynomial time.

Proposition 2.6 ([27, 32]) *Deciding whether a given EDTD is a EDTDst, EDTD^{rc}, EDTD^{bud}, or EDTD^{un} is in PTIME.*

Proof The result for EDTDst and EDTD^{rc} is proved in [27]. Testing whether an EDTD is a EDTD^{bud} simply boils down to testing emptiness of finite string automata, which is in PTIME.

We can translate an EDTD to a ranked tree automaton using a variant of the standard first-child next-sibling encoding (in polynomial time), and the resulting ranked tree automaton is unambiguous if and only if the original EDTD is unambiguous. In [32], Seidl has shown that testing whether a ranked tree automaton is unambiguous (i.e., 1-ambiguous in Seidl's terminology) is in PTIME. Therefore, testing whether an EDTD is an EDTD^{un} is also in PTIME. □

3 Counting General Tree Languages

In this section, we consider the counting problem for tree languages $\mathcal{L}(D)$, where D is an EDTD or EDTD-DFA. In general, the counting problem for languages can be stated as follows:

Definition 3.1 For a class of languages $\mathcal{C}$, given a language $C \in \mathcal{C}$ and $m \in \mathbb{N}$ in unary notation, we define $\#\mathcal{C}$ as the problem of finding the number of members in C of size m.

For instance, #DFA reduces to counting the number of paths in a graph, a PTIME process. In contrast, #NFA is known to be #P-complete [24]. In this section, we first show that the counting problem #EDTD is #P-complete. Next, in view of this intractability result, we provide a randomized approximation scheme for #EDTD. We conclude this section by showing that similar results hold for #EDTD-DFA.

3.1 Intractability of #EDTD

We first establish the intractability of #EDTD. More specifically, we show #P-hardness for #EDTD by a reduction from the #NFA problem, which is known to be #P-complete [24]. The matching upper bound is established by providing a #P-algorithm for #EDTD.

Proposition 3.2 *#EDTD is #P-complete.*

Proof Recall that, given $m \in \mathbb{N}$ and NFA N, #NFA is the problem of finding $|\mathcal{L}(N)^{=m}|$. Let $A = (\Sigma, Q, I, F, \delta)$ be any NFA. We define an EDTD D_A such that there is a bijection between words accepted by A and trees accepted by D_A. In addition, the EDTD D_A is derived from A in polynomial time. From this, the #P-hardness of #EDTD then readily follows.

We define D_A to be the EDTD $(\Sigma \uplus \{\sigma_0\}, \Delta, R, d, S_d, \mu)$ as follows. The symbol σ_0 is a distinguished symbol not in Σ. The set of types Δ is $Q \times (\Sigma \uplus \{\sigma_0\})$. Furthermore, for any $\tau \in \Delta$, if $\tau = (q, a)$ then $\mu(\tau) = a$, and d is defined as the function mapping any $\tau = (q, a)$ to the disjunction of the elements of the set $\{(q', a') \mid q' \in \delta(q, a'), a' \in \Sigma\} \cup \{\varepsilon \mid \text{if } q \in F\}$.

Since each $d((q, a))$ is a finite set of elements from Δ, it can be represented by a polynomial-size deterministic regular expression, which is just a disjunction of the elements in $d((q, a))$. We claim that for any string $w = a_1 \cdots a_n \in \Sigma^*$, $w \in \mathcal{L}(A) \Leftrightarrow \sigma_0(a_1(\ldots(a_n(\varepsilon)))) \in \mathcal{L}(D_A)$.

For the *only if* direction, suppose that for some string $w = a_1 \cdots a_n$, $w \in \mathcal{L}(A)$. Then there is a successful run $\rho = q_0, \ldots, q_n$ of A on w. To show that $t = \sigma_0(a_1(\ldots(a_n(\varepsilon))))$ is a tree that satisfies D_A, it suffices to show that there is a tree $t' = \tau(\tau_1(\ldots(\tau_n(\varepsilon))))$ that is a witness to t with $\tau \in S_d$ and such that for all $i \in [1..n]$, $\mu(\tau_i) = a_i$. But $t' = (q_0, \sigma_0)((q_1, a_1)(\ldots((q_n, a_n))))$ is such a tree by definition.

For the *if* direction, suppose that a tree $t = \sigma_0(a_1(\ldots(a_n(\varepsilon))))$ is in $\mathcal{L}(D_A)$. Notice that the letter of the root of t is always labeled by σ_0 by definition of D_A. We want to show that there is an accepting run ρ of A on $w = a_1 \cdots a_n$. Let $t' = (q, \sigma_0)((q_1, a_1)(\ldots((q_n, a_n))))$ be a witness of t. Then the run $\rho = q, q_1, \ldots, q_n$ is an accepting run of A on w.

Note that the above translation from NFAs to EDTDs can be performed in PTIME and therefore #EDTD is #P-hard.

For the upper bound it suffices to observe that deciding whether there is a tree of a given size in the language of an EDTD is in NP. Therefore, the counting problem #EDTD is in #P. □

3.2 Approximating #EDTD

In view of the intractability of #EDTD we next provide a randomized approximation scheme for #EDTD. We first recall the notion of randomized approximation schemes for languages from Gore et al. [21]. A *randomized approximation scheme for languages* is a randomised procedure that takes as input a description for a language $L \subseteq \Sigma^*$ and a tolerance $\varepsilon > 0$, and produces as output a number $\hat{L}$ such that $(1+\varepsilon)^{-1}|L| \leq \hat{L} \leq (1+\varepsilon)|L|$ with probability at least $\frac{3}{4}$. For instance, an approximation scheme exists for context-free grammars (CFGs). We refer to Sect. 3.3 for the definition of CFGs.

Theorem 3.3 ([21]) *There is a randomized approximation scheme for #CFG, i.e., finding the number of elements of size m of a language defined by a given CFG G, with running time* $\varepsilon^{-2}(m|G|)^{O(\log m)}$.

Here, $|G|$ can be taken as the sum of the sizes of the regular expressions that appear in the productions of the CFG G [21]. To obtain a randomized approximation scheme for #EDTD, we proceed as follows. We first establish a translation from EDTDs to CFGs in which the number of trees of a certain size that are accepted by the EDTD are closely related to the number of strings of a certain size that are accepted by the associated CFG. More specifically, we show the following.

Lemma 3.4 *For every EDTD* $D = (\Sigma, \Delta, R, d, S_d, \mu)$ *there is a CFG* $G = (N, \Sigma', R', S')$ *such that for all* $n \in \mathbb{N}$, $|\mathcal{L}(D)^{=n}| = |\mathcal{L}(G)^{=3n}|$.

The approximation scheme for #EDTDs then immediately follows from Theorem 3.3 and Lemma 3.4.

Corollary 3.5 *For an EDTD D, there is a randomized approximation scheme for finding the number of elements of size n of the language* $\mathcal{L}(D)$, *which runs in time* $\varepsilon^{-2}(3n|D|)^{O(\log n)}$.

The next section is dedicated to the translation from EDTDs to CFGs and the proof of Lemma 3.4.

3.3 From EDTDs to CFGs

We first recall the definition of CFGs. A *context-free grammar (CFG) G* is a tuple (N, Σ, R, S) such that N is a finite set of non-terminal symbols, Σ is a set of terminal symbols, the set of derivation rules R is a subset of $N \times (N \cup \Sigma)^*$ and $S \in N$ is the start symbol. We denote the tuples $(V, w) \in R$ by $V \to w$. When $w, w_1, w_2 \in (N \cup \Sigma^*)$ and $V \in N$, we write $w_1 V w_2 \Rightarrow w_1 w w_2$, if $V \to w$ in G. Intuitively, this means that $w_1 w w_2$ can be obtained from $w_1 V w_2$ by applying the derivation rule $V \to w$. We write $w \Rightarrow^k w'$ to abbreviate that there exist $w_1, \ldots, w_{k-1}$ such that $w \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_{k-1} \Rightarrow w'$. For technical reasons, we write sometimes $w \Rightarrow^0 w'$ when $w = w'$. The transitive and reflexive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. The

language accepted by G is the set of strings $w \in \Sigma^*$ such that $S \Rightarrow^* w$, and is denoted by $\mathcal{L}(G)$. If $T \in N$, then $\mathcal{L}(G_T)$ is the set of strings w such that $T \Rightarrow^* w$. A derivation tree of a context-free grammar $G = (N, \Sigma, R, S)$, is a tree over the alphabet $N \cup \Sigma$, whose root is labeled by S and where each non-leaf node is labeled by a non-terminal, each leaf node is labeled by a terminal or ε, and for each node of the tree with label V whose children are labeled $V_1, \ldots, V_n$ respectively from left to right, it holds that $V \to V_1 \ldots V_n$.

In order to translate EDTDs to CFGs, we first need to show how to associate CFGs to REs. Let $\mathcal{RECG}_\Sigma$ denote the class of regular expressions over the alphabet Σ. Let N be an infinite set of non-terminal symbols. Also, let $R_{N,\Sigma}$ denote the class of context-free grammar rules over the set of terminal symbols Σ and the set of non-terminal symbols N. We define the function $\varphi : N \times \mathcal{RECG}_\Sigma \to 2^{(R_{N,\Sigma})}$ to be a partial function mapping pairs of a non-terminal symbol and regular expression to a set of context-free grammar rules.

To this end, we slightly adapt a translation given by Hopcroft et al. [23]. The function is defined inductively as follows:

$$
\begin{array}{ll}
\varphi(R, r) = \emptyset & \text{if } r = \emptyset, \\
\varphi(R, r) = \{R \to \varepsilon\} & \text{if } r = \varepsilon, \\
\varphi(R, r) = \{R \to \sigma\} & \text{if } r = \sigma, \\
\varphi(R, r) = \{R \to R_1, R \to R_2\} \cup \varphi(R_1, r_1) \cup \varphi(R_2, r_2) & \text{if } r = r_1 + r_2, \\
\varphi(R, r) = \{R \to R_1 R_2\} \cup \varphi(R_1, r_1) \cup \varphi(R_2, r_2) & \text{if } r = r_1 \cdot r_2, \\
\varphi(R, r) = \{R \to R R_1, R \to \varepsilon\} \cup \varphi(R_1, r_1) & \text{if } r = r_1^*,
\end{array}
$$

where the non-terminal symbols introduced in the rules above, are not used elsewhere. The following lemma is readily verified.

Lemma 3.6 *Let r be a regular expression over the alphabet Δ. Then we can construct in linear time a CFG G with start symbol S such that $\tau_1 \cdots \tau_n \in \mathcal{L}(r)$ if and only if $S \Rightarrow^* \tau_1 \cdots \tau_n$ and thus $\tau_1 \cdots \tau_n \in \mathcal{L}(G)$.*

Given the translation from REs to CFGs, we next turn to the translation from EDTDs to CFGs. For this, we need to show how CFGs obtained from REs that occur in an EDTD can be combined. Let $D = (\Sigma, \Delta, R, d, S_d, \mu)$ be an EDTD. We use the following notation: if r is a regular expression, then we denote by $\mathrm{CFG}(r, V)$ the set of CFG rules obtained by taking rules of G from Lemma 3.6, replacing the start symbol S by V, and replacing each terminal symbol $\tau \in \Delta$ in the derivation rules by a non-terminal T_τ. In this way, we have that $\tau_1 \cdots \tau_n \in \mathcal{L}(r)$ if and only if $V \Rightarrow^* T_{\tau_1} \cdots T_{\tau_n}$ in $\mathrm{CFG}(r, V)$. We again use $R_{N,\Sigma}$ to refer to the class of context-free grammar rules over the set of terminal symbols Σ and the set of non-terminal symbols N. Let $\psi_D : \Delta \to 2^{(R_{N,\Sigma'})}$, for $\Sigma' = \Sigma \cup \{[,]\}$, be a function mapping types to sets of CFG rules, defined as:

$$\psi_D(\tau) = \big\{T_\tau \to \sigma[R_\tau]\big\} \cup \mathrm{CFG}\big(d(\tau), R_\tau\big),$$

where $\sigma = \mu(\tau)$. In the following, we assume that the non-terminals in the rules $\mathrm{CFG}(d(\tau), R_\tau)$ that are not of the form T_τ, for some $\tau \in \Delta$, are not used elsewhere. This can always be achieved by renaming non-terminals accordingly. Let

Ψ_D be the set $(\bigcup_{\tau\in\Delta}\psi_D(\tau)) \cup \{S \to T_\tau \mid \tau \in S_d\}$. Notice that, for each type τ there exists exactly one rule in Ψ_D whose left-hand side is T_τ. For an EDTD $D = (\Sigma, \Delta, R, d, \{s_d\}, \mu)$ we define $G = (N, \Sigma \cup \{[,]\}, \Psi_D, T_{s_d})$ as its associated context-free grammar. As a first step towards the proof of Lemma 3.4 we establish the following property.

Lemma 3.7 *Let $D = (\Sigma, \Delta, R, d, \{s_d\}, \mu)$ be an EDTD and let $G = (N, \Sigma \cup \{[,]\}, \Psi_D, T_{s_d})$ be its associated context-free grammar. Then, for every $R \in N$, and for every $w \in (\Sigma \cup \{[,]\})^*$ such that $R \Rightarrow^* w$, it holds that $|w| = 3n$ for some $n \in \mathbb{N}$.*

Proof Let D be an EDTD and let G be its associated context-free grammar, as specified above. Furthermore, let R be any non-terminal symbol in N. Let $R \Rightarrow^* w$. We proceed by induction on the number of derivation steps m. Suppose that $m = 1$ and hence that $R \Rightarrow w$. By definition, the only rule in Ψ_D that produces a string of only terminal symbols, is $R \Rightarrow \varepsilon$, and $|\varepsilon| = 0 = 3 \cdot 0$. Suppose then that the statement holds for all $k < K$ derivation steps, for some $K \in \mathbb{N}$, and consider the case where the number of derivation steps is K. Suppose that $R \Rightarrow R_1 R_2$ and therefore there exist w_1, w_2 such that $w = w_1 w_2$ and $R_1 \Rightarrow w_1$ and $R_2 \Rightarrow w_2$. Then by the inductive hypothesis, $|w_1| = 3n_1$ and $|w_2| = 3n_2$ for $n_1, n_2 \in \mathbb{N}$. Therefore, $|w| = 3(n_1 + n_2)$. The argument is similar for the other cases that are in the image of φ given some regular expression r. The only remaining case is where $R \Rightarrow \sigma \cdot [\cdot R' \cdot]$, where $R' \in N$. Therefore, if $R \Rightarrow^K w$, there exists w' such that $w = \sigma \cdot [\cdot w' \cdot]$ and $R' \Rightarrow^{(K-1)} w'$. By the inductive hypothesis, $|w'| = 3n$ for some $n \in \mathbb{N}$ and therefore $|w| = 3n + 3 = 3(n + 1)$. □

We are now ready to prove Lemma 3.4. Recall that we need to show that for every EDTD $D = (\Sigma, \Delta, R, d, S_d, \mu)$ there is a CFG G such that for all $n \in \mathbb{N}$, $|\mathcal{L}(D)^{=n}| = |\mathcal{L}(G)^{=3n}|$. We show that the CFG $G = (N, \Sigma \cup \{[,]\}, \Psi_D, S)$, associated with D, has the desired property. We show this by establishing a bijection $\mathtt{str}$ between the languages $\mathcal{L}(D)$ and $\mathcal{L}(G)$. More precisely, $\mathtt{str} : \mathsf{Trees}_\Sigma \to (\Sigma \cup \{[,]\})^*$ is inductively defined as follows:

$$\begin{aligned} \mathtt{str}(\sigma(\varepsilon)) &= \sigma \cdot [\cdot], \\ \mathtt{str}(\sigma(t_1, \ldots, t_m)) &= \sigma \cdot \big[\cdot \mathtt{str}(t_1) \cdots \mathtt{str}(t_m) \cdot\big]. \end{aligned}$$

We will show by induction on n that, for any $n \in \mathbb{N}$ and any type τ, $\mathtt{str}$ is a bijection between $\mathcal{L}(D_\tau)^{=n}$ and $\mathcal{L}(G_{T_\tau})^{=3n}$. Recall that D_τ and G_{T_τ} denote the EDTD D and the grammar G with start symbols τ and T_τ, respectively. Since the function $\mathtt{str}$ is injective and well-defined, $|\mathcal{L}(D)^{=n}| = |\mathcal{L}(G)^{=3n}|$ holds. In order to show that $\mathtt{str}$ is a bijection it suffices to show:

$\boxed{\Rightarrow}$ For any tree t, if $t \in \mathcal{L}(D_\tau)^{=n}$ then $\mathtt{str}(t) \in \mathcal{L}(G_{T_\tau})^{=3n}$; and
$\boxed{\Leftarrow}$ For any string w, if $w \in \mathcal{L}(G_{T_\tau})^{=3n}$ then $w = \mathtt{str}(t)$ for some tree t such that $t \in \mathcal{L}(D_\tau)^{=n}$.

We show these two directions by induction on n.

Consider the base case when $n = 1$. For the $\boxed{\Rightarrow}$ direction, let t be a tree in $\mathcal{L}(D_\tau)^{=1}$. Then $t = \sigma(\varepsilon)$ for $\sigma = \mu(\tau)$. By definition of $\psi_D(\tau)$, the string $\sigma[]$ is in $\mathcal{L}(G_{T_\tau})^{=3}$ if and only if $\varepsilon \in \mathcal{L}(G_{R_\tau})$. We know that $\varepsilon \in \mathcal{L}(d(\tau))$ and therefore by induction on the structure of $d(\tau)$, Lemma 3.6 implies that $\varepsilon \in \mathcal{L}(G_{R_\tau})$. For the $\boxed{\Leftarrow}$ direction, let τ be a type and w be a string in $\mathcal{L}(G_{T_\tau})^{=3}$. Notice that each rule in G that produces terminal symbols, produces 3 terminal symbols and is of the form $T_\tau \to \sigma[R]$, for some non-terminal symbol R and terminal symbol σ. So suppose that $\sigma[] \in \mathcal{L}(G_{T_\tau})^{=3}$. Then, since the set of rules of G is equal to $\{S \to T_{\tau'} \mid \tau' \in S_d\} \cup \bigcup_{\tau' \in \Delta} \psi_D(\tau')$ with $\psi_D(\tau') = \{T_{\tau'} \to \sigma[R_{\tau'}]\} \cup \mathrm{CFG}(d(\tau'), R_{\tau'})$, where $\sigma = \mu(\tau')$, it holds by induction on the structure of $d(\tau')$ and Lemma 3.6, that $\varepsilon \in \mathcal{L}(d(\tau))$ and therefore $\sigma(\varepsilon) \in \mathcal{L}(D_\tau)$.

Consider next the general case. That is, suppose that both directions hold for all $k < n$ for some $n \in \mathbb{N}$. We want to show that the directions also hold for n. For the $\boxed{\Rightarrow}$ direction, suppose that, for some tree t and type τ, $t \in \mathcal{L}(D_\tau)^{=n}$. Then $t = \sigma_0(t_1, \ldots, t_m)$ for some m such that for each $i \in [1, m]$, $t_i \in \mathcal{L}(D_{\tau_i})^{=n_i}$, where $n_i < n$, $\sum_{i=1}^m n_i = n - 1$, and where $\tau_1 \ldots \tau_m \in d(\tau)$. By the inductive hypothesis, $\mathtt{str}(t_i) \in \mathcal{L}(G_{T_{\tau_i}})^{=3n_i}$ for each $i \in [1, m]$. Notice that $n = 1 + \sum_{i=0}^m n_i$ and therefore $3n = 3 + \sum_{i=0}^m 3n_i$. By definition of $\psi_D(\tau) = \{T_\tau \to \sigma[R_\tau]\} \cup \mathrm{CFG}(d(\tau), R_\tau)$, it suffices to show that $\mathtt{str}(t_1) \cdots \mathtt{str}(t_m) \in \mathcal{L}(G_{R_\tau})$. From the assumption above, that for all $i \in [1, m]$, $\mathtt{str}(t_i) \in \mathcal{L}(G_{T_{\tau_i}})^{=3n_i}$, it is enough to show that $R_\tau \Rightarrow^* T_{\tau_1} \ldots T_{\tau_m}$ by the rules in G. However, this is true by construction of G and by Lemma 3.6.

For the $\boxed{\Leftarrow}$ direction, suppose that for some type τ and string w, $w \in \mathcal{L}(G_{T_\tau})^{=3n}$. Notice that the set of rules of G is equal to $\{S \to T_{\tau'} \mid \tau' \in S_d\} \cup \bigcup_{\tau' \in \Delta} \psi_D(\tau')$ and $\psi_D(\tau) = \{T_\tau \to \sigma[R_\tau]\} \cup \mathrm{CFG}(d(\tau), R_\tau)$ where $\sigma = \mu(\tau)$, and furthermore, there is a unique rule whose left-hand side is T_τ. Therefore, it holds that $w \in \mathcal{L}(G_{T_\tau})^{=3n}$ if and only if there exists string w' such that $w = \sigma \cdot [\cdot w' \cdot]$ and $w' \in \mathcal{L}(G_{R_\tau})^{=3(n-1)}$. Furthermore, $w' \in \mathcal{L}(G_{R_\tau})^{=3(n-1)}$ if and only if $R_\tau \Rightarrow^* T_{\tau_1} \ldots T_{\tau_m}$ for some m, and $w' = w_1 \ldots w_m$ such that for each $i \in [1, m]$, $w_i \in \mathcal{L}(G_{T_{\tau_i}})^{=n_i}$, where $\sum_{i=1}^m n_i = 3(n-1)$. Notice that for each $i \in [1, m]$, $n_i = 3n_i'$ for some n_i' by Lemma 3.7, and therefore $\sum_{i=1}^m n_i' = n - 1$. By the inductive hypothesis, there exist trees $t_1, \ldots, t_m$ such that for all $i \in [1, m]$, $w_i = \mathtt{str}(t_i)$ and $t_i \in \mathcal{L}(D_{\tau_i})^{=n_i'}$. Furthermore, by induction on the structure of $d(\tau)$ and Lemma 3.6, $R_\tau \Rightarrow^* T_{\tau_1} \ldots T_{\tau_m}$ if and only if $\tau_1 \ldots \tau_m \in \mathcal{L}(d(\tau))$. Therefore, $t = \sigma(t_1, \ldots, t_m) \in \mathcal{L}(D_\tau)$ and in addition $t \in \mathcal{L}(D_\tau)^{=n}$. This concludes the proof of Lemma 3.4.

3.4 #EDTD-DFA

We next turn our attention to #EDTD-DFA. Along the same lines as the proof of Proposition 3.2, it is readily verified that #EDTD-DFA is #P-complete. Furthermore, similarly to #EDTD, the counting problem #EDTD-DFA admits a randomized approximation scheme. This is verified in precisely the same way as for #EDTD. More specifically, it suffices to show that one can translate EDTD-DFAs into CFGs such that Lemmas 3.7 and 3.4 hold. Since EDTD-DFAs use DFAs rather than REs to represent regular languages, we only need to describe how DFAs can be translated into CFGs. Let $A = (\Sigma, Q, q_0, F, \delta)$ be a DFA. Then, a context-free grammar G^A such that $L(G^A) = L(A)$ can be constructed as follows. The grammar

$G^A = (N_A, \Sigma, R_A, S_A)$ is defined with $N_A = S_A \uplus \{R_q \mid q \in Q\}$ and the set of rules R_A which is defined as the union of all rules of the form

- $S_A \to R_{q_0}$, for the initial state q_0;
- $R_q \to \varepsilon$ for every $q \in F$; and
- $R_{q_1} \to a R_{q_2}$ for every $a \in \Sigma$ and $q_2 \in Q$ such that $q_2 \in \delta(q_1, a)$.

Notice that this is a standard translation from automata to a (left-linear) context-free grammar. It is easily verified that, with this translation, Lemma 3.6 holds when considering DFAs instead of regular expressions. Furthermore, we can associate a CFG to an EDTD-DFA in the same way as before and only a minor modification of the proof of Lemma 3.7 is required. Indeed, in the inductive step in that proof we need to consider a production of the form $R \Rightarrow T_\tau R'$. Again, by induction, we have that there exist w_1, w_2 such that $w = w_1 w_2$ and $T_\tau \Rightarrow w_1$ and $R' \Rightarrow w_2$ and by the inductive hypothesis, $|w_1| = 3n_1$ and $|w_2| = 3n_2$ for $n_1, n_2 \in \mathbb{N}$. Hence, Lemma 3.7 holds for EDTD-DFAs as well. The proof of Lemma 3.4 for EDTD-DFAs is analogous to its EDTD counterpart. We may thus conclude:

Corollary 3.8 *For an EDTD-DFA D, there is a randomized approximation scheme for finding the number of elements of size n of the language $\mathcal{L}(D)$, which runs in time $\varepsilon^{-2}(3n|D|)^{O(\log n)}$.*

4 Counting Unambiguous Tree Languages

In this section we show that counting unambiguous tree languages is in PTIME. More specifically, we provide an efficient algorithm for #EDTD-DFA^{un}. Given an EDTD-DFA^{un} D, the algorithm (1) translates D into an unambiguous CFG G_D similar to the translation given in Sect. 3.3; (2) translates G_D into a so-called combinatorial specification [18]; and (3) leverages the available PTIME counting algorithm for the corresponding combinatorial class [18]. We further consider two applications of the translation into a combinatorial specification: the counting of trees that adhere to some shape constraints; and the efficient computation of the similarity between two unambiguous tree languages. As we will see in the next section, the combinatorial specification has as additional advantage that one obtains a sampling procedure for trees in $\mathcal{L}(D)$. Before describing the algorithm for #EDTD-DFA^{un} in more detail, we define the notion of combinatorial specification (see [18] for more details). Since single-type-, restrained competition-, and bottom-up deterministic EDTD-DFAs are EDTD-DFA^{un}s, the results presented here thus apply to those classes as well.

4.1 Combinatorial Specifications

A *combinatorial class* is a finite or denumerable set on which a size function is defined, satisfying the following two conditions:

(i) the size of an element is a non-negative integer,
(ii) the number of elements of any given size is finite.

If $\mathcal{A}$ is a class, the size of an element $a \in \mathcal{A}$ is denoted by $|a|$. The set of objects in $\mathcal{A}$ of size n is denoted by $\mathcal{A}_n$. The *counting sequence* of a combinatorial class is the sequence of integers $(A_n)_{n\geq 0}$ where $A_n = |\mathcal{A}_n|$ is the number of objects in class $\mathcal{A}$ that have size n. Two combinatorial classes $\mathcal{A}$ and $\mathcal{B}$ are said to be *combinatorially isomorphic*, written $\mathcal{A} \cong \mathcal{B}$ if and only if their counting sequences are identical. This condition is equivalent to the existence of a bijection from $\mathcal{A}$ to $\mathcal{B}$ that preserves size.

A calculus for combinatorial classes introduced in [18], is presented below. Here, $\mathcal{E}$ and $\mathcal{Z}$ are atoms that denote the classes containing exactly one object of size 0 and size 1 respectively. (We remark that $\mathcal{E}$ is denoted as **1** in [18].) In the following, we allow different instantiations $\mathcal{Z}_a, \mathcal{Z}_b, \ldots$ of the same atom $\mathcal{Z}$. Let $\mathcal{B}$ and $\mathcal{F}$ be combinatorial classes. Then the combinatorial class $\mathcal{A} = \mathcal{B} + \mathcal{F}$ is the disjoint union of the classes $\mathcal{B}$ and $\mathcal{F}$. In particular, $\mathcal{Z} + \mathcal{Z}$ contains two objects of size 1. Furthermore, $\mathcal{A} = \mathcal{B} \times \mathcal{F}$ denotes the combinatorial class $\{\alpha = (\beta, \gamma) \mid \beta \in \mathcal{B}, \gamma \in \mathcal{F}\}$ and for each $\alpha = (\beta, \gamma) \in \mathcal{A}$, the size of α is the sum of the sizes of β and γ. More generally, if $\alpha = (\beta_1, \ldots, \beta_n) \in \mathcal{A}$ then the size of α is the sum of the sizes of β_i for $i \in [1, n]$. For all n the following hold when $\mathcal{A}$ is a combinatorial class:

$$\text{if } \mathcal{A} = \mathcal{B} + \mathcal{F} \quad \text{then } |A_n| = |B_n| + |F_n|,$$

$$\text{if } \mathcal{A} = \mathcal{B} \times \mathcal{F} \quad \text{then } |A_n| = \sum_{k=0}^{n} |B_{n-k}| \cdot |F_k|$$

We assume an infinite set of variables $C, C_0, C_1, \ldots$. Each variable will define a combinatorial class $\mathcal{L}(C)$.

Definition 4.1 ([18]) A *specification* for $(C_1, \ldots, C_n)$ is a collection of n equations, with the i-th equation being of the form

$$C_i := \Psi_i(C_1, \ldots, C_n)$$

where $\Psi_i(C_1, \ldots, C_n)$ is a term built from $\mathcal{E}$, $\mathcal{Z}$ and the C_j, using the constructors $+$ and $\times$. For each $j \in [1, n]$, let $C_j^0 = \emptyset$ and for each $i \in \mathbb{N}$, let $C_j^{i+1} = \Psi_j(C_1^i, \ldots, C_n^i)$. Then $\mathcal{L}(C_j)$ is defined to be $\bigcup_{i\geq 0} \mathcal{L}(C_j^i)$. For $k \in \mathbb{N}$, we denote by $\mathcal{L}(C_j)^{=k}$ the objects in $\mathcal{L}(C_j)$ of size k.

We say that a specification is in *normal form* if each equation is either a single atom, or a single operation $C_i := C_j + C_k$ or $C_i := C_j \times C_k$.

Theorem 4.2 ([18]) *Given a specification for $(C_1, \ldots, C_n)$ in normal form and an integer k, the counting sequence up to size k can be computed in $O(nk^2)$ arithmetic operations.*

As an example of specifications, we consider the class of unambiguous CFGs. Recall that a context-free grammar $G = (N, \Sigma, R, S)$ is *unambiguous* if, for every string $w \in \mathcal{L}(G)$, w has exactly one derivation tree for G. A CFG G is in Chomsky normal form if every $V \to w$ in R is either of the form $V \to \varepsilon$, $V \to \sigma$ for $\sigma \in \Sigma$, or $V \to V_1 \cdot V_2$ with V_1 and V_2 in N. It is well-known [18] that an unambiguous CFG

G with n non-terminals in Chomsky normal form can be translated into a linear-size combinatorial specification $(C_1, \ldots, C_n)$ in normal form by simply replacing concatenation $(\cdot)$ by $\times$, disjunction $(\cup)$ of rules with the same left hand side by $+$, ε by $\mathcal{E}$, and finally by replacing each $\sigma \in \Sigma$ by $\mathcal{Z}_\sigma$. The following lemma is readily verified.

Lemma 4.3 ([18]) *Let G be an unambiguous CFG in Chomsky normal form with n non-terminals and let $(C_1, \ldots, C_n)$ be the corresponding combinatorial specification. Then, $|\mathcal{L}(C_1)^{=k}| = |\mathcal{L}(G)^{=k}|$ for all $k \in \mathbb{N}$, where C_1 corresponds to the start symbol $S \in N$. Furthermore, $|\mathcal{L}(G)^{=k}|$ can be computed by using $O(nk^2)$ arithmetic operations.*

4.2 #EDTD-DFA^{un}

The previous lemma tells that a PTIME algorithm for #EDTD-DFA^{un} can be obtained by providing a polynomial time computable translation from EDTD-DFA^{un}s to unambiguous CFGs in Chomsky normal form. We first show that the translation from EDTD-DFAs to CFG given in Sect. 3.3 preserves unambiguity. More specifically, given an EDTD-DFA^{un} $D = (\Sigma, \Delta, A, d, S_d, \mu)$, we show that its associated CFG $G_D = (N, \Sigma \cup \{[,]\}, \Psi_D, S)$ is unambiguous (Lemma 4.5). Secondly, we verify that the number of non-terminals of the Chomsky normal form G'_D of G_D depends in a polynomial way on the size of G_D (Lemma 4.6). More specifically, we show that G'_D has $O(|\Delta||Q_{\max}|)$ non-terminals where $Q_{\max}$ denotes the largest set of states of an automaton in A. From this, together with Lemma 3.4 and 4.3 we obtain that $\mathcal{L}(D)^{=k}$ can be computed using $O(|\Delta||Q_{\max}|(3k)^2) = O(|\Delta||Q_{\max}|k^2)$ arithmetic operations, from which the tractability of #EDTD-DFA^{un} follows. The following theorem summarizes the main result of this section.

Theorem 4.4 *For an EDTD-DFA^{un} $D = (\Sigma, \Delta, A, d, S_d, \mu)$, the sequence of numbers of trees in $\mathcal{L}(D)$ of size m, for all $m \leq k$ can be computed using $O(|\Delta||Q_{\max}|k^2)$ arithmetic operations, where $Q_{\max}$ denotes the largest set of states of an automaton in A.*

It remains to verify Lemma 4.5 and 4.6

Lemma 4.5 *If $D = (\Sigma, \Delta, A, d, S_d, \mu)$ is an EDTD-DFA^{un}, then its associated context-free grammar $G = (N, \Sigma \cup \{[,]\}, \Psi_D, S)$ is unambiguous.*

Proof Let w be a string in $\mathcal{L}(G)$ and let $\Sigma' = \Sigma \uplus \{[,]\}$. We want to show that there exists a unique derivation tree for w. We show the equivalent statement that each w has a unique *left derivation*, i.e., a derivation in which at each step the leftmost non-terminal is replaced.

We show by induction on n that, for any $B \in N \cup \Sigma'$ and any string $w \in (N \cup \Sigma')^*$, $B \Rightarrow^n w$ implies that there is a unique left derivation for w from B. Let $n = 0$. Then $w = \sigma$ for some $\sigma \in \Sigma$ or $w = \varepsilon$. In both cases, $B = w$ and therefore there is a unique left derivation for w. Suppose then that the statement is true for all $k < K$ for some $K \in \mathbb{N}$, and let $B \Rightarrow^K w$.

Suppose first that B is equal to R_{q_τ} for some $\tau \in \Delta$ and $q_\tau \in A_\tau$, for $A_\tau \in A$. There are two cases: either $w = \varepsilon$ or $w = w_1 \cdot w_2$ for some $w_1 \in \Sigma'^+$. In the first case, q_τ is a final state of A_τ and there is only one possible left derivation for $w = \varepsilon$ from B. In the second case, suppose there are two distinct sequences $T_{\tau_1}, \ldots, T_{\tau_m}$, and $T_{\tau'_1}, \ldots, T_{\tau'_{m'}}$, and two sequences $w_1, \ldots, w_m \in \Sigma'^*$ and $w'_1, \ldots, w'_{m'} \in \Sigma'^*$ with $w = w_1 \cdots w_m = w'_1 \cdots w'_{m'}$, such that

$$T_{\tau_i} \quad \Rightarrow^{\ell_i} \quad w_i, \quad \text{and} \quad T_{\tau'_j} \quad \Rightarrow^{\ell'_j} \quad w'_j,$$

with $\ell_i, \ell'_j < K$ for all $i \in [1, m]$, $j \in [1, m']$, and furthermore

$$R_{q_\tau} \quad \Rightarrow \quad T_{\tau_1} \cdots T_{\tau_m} \quad \text{and} \quad R_{q_\tau} \quad \Rightarrow \quad T_{\tau'_1} \cdots T_{\tau'_{m'}}.$$

Now, by definition of the bijection $\mathtt{str}$, for every type $\tau \in \Delta$ and every word w, if $T_\tau \Rightarrow w$, then $w = \sigma \cdot [\cdot w' \cdot]$, where $w' \in \Sigma'^*$ is a well-nested string in terms of the symbols '[' and ']'. Therefore there is a unique way to split the word w into subwords to which there is a derivation from some $T_{\tau''}$, $\tau'' \in \Delta$. This means that $m = m'$ and $w_i = w'_i$, for all $i \in [1, m]$.

If the sequences $T_{\tau_1}, \ldots, T_{\tau_m}$ and $T_{\tau'_1}, \ldots, T_{\tau'_m}$ are distinct, we have that $\tau_j \neq \tau'_j$ for some $j \in [1, m]$. By definition of the bijection $\mathtt{str}$, and since $T_{\tau_j} \Rightarrow w_j$ and $T_{\tau'_j} \Rightarrow w_j$, there is a tree $t \in \mathcal{L}(D_{\tau_j}) \cap \mathcal{L}(D_{\tau'_j})$. But then there are two distinct trees t_1 and t_2 that are witnesses to t being in $\mathcal{L}(D_{\tau_j})$ and $\mathcal{L}(D_{\tau'_j})$ respectively. These trees t_1 and t_2, together with the trees $s_i = \mathtt{str}^{-1}(w_i)$ for $i \neq j$, can then be used to construct two distinct witnesses for some tree $t' \in \mathcal{L}(D)$, which is a contradiction since D is unambiguous.

Hence there is a unique such sequence $T_{\tau_1}, \ldots, T_{\tau_m}$. Since for all $i \in [1, m]$, $T_{\tau_i} \Rightarrow^{\ell_i} w_i$ with $\ell_i < K$, there is a unique left derivation of w_i from T_{τ_i} and hence there is also a unique left derivation from R_{q_τ} to w.

Suppose then that B is equal to T_τ for some τ. For each τ there is a unique rule $T_\tau \to \sigma[R_{A_\tau}]$ with T_τ at the left hand side and, therefore, w is of the form $\sigma \cdot [\cdot w' \cdot]$, where $R_{A_\tau} \Rightarrow^{(K-1)} w'$. By the inductive hypothesis, w' has a unique left derivation from R_{A_τ} and therefore so does w from $B = T_\tau$.

Let B be equal to R_{A_τ} for some $\tau \in \Delta$ and $A_\tau \in A$. Since D is a EDTD-DFA^{un}, A_τ is a DFA for each $\tau \in \Delta$ and only has a single initial state q_{init}. Therefore, there exists a unique $q_{\text{init}} \in A_\tau$ such that $R_{A_\tau} \to R_{q_{\text{init}}}$. Therefore, $R_{q_{\text{init}}} \Rightarrow^{(K-1)} w$ and, by the inductive hypothesis, w has a unique left derivation from $R_{q_{\text{init}}}$ and therefore also from R_{A_τ}.

Finally, let B be equal to the start symbol S. Towards a contradiction, assume that there exist two types $\tau_1, \tau_2 \in S_d \subseteq \Delta$ such that $S \to T_{\tau_1}$, $S \to T_{\tau_2}$, $T_{\tau_1} \Rightarrow^{K-1} w$, and $T_{\tau_2} \Rightarrow^{K-1} w$. Then, by definition of the bijection $\mathtt{str}$, there exist a tree t such that $\mathtt{str}(t) = w$, $t \in \mathcal{L}(D_{\tau_1})$, and $t \in \mathcal{L}(D_{\tau_1})$. But since $\tau_1, \tau_2 \in S_d$, this means that $t \in \mathcal{L}(D)$ and there are two witness trees t_1, t_2 for t that differ at least at their root, with t_1 having τ_1 as a root label and t_2 having τ_2 as a root label.

Therefore, there are no two distinct types $\tau_1, \tau_2 \in \Delta$ such that $\mu(\tau_1) = \mu(\tau_2)$ and $\tau_1, \tau_2 \in S_d$. Suppose that $w = \sigma \cdot w'$. Then, as we proved above, there exists a unique type $\tau \in S_d$ such that $T_\tau \Rightarrow \sigma[S_{A_\tau}]$. In particular, $T_\tau \Rightarrow^{(K-1)} w$ and therefore there

is a unique left derivation of w from T_τ by the inductive hypothesis. Hence w has a unique left derivation from S. □

Lemma 4.6 *Let $G_D = (N, \Sigma \cup \{[,]\}, \Psi_D, S)$ be the unambiguous CFG associated with an EDTD-DFA^{un} D. Then the Chomsky normal form G'_D of G_D has at most $O(|\Delta||Q_{\max}|)$ non-terminals.*

Proof We first observe that the number of non-terminals $|N|$ of G_D is $O(|\Delta||Q_{\max}|)$, where $Q_{\max}$ is the largest set of states in any automaton in A. Consider the different types of rules in G_D. We remind the reader that $\Psi_D = \bigcup_{\tau\in\Delta} \psi_D(\tau) \cup \{S \to T_\tau \mid \tau \in S_d\}$, where $\psi_D(\tau) = \{T_\tau \to \sigma[R_\tau]\} \cup \mathrm{CFG}(d(\tau), R_\tau)$.

We now argue how G_D can be brought into a normal form G'_D in which every right hand side of a production either consists of (i) ε, (ii) a single terminal, or (iii) two non-terminals. Furthermore, by analyzing the increase in the number of non-terminals incurred by the normalization process, we obtain the desired bound on the non-terminals in G'_D.

Notice that production rules whose right hand side consists of a single *non-terminal*, such as $R_\tau \to R_q$ and $S \to T_\tau$, can be eliminated by a standard size-reduction algorithm [23]. For rules of the form $T_\tau \to \sigma[R_\tau]$, we add the non-terminals $N_[$ and $N_]$ with rules $N_[\to [$ and $N_] \to]$, and for each $\sigma \in \Sigma$ we add the non-terminal N_σ and the rule $N_\sigma \to \sigma$. For each rule of the form $T_\tau \to \sigma[R_\tau]$, we finally add a non-terminal $N_{\sigma\cdot[}$ and $N_{R_\tau\cdot]}$, with the respective rules $N_{\sigma\cdot[} \to N_\sigma \cdot N_[$ and $N_{R_\tau\cdot]} \to R_\tau \cdot N_]$. We then replace each rule $T_\tau \to \sigma[R_\tau]$ with $T_\tau \to N_{\sigma\cdot[} \cdot N_{R_\tau\cdot]}$. Since for each $\tau \in \Delta$ there is only one rule with T_τ as a left hand side, we add in total $2 + |\Sigma| + 2 \cdot |\Delta| < 2 + |\Delta| + 2 \cdot |\Delta|$ non-terminals for the rules of this form. The remaining forms of rules, namely rules of the form $R_{q_1} \to T_{a_i} \cdot R_{q_2}$ and $R_q \to \varepsilon$, are already in Chomsky normal form and thus need no further processing.

As a consequence, G'_D has $O(|\Delta||Q_{\max}| + 2 + |\Delta| + 2 \cdot |\Delta|) = O(|\Delta||Q_{\max}|)$ non-terminals. □

We stress that the size of the numbers $|\mathcal{L}(D)^{=k}|$ can grow very fast. To implement the algorithm underlying Theorem 4.4, a mathematical software package is needed. Actually, Maple provides an implementation for combinatorial specifications in the combstruct module.[1] We implemented our specification for the EDTD-DFA^{un} D_1 given in Example 2.2. As an illustration, we computed the number of trees of size 1001 valid with respect to D_1 and obtained a number with 314 decimals:

```
51879502371239317320511752369544517561698193655988404231585212 14
81908948889495358432656815934343950208100024435828682335203876 50
92543737284388062928765258453029470320709909346697782409585624 32
23188522684389654317803723666450135945868706080790349000020103 71
20152303965795554922650323287553303269884549851688819208474
```

The computation remains under the 60 seconds on a 1.8 GHz iMac with 1 GB of RAM.

[1] http://www.maplesoft.com/support/help/Maple/view.aspx?path=combstruct.

For $\tau \in \Delta$, $(\delta, w) \in \mathbb{N}^2$, $\delta \geq 1$:

$$T_\tau^{(\leq\delta,\leq w)} := \mathcal{Z}_{\mu(\tau)} \times \mathcal{Z}_[\times R_{\mathsf{init}(d(\tau))}^{(\leq\delta-1,\leq w)} \times \mathcal{Z}_]$$

For $q \in Q_{\tau'}$, $(\delta, w) \in \mathbb{N}^2$, $\delta \geq 1$:

$$R_q^{(\leq\delta,\leq w)} := \sum_{\substack{\tau\in\Delta, q'\in Q \\ q'\in\delta_{\tau'}(q,\tau)}} \left(T_\tau^{(\leq\delta,\leq \mathsf{w})} \times R_{q'}^{(\leq\delta,\leq w-1)}\right) \underbrace{+\ \mathcal{E}}_{\text{iff } q\in F_{\tau'}}$$

For $q \in Q_{\tau'}$, $w \in \mathbb{N}^2$:

$$R_q^{(\leq 0,\leq w)} := \mathcal{E} \quad \text{iff} \quad q \in F_{\tau'}$$

Fig. 1 Combinatorial specification of CFG G_D In the presence of shape constraints

4.3 Shape Constraints

As a first application of the translation of EDTD-DFA^{un}s into combinatorial specifications, we consider the counting problem of languages in the presence of shape constraints. Given an EDTD-DFA^{un} $D = (\Sigma, \Delta, A, d, S_d, \mu)$, it is often desirable to count the number of trees in $\mathcal{L}(D)$ that satisfy certain *shape constraints*. Here, by shape constraints we mean certain restrictions on the allowed combinations of the size, depth and/or width of trees in the language. More formally, a shape constraint on the depth (δ) (resp. branching width (w)) of trees consists of a function $\phi_\delta(k)$ (resp. $\phi_w(k)$) that assigns to each tree of size k its maximal allowed depth (resp. branching width). For instance, to avoid string-like trees one can take $\phi_\delta(k) = \log k$; to only consider binary trees one simply lets $\phi_w(k) = 2$. As previously described, the counting sequences of trees in $\mathcal{L}(D)$ can be computed using the combinatorial specification corresponding to the CFG G_D. In the presence of shape constraints, we need to augment this specification with parameters corresponding to the depth and width of objects.

We next describe in detail the specification for G_D: Let $D = (\Sigma, \Delta, A, d, S_d, \mu)$ be an EDTD-DFA^{un}. Let $A = \{A_\tau \mid \tau \in \Delta\}$ such that, for each $\tau \in \Delta$, $A_\tau = (\Delta, Q_\tau, \delta_\tau, q_{\tau,0}, F_\tau)$ is the DFA such that $d(\tau) = A_\tau$ and let $Q_\tau = \{q_{\tau,0}, \ldots, q_{\tau,m_\tau}\}$. We assume w.l.o.g. that the Q_τ are pairwise disjoint and also disjoint with Δ. Also, let $\mathsf{init} : \{A_\tau \mid \tau \in \Delta\} \to \{q_{\tau,0} \mid \tau \in \Delta\}$ be the function mapping each automaton to its initial state. Finally, we let $Q = \bigcup_{\tau\in\Delta} Q_\tau$. Given the maximal tree depth d and width w, the specification is defined over the set of variables $\mathbf{Var}(\mathsf{d}, \mathsf{w}) = \{T_\tau^{(\leq\delta,\leq w)}, R_q^{(\leq\delta,\leq w)}, \mathcal{Z}_{\mu(\tau)} \mid \tau \in \Delta, q \in Q, \delta \in [1, \mathsf{d}], w \in [0, \mathsf{w}]\}$ and equations shown in Fig. 1. This specification is obtained as described in the previous section, by translating D into G_D, followed by the translation of G_D into a combinatorial specification. Note that Fig. 1 represents G_D before its normalization in Chomsky normal form.

We denote by $\mathcal{L}(D)^{(=k,\leq\mathsf{d},\leq\mathsf{w})}$ the set of trees of size k, maximal depth d and maximal width w. A straightforward generalization of Lemma 3.4 then shows that $|\mathcal{L}(D)^{(=k,\leq\mathsf{d},\leq\mathsf{w})}| = |(\mathcal{L}(\sum_{\tau\in S_d} T_\tau^{(\leq\mathsf{d},\leq\mathsf{w})}))^{=3k}|$. Furthermore, it is easily verified that

the normalization of the specification $\sum_{\tau\in S_d} T_\tau^{(\leq \mathsf{d},\leq \mathsf{w})}$ contains $O(|\Delta||Q_{\max}|\mathsf{dw})$ non-terminals. Hence, similarly to Theorem 4.4 we obtain:

Corollary 4.7 *For an EDTD-DFA^{un} $D=(\Sigma,\Delta,A,d,S_d,\mu)$, size k, depth d and width w, the cardinality of $\mathcal{L}(D)^{(=k,\leq \mathsf{d},\leq \mathsf{w})}$ can be computed using $O(|\Delta||Q_{\max}|\mathsf{dw}k^2)$ arithmetic operations, where $Q_{\max}$ denotes the largest state space of an automaton in A.*

We remark that when shape constraints $\phi_\delta(k)$ and $\phi_w(k)$ are provided, $|\mathcal{L}(D)^{(=k,\phi_\delta(k),\phi_w(k))}|$ is easily obtained from Corollary 4.7. Moreover, when only $\phi_\delta(k)$ or $\phi_w(k)$ is provided one simply removes the w or δ parameter, respectively, from the above specification and the complexity is adjusted correspondingly. Finally, we observe that when no shape constraint is specified, the specification reduces to the one for G_D.

4.4 Similarity Measure

The translation of EDTD-DFA^{un}s into combinatorial specifications further allows for the computation of the similarity between two tree languages as defined in the introduction. More specifically, for tree languages S and T, define,

$$\mathrm{sim}_{\leq n}(S,T) := \frac{\sum_{k=0}^{n}|(S\cap T)^{=k}|}{\sum_{k=0}^{n}|(S\cup T)^{=k}|},$$

where $\frac{0}{0}$ is taken to be 1.

Then following result is then readily verified:

Proposition 4.8 *Assume S and T are specified as unambiguous EDTD-DFAs. Then for any n, $\mathrm{sim}_{\leq n}(S,T)$ can be computed using $O(|\Delta||Q_{\max,S}||Q_{\max,T}|n^2)$ arithmetic operations, where $Q_{\max,S}$ and $Q_{\max,T}$ denote the largest state space of an automaton in S and T, respectively.*

Proof Since $\mathrm{sim}_{\leq n}(S,T)$ requires both $|(S\cap T)^{=k}|$ and $|(S\cup T)^{=k}|$, for $k\in[0,n]$, it suffices to bound the operations needed to compute these quantities. By Proposition 2.5, EDTD^{un}s can be computed for $S\cap T$ and $S\cup T$ in quadratic time. Hence, all $|(S\cap T)^{=k}|$ for $k\in[0,n]$ can be computed from the specification of $S\cap T$ using $O(|\Delta||Q_{\max,S}||Q_{\max,T}|n^2)$ operations, where $Q_{\max,S}$ and $Q_{\max,T}$ denote the largest state space of an automaton in S and T, respectively. Indeed, this follows from Theorem 4.4 and the fact that the automata in $S\cap T$ consist of product automata of S and T. Due to trees common to S and T, we cannot use $S\cup T$. Instead, we simply use $|S^{=k}|+|T^{=k}|-|(S\cap T)^{=k}|$ for the counting sequence of the union of S and T. From Theorem 4.4 it follows again that these quantities can be computed up to $k=n$ using $O(|\Delta_S||Q_{\max,S}|n^2)$, $O(|\Delta_T||Q_{\max,T}|n^2)$ and $O(|\Delta||Q_{\max,S}||Q_{\max,T}|n^2)$ operations, respectively. As a consequence, $\mathrm{sim}_{\leq n}(S,T)$ requires $O(|\Delta||Q_{\max,S}||Q_{\max,T}|n^2)$ operations. □

To illustrate feasibility, we used our implementation in Maple to compute $\mathrm{sim}_{\leq 100}(D_1, D_2) = 2.405906249 \cdot 10^{-7}$ taking D_1 and D_2 as defined in Example 2.2. The score was computed in as little as a few seconds.

5 Sampling Tree Languages

We next turn to the problem of sampling trees of a certain size in a tree language $\mathcal{L}$ uniformly at random.

Definition 5.1 For a class of languages $\mathcal{C}$, we say that $\mathcal{C}$ admits uniform sampling if for every $C \in \mathcal{C}$ and $m \in \mathbb{N}$, there is an algorithm that generates each element $t \in \mathcal{L}(C)$ of size m with probability $1/|\mathcal{L}(C)^{=m}|$. We say that $\mathcal{C}$ admits tractable uniform sampling if the sampling algorithm runs in PTIME in the size of C and m, where m is given in unary notation.

In this section we observe that EDTD-DFA^{un}s admit tractable uniform sampling by leveraging the translation from EDTD-DFA^{un}s into combinatorial specifications as described in the previous section. More specifically, we have shown in that section that an EDTD-DFA^{un} D can be translated into a specification $(C_1, \ldots, C_n)$ such that $|\mathcal{L}(D)^{=m}| = |\mathcal{L}(C_1)^{=3m}|$ for all $m \in \mathbb{N}$. Here, n is a parameter that depends on the number of non-terminals in the unambiguous CFG in Chomsky normal form used in the translation. Clearly, if we can generate each object c in $\mathcal{L}(C_1)^{=3m}$ with equal probability $1/|\mathcal{L}(C_1)^{=3m}|$, then we can also generate each tree $t \in \mathcal{L}(D)^{=m}$ with probability $1/|\mathcal{L}(D)^{=m}|$. Indeed, we simply need to convert $c \in \mathcal{L}(C_1)$ into a tree $t \in \mathcal{L}(D)$. We note that this conversion is implicit in the translation given in the previous section. The following general result for sampling a combinatorial class then provides the necessary machinery to conclude that EDTD-DFA^{un}s admit tractable uniform sampling.

Theorem 5.2 ([18]) *Any combinatorial specification for $(C_1, \ldots, C_n)$ in normal form has a random generation routine for objects of size k, that uses precomputed tables of size $O(nk)$ and achieves $O(nk \log k)$ worst case time complexity. The computation of the tables requires $O(nk^2)$ operations.*

A sampling procedure for combinatorial specifications is built by combining sampling procedures from smaller components in the specification and heavily relies on the computation of the number of objects in a class of a certain size. For instance, when $\mathcal{C} = \mathcal{A} + \mathcal{B}$, then an object of size k is drawn uniformly at random from $\mathcal{C}$ by drawing an object from $\mathcal{A}$ with probability A_k/C_k and an object from $\mathcal{B}$ with probability B_k/C_k. A more detailed description can be found in [18]. Again, as numbers can grow very fast, a Mathematical Software package is needed. Luckily, Maple provides an implementation of the just sketched sampling technique for a combinatorial specification in its `combstruct` module. So, to write a uniform generator for any class of objects, only a parser needs to be written which translates that class into a combinatorial specification.

6 Uniform XSD Generation

We next turn our attention to the generation of languages in some given class of tree languages. In general, the generation problem for classes of languages can be stated as follows:

Definition 6.1 For a class of languages $\mathcal{C}$, let $s : \mathcal{C} \to \mathbb{N}$ be a function that assigns to each language $\mathcal{L} \in \mathcal{C}$ its size $s(\mathcal{L})$. We say that $\mathcal{C}$ admits uniform generation if for every $m \in \mathbb{N}$, there is an algorithm that generates each language $\mathcal{L} \in \mathcal{C}$ of size m with equal probability $1/p$, where p denotes the number of languages in $\mathcal{L}' \in \mathcal{C}$ with $s(\mathcal{L}') = m$.

Observe that in contrast to the previous section, we here consider the generation of languages rather than trees in the language.

For example, Almeida et al. show that the class of languages defined by connected complete DFAs admits uniform generation [2]. This is even shown to hold when relaxing the completeness assumption by Bassino et al. [4]. In both cases, $s(\mathcal{L})$ is given in terms of the number n of states of the DFA and the size k of the alphabet. More specifically, we define the size of the language $\mathcal{L}$ specified by a DFA with n states and k alphabet symbols as $s(\mathcal{L}) = 2^n 3^k$. Given a language $\mathcal{L}$ of size $s(\mathcal{L}) = m$, there is a unique pair n and k such that $m = 2^n 3^k$. We can thus, without ambiguity, measure the size of a language $\mathcal{L}$ in terms of n and k, or in terms of $s(\mathcal{L})$.

In this section, we consider the problem of generating uniformly at random languages defined by XSDs of a given size. In order to define the size of an XSD one must choose a representation of XSDs. Here, a natural choice is to regard XSDs as single-typed EDTDs [27, 30]. However, little is known on how to uniformly generate the regular expressions present in the EDTDs. In view of the existing generation algorithms for DFAs [2, 4], it will thus be more convenient to represent XSDs entirely by means of DFAs.

We do this as follows: First, we use the representation of XSDs as DFA-based XSDs [25]. In this representation, part of the structure is specified by a DFA whereas the so-called context models are specified by regular expressions. We do not lose generality since it is known that single-typed EDTDs have equivalent expressive power as DFA-based XSDs. Indeed, they can be translated back and forth in linear time [20, 25].[2] Second, we represent the regular expressions in the content models of the DFA-based XSD by means of DFAs as well. However, as it was noted in [9, 10], regular expressions in real-world XSDs can have large alphabets but each of these alphabet symbols typically occurs only a small number of times. The automata counterpart of such expressions will be formalized below as k-occurrence automata (k-OA). To

[2]The translation algorithm in Lemma 7 of [20] only claims quadratic time, but it uses a different definition of single-type EDTDs. In the definition there, the set of types in an EDTD is always of the form $\{a^i \mid a \in \Sigma, i \in \Delta\}$ and, for each such type, $\mu(a_i) = a$. For the definition we use in this paper, the translation can be easily adapted to run in linear time.

summarize, we will represent XSDs by DFA-based XSDs with k-OAs as content models.

This section is organized as follows: We first define DFA-based XSD with k-OAs content models in Sect. 6.1. An algorithm to generate DFA-based XSD with k-OAs uniformly at random is given in Sect. 6.2. In the same section, we also show that XSDs admit uniform generation. Section 6.3 concludes by providing algorithms for counting and generating k-OAs; these are needed when generating XSDs.

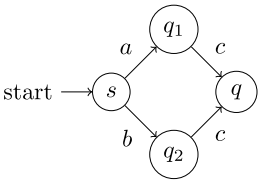
6.1 DFA-Based XSD with k-OAs Content Models

We start by defining DFA-based XSDs with general content models.

Definition 6.2 Let $\mathcal{C}$ be a class of languages over alphabet Σ. A *DFA-based XSD with content model* $\mathcal{C}$ is a tuple $D = (\Sigma, A, \lambda, S)$, where S is the set of start symbols and $A = (\Sigma, \Delta, q_0, F, \delta)$ is a DFA with set of states Δ over the alphabet Σ, λ is a function mapping each non-initial state q of A to some language $C \in \mathcal{C}$, and $F = \emptyset$. A tree t is valid with respect to D if its root is labeled with a symbol from S and, for every node v of t, the sequence $a_1 \cdots a_n$ of labels of its children is in the language $\mathcal{L}(\lambda(q))$, where $q = \delta(q_0, a'_1 \cdots a'_m)$ where $a'_1 \cdots a'_m$ is the sequence of labels of the nodes on the path from the root to v, including the label of the root and the label of v.

When context models are given by regular expressions, it is known that the corresponding DFA-based XSDs equally expressive as XSDs (single-typed EDTDs) since they can be translated back and forth in linear time [20, 25]. Note that in Definition 6.2, the DFA A is possibly incomplete. Furthermore, since no final states are present, the standard way of completing an incomplete DFA by adding a new final sink state cannot be applied. We can, however, assume that the DFA is connected since states that cannot be reached from the initial state can be disregarded. In the following, we assume A to be connected. Recall that an NFA $A = (\Sigma, Q, I, F, \delta)$ is connected if for every state $q \in Q$, there is a string w and a run $\rho = q_0, \ldots, q_n$ of A on w, such that $q = q_i$ for some $i \in [0, n]$. We also observe that given a $D = (\Sigma, A, \lambda, S)$, we may assume that for every non-initial state q in the DFA A, the language $\lambda(q)$ is defined over the alphabet consisting of all $\sigma \in \Sigma$ for which $\delta(q, \sigma)$ is defined, and if q has no outgoing transitions, then $\lambda(q)$ should be either the trivial language $\{\varepsilon\}$ or the empty language. If this holds, we say that the content models in D are redundancy-free. It is readily verified that every D can be converted into a D' with redundancy-free content models such that $\mathcal{L}(D) = \mathcal{L}(D')$. In the following, we assume D to have redundancy-free content models. We next illustrate the notion of DFA-based XSDs with REs as content models by means of the following example.

Example 6.3 Consider $D = (\Sigma, A, \lambda, S)$ with $\Sigma = \{a, b, c, d\}$ and $S = \{a, b\}$, where A is a DFA (without final states) as depicted below:



Furthermore, we define $\lambda(q_1) = c$, $\lambda(q_2) = cc$, and $\lambda(q) = \varepsilon$. This DFA-based XSD accepts two trees, namely $a(c)$ and $b(c, c)$. Note that A is connected but incomplete, and that D has redundancy-free content models.

As mentioned above, in content models of XSDs that occur in practice, alphabets can be large but each alphabet symbol occurs only a small number of times. Thereto, Bex et al. [9] introduced the notion of a k-occurrence regular expression (or k-*ORE*). These are regular expressions in which every alphabet symbol occurs at most k times. For instance, $a \cdot (a + b)^*$ is a 2-ORE. As we explained previously, little is known about the uniform generation of regular expressions. Therefore, we will not consider k-OREs as such but instead we turn to the corresponding (but slightly larger) class of k-occurrence automata (k-OAs) as defined next. Such automata are state labeled and at most k states can be labeled by the same alphabet symbol. As every k-ORE can be defined by a corresponding k-OA (although not vice versa [9]), we choose to represent content models by k-OAs rather than by unconstrained DFAs.

We formalize this notion by means of k-occurrence automata which are defined as follows:

Definition 6.4 A *k-occurrence automaton* A (k-OA) over Σ is a tuple $(V, E, I, F, \mathsf{lab}, e)$ where V is a finite set of states, $E \subseteq (V \times V)$ is the edge relation, $I \subseteq V$ is the initial set of states, $F \subseteq V$ is the set of final states, $\mathsf{lab} : V \to \Sigma$ is the labeling function, and e is a Boolean which is true when A accepts the empty string. We have the additional requirement that every Σ-symbol labels at most k states.

Since k-OAs are defined as state-labeled automata (rather than in the usual edge-labeled way), we need to redefine how k-OAs define languages. Given a word $w = w_1 \cdots w_n \in \Sigma^+$, and a k-OA A, a run of A on w is an assignment ρ of nodes of A to positions of the word, such that for all $i \in [1, n]$, $\mathsf{lab}(\rho(w_i)) = w_i$, for all $i \in [1, n-1]$, $(\rho(w_i), \rho(w_{i+1})) \in E$ and $\rho(w_1) \in I$. The run is accepting when $\rho(w_n) \in F$. The language defined by a k-OA A, denoted by $\mathcal{L}(A)$, is the set of words w on which there is an accepting run of A together with the word ε exactly when $e = 1$.

In addition, in order to be able to generate k-OAs we will require them to be *admissible*, i.e., deterministic, connected and complete. A k-OA A is *deterministic* if for every state $s \in V$ and $\sigma \in \Sigma$ there is at most one state $s' \in V$ such that $(s, s') \in E$ and $\mathsf{lab}(s') = \sigma$, and there are no two distinct states $s, s' \in I$ with the same label. A k-OA A is *complete* if for every state $s \in V$ and every $\sigma \in \Sigma$, there exists a state $s' \in V$ such that $\mathsf{lab}(s') = \sigma$ and $(s, s') \in E$, and for every label σ there is a state $s \in I$ with $\mathsf{lab}(s) = \sigma$. Finally, a k-OA A is *connected*, if every state s is reachable from an initial state in I.

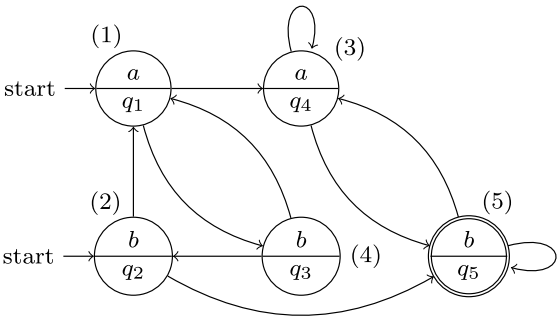


Fig. 2 An example of an admissible k-occurrence automaton (k-OA)

Example 6.5 Consider the k-OA $A = (V, E, I, F, \mathsf{lab}, e)$ shown in Fig. 2, where $V = \{q_1, q_2, q_3, q_4, q_5\}$, $I = \{q_1, q_2\}$, $F = \{q_5\}$, and E and lab are as shown in the figure, and e is true, which is not represented in the figure. It is readily verified that A is deterministic, connected and complete and thus admissible.

We are now ready to define the representation of XSDs that will be used in the uniform generation algorithm.

Definition 6.6 A *DFA-based k-OA XSD* $D = (\Sigma, A, \lambda, S)$ is a DFA-based XSD with k-OAs as content model. The size of the language $\mathcal{L}(D)$ is given by $s(\mathcal{L}(D)) = 2^n 3^\ell$, where n is the number of states of the DFA A and ℓ is the size of the alphabet Σ.

One may wonder why $s(\mathcal{L}(D))$ does not explicitly depend on the number of states and alphabet sizes of the component k-OAs. First of all, since we consider DFA-based XSDs with redundancy-free content models, we can bound the size of the alphabets of the k-OAs as follows:

Property 6.7 *Let $D = (\Sigma, A, \lambda, S)$ be a DFA-based k-OA XSD, and let q be state of the DFA A. Then, the size of the alphabet of the k-OA $\lambda(q)$ is equal to the number of outgoing transitions of state q in A.*

Second, given the size of alphabet for a k-OA and assuming that the k-OA is admissible, we can also bound its number of states:

Property 6.8 *Let $B = (V, E, I, F, \mathsf{lab}, e)$ be a k-OA over an alphabet Σ of size ℓ. Then the number of states for B is at most $k \cdot \ell$. Furthermore, if B is admissible then the number of states of B is at least ℓ.*

Proof Let B be a k-OA over Σ of size ℓ. Since each symbol in B can label at most k states, there can be at most $k \cdot \ell$ states. If B is admissible, then for each symbol in Σ, there is an initial state labeled by it. Therefore there are at least ℓ states. □

Finally, in order to generate languages of a certain size that are defined by DFA-based k-OA XSDs, we will need to consider minimal DFA-based k-OA XSDs (see Sect. 6.2 for more details of the generation algorithm).

To this end, we define the *state size* of a DFA-based k-OA XSD $D = (\Sigma, A, \lambda, S)$ to be if $|A| + \sum_{q \in \Delta} |\lambda(q)|$, where Δ are the states in A, $|A|$ denotes the number of states of A and $|\lambda(q)|$ denotes the number of states of the k-OA $\lambda(q)$. A DFA-based k-OA XSD D is *minimal* if there does not exist a DFA-based k-OA XSD D' such that $\mathcal{L}(D) = \mathcal{L}(D')$ for which the state size of D' is smaller than the state size of D.

We will show below that a minimal DFA-based k-OA XSD is unique, up to isomorphism. Here, two DFA-based k-OA XSDs $D_1 = (\Sigma, A_1, \lambda_1, S_1)$ and $D_2 = (\Sigma, A_2, \lambda_2, S_2)$ are said to be isomorphic if there is an isomorphism $\iota : A_1 \to A_2$ such that for all states q in A_1, the k-OAs $\lambda_1(q)$ and $\lambda_2(\iota(q))$ are isomorphic. Two k-OAs $B_1 = (V_1, E_1, I_1, F_1, \mathsf{lab}_1, e_1)$ and $B_2 = (V_2, E_2, I_2, F_2, \mathsf{lab}_2, e_2)$ are isomorphic, if there exists a bijective function $\beta : V_1 \to V_2$ such that for all $v_1, v_2 \in V_1$, $(v_1, v_2) \in E_1$ if and only if $(\beta(v_1), \beta(v_2)) \in E_2$, $v_1 \in I_1$ if and only if $\beta(v_1) \in I_2$, $v_2 \in F_1$ if and only if $\beta(v_2) \in F_2$, $\mathsf{lab}_1(v_1) = \mathsf{lab}_2(\beta(v_1))$ and $e_1 = e_2$.

The following proposition will be crucial in showing that XSDs admit uniform generation.

Proposition 6.9 *Let $D = (\Sigma, A, \lambda, S)$ be a DFA-based k-OA XSD. Then,*

1. *one can check in PTIME whether or not D is minimal*; *and*
2. *if D is minimal, then D is unique up to isomorphism.*

Proof It is proved in [28] that there exists an algorithm that, for a given single-type EDTD-DFA, computes the unique minimal equivalent one. Given a single-type EDTD-DFA $D = (\Sigma, \Delta, A, d, S_d, \mu)$, this minimization algorithm performs the following three steps:

1. Transform D to an equivalent *reduced* single-type EDTD-DFA D by removing types that do not occur in any witness tree.
2. Test, for each $(\tau_1, \tau_2) \in \Delta^2$, whether $\mathcal{L}(D_{\tau_1}) = \mathcal{L}(D_{\tau_2})$. If so, replace all occurrences of τ_2 by τ_1 in the definition of D and remove τ_2 from Δ.
3. For each remaining $\tau \in \Delta$, minimize the DFA A_τ.

To see how this algorithm can be used for DFA-based k-OA XSDs, we make several observations. The first is that each DFA-based k-OA XSD $D = (\Sigma, A, \lambda, S)$ with $A = (\Sigma, \Delta, q_0, \emptyset, \delta)$ can be translated into an equivalent single-type EDTD $\varphi(D) = (\Sigma, \Delta', B, d, S_d, \mu)$ as follows:

- $\Delta' = \{(a, q) \in \Sigma \times \Delta \mid \delta(p, a) = q \text{ in } A\}$;
- $S_d = \{(a, q) \mid \delta(q_0, a) = q \text{ in } A\}$;
- $\mu((a, q)) = a$ for every $(a, q) \in \Delta'$;
- For each $(a, q) \in \Delta'$, d defines $B_{(a,q)}$ to be the automaton recognizing the language $\{(a_1, q_1) \cdots (a_n, q_n) \in \Delta^* \mid a_1 \cdots a_n \in \mathcal{L}(\lambda(q))$ and, for each a_i, $\delta(q, a_i) = q_i$ in $A\}$.

We note that this translation is essentially the one of Lemma 7 in [20]. Notice that this translation defines a one-one correspondence between transitions in A and types in Δ'. Furthermore, for each (a, q), the automaton $B_{(a,q)}$ can be defined to behave exactly the same as $\lambda(q)$, since the Δ-symbols q_i are determined by q and a_i. In other words, there exists a k-OA for $\lambda(q)$ if and only if there exists a k-OA for $\mathcal{L}(B_{(a,q)})$.

Algorithm 1 generate-k-OA-XSD

Require: n, ℓ
Ensure: DFA-based k-OA XSD (Σ, A, λ, S)
1: $A =$ generate-DFA(n, ℓ);
2: **for** every state $j \in [2, n]$ of A **do**
3: $\quad \ell_j = |\{\sigma \mid \delta_A(j, \sigma) \text{ is defined in } A\}|$
4: **end for**
5: **for** every state $j \in [2, n]$ of A **do**
6: $\quad \max = \sum_{i=\ell_j}^{k \cdot \ell_j} k$-OA-uniform-count$(i, \ell_j)$
7: $\quad$ choose $n_j \in [\ell_j, k \cdot \ell_j]$ with probability k-OA-uniform-count$(n_j, \ell_j)/\max$
8: $\quad \lambda(j) =$k$-$OA-uniform-generation(n_j, ℓ_j);
9: **end for**
10: $S = \{a \mid \delta_A(q_0, a) \neq \emptyset$, where q_0 is the initial state of $A\}$

This implies that transforming a DFA-based k-OA XSD into a reduced one can be done by the same algorithm as for single-type EDTD-DFAs. Furthermore, also the second step of the above algorithm can be performed on DFA-based k-OAs in the same manner than for EDTD-DFAs.

The third step of the algorithm can also be performed directly on DFA-based k-OA XSDs, since deterministic k-OAs can be minimized by a simple adaptation of the standard DFA minimization algorithms. Since this minimization algorithm only merges states and transitions, the resulting automaton remains a k-OA. Finally, due to the above mentioned translation and due to the uniqueness up to isomorphism of minimal single-type EDTD-DFAs [28], we also obtain that the minimal DFA-based k-OA XSD for a given language is unique. □

6.2 Generating XSDs Uniformly at Random

In this section we show that XSDs admit uniform generation. More specifically, we first provide an algorithm that, given n, k and ℓ, generates uniformly at random DFA-based k-OA XSDs of size n over alphabets of size ℓ. Second, we use rejection sampling to filter out DFA-based k-OA XSDs that are not minimal. Indeed, in view of Proposition 6.9 tree languages can be uniquely specified by means of minimal DFA-based k-OA XSDs. All combined, this provides a way of generating languages defined by XSDs.

Generating DFA-Based k-OA XSDs We describe an algorithm, generate-k-OA-XSD(n, ℓ), that generates, uniformly at random, non-isomorphic DFA-based k-OA XSDs over an alphabet of size ℓ, in which (i) the inner DFA is connected and consists of n states; and (ii) all of its k-OAs are admissible. Algorithm 1 provides the pseudocode for the algorithm.

The algorithm generate-k-OA-XSD takes as input parameters n and ℓ, and first generates the inner DFA A with n states over the alphabet $[0, \ell - 1]$ uniformly at random. It does so by calling the procedure generate-DFA(n, ℓ) (line 1) from Bassino et al. [4]. This method is particularly well suited since it produces possibly incomplete connected DFAs, as needed for DFA-based k-OA XDSs. In addition, only non-

isomorphic DFAs are generated. We assume that the states of A are numbered from 1 to n and its initial state is 1.

Then, for each $j \in [2, n]$, i.e., for each state of A except the initial state, we compute the alphabet size ℓ_j of the k-OA that will be associated to state j. Property 6.7 tells that in a DFA-based k-OA XSD, ℓ_j is equal to the number of outgoing transitions of state j (line 3).

Next, generate-k-OA-XSD maps each $j \in [2, n]$ to a k-OA $\lambda(j)$ over an alphabet of size ℓ_j (lines 6, 7, 8). For this, the algorithm relies on two procedures, k-OA-uniform-count(m, κ) and k-OA-uniform-generation(m, κ), that count and generate all non-isomorphic k-OAs that have m states and have alphabet Σ of size κ, respectively. These procedures will be described in detail in Sect. 6.3.

Let $j \in [2, n]$ and let ℓ_j be the size of the alphabet. To generate a k-OA $\lambda(j)$ we need to uniformly range over the sizes of the k-OAs of alphabets of size ℓ_j (lines 6, 7). Indeed, k-OA-uniform-generation requires both the number of states of the k-OA and the size of the alphabet. By Property 6.8, the possible number of states of k-OAs over alphabets of size ℓ_j, ranges between ℓ_j and $k \cdot \ell_j$. Consequently, the probability that a k-OA has $n_j \in [\ell_j, k \cdot \ell_j]$ states is given by ratio of the number of k-OAs with n states over the total number of k-OAs whose number of states lie in $[\ell_j, k \cdot \ell_j]$. These ratios are computed by means of k-OA-uniform-count, followed by a corresponding draw of n_j from $[\ell_j, k \cdot \ell_j]$ with the appropriate probability (lines 6, 7). Given n_j and ℓ_j, k-OA-uniform-generation(n_j, ℓ_j) then generates a k-OA with n_j states and over an alphabet of size ℓ_j, uniformly at random (line 8).

Finally, generate-k-OA-XSD instantiates the initial states of the generated DFA-based k-OA XSD (line 10).

We next show the correctness of the algorithm generate-k-OA-XSD:

Theorem 6.10 *Algorithm* generate-k-OA-XSD *generates, given n and ℓ, uniformly at random non-isomorphic DFA-based k-OA XSDs with n states and alphabet size ℓ.*

Proof Since the procedures generate-DFA and k-OA-uniform-generation generate DFAs and k-OAs uniformly at random, and since the algorithm selects the number of states in the k-OAs uniformly at random, we may conclude that generate-k-OA-XSD generates DFA-based k-OA XSDs uniformly at random as well. It remains to be shown that only non-isomorphic DFA-based k-OA XSDs are generated. However, since both generate-DFA and k-OA-uniform-generation generate non-isomorphic DFAs and k-OAs, the only way two isomorphic DFA-based k-OA XSDs can be generated, with different representation, is if there is a non-trivial automorphism on the representation of the DFA mapping a state q to a state q' and where in one representation q is associated to the k-OA B and q' to the k-OA B', and where in the other representation, q is associated to the k-OA B' and q' to the k-OA B. But, since no two states can be reached from a single state using a transition labeled by the same symbol, there can be no non-trivial automorphism on a DFA. □

Generating Languages Defined by XSDs We next show that XSDs admit uniform generation. Note that generate-k-OA-XSD does not necessarily generate minimal DFA-based k-OA XSDs. Indeed, two DFA-based k-OA XSDs A_1 and A_2 can be

generated that are equivalent, that is, they define the same tree language. To uniquely generate languages $\mathcal{L}$ defined by DFA-based k-OA XSDs uniformly at random, we need to only retain the minimal DFA-based k-OA XSDs generated by generate-k-OA-XSD. Indeed, Proposition 6.9 tells that minimal DFA-based k-OA XSDs are unique, up to isomorphism. Furthermore, checking the minimality of DFA-based k-OA XSDs can be done in PTIME.

We therefore use generate-k-OA-XSD in combination with a rejection stage that checks if the resulting automaton is minimal, and keep generating such automata until a minimal one is found. In view of Theorem 6.10, each minimal DFA-based k-OA XSD will be generated once, as generate-k-OA-XSD only produces non-isomorphic DFA-based k-OA XSDs, and hence may thus conclude that XSDs admit uniform generation.

Observe that the efficiency of the generation procedure is prone to optimization. Indeed, one could incorporate a minimization step for the k-OAs in generate-k-OA-XSD, hereby reducing the number DFA-based k-OA XSDs that will be generated and tested for minimality.

One may wonder whether a similar uniform generation procedure is also possible for some of the more expressive subclasses of EDTDs that we considered in the paper. We note that, already for EDTD-DFA^{bud}s, we believe that such a generation procedure is likely to be more difficult. The two underlying reasons why we believe this to be the case are that (i) in contrast to DFA-based k-OA XSDs, state-minimal EDTD-DFA^{bud}s are no longer unique for a given language [28]; and (ii) testing whether a EDTD-DFA^{bud} is minimal is coNP-complete [12, 28]. As such, the computational complexity of the above mentioned rejection stage is likely to become much higher.

6.3 Counting and Generating k-OAs

In this section we describe the procedures k-OA-uniform-count(n, ℓ) and k-OA-uniform-generation(n, ℓ) that are used in algorithm generate-k-OA-XSD, presented in the previous section. These procedures count and generate all non-isomorphic k-OAs that have n states and have alphabet Σ of size ℓ. We obtain both procedures as a direct result from a translation from k-OAs into a combinatorial specification (see Sect. 4.1 for the definition of such specifications). This translation is canonical in the sense that isomorphic k-OAs are translated in the same specification, and vice versa, distinct specifications correspond to non-isomorphic k-OAs. The translation is obtained as follows: first we provide a (canonical) string encoding of k-OAs; then we characterize the set of strings that correspond to encodings of k-OAs; and finally we use this characterization to obtain a specification of strings that encode k-OAs. We start by providing the string encoding of k-OAs.

String Encoding of k-OAs The string encoding is inspired by [2] and is computed as follows: Let $A = (V, E, I, F, \mathsf{lab}, e)$ be an admissible k-OA and assume that the alphabet Σ has size ℓ and V consists of M states. We denote by $o : \Sigma \to [0, \ell - 1]$ a total order over the symbols of Σ. In other words, for any two symbols $\sigma_1, \sigma_2 \in \Sigma$, σ_1 comes before σ_2 in the order o iff $o(\sigma_1) < o(\sigma_2)$, where $<$ denotes the standard order on the natural numbers. Given o, we define a canonical total order $c : V \to$

$[1, M]$ as follows. First, for each state $s \in I$ let $c(s) = o(\mathsf{lab}(s)) + 1$. Then, traverse the automaton in a breadth-first way, where at each state s, assign to each of the neighbours of s, i.e., those $s' \in V$ such that $(s, s') \in E$, that are not yet in the domain of c, and in the order induced by o, the smallest number $n \in [1, M]$ that has not been assigned to a state.

Example 6.11 Consider the k-OA A shown in Fig. 2, over the alphabet $\Sigma = \{a, b\}$, where $o(a) = 0$ and $o(b) = 1$, and let $\varepsilon \in \mathcal{L}(A)$. The canonical total order $c : V \to [1, 5]$ is defined as follows. The initial state q_1 with label a is assigned the number 1, and the initial state q_2 is assigned the number 2. Now, traversing the automaton in a breadth-first order, the number 3 is assigned to the state q_4, which is a neighbour of q_1 and has label a. Similarly, the state q_3, which is a neighbour of q_1 with label b is assigned the number 4. Finally, proceeding to the neighbours of q_2, the state q_5 is assigned the number 5 and all states are now ordered. The ordering of the states is annotated between parentheses in Fig. 2.

Given an admissible k-OA A over Σ with M states, the string encoding of A is a string $\mathsf{enc}(A) = z_1 \cdot z_2 \cdot s$ of length $(M + 1) \cdot \ell + M + 1$, where z_1 is the substring encoding the transitions of the automaton and is of length $(M + 1) \cdot \ell$, z_2 is the substring encoding the set of final states and is of length M, and finally s is 1 if A accepts the empty string and 0 otherwise. The substring $z_1 = s_0 \cdots s_{(M+1)\cdot\ell-1}$ is such that for each $j \in [0, \ell - 1]$ $s_j = c(i)$ where $i \in I$ and $o(\mathsf{lab}(i)) = j$, and for each $j' \in [1, M]$ and $j \in [0, \ell - 1]$, $s_{j'\cdot\ell+j}$ is equal to $c(v)$ where v is the state of A such that $\mathsf{lab}(v) = o^{-1}(j)$ and $(c^{-1}(j'), v) \in E$. Notice that according to the above, the prefix $s_0 \ldots s_{\ell-1}$ is equal to the sequence $1 \ldots \ell$, and informally, for $j' \in [1, M]$ the above expresses that $s_{j'\cdot\ell+j}$ is the number corresponding to the state reached from state with number j' reading the alphabet symbol $o^{-1}(j)$. For the substring $z_2 = s_{(M+1)\cdot\ell} \cdots s_{(M+1)\cdot\ell+M-1}$ encoding the set of final states, for each $j \in [0, M - 1]$, $s_{(M+1)\cdot\ell+j}$ is 1 if the state $c^{-1}(j)$ is final and is 0 otherwise.

Example 6.12 For example, the string encoding for the k-OA A shown in Fig. 2, is the string

$$\underbrace{12}_{0}\,\underbrace{34}_{1}\,\underbrace{15}_{2}\,\underbrace{35}_{3}\,\underbrace{12}_{4}\,\underbrace{35}_{5}\,\underbrace{00001}_{\text{final}}\,\underbrace{1}_{\varepsilon}.$$

The first two digits of the string encode that state 1 (q_1) is the initial state with label a and that state 2 (q_2) is the initial state with label b. The next two digits encode the outgoing transitions of state 1. The outgoing a-transition goes to state 3 (q_4) and the outgoing b-transition goes to state 4 (q_3). The outgoing transitions for states 2–5 are encoded analogously.

Notice that if the given k-OA is not deterministic or not complete, then the string encoding defined above is not well-defined. Indeed, for a k-OA with n states, if it is not deterministic then for some $j' \leq n$ and some $j < \ell$, there will be more than two possible values for $s_{j'\cdot\ell+j}$, and if the k-OA is not complete, there will be no suitable value for $s_{j'\cdot\ell+j}$, for some $j' \leq n$ and $j < \ell$.

We next show that the encoding is canonical: Isomorphic and admissible k-OAs have the same string encoding; and a string encoding of an admissible k-OA uniquely specifies the k-OA, up to isomorphism. It is readily verified that isomorphic and admissible k-OAs lead to the same string encoding. The other direction is more challenging and is shown in the next lemma.

Lemma 6.13 *For two admissible k-OAs A_1 and A_2 with n states over an alphabet Σ of size ℓ, with a total order $o : \Sigma \to [0, \ell - 1]$ defined on this alphabet, if $\mathsf{enc}(A_1) = \mathsf{enc}(A_2)$ then A_1 is isomorphic to A_2.*

Proof We first show that for admissible k-OAs with M states, the canonical total order $c : V \to [1, M]$ defined above is a bijection. Let A be an admissible k-OA with M states. Since $c(I) = [1, \ell]$, every state s is mapped to some $n \in [1, M]$. To show that every $n \in [1, M]$ is in the image of c, we need to show that c is injective, since $|V| = M$. But notice that every time a number is assigned to a state, it is the smallest number that has not yet been assigned to a state.

Now, let $A_i = (V_i, E_i, I_i, F_i, \mathsf{lab}_i, e_i)$ for $i \in [1, 2]$, let $s = \mathsf{enc}(A_1)$ and let $A = (V, E, I, F, \mathsf{lab}, e)$ be the following k-OA. The set of states V is equal to $[1, n]$, where n is the largest number appearing in s, for any state $v \in V$, $\mathsf{lab}(v) = o^{-1}(j \pmod{\ell})$, where j is the first position in s where v appears, for any two states $v_1, v_2 \in V$, let $(v_1, v_2) \in E$ if $v_2 = s_{v_1 \cdot \ell + o(\mathsf{lab}(v_2))}$ and let $I = [1, \ell]$. Note that, $\mathsf{enc}(A) = \mathsf{enc}(A_1)$.

We want to show that A_1 is isomorphic to A. By symmetry it will follow that A_2 is also isomorphic to A and therefore isomorphic to A_1. Let c be the ordering of the states of A_1 and notice that the ordering of the states of A is the identity function. We show that the function $c : V_1 \to V$, in addition to being bijective, it is also an isomorphism for the two automata A and A_1. Suppose that this is not the case. Then one of the following cases hold, each of which leads to a contradiction:

- There exists $k \in [1, n]$ such that $\mathsf{lab}(k) \neq \mathsf{lab}_1(c^{-1}(k))$. Then $o(\mathsf{lab}(k)) \neq o(\mathsf{lab}_1(c^{-1}(k)))$ and the first position in $\mathsf{enc}(A)$ that k occurs is different from the first position that k occurs in $\mathsf{enc}(A_1)$. This contradicts that $\mathsf{enc}(A) = \mathsf{enc}(A_1)$.
- The final states of A are different than the final states of A_1. Again, this contradicts that $\mathsf{enc}(A) = \mathsf{enc}(A_1)$.
- It is not the case that $\varepsilon \in A \Leftrightarrow \varepsilon \in A_1$. But then the last letter in $\mathsf{enc}(A)$ is different from the last letter in $\mathsf{enc}(A_1)$. This contradicts that $\mathsf{enc}(A) = \mathsf{enc}(A_1)$.
- There exist $k, k' \in [1, n]$ such that k has a transition to k' but the state $s_1 = c^{-1}(k)$ does not have a transition to the state $s_2 = c^{-1}(k')$. Then the $(k \cdot \ell + o(\mathsf{lab}(k')))$-th position in the string $\mathsf{enc}(A)$ is k' and the same position in $\mathsf{enc}(A_1)$ is not k', and hence $\mathsf{enc}(A) \neq \mathsf{enc}(A_1)$. This contradicts that $\mathsf{enc}(A) = \mathsf{enc}(A_1)$.

As a consequence, A_1 is isomorphic to A. □

Characterisation of String Encodings We next provide a characterisation of strings that correspond to encodings of admissible k-OAs.

For every string $s = s_0 \dots s_{(n+1)\cdot\ell-1}$, $n \in \mathbb{N}$, we let $(f_i)_{i \in [1,n]}$ be a sequence of numbers such that for each $i \in [1, n]$, f_i denotes the first position in the string s where the number i appears. Then consider the strings $s_0, \dots, s_{(n+1)\cdot\ell-1}$ that satisfy the following 4 rules:

$(A1)$ $\forall i \in [1, n-1]$, $f_i < f_{i+1}$,
$(A2)$ $\forall i \in [1, n]$, $f_i < i \cdot \ell$,
$(A3)$ $\forall i \in [0, \ell-1]$, $|\{s_p \mid p = i \pmod{\ell}\}| \leq k$,
$(A4)$ $\forall i, i' \in [0, \ell-1]$, where $i \neq i'$, $\{s_p \mid p = i \pmod{\ell}\} \cap \{s_p \mid p = i' \pmod{\ell}\} = \emptyset$.

Intuitively, the above rules express the following: Rule $(A1)$ expresses that for each state i, the first time i appears in the string is before the first time state $i+1$ appears in the string. Rule $(A2)$ expresses that for each state i, the first occurrence of i is in the part of the string that is encoding the transitions of the first $i-1$ states, which means that state i is reachable from a state $i' < i$. Rule $(A3)$ expresses that for any symbol σ, there are at most k different states that can be reached by reading σ. Finally, rule $(A4)$ expresses that each state has a unique label.

Notice that the rules $(A1)$–$(A4)$ imply that the substring $s_0 s_1 \dots s_{\ell-1}$ is equal to $1 \dots \ell$. This is because rule $(A2)$ implies that 1 must be in the first ℓ positions, and rule $(A1)$ ensures that $s_0 = 1$. Furthermore, rule $(A4)$ ensures that 1 appears nowhere else in the first ℓ positions of the string. Following the same reasoning for $s_1 \dots s_{\ell-1}$, it follows that $s_0 \dots s_{\ell-1} = 1 \dots \ell$.

The precise relationship between the set of strings satisfying $(A1)$–$(A4)$ and k-OAs is given by the following lemma:

Lemma 6.14 *For each $n \in \mathbb{N}$, enc is a bijection from the set of non-isomorphic admissible k-OAs with n states, over an alphabet Σ of size ℓ to the strings of size $(n+1) \cdot \ell + n + 1$ whose prefix $s = s_0 \dots s_{(n+1)\cdot\ell-1}$ satisfies the rules $(A1)$–$(A4)$ above and whose suffix $s' = s_{(n+1)\cdot\ell} \cdots s_{(n+1)\cdot\ell+n}$ uses only 0 and 1.*

Proof From Lemma 6.13, it follows that the function enc is injective, so it remains to be shown that it is also surjective. Consider any string of length $(n+1) \cdot \ell + n + 1$, whose prefix of size $(n+1) \cdot \ell$ satisfies $(A1)$–$(A4)$, and whose suffix of length $n+1$ uses only 0 and 1. Let A be the automaton obtained by letting the set of states be $[1, n]$, and for each $i \in [0, \ell-1]$, let S_i be the set of states $\{s_j \mid j < (n+1) \cdot \ell,\ j = i \pmod{\ell}\}$. Then for each $s \in S_i$ let $\mathsf{lab}(s) = i$. According to rule $(A3)$ for each $i \in [0, \ell-1]$, $|S_i| \leq k$. Furthermore, for each $i \in [\ell, (n+1) \cdot \ell - 1]$, if $i = j' \cdot \ell + j$, with $j < \ell$, let (j', s_i) be a member of E. Finally, let $I = [1, \ell]$ and let e and the final states of A be determined by the suffix of length $n+1$ of the string. It is readily verified that applying the function enc on A returns the original string. This automaton is complete and deterministic, and it remains to show that it is connected. But from the remark above, the prefix $s_0 \dots s_{\ell-1}$ is equal to $1 \dots \ell$ and from the rule $(A2)$ for each $i \in [1, n]$, there is $j < i \cdot \ell$ such that $s_j = i$, and therefore state i in A is reachable from state $\lfloor \frac{j}{\ell} \rfloor$. By induction, every state is reachable from one of the states in $[1, \ell]$, the initial states. □

In other words, Lemma 6.14 characterizes the strings corresponding to encodings of k-OAs. We next use this characterization to build a combinatorial specification for the set of strings encoding k-OAs.

Combinatorial Specification of k-OAs Let Σ be an alphabet of size ℓ and let A be an admissible k-OA over Σ with n states. Recall that $\mathsf{enc}(A) = z_1 \cdot z_2 \cdot s$ is a string

For $m \geq \ell$, $j' \leq m-1$ and $j < \ell$:

$$\mathcal{S}_m^{(j',j)}[\overline{W}^m] := Q_1 + Q_2$$

$$Q_1 := (\textstyle\prod_{i=j+1}^{\ell+j} \mathcal{W}_{i \pmod \ell}) \times \mathcal{S}_m^{(j'+1,j)}[\overline{W}^m]$$

$$Q_2 := \textstyle\sum_{i=1}^{\ell} \big(\prod_{i'=1}^{i-1} \mathcal{W}_{j+i' \pmod \ell}\big) \times \mathcal{Z}_{m+1} \times \mathcal{S}_{m+1}^{(j',j+i)}[\overline{W}^m_{m+1,\, j+i \pmod \ell}]$$

For $m \geq \ell$, $j' = m$ and $j < \ell$:

$$\mathcal{S}_m^{(j',j)}[\overline{W}^m] := Q_3$$

$$Q_3 := \textstyle\sum_{i=1}^{\ell-j-1} \big(\prod_{i'=1}^{i-1} \mathcal{W}_{j+i' \pmod \ell}\big) \times \mathcal{Z}_{m+1} \times \mathcal{S}_{m+1}^{(j',j+i)}[\overline{W}^m_{m+1,\, j+i \pmod \ell}]$$

For $m = n$, $j' = n$ and $j < \ell$:

$$\mathcal{S}_n^{(n-1,j)}[\overline{W}^n] := \mathcal{W}_{j+1} \times \cdots \times \mathcal{W}_{\ell-1}$$

For $j < \ell$ and when $\overline{W}^m$ is not a valid partition w.r.t. k for m, or when $j' > m$ or when $m > n$:

$$\mathcal{S}_m^{(j',j)}[\overline{W}^m] := \emptyset$$

Fig. 3 Combinatorial specification of k-OAs with n states

of length $(n+1)\ell + n + 1$, where z_1 is the substring of length $(n+1)\ell$ encoding the transitions of A, z_2 is the substring of length n encoding the set of final states, and finally s is 1 if A accepts the empty string and 0 otherwise. Furthermore, the prefix of z_1 of size ℓ is equal to $1 \cdot 2 \cdots \ell$ and the suffix $z_2 \cdot s$ solely contains 0 or 1. Such fixed strings can easily be combinatorially specified. Indeed, for $i \in [1, n]$, let $\mathcal{Z}_i$ denote the atom corresponding to state i. Then, $\mathcal{Z}_1 \times \cdots \times \mathcal{Z}_\ell$ corresponds to the prefix $1 \cdot 2 \cdots \ell$, whereas for $\mathcal{B} := \mathcal{Z}_0 + \mathcal{Z}_1$, we have that $\mathcal{B}^n \times \mathcal{B}$ corresponds to the set of all possible suffixes of size $n+1$ in encodings of k-OAs.

Given these, it remains to specify the remaining symbols in z_1. We need the following notation: For $m \in [\ell, n]$, let $\overline{W}^m = [W_0, \ldots, W_{\ell-1}]$ be a partition of $[1, m]$ in which each part is of cardinality at most k and for each $i \in [1, \ell]$, $i \in W_{i-1}$. We call such a partition, *a valid partition* w.r.t. k for m. We denote by $\overline{W}^m_{m+1,\, j}$ the partition of $[1, m+1]$ obtained from $\overline{W}^m$ by adding $\{m+1\}$ to W_j. Finally, if W is a subset of $[1, n]$ then $\mathcal{W}$ denotes the specification $\sum_{i \in W} \mathcal{Z}_i$ that represents W.

Consider the specification given in Fig. 3. Intuitively, for $\ell < m \leq n$, the combinatorial class $\mathcal{S}_m^{(j',j)}[\overline{W}^m]$ corresponds to the class of strings of length $(n+1) \cdot \ell$, with a prefix $s_0, \ldots, s_{j' \cdot \ell + j}$, which satisfy the rules $(A1)$–$(A4)$, and are such that for each $i \in [0, \ell - 1]$, and $p \in [0, j' \cdot \ell + j]$, it holds that $|\{s_p \mid p = i \pmod \ell\}| = |W_i|$. This class of strings consists of two sets of strings: those strings in which the state $m+1$ does not appear in any of the ℓ positions after position $j' \cdot \ell + j$, and the strings in which the state $m+1$ does appear in one of those. In the specification shown in Fig. 3, this is represented by the two classes Q_1 and Q_2, respectively.

We further need to address two boundary cases: First, in case that $j' = m$, the rule $(A2)$ implies that symbol $m+1$ must appear soon afterwards, namely in one of

the remaining positions up to position $(m+1)\cdot\ell-1$. For $j'=m$ and $j<\ell$, these positions are those ranging from $j'\cdot\ell+j+1$ up to $(j'+1)\cdot\ell-1$. The specification for the class defined by Q_3 reflects that there are $\ell-j-1$ such positions. Second, when $m=n$ and $j'=n$, the end of string is nearby and the recursive specification must end. The pre-ultimate equation in the specification deals with this case. Indeed, it simply pads the string with symbols until the desired length of $(n+1)\cdot\ell$ is reached. The final equation in the specification simply says to ignore invalid inputs, that is, wrong partitions and positions that are out of bound. Notice that k is an implicit parameter in the specification and in particular it is a parameter of a valid partition. A partition is invalid if any of the partition sets it comprises contains more than k elements.

Assume for now that the specification in Fig. 3 is correct. In other words, the combinatorial class $S_{\ell+1}^{(j',j)}[\overline{W}^{\ell}_{\ell+1,j\ (\mathrm{mod}\ \ell)}]$ consists of all strings of length $(n+1)\cdot\ell$ satisfying rules (A1)–(A4), with a prefix $s_0\ldots s_{j'\cdot\ell+j}$, where exactly the states $[1,\ell+1]$ occur in the string up to position $j'\cdot\ell+j$, and for each $i\in[0,\ell-1]$, and $q\in[0,j'\cdot\ell+j]$, it holds that $\{s_q \mid q=i \pmod\ell\}=W_i$. Then, Lemma 6.14 implies that the class of strings corresponding to k-OAs with n states can be combinatorially specified as:

$$\begin{aligned}\mathcal{OA}_n := {} & \mathcal{Z}_1\times\cdots\times\mathcal{Z}_\ell \\ & \times\left(\sum_{p=\ell}^{(\ell+1)\cdot\ell-1}\left(\prod_{i=0}^{p-\ell-1}\mathcal{Z}_{i\ (\mathrm{mod}\ \ell)}\right)\times\mathcal{Z}_{\ell+1}\times S_{\ell+1}^{(j',j)}[\overline{W}^{\ell}_{\ell+1,j}]\right) \\ & \times\mathcal{B}^n\times\mathcal{B},\end{aligned}$$

where j' and j are such that $p=j'\ell+j$ and $j<\ell$. Intuitively, $\mathcal{OA}_n$ is the class of strings where the first ℓ positions are the states 1 to ℓ, the last $n+1$ positions are either 0 or 1, and the positions in between, are labeled according to which is the first position with the state $\ell+1$, which must be before the position $(\ell+1)\cdot\ell-1$.

The following two lemmas tell that the specification shown in Fig. 3 is correct.

Lemma 6.15 *For $\ell,k,n,j',j,n_0,\ldots,n_{\ell-1}\in\mathbb{N}$, $m\geq\ell$, and $\overline{W}^m$ a valid partition w.r.t. k for m, the class $\mathcal{S}_m^{(j',j)}[\overline{W}^m]$ defined by the specification in Fig. 3 is combinatorially isomorphic to the class of strings of length $(n+1)\cdot\ell$ satisfying rules (A1)–(A4), with a prefix $s_0\ldots s_{j'\cdot\ell+j}$, where exactly the states $[1,m]$ occur in the string at positions up to position $j'\cdot\ell+j$, and for each $i\in[0,\ell-1]$, and $q\in[0,j'\cdot\ell+j]$, it holds that $\{s_q \mid q=i \pmod\ell\}=W_i$.*

Proof We defer the proof of this lemma to the Appendix. □

From Lemma 6.15 and the definition of $\mathcal{OA}_n$ it then readily follows that:

Lemma 6.16 *Let Σ be an alphabet of size ℓ. For each $n\in\mathbb{N}$, there is a bijection from the set of objects in $\mathcal{OA}_n$ to the set of strings over Σ of* (i) *size $(n+1)\cdot\ell+n+1$;* (ii) *with a prefix $s=s_0\ldots s_{(n+1)\cdot\ell-1}$ satisfying the rules (A1)–(A4) given earlier; and* (iii) *with a suffix $s'=s_{(n+1)\cdot\ell}\cdots s_{(n+1)\cdot\ell+n}$ that uses only 0 and 1.*

As a consequence from Lemma 6.14 and Lemma 6.16 we may thus conclude:

Theorem 6.17 *Let Σ be an alphabet of size ℓ. For each $n \in \mathbb{N}$, there is a bijection between the set of non-isomorphic k-OAs with n states over Σ and the set of objects in $\mathcal{OA}_n$ of size $(n+1)\cdot\ell+n+1$.*

In other words, the specification $\mathcal{OA}_n$ consists of all strings that correspond to the encoding of a k-OA with n states and alphabet of size ℓ.

Uniform Generation Procedure for k-OAs With Theorem 6.17 in place, we have a canonical translation from k-OAs to the combinatorial specification $\mathcal{OA}$. As a consequence, we now get the procedures k-OA-uniform-count(n,ℓ) and k-OA-uniform-generation(n,ℓ) for free from Theorems 4.2 and 5.2. The procedure k-OA-uniform-generation(n,ℓ) requires an additional step to translate the returned string encodings back to k-OAs, a linear time process. The exact complexity of these procedures is left for future work.

7 Related Work

Sampling Our approach towards counting of tree languages is based on the recursive method, which was initiated by Nijenhuis and Wilf [31], and then formalized by Flajolet, Zimmermann and Van Cutsem [18] in the more general setting of combinatorial specifications. In the current paper we only use a restricted class of specifications (called context-free) in that only atoms, union and product are allowed. General recursive specifications allow many more operators (cf. [18]).

The computational complexity of variants of computing the number of strings of a given length in context-free languages is investigated by Bertoni et al. [6]. We choose to employ the method of going through the combinatorial specification as it gives rise to an immediate implementation in Maple and is versatile enough to incorporate shape constraints (which are not context-free definable).

Sampling of trees that adhere to a probabilistic tree model is investigated in [16, 17]. In particular in [17] a sampling procedure for trees that additionally satisfy a bottom-up tree automaton is provided. These methods differ from ours in that trees are sampled in accordance with their probabilities specified by the probabilistic model rather than uniformly.

XSD Generation There has been substantial work on the uniform generation of regular languages represented by DFAs. To the best of our knowledge, no algorithm exists that uniformly generates *minimal* DFAs. The works [2, 5] and [4] do consider the non-isomorphic uniform generation of admissible and connected DFAs, respectively, but need rejection sampling to sample minimal DFAs. This approach, however, has no proven guarantees on running times.

Our string encoding for k-OAs is inspired by [2]. For the generation procedure, however, we rely on a sampling procedure for combinatorial specifications [18]. Although Héam, Nicaud and Sylvain [22] show that non-isomorphic deterministic tree-walking automata can be generated uniformly at random through an encoding into string transducers, it is not clear how to use their results to generate XSDs.

To the best of our knowledge, the present paper presents the first step towards uniform XSD generation. The papers [8, 10] only dealt with DTDs which reduce to regular expressions. In [11], the experimental validation used one real-world XSD and 8 hand-crafted XSDs. The XSD generation algorithm presented in this paper could be used to generate a benchmark of XSDs. We did not address generation of XML corpora adhering a given schema as is for instance implemented in ToXGene [3].

8 Conclusion

In this paper, we presented a first step towards the foundation for an experimental testbed for XSD generating algorithms. We addressed uniform XSD generation as well as the machinery to compute similarity measures based on the counting of trees of a certain size in tree languages. Finally, we provided a sampling procedure for (unambiguous) tree languages using the formalism of combinatorial specifications.

An initial implementation in Maple shows that the approach through combinatorial specifications is promising. Although the approach to assess similarity through counting of the number of different and common trees is intuitive, in depth experimental validation of efficiency and effectiveness remains needed to obtain a robust similarity measure.

Directions for future work include the following:

The complexity of the generation algorithm for k-OAs and DFA-based k-OA XSDs remains to be determined. We expect, however, an exponential behavior in the size of the alphabet. The main reason for this is that during the generation process, we implicitly need to remember how many times each alphabet symbol already occurred. Fortunately, in real-world content models the far majority of the symbols occur only once. It would be interesting to see how this constraint can be incorporated into the algorithm.

Furthermore, it we would like to extend the XSD generation algorithm to generate k-OREs rather than k-OAs. This, however, would require a useful canonical representation for regular expressions.

Finally, we want to explore the possibility of using specifications (possibly extended with probabilities) to get non-uniform sampling procedures of trees in a tree language. This would be particularly useful in the probabilistic XML setting, among others.

Appendix: Proof of Lemma 6.15

We show that the specification given in Fig. 3 is combinatorially isomorphic to the strings of length satisfying rules $(A1)$–$(A4)$ given in Sect. 6.

Lemma 6.15 *For $\ell, k, n, j', j, n_0, \ldots, n_{\ell-1} \in \mathbb{N}$, $m \geq \ell$, and $\overline{W}^m$ a valid partition w.r.t. k for m, the class $\mathcal{S}_m^{(j',j)}[\overline{W}^m]$ defined by the specification in Fig. 3 is combinatorially isomorphic to the class of strings of length $(n+1)\cdot\ell$ satisfying rules $(A1)$–$(A4)$, with a prefix $s_0 \ldots s_{j'\cdot\ell+j}$, where exactly the states $[1, m]$ occur in the string at*

positions up to position $j' \cdot \ell + j$, *and for each* $i \in [0, \ell - 1]$, *and* $q \in [0, j' \cdot \ell + j]$, *it holds that* $\{s_q \mid q = i \pmod{\ell}\} = W_i$.

Proof We first count the number of valid extensions of such a given prefix (Lemma A.1) and then show that this number coincides with the number of objects in the corresponding class (Lemma A.2).

We need the following notations. Let Σ be an alphabet of size ℓ. For ℓ and k as above, and $m, j', j, n, n_0, \ldots, n_{\ell-1} \in \mathbb{N}$, let $N(m, j' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1})$ be defined inductively as follows:

$$\begin{aligned}
&N\big(m, j'\ell + j, n, n_0, \ldots, n_{\ell-1}\big) = 0 \quad \text{if } \exists i \in [0, \ell - 1] \text{ s.t. } n_i > k,\\
&N\big(m, j'\ell + j, n, n_0, \ldots, n_{\ell-1}\big) = 0 \quad \text{if } j' > m,\\
&N\big(m, j'\ell + j, n, n_0, \ldots, n_{\ell-1}\big) = 0 \quad \text{if } m > n,\\
&N\big(m, j'\ell + j, n, n_0, \ldots, n_{\ell-1}\big) = 0 \quad \text{if } m \neq \textstyle\sum_{i=0}^{\ell-1} n_i.
\end{aligned} \tag{A.1}$$

$$N(n, n\ell + j, n, n_0, \ldots, n_{\ell-1}) = \prod_{i=j+1}^{\ell-1} n_i, \tag{A.2}$$

and for $j' \leq m - 1$ and $j < \ell$,

$$N\big(m, j' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}\big) = N_1 + N_2, \tag{A.3}$$

and for $j' = m$ and $j < \ell$,

$$N\big(m, j' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}\big) = N_3, \tag{A.4}$$

where:

$$\begin{aligned}
N_1 &= (n_0 \cdot \ldots \cdot n_{\ell-1}) \cdot N\big(m, \big(j' + 1\big) \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}\big),\\
N_2 &= \sum_{i=1}^{\ell} \Bigg(\Bigg(\prod_{i'=1}^{i-1} (n_{j+i' \pmod{\ell}}) \Bigg)\\
&\qquad \cdot N\big(m + 1, j'\ell + j + i, n, n_0, \ldots, n_{j+i \pmod{\ell}} + 1, \ldots, n_{\ell-1}\big) \Bigg),\\
N_3 &= \sum_{i=1}^{\ell-j-1} \Bigg(\Bigg(\prod_{i'=1}^{i-1} (n_{j+i' \pmod{\ell}}) \Bigg)\\
&\qquad \cdot N\big(m + 1, j'\ell + j + i, n, n_0, \ldots, n_{j+i \pmod{\ell}} + 1, \ldots, n_{\ell-1}\big) \Bigg).
\end{aligned}$$

Informally, and in a similar manner as above, the number of strings satisfying the rules $(A1)$–$(A4)$ where exactly the states in $[1, m]$ appear at the positions up to an including position $j' \cdot \ell + j$, is equal to the sum of the number of possible strings where $m + 1$ appears for the first time in one of the positions between $j' \cdot \ell + j + 1$ and $(m + 1) \cdot \ell - 1$. In equation (A.3), N_1 covers the case where between the positions

$j' \cdot \ell + j + 1$ and $(j'+1) \cdot \ell + j$ no new state $m+1$ is introduced, and N_2 covers the cases where at some position J between the positions $j' \cdot \ell + j + 1$ and $(j'+1) \cdot \ell + j$, the state $m+1$ is introduced. In the case where $j' = m$, since $N(m, (j'+1) \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}) = 0$ by the definition of the base case that also complies with the rule $(A2)$, the equation is different in order to take into account the remaining positions until position $m \cdot \ell - 1$. This is reflected in (A.4).

The next lemma tells that the function $N(\cdot)$ defined above, correctly counts the strings satisfying rules $(A1)$–$(A4)$.

Lemma A.1 *For $\ell, k, m, n, j', j, n_0, \ldots, n_{\ell-1} \in \mathbb{N}$ with $1 \leq m \leq n$, the number $N(m, j' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1})$ is the number of strings of length $(n+1) \cdot \ell$ satisfying rules $(A1)$–$(A4)$, with a prefix $s_0 \ldots s_{j' \cdot \ell + j}$, where exactly the states in $[1, m]$ occur up to the position $j' \cdot \ell + j$, and for each $i \in [0, \ell - 1]$, and $p \in [0, j' \cdot \ell + j]$, it holds that $|\{s_p \mid p = i \pmod{\ell}\}| = n_i$.*

Proof We proceed by inverse induction on m to show that the statement above holds. For the base case, let $m = n$. Notice that if $m = n+1$, $N(n+1, J, n, n_0, \ldots, n_{\ell-1}) = 0$ for all values of the other parameters. We show that $N(n, J, n, n_0, \ldots, n_{\ell-1})$ is the number of strings described by the lemma. For $J \geq (n+1) \cdot \ell$, $N(n, J, n, n_0, \ldots, n_{\ell-1}) = 0$, as required by the rules. We proceed by inverse induction on $J \leq (n+1) \cdot \ell - 1$.

If $J = n \cdot \ell + j$ for some $j \in [0, \ell - 1]$, then

$$N(n, n \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}) = \prod_{i=j+1}^{\ell-1} n_i,$$

by (A.2), which is the correct number according to the rules $(A1)$–$(A2)$.

For the inductive case, suppose that for all $r > J'$ for some $J' < n \cdot \ell$, $N(n, r, n, n_0, \ldots, n_{\ell-1})$ for all n_i with $n = \sum_{i=0}^{\ell-1} n_i$, is the correct number of strings. Consider the number $N(n, J', n, n_0, \ldots, n_{\ell-1})$. If for any $i \in [0, \ell - 1]$, $n_i > k$ then this number is equal to 0 which complies with the rule $(A3)$ of strings. Suppose then that for all $i \in [0, \ell - 1]$, $n_i \leq k$ and let $J' = J'_0 \cdot \ell + J'_1$, for $J'_0, J'_1 \in \mathbb{N}$ and J'_0 maximal. Notice that, since all states $[1, n]$ occur at a position up to position J', no state number can appear in the string at a position to the right of the position J' that has not appeared to the left or exactly at the position J'. Therefore, from the rules $(A3)$ and $(A4)$, each position to the right of position J' can be occupied by states that have already appeared before that position, and furthermore, have appeared at positions associated with the appropriate label. Consider therefore, the next ℓ positions, starting with $J'_0 \cdot \ell + J'_1 + 1$. For this position there are $n_{J'_1+1 \pmod{\ell}}$ possible values for the string to comply with the rules $(A1)$–$(A4)$. Similarly, for the position $J'_0 \cdot \ell + J'_1 + 2$ there are $n_{J'_1+2 \pmod{\ell}}$ possible symbols, and so on, until position $J'_0 \cdot \ell + J'_1 + \ell = (J'_0 + 1) \cdot \ell + J'_1$ for which there are $n_{J'_1}$ possible values. By the inductive hypothesis, $N(n, J'_0 \cdot \ell + J'_1 + \ell, n, n_0, \ldots, n_{\ell-1})$ is the number of possible strings with a prefix $s_0, \ldots, s_{(J'_0 \cdot \ell + J'_1 + \ell)}$, that satisfy the rules $(A1)$–$(A4)$ and exactly the states $[1, n]$ appear up to position $J'_0 \cdot \ell + J'_1 + \ell$. Therefore, $N(n, J'_0 \cdot \ell +$

$J'_1, n, n_0, \ldots, n_{\ell-1}) = n_0 \cdot \ldots \cdot n_{\ell-1} \cdot N(n, (J'_0 + 1) \cdot \ell + J'_1, n, n_0, \ldots, n_{\ell-1})$, which is also what N_1 is defined to be according to (A.3). Notice that $N_2 = 0$, according to (A.1), which complies with the fact that no new state can appear to the right of J'.

Suppose then that for some $M < n$ and all $m > M$, the number $N(m, J, n, n_0, \ldots, n_{\ell-1})$, for all n_i such that $m = \sum_{i=0}^{\ell-1} n_i$, is the correct number for all $J \in \mathbb{N}$, and consider the value of $N(M, J', n, n_0, \ldots, n_{\ell-1})$ for the different values of $J' \in \mathbb{N}$. We show that this is the correct number of strings. First, for $J' \geq (M+1) \cdot \ell$, the number $N(M, J', n, n_0, \ldots, n_{\ell-1})$ is equal to 0 which complies with rule $(A2)$. We proceed by inverse induction on $J' < (M+1) \cdot \ell$. For the base cases, suppose $J' = M \cdot \ell + J'_1$, for $J'_1 \in [0, \ell - 1]$. Then, $N(M, M \cdot \ell + J'_1, n, n_0, \ldots, n_{\ell-1})$ is determined by (A.4). By rule $(A2)$, the number of strings where exactly the states $[1, M]$ appear at positions up to $M \cdot \ell + J'_1$ is equal to the sum of the number of strings where state $M+1$ appears for the first time in some position in the next $\ell - J'_1 - 1$ positions, and this is the number given by (A.4).

For the inductive hypothesis, suppose that $N(M, r, n, n_0, \ldots, n_{\ell-1})$, for all n_i such that $M = \sum_{i=0}^{\ell} n_i$, is the correct number of strings for all $r > J'$ for some $J' < M \cdot \ell$. Consider $N(M, J', n, n_0, \ldots, n_{\ell-1})$. If for any $i \in [0, \ell - 1]$, $n_i > k$ then this number is equal to 0 which complies with the rule $(A3)$ of strings.

Otherwise, the possible strings with n states that satisfy rules $(A1)$–$(A4)$ and with prefix $s_0 \cdots s_{J'}$, where exactly the states $[1, M]$ occur at positions up to position J', are the following. Either, $M+1$ does not appear in the following ℓ positions to the right of J', or it appears in at least one of them. For the first case, the number of possible strings is $n_0 \cdot \ldots \cdot n_{\ell-1} \cdot N(M, J' + \ell, n, n_0, \ldots, n_{\ell-1})$, where by the inductive hypothesis, $N(M, J' + \ell, n, n_0, \ldots, n_{\ell-1})$ is the correct number of the appropriate strings. This is the number given by the term N_1 of (A.3). For the second case, let $J' = J_0 \cdot \ell + J_1$ and let us consider all possible ℓ positions to the right of position J' where the state $M+1$ appears for the first time. Suppose that this position is $J' + i$ for $i \in [1, \ell]$. Then, the number of possible strings complying with rules $(A1)$–$(A4)$ is the following. For position $J' + 1$ there are $n_{J_1+1 \pmod{\ell}}$ possible values, for position $J' + 2$ there are $n_{J_1+2 \pmod{\ell}}$ possible values, and so on until the position $J' + i$ which is labeled by $M + 1$. The number of allowed strings with prefix $s_0 \cdots s_{J'+i}$, and where the states $[1, M+1]$ appear at positions up to position $J' + i$, is given by $N(M+1, J' + i, n, n_0, \ldots, n_{J_1+i \pmod{\ell}}, \ldots, n_{\ell-1})$, by the inductive hypothesis. Considering all possible values where the new symbol can appear, we get a sum equal to the term N_2 of (A.3). Notice that any string counted in one of the terms of this sum, is not counted in any other term of this sum. Therefore, $N(M, J', n, n_0, \ldots, n_{\ell-1})$ is equal to $N_1 + N_2$, which is what is described by (A.3). □

We next show that $N(\cdot)$ also correctly counts the number of objects in the specification given in Fig. 3.

Lemma A.2 *Let $\ell, k, m, n, j', j, n_0, \ldots, n_{\ell-1} \in \mathbb{N}$, $\ell < m$. The number of objects in $\mathcal{S}_m^{(j',j)}[\overline{W}^m]$ is equal to $N(m, j' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1})$, where for each $i \in [0, \ell - 1]$, $n_i = |W_i|$.*

Proof Firstly, for $m > n$, $N(m, j' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}) = 0$ and there are no objects in $\mathcal{S}_m^{(j',j)}[\overline{W}^m]$. We proceed by reverse induction on $m \leq n$ to show that the statement holds.

Suppose first that $m = n$. Then for all $j' > m = n$, $N(n, j' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}) = 0$, and $\mathcal{S}_n^{(j',j)}[\overline{W}^n]$ has no objects. We then show that the statement holds by reverse induction on $j' \leq m = n$.

For the base cases, suppose that $j' = n$ and $j \in [0, \ell - 1]$. Then $\mathcal{S}_n^{(n,j)}[\overline{W}^n] := \prod_{i=j+1}^{\ell-1} \mathcal{W}_i$, and $N(n, n \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}) = \prod_{i=j+1}^{\ell-1} n_i$. For each i in $[0, \ell-1]$, the number of objects in $\mathcal{W}_i$ is equal to n_i, and hence the statement holds for $j' = n$.

Next, assume that the statement holds for $m = n$ and $j' > J'$ for some $J' < n$ and consider the case where $j' = J'$ and $j \in [0, \ell - 1]$. The class $\mathcal{S}_n^{(J',j)}[\overline{W}^n]$ is given by $Q_1 + Q_2$, and $N(n, J' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1})$ is given by $N_1 + N_2$. Consider first Q_1 and N_1. The class Q_1 is defined as

$$Q_1 = \left(\prod_{i=j+1}^{\ell+j} \mathcal{W}_{i \pmod \ell} \right) \times \mathcal{S}_n^{(J'+1,j)}\big[\overline{W}^n\big],$$

and for $m = n$,

$$N_1 = (n_0 \cdot \ldots \cdot n_{\ell-1}) \cdot N\big(n, \big(J' + 1\big) \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}\big).$$

From the inductive hypothesis, we may conclude that the number of objects in Q_1 is equal to N_1 for $m = n$, $j' = J'$ and $j \in [0, \ell - 1]$.

Similarly, for $m = n$ and $j' = J'$, the class Q_2 is empty by the equation

$$Q_2 := \sum_{i=1}^{\ell} \left(\prod_{i'=1}^{i-1} \mathcal{W}_{j+i' \pmod \ell} \right) \times \mathcal{Z}_{n+1} \times \mathcal{S}_{n+1}^{(J',j+i)}\big[\overline{W}^n_{n+1,j+i \pmod \ell}\big]$$

and N_2 is defined to be equal to

$$\sum_{i=1}^{\ell} \left(\left(\prod_{i'=1}^{i-1} (n_{j+i' \pmod \ell}) \right) \cdot N\big(n+1, J' \cdot \ell + j + i, n, n_0, \ldots, n_{j+i \pmod \ell} + 1, \ldots, n_{\ell-1}\big) \right).$$

We have that Q_2 is undefined whereas N called for $n + 1$ is equal to 0 by definition. Hence, the number of objects in $Q_1 + Q_2$ is equal to $N_1 + N_2$ for $m = n$ and $j' = J'$, and hence the statement holds.

Suppose next that the statement holds for all $m > M$ for some $M < n$, and consider the case where $m = M$. For all $j' > M$, $N(M, j' \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}) = 0$, and $\mathcal{S}_M^{(j',j)}[\overline{W}^M]$ has no objects. We then show that the statement holds by reverse induction on $j' \leq M$.

For the base cases, let $j' = M$, and $j \in [0, \ell - 1]$. Then, we have that the class $\mathcal{S}_M^{(j',j)}[\overline{W}^M]$ is defined by

$$Q_3 := \sum_{i=1}^{\ell-j-1} \left(\prod_{i'=1}^{i-1} \mathcal{W}_{j+i' \pmod{\ell}} \right) \times \mathcal{Z}_{M+1} \times \mathcal{S}_{M+1}^{(j',j+i)}\big[\overline{W}^M_{M+1,j+i \pmod{\ell}}\big],$$

whose number of elements is equal to

$$N_3 = \sum_{i=1}^{\ell-j-1} \left(\left(\prod_{i'=1}^{i-1} (n_{j+i' \pmod{\ell}}) \right) \cdot N\big(M+1, j' \cdot \ell + j + i, n, n_0, \ldots, n_{j+i \pmod{\ell}} + 1, \ldots, n_{\ell-1}\big) \right)$$

by the inductive hypothesis. Finally, assume that the statement holds for all $j' > J'$ for some $J' < M$, and consider the case where $j' = J'$. The class $\mathcal{S}_M^{(J',j)}[\overline{W}^M]$ is defined by the equation $Q_1 + Q_2$. Since

$$Q_1 := \left(\prod_{i=j+1}^{\ell+j} \mathcal{W}_{i \pmod{\ell}} \right) \times \mathcal{S}_M^{(J'+1,j)}\big[\overline{W}^M\big],$$

its number of elements is equal to

$$N_1 = \big(n_0 \cdot \ldots \cdot n_{\ell-1}\big) \cdot N\big(M, (J'+1) \cdot \ell + j, n, n_0, \ldots, n_{\ell-1}\big),$$

by the inductive hypothesis. Similarly, the number of elements in Q_2 defined by

$$Q_2 = \sum_{i=1}^{\ell} \left(\prod_{i'=1}^{i-1} \mathcal{W}_{j+i' \pmod{\ell}} \right) \times \mathcal{Z}_{M+1} \times \mathcal{S}_{M+1}^{(J',j+i)}[\overline{W}^M_{M+1,j+i \pmod{\ell}}]$$

is equal to

$$N_2 = \sum_{i=1}^{\ell} \left(\left(\prod_{i'=1}^{i-1} (n_{j+i' \pmod{\ell}}) \right) \cdot N\big(M+1, J' \cdot \ell + j + i, n, n_0, \ldots, n_{j+i \,(\mathrm{mod}\, \ell)} + 1, \ldots, n_{\ell-1}\big) \right),$$

by the inductive hypothesis. □

Clearly, since $N(\cdot)$ both correctly counts strings and the corresponding objects in the specification, the lemma readily follows.

References

1. Albert, J., Giammerresi, D., Wood, D.: Normal form algorithms for extended context free grammars. Theor. Comput. Sci. **267**(1–2), 35–47 (2001)
2. Almeida, M., Moreira, N., Reis, R.: Enumeration and generation with a string automata representation. Theor. Comput. Sci. **387**(2), 93–102 (2007)
3. Barbosa, D., Mendelzon, A.O., Keenleyside, J., Lyons, K.A.: ToXgene: a template-based data generator for XML. In: International Symposium on Management of Data (SIGMOD), p. 616 (2002)
4. Bassino, F., David, J., Nicaud, C.: Enumeration and random generation of possibly incomplete deterministic automata. Pure Math. Appl. **19**(2–3), 1–16 (2008)
5. Bassino, F., Nicaud, C.: Enumeration and random generation of accessible automata. Theor. Comput. Sci. **381**(1–3), 86–104 (2007)
6. Bertoni, A., Goldwurm, M., Sabadini, N.: The complexity of computing the number of strings of given length in context-free languages. Theor. Comput. Sci. **86**(2), 325–342 (1991)
7. Bex, G.J., Gelade, W., Martens, W., Neven, F.: Simplifying XML schema: effortless handling of nondeterministic regular expressions. In: International Symposium on Management of Data (SIGMOD), pp. 731–744 (2009)
8. Bex, G.J., Gelade, W., Neven, F., Vansummeren, S.: Learning deterministic regular expressions for the inference of schemas from XML data. In: International World Wide Web Conference (WWW), pp. 825–834 (2008)
9. Bex, G.J., Gelade, W., Neven, F., Vansummeren, S.: Learning deterministic regular expressions for the inference of schemas from XML data. ACM Trans. Web **4**(4) (2010)
10. Bex, G.J., Neven, F., Schwentick, T., Vansummeren, S.: Inference of concise regular expressions and DTDs. ACM Transactions on Database Systems (2010)
11. Bex, G.J., Neven, F., Vansummeren, S.: Inferring XML schema definitions from XML data. In: International Conference on Very Large Data Bases (VLDB), pp. 998–1009 (2007)
12. Björklund, H., Martens, W.: The tractability frontier for NFA minimization. In: International Colloquium on Automata, Languages and Programming (ICALP), pp. 27–38 (2008)
13. Brüggemann-Klein, A.: Regular expressions into finite automata. In: Latin American Symposium on Theoretical Informatics (LATIN), pp. 87–98 (1992)
14. Brüggemann-Klein, A., Murata, M., Wood, D.: Regular tree and regular hedge languages over unranked alphabets: version 1, April 3. Technical report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology (2001)
15. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Inf. Comput. **142**(2), 182–206 (1998)
16. Cohen, S., Kimelfeld, B., Sagiv, Y.: Incorporating constraints in probabilistic XML. ACM Trans. Database Syst. **34**(3), 1–45 (2009)
17. Cohen, S., Kimelfeld, B., Sagiv, Y.: Running tree automata on probabilistic XML. In: International Symposium on Principles of Database Systems (PODS), pp. 227–236 (2009)
18. Flajolet, P., Zimmermann, P., Van Cutsem, B.: A calculus for the random generation of labelled combinatorial structures. Theor. Comput. Sci. **132**(2), 1–35 (1994)
19. Gelade, W., Idziaszek, T., Martens, W., Neven, F.: Simplifying XML schema: single-type approximations of regular tree languages. In: International Symposium on Principles of Database Systems (PODS) (2010)
20. Gelade, W., Neven, F.: Succinctness of pattern-based schema languages for XML. J. Comput. Syst. Sci. **77**(3), 505–519 (2011)
21. Gore, V., Jerrum, M., Kannan, S., Sweedyk, Z., Mahaney, S.R.: A quasi-polynomial-time algorithm for sampling words from a context-free language. Inf. Comput. **134**(1), 59–74 (1997)
22. Héam, P.-C., Nicaud, C., Schmitz, S.: Random generation of deterministic tree (walking) automata. In: International Conference on Implementation and Application of Automata (CIAA), pp. 115–124 (2009)
23. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley, Reading (2007)
24. Kannan, S., Sweedyk, Z., Mahaney, S.R.: Counting and random generation of strings in regular languages. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 551–557 (1995)
25. Martens, W., Neven, F., Schwentick, T.: Simple off the shelf abstractions of XML schema. SIGMOD Rec. **36**(3), 15–22 (2007)
26. Martens, W., Neven, F., Schwentick, T.: Complexity of decision problems for XML schemas and chain regular expressions. SIAM J. Comput. **39**(4), 1486–1530 (2009)

27. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML schema. ACM Trans. Database Syst. **31**(3), 770–813 (2006)
28. Martens, W., Niehren, J.: On the minimization of XML Schemas and tree automata for unranked trees. J. Comput. Syst. Sci. **73**(4), 550–583 (2007)
29. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: FOCS, pp. 188–191. IEEE, New York (1971)
30. Murata, M., Lee, D., Mani, M., Kawaguchi, K.: Taxonomy of XML schema languages using formal language theory. ACM Trans. Internet Technol. **5**(4), 660–704 (2005)
31. Nijenhuis, A., Wilf, H.: Combinatorial Algorithms. Academic Press, San Diego (1979)
32. Seidl, H.: Deciding equivalence of finite tree automata. SIAM J. Comput. **19**(3), 424–437 (1990)