Optimal Probabilistic Generation of XML Documents

Serge Abiteboul, Yael Amsterdamer, Daniel Deutch, Tova Milo, Pierre Senellart

Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Optimal Probabilistic Generation of XML Documents

Serge Abiteboul · Yael Amsterdamer · Daniel Deutch · Tova Milo · Pierre Senellart

the date of receipt and acceptance should be inserted later

Abstract We study the problem of, given a corpus of XML documents and its schema, finding *an optimal (generative) probabilistic model*, where optimality here means maximizing the likelihood of the particular corpus to be generated. Focusing first on the structure of documents, we present an efficient algorithm for finding the best generative probabilistic model, in the absence of constraints. We further study the problem in the presence of integrity constraints, namely key, inclusion, and domain constraints. We study in this case two different kinds of generators. First, we consider a *continuation-test generator* that performs, while generating documents, tests of schema satisfiability; these tests prevent from generating a document violating the constraints but, as we will see, they are computationally expensive. We also study a *restart generator* that may generate an invalid document and, when this is the case, restarts and tries again. Finally, we consider the injection of data values into the structure, to obtain a full XML document. We study different approaches for generating these values.

S. Abiteboul
INRIA Saclay and ENS Cachan

Y. Amsterdamer
Tel Aviv University

D. Deutch
Ben Gurion University

T. Milo
Tel Aviv University

P. Senellart
Institut Mines–Télécom; Télécom ParisTech; CNRS LTCI

1 Introduction

We study in this paper the problem of finding a model that best fits a given corpus of XML documents. We focus on *probabilistic*, *generative* models, and our objective is to find an instance of the model that maximizes the likelihood of observing the corpus given the model instance. A solution to this learning problem consists of two components. The first is the discovery of a schema (e.g., in a language such as DTD or XSD) that the documents conform to. This component has been intensively studied; see, e.g., [30, 28, 20, 11, 23, 22]. The second component requires, given such a schema and a corpus, associating probabilities with the different choices in the schema, to obtain a *probabilistic generator* that in some sense (to be made precise in the sequel) maximizes the likelihood of the particular corpus.

Such a probabilistic model has a variety of usages:

Testing. The model can be used to generate (many) samples of the documents for test purposes. For instance, the documents may describe some workflow sessions and the samples be used to stress-test a new functionality.

Explaining. The schema may be useful for explaining the corpus to users. The probabilities provide extra information on the semantics of data. For example, in DBLP, how many journal vs. conference articles there are, or how many authors a paper has on average.

Querying. One can get an approximation of query answers by evaluating queries on this model in the style of query answering on probabilistic databases. For instance, one can assess the probability that journal articles have more than three authors from a particular institute.

Schema mining. Given a corpus, there may be many possible schemas that accept all the documents in the corpus. To choose between those schemas, one can use measures such as compactness [27] (how small the schema is) or precision (how much it rules out documents outside of the corpus). It turns out one can also use, as a quality measure, how well a probabilistic model for this schema fits the corpus.

Auto-completion. The model can be used to generate partial XML documents that form completion of a document prefix edited manually. These completions may then be proposed to the user (editor), assisting in the editing process [2].

These usages are motivations for the present work. We next overview some of the main notions used in this paper, as well as the paper contributions.

We start by describing the notion of schemas used in this work. We use a very general notion of such schemas, essentially based on automata specifying the labels of the children of nodes with a certain label. This classical notion suggests the following *nondeterministic generator* for the documents satisfying a particular schema. Start with a single node whose label is the root label. The children of a node with label a are generated using the automaton A_{a}: starting from the initial state of A_{a} the generator nondeterministically chooses an accepting run of the automaton generating some word $\mathsf{a}_1 \ldots \mathsf{a}_n\$$ in $L(A_{\mathsf{a}})$

(where \$ is a special terminating symbol). Accordingly, the node will have a sequence of n children labeled $\mathsf{a}_1 ... \mathsf{a}_n$.

To obtain a *probabilistic generator*, it suffices to associate probabilities with the transitions in the different automata. These are the probabilities of the transitions to be selected in the course of generation. The resulting generator provides *skeletons* of the document. To obtain full documents, one also needs to feed in data values (at the leaves). The entire generation process we describe may also be interpreted as tree rewriting specified as ActiveXML documents [3]. Our contribution consists in determining the "best" such generator for a given corpus of documents and a specific schema. More precisely, we need to determine the probabilities to attach to the automata transitions that make the corpus most likely given the generator.

We will study the problem with and without semantic constraints on the documents, focusing first on the generation of document skeletons, and then on generating data values for the leaves.

Case without constraints. In the absence of constraints, we introduce a simple and elegant way of determining these probabilities, as follows. The documents of a particular corpus are type-checked (i.e., checked to be valid with respect to the schema). For each automaton, we count the number of times each transition is chosen. We prove that using the relative frequencies of the transitions yields probabilities that optimize the generation of the corpus, and moreover guarantee termination of the generation process.

Case with constraints. Real applications often involve (in addition to schemas) semantic constraints, which greatly complicate the issue. We study three main kinds of constraints considered in practice, namely (unary) key, inclusion, and domain constraints. The main difficulty is that, during generation, we may reach states where some of the transitions do not constitute real alternatives: following a particular transition, there is no chance of generating an instance obeying the constraints. This motivates our definition of two kinds of generators, *restart* generators and *continuation-test* generators, as follows.

A *restart generator* ignores the constraints and generates a skeleton, then checks whether there exists a value assignment for this skeleton so that the resulting document satisfies the constraints. If this fails, it restarts. Unfortunately, we show that for some input instances, there is virtually no chance of generating a skeleton that can be turned into a document satisfying the constraints, rendering restart-generators a problematic solution in general although they may be very efficient in some cases. In contrast, a *continuation-test generator* is somewhat more complex. At every point of generation where there is more than one option, such generator invokes a continuation test to check which of the options are feasible, i.e., for which options there are continuations of the generation that lead to a document satisfying the constraints. Thus we never choose a transition that takes us to a dead end and document generation always succeeds. The price that we pay for this is performing the continuation test, which we show, following the work of [17] on schema satisfiability, to be NP-complete.

To compute the optimal continuation-test generator, we have to assume that choices are binary. (We will explain why.) Again, we type-check the documents of the corpus. We count the number of times each transition was chosen, but this time we only count a transition in cases where there was more than one option with continuation. We prove that this gives optimal probabilities. However, we also analyze the termination probability of such generators, and show that termination is not guaranteed even in very simple cases.

Generating data values. Finally, we consider the generation of data values to be injected at the leaves of the generated document skeletons, following given probabilistic distributions. We present a general algorithm for generating values that conform to the schema constraints.

We then study a particular promising approach for the generation of data values. This approach is based on the idea of annotating skeleton leaves with *old* and *new* annotations. The former implies that, upon generation of values, the value for the leaf should be drawn out of the set of existing leaf values, while the latter implies that this value is a new one. We then provide and analyze two algorithms for the generation of such *old* and *new* annotations for the document skeleton leaves: an *offline* algorithm that operates on a document skeleton (generated, e.g., by one of the generators suggested above), and an *online* algorithm that is embedded into the document skeleton generation process. We further provide algorithms for setting probabilities for both algorithms, which we prove to be optimal. A full comparative analysis of the algorithms shows that neither of the constructions is “superior", in the sense that each type of algorithm achieves better quality w.r.t. different inputs.

Conference version. A preliminary version of this article was published in the proceedings of ICDT 2012 [1]. New contributions of the current version include, among others: (1) full details of all proofs, accompanied with illustrative examples (throughout the article); (2) soundness and completeness of the nd-generator model (in Section 2); (3) an extensive discussion of the generation of data values (in Section 6.2), including examples and an explicit algorithm for learning optimal probabilities for offline generators, absent from [1]; and (4) extensions of the techniques and results to a typed schema model (Section 7).

Outline. In Section 2, we provide the definitions and background for the rest of the paper. Generators are defined in Section 3. In Sections 4 and 5, we study the problem of finding the best probabilistic generators without and with constraints respectively. We discuss value generation in Section 6. Extension of the results to a typed schema model is considered in Section 7. Related work is presented in Section 8, and Section 9 is a conclusion.

2 Preliminaries

In this section, we first introduce basic definitions for XML document and document corpora. We then consider schemas and constraints.

2.1 XML Documents and Corpus

An *XML document* is abstractly modeled as an unranked, ordered, and labeled tree. Given an XML document $d = (V, E)$, we use $\mathrm{root}(d)$ for the root node of d. Let $\mathcal{L} = \mathcal{L}_{\mathrm{leaf}} \cup \mathcal{L}_{\mathrm{inner}}$ be a finite domain of labels, where $\mathcal{L}_{\mathrm{leaf}}$ and $\mathcal{L}_{\mathrm{inner}}$ are two disjoint sets of labels for leaves and inner nodes (i.e., nodes that are not leaves), respectively. We denote by $\mathrm{lbl} : V \to \mathcal{L}$ the labeling function of the nodes, mapping leaf (inner) nodes to leaf (inner) labels. Given a node $v \in V$, $\mathrm{lbl}_{\downarrow}(v) \in \mathcal{L}^*\$$ is the sequence of labels of the children of v, from left to right, with an additional terminating symbol $\$ \notin \mathcal{L}$. We assume that (only) the leaves are further assigned values from a countably infinite domain $\mathcal{U}$ by the function val.

Example 1 Consider the following XML document d_0, viewed as a tree in the standard manner.

```
<Dept>
  <Head>Martha B.</Head>
  <Seniors>
    <Emp>
      <Name>Martha B.</Name>
      <Tel>123-5234</Tel>
      <Tel>123-5357</Tel>
    </Emp>
  </Seniors>
  <Juniors></Juniors>
</Dept>
```

This document describes the phone book of a department containing one senior employee as a member (who is also the department head), Martha B.: The root node v_0 is the one labeled with Dept, i.e., $\mathrm{root}(d_0) = v_0$ and $\mathrm{lbl}(v_0) = \mathsf{Dept}$. Let v_1 be the node such that $\mathrm{lbl}(v_1) = \mathsf{Emp}$. Then $\mathrm{lbl}_{\downarrow}(v_1) = \mathsf{Name\,Tel\,Tel}\,\$$. Similarly, if $\mathrm{lbl}(v_2) = \mathsf{Name}$, then $\mathrm{lbl}_{\downarrow}(v_2) = \$$ (i.e., this is a leaf node with no children), but this node has a value, $\mathrm{val}(v_2) =$ "Martha B.".

An *XML corpus* is then a finite bag of documents. Let $\mathcal{D}$ be the universal domain of all documents over $\mathcal{L}$, $\mathcal{U}$. A corpus is represented by a function $D : \mathcal{D} \to \mathbb{N}$, which maps each document d to the number of times d appears in the corpus. We denote by $|D|$, the bag size counting duplicates (recall that the bag is finite), and by $\mathrm{supp}(D)$, the set of unique documents in D.

2.2 Schema

We start by recalling the notion of schemas as specifications of valid XML documents. We consider first schemas with no constraints, and then in Section 2.3 we extend our definition to the general case where constraints are considered. Also, to simplify the definitions, our model follows that of Document Type Definitions (DTDs). *However, we stress the model can be extended in a straightforward manner to a schema defined in the XML Schema language*, see Section 7.

Let $\mathcal{Q}$ be a finite domain of states.

Definition 1 A *schema* S is a tuple $(\mathsf{r}, \mathcal{A}_{\downarrow})$, where $\mathsf{r} \in \mathcal{L}_{\text{inner}}$ is the root label, and $\mathcal{A}_{\downarrow}$ is a partial function mapping an inner label $\mathsf{a} \in \mathcal{L}_{\text{inner}}$ to a deterministic finite-state automaton (DFA) $\mathcal{A}_{\downarrow}(\mathsf{a}) = A_{\mathsf{a}}$,[1] whose language is $L(A_{\mathsf{a}}) \subseteq \mathcal{L}^{*}\$$. An XML document d is said to be *accepted* by a schema S if $\text{lbl}(\text{root}(d)) = \mathsf{r}$ and for every inner node v of d, $\mathsf{a} = \text{lbl}(v) \in \mathcal{L}_{\text{inner}}$ and $\text{lbl}_{\downarrow}(v) \in L(A_{\mathsf{a}})$.

We refer to the DFA A_{a} as *the deriving automaton* of a, and to the set of all such automata for the labels of a document d as the deriving automata of d.

Remark 1 Note that, by the definition, every word accepted by the automata must terminate with a \$ and contain no other \$'s. To simplify further definitions, we assume that the states of two deriving automata are disjoint subsets of $\mathcal{Q}$.

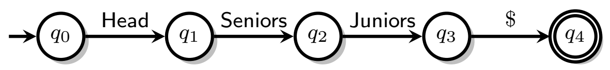

Fig. 1: The A_{Dept} DFA

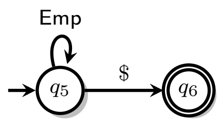

Fig. 2: The A_{Seniors}/ A_{Juniors} DFAs

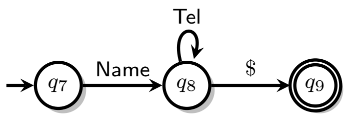

Fig. 3: The A_{Emp} DFA

Example 2 Consider the schema S_0 for documents that describe a department of employees, like in Example 1. In this case, assume that $\mathcal{L}_{\text{inner}} = \{\mathsf{Dept}, \mathsf{Seniors}, \mathsf{Juniors}, \mathsf{Emp}\}$, $\mathcal{L}_{\text{leaf}} = \{\mathsf{Head}, \mathsf{Name}, \mathsf{Tel}\}$, and $\mathsf{r} = \mathsf{Dept}$. A_{Dept} (depicted in Fig. 1) is simply composed of a sequence of states q_0 to q_4, and $L(A_{\mathsf{Dept}}) = \mathsf{Head\,Seniors\,Juniors}\,\$$. A_{Seniors}, A_{Juniors} (depicted in Fig. 2), and A_{Emp} (depicted in Fig. 3), are such that $L(A_{\mathsf{Seniors}}) = L(A_{\mathsf{Juniors}}) = \mathsf{Emp}^{*}\,\$$ and $L(A_{\mathsf{Emp}}) = \mathsf{Name\,Tel}^{*}\,\$$. Note that S_0 accepts the document d_0 from Example 1.

[1] It is common to use *regular expressions* for the allowed sequences of children labels in a schema [29,26]; the reasons for our choice of automata instead will become apparent when we discuss generators further.

2.3 Introducing Constraints

We continue by adding global constraints to the model. Following previous work on constraints in XML schema languages, we consider three major types of constraints on the values of the leaves.

Definition 2 A *schema with constraints* is defined by a pair $\langle S^{\mathrm{u}}, C\rangle$, where S^{u} is a schema (without constraints) and C is a set of constraints on labels from $\mathcal{L}_{\mathrm{leaf}}$, of the following three types.

Key constraint. Given a label $\mathsf{a} \in \mathcal{L}_{\mathrm{leaf}}$, we denote by $\mathrm{uniq}(\mathsf{a})$ the constraint that the value of each a-labeled leaf is unique (among all values of a-labeled leaves in the document)[2].

Inclusion constraint. Given two labels $\mathsf{a}, \mathsf{b} \in \mathcal{L}_{\mathrm{leaf}}$, we denote by $\mathsf{a} \subseteq \mathsf{b}$ the constraint that the values of a-labeled leaves are included in those of b-labeled leaves.

Domain Constraint. Given a label $\mathsf{a} \in \mathcal{L}_{\mathrm{leaf}}$, we denote by $\mathsf{a} \subseteq \mathrm{dom}(\mathsf{a})$ the constraint that in any document, the values of a-labeled nodes are in $\mathrm{dom}(\mathsf{a})$, a subset of $\mathcal{U}$.

We will assume that inclusion constraints $\mathsf{a} \subseteq \mathsf{b}$ are only given when $\mathrm{dom}(\mathsf{a}) = \mathrm{dom}(\mathsf{b})$, or when there are no domain constraints on a, b. When that is not the case, the combination of domain and inclusion constraints may change the domain of possible values for some of the labels, e.g., the "actual" domain of a may become $\mathrm{dom}(\mathsf{a}) \cap \mathrm{dom}(\mathsf{b})$ and must be re-computed.

3 Generators

In this section, we consider various generators. First we consider nondeterministic generators, then probabilistic ones, and finally generators under constraints.

3.1 Nondeterministic Generator

Schemas are typically considered as *acceptors* for verifying XML documents. But it is also possible to see a schema as a *nondeterministic generator* (nd-generator). This is in the same sense that a DFA can be also seen as a word generator. For each node of label a, we can use the automaton A_{a} to nondeterministically generate the node children. Similarly to a schema not performing verification on the leaf values, an nd-generator generates *XML document skeletons*, consisting only of the labeled nodes, and into which leaf values can later be injected (see Section 6). Since this is the main focus of this paper and unless stated otherwise, from now on, when we speak of documents and corpora, we

[2] We are considering here only *unary* keys, defined on single values and not combinations of such values.

mean document skeletons and corpora of document skeletons.

Order of generation. We assume, for all the generator types considered in the sequel, that the node generation is done in a fixed particular order, namely *Breadth-First Left-To-Right (BF-LTR)*. I.e., we first generate the root, then the root's children from left to right, then the children of the root's children, starting from the root's left-most child, and so on. This fixed order of generation is used in the comparison of the different generators types.

Generating a document d can be described as follows:

1. Generate a new root $\mathrm{root}(d)$ with a label r and add it to a *todo* queue Q.
2. While Q is not empty, pop the node v at the head of the queue. Let a be the label of v and q the initial state of A_{a}.
3. Nondeterministically choose one transition (q, b) in A_{a}.
4. If $\mathsf{b} = \$$ (i.e., we have finished generating children for v) return to Step 2.
5. Otherwise $\mathsf{b} \in \mathcal{L}$. Generate v', a child for v such that $\mathrm{lbl}(v') = \mathsf{b}$. If $\mathsf{b} \in \mathcal{L}_{\mathrm{inner}}$ add v' to Q. Set $q \leftarrow q'$ and return to Step 3.

The generation process ends when the *todo* queue at Step 2 is empty, i.e., the deriving automata of all the generated inner nodes reached an accepting state. This means that the inner nodes generated last have only leaves as children (since we are going in a BF-LTR order). In what follows, we say that a generator *conforms* to a schema (also for other types of generators) if they have the same structure (deriving automata and root label).

Example 3 Reconsider the automaton A_{Emp} depicted in Figure 3 as a generator. Assume that we have already generated an Emp-labeled node v, and now we are generating its children. We start from state q_7 and when v has no children. We have only one option for the next transition, moving to q_8. Since the transition is annotated with Name, we generate the first child node and label it with Name. From q_8 we have two options: a transition to itself, in which case we generate an additional child, labeled Tel, and a transition to q_9, in which case no more children are generated for v.

Remark 2 Given such a nondeterministic generator, one can easily construct an Active XML [3] document that generates the same documents. Active XML is much more general and in particular allows specifying generators that will be introduced later in this paper.

Next, we define the notion of a *generation trace*, which describes the process of document generation in terms of the nondeterministic choices taken by the generator.

Definition 3 A *generation trace of a node* v, whose deriving automaton is A and where $\mathrm{lbl}_{\downarrow}(v) = \mathsf{a}_1 ... \mathsf{a}_n\$$, is a sequence $\langle q_0, \mathsf{a}_1\rangle, \langle q_1, \mathsf{a}_2\rangle, ..., \langle q_n, \$\rangle$ where $q_0, ..., q_n \in \mathcal{Q}$ and the transition function δ of A is such that $\delta(q_{i-1}, \mathsf{a}_i) = q_i$ for all $1 \leqslant i \leqslant n$ and $\delta(q_n, \$)$ is an accepting state. A *generation trace of a document* is then the concatenation of all the generation traces of all its inner nodes, in the order they were performed.

An nd-generator generates exactly the documents accepted by the corresponding schema, i.e. the generator is sound and complete, as indicated by the following proposition:

Proposition 1 *For a schema S, the set D of all documents that an nd-generator conforming to S generates is exactly the set of all documents S accepts.*

Proof Assume that S accepts some document d. For each inner node v in d there is an automaton $A_{\mathsf{a}} = \mathcal{A}_{\downarrow}(\mathrm{lbl}(v))$; A_{a} performs a sequence of state transitions on $\mathrm{lbl}_{\downarrow}(v)$ and reaches an accepting state (since S accepts d). Take the sequence of pairs of state and transition label to be the generation trace for v. Concatenating all the generation traces of the inner nodes according to our BF-LTR order will give a valid generation trace of S for d. The other direction is also simple – it is easy to see that if an automaton generates nondeterministically a sequence of child labels, it also accepts this sequence of labels; hence if a document d is generated by S it is also accepted by it. $\square$

3.2 Probabilistic Generator

For practical purposes, we are not only interested in generating all possible finite documents that match some XML schema, but rather we want to generate them according to some probability distribution. For that we introduce the notion of probabilistic generator, where the nondeterministic choices are associated with probabilities.

Definition 4 A *probabilistic generator* (p-generator) S is a pair $\langle S^{\mathrm{u}}, \text{t-prob}\rangle$, where S^{u} is a schema, and t-prob is a function $\mathcal{Q} \times \mathcal{L} \to [0;1]$ mapping the transitions of the deriving automata of d to probabilities, such that for every $q \in \mathcal{Q}$, $\sum_{\mathsf{a} \in \mathcal{L}} \text{t-prob}(q, \mathsf{a}) = 1$, and for every transition (q, a) which is not a part of any automaton, $\text{t-prob}(q, \mathsf{a})$ is 0.

The probabilistic generation process is then very similar to the nondeterministic one, except that from each automaton state q, the generator *randomly chooses* the next transition (q, a), according to t-prob, independently of other choices.

Document probability. Let d be a document skeleton. For each inner node v in d, the probability of $\mathrm{lbl}_{\downarrow}(v)$ is the product of probabilities of all transitions in its generation trace; the probability of d is the product of all such probabilities over all its nodes. Note that we assume for now independence of the probabilistic events associated with transitions (and independence in generation of different documents).

Example 4 Let us assign probabilities to the transitions in the schema described in Example 2. Assume that $\text{t-prob}(q_5, \mathsf{Emp}) = 0.3$, $\text{t-prob}(q_5, \$) = 0.7$, t-prob($q_8$, Tel) = 0.6 and t-prob($q_8$, \$) = 0.4 (all other transitions have probability 1). We can now compute the probability of generating the document skeleton d_0 in Example 1. The following table shows for each node its generation trace and the computation of generation probability. Since all inner nodes in d_0 have unique labels, we use them here as node identifiers.

Node	**Generation trace**	**Probability**
Dept	$\langle q_0, \mathsf{Head}\rangle, \langle q_1, \mathsf{Seniors}\rangle, \langle q_2, \mathsf{Juniors}\rangle, \langle q_3, \$\rangle$	$1 \cdot 1 \cdot 1 \cdot 1 = 1$
Seniors	$\langle q_5, \mathsf{Emp}\rangle, \langle q_5, \$\rangle$	$0.3 \cdot 0.7 = 0.21$
Juniors	$\langle q_5, \$\rangle$	0.7
Emp	$\langle q_7, \mathsf{Name}\rangle, \langle q_8, \mathsf{Tel}\rangle, \langle q_8, \mathsf{Tel}\rangle, \langle q_8, \$\rangle$	$1 \cdot 0.6 \cdot 0.6 \cdot 0.4 = 0.144$
Total		$0.21 \cdot 0.7 \cdot 0.144 \approx 0.021$

The last row shows the probability to generate d_0 with the p-generator, which is the product of the probabilities of the inner nodes.

3.3 Generators with Constraints

In presence of constraints, a generator that only makes independent choices may be unsuitable, as shown next.

Example 5 Let us now consider a schema based on S_0 from Example 2, but with the following additional constraints on the values:

(i) uniq(Name): the employee names are unique.
(ii) Tel $\in$ 123–5$\{0,..,9\}^3$: the department phone numbers always start with 123–5, and then some three digits.
(iii) Head $\subseteq$ Name: the name of the department head must be the name of an employee in the department.

Note that a document generated according to our schema may list a head but no member employees, in violation of constraint (iii). We can try enforcing that there is at least one employee, by setting t-prob(q_5, Emp) to 1 (either in A_{Seniors} or A_{Juniors}). However, such a generator will never halt. Another possibility would be to modify the automaton itself to guarantee e.g. at least one junior or senior employee; but the resulting generator will no longer correspond to the schema and in particular will not generate d_0 from Example 1 (or a similar document, where Martha B. is a junior employee).

We suggest two kinds of generators dealing with this problem: *restart generators* which try to generate a document, check if it is invalid, and if so start the process over again; and *continuation-test generators*, that may perform tests for the existence of continuations leading to valid documents, to avoid generating invalid ones.

Restart generators. We start by defining more formally the notion of a restart generator (r-generator). An r-generator G is a pair $\langle G^{\mathrm{p}}, C\rangle$, where G^{p} is a p-generator, and C is a set of constraints. The operation of G is composed of two main steps which may be repeated.

1. Generating, probabilistically, a document skeleton d matching the schema of G^{p}. This step can be done simply by invoking G^{p}.
2. Checking, given d and C, whether there *exists a valid value assignment to the leaves of* d. If not, d is discarded and we start over.

An important question is whether the test in the second step can be performed efficiently. We show that this is the case, in Section 5.2.

An r-generator is very simple, but may generate many invalid documents before generating a valid one. This leads us to consider the next kind of generators.

Continuation-test generators. We next consider generators that are guaranteed to generate valid documents (without restarting). For that, we introduce the notion of continuation testing. We say that a partial generation trace is *valid* for a schema S if it is a prefix of a generation trace of a valid document skeleton by an nd-generator conforming to S.

Definition 5 Given (1) a schema with constraints S, (2) a partial generation trace ξ valid for S, and (3) $\mathsf{a} \in \mathcal{L} \cup \{\$\}$, a possible next choice, the $\mathtt{CONT}(S, \xi, \mathsf{a})$ *problem* is to decide whether $\xi, \langle q, \mathsf{a}\rangle$ is valid for S, where q is the current state of the nd-generator conforming to S after ξ.

A continuation-test generator (ct-generator) is then a probabilistic generator that (A) conforms to a given schema, (B) generates only documents that are valid with respect to the schema and constraints, and (C) when reaching a certain (non-accepting) state checks, using a *continuation test* that solves CONT, which of the transitions from this state may lead to a valid document; all the transitions that lead to a dead end are ignored; then the generator chooses between the remaining transitions with continuations (there must be at least one), according to their assigned probabilities (normalized to sum up to one).

Intuitively, the continuation test guides the generator by testing if a possible next step can lead (eventually) to a valid document; if not, then the generator will not make this step. In a sense, the continuation test is the only reasonable Boolean test to perform here: if the test returns true when there is no continuation, an invalid document will be generated; in contrast, if the test returns false when there is a continuation, there are some valid documents (that may be in the corpus) that will never be generated, regardless of the probabilities assigned to transitions.

Note that, in the absence of constraints (when $C = \varnothing$), there are no invalid document skeletons and both r-generators and ct-generators are the same as p-generators.

3.4 Quality and Optimality Measures

For a given XML schema, there are many possible generator instances (for each model described above). We define the quality of a generator instance G based on the likelihood of observing a corpus of example documents, under the assumption that it was generated by G. This follows the general notion of maximum likelihood estimation, commonly used for tuning the parameters of probabilistic models (see [13]). Formally,

Definition 6 Given a generator G and for every document skeleton d, let $\Pr(d \,|\, G)$ be the probability for G to generate d. Let D be a document skeleton corpus. Then the *quality* of G with respect to D, denoted $\text{quality}(G, D)$, is $\prod_{d \in \text{supp}(D)} \Pr(d \,|\, G)^{D(d)}$ (recall that $D(d)$ is the number of occurrences of d in D).

Note that if we multiply $\text{quality}(G, D)$ by the multinomial coefficient of D as a bag,[3] the result is exactly the probability for G to generate D.

Optimal generator. Given a schema S, a class $\mathcal{G}$ of generators conforming to S, and a document corpus D, we then say that a generator $G \in \mathcal{G}$ is *optimal* for S, $\mathcal{G}$, D if for each generator $G' \in \mathcal{G}$, $\text{quality}(G, D) \geqslant \text{quality}(G', D)$. When $\mathcal{G}$ is understood, we say that it is optimal for S, D. We call the problem of finding the optimal generator (for given S and D) `OPT-GEN`.

4 The Unconstrained Case

In this section, we first show quality bounds for generators, then study optimal generators for schemas without constraints. The results obtained here are similar to those of [14] for maximum likelihood estimators of probabilistic context-free grammars, but the explicit construction will be useful when we introduce constraints in Section 5.

4.1 An Upper Bound for Quality

We start by considering an upper bound of quality for a corpus. We will later discuss whether this bound can be achieved by the kinds of generators we defined, or by others.

Given a corpus D, consider a generator that would generate each document d in D with probability $\frac{D(d)}{|D|}$, i.e., according to its relative frequency. The quality of this generator would be $q_D = \prod_{d \in \text{supp}(D)} \left(\frac{D(d)}{|D|}\right)^{D(d)}$. We can show that this is indeed an upper bound for the possible quality of a generator for D, *independently* from the type of generator and the schema it conforms to, as the following theorem holds.

[3] The multinomial coefficient is the number of distinct permutations of the bag elements (specifically, it is $|D|!$ if D is a set).

Theorem 1 *Let D be a corpus and G a generator. Then* $\text{quality}(G, D) \leqslant q_D$.

Proof The proof is based on the following lemma:

Lemma 1 *Let $\alpha_1 \dots \alpha_n$ be n positive integers. We define the function f : $[0;1]^n \to [0;1]$ as $(p_1, \dots, p_n) \mapsto f(p_1, \dots, p_n) = \prod_{i=1}^{n} p_i^{\alpha_i}$. Then the maximum of f under the constraint $\sum_{i=1}^{n} p_i \leqslant 1$ is obtained when $p_i = \frac{\alpha_i}{\sum_{k=1}^{n} \alpha_k}$ for $1 \leqslant i \leqslant n$ and only then.*

Proof First note that as a real-valued continuous function with a compact domain, f has a maximum. Since $f(p_1, \dots, p_n) = 0$ if and only if one of the p_i's is 0, this maximum under $\sum_{i=1}^{n} p_i \leqslant 1$ is obtained for some $(p_1^*, \dots, p_n^*) \in (0;1]^n$. This point is also (under the same constraint) a maximum of the $\log f$ function defined over $(0;1]^n$ by:

$$(p_1, \dots, p_n) \mapsto (\log f)(p_1, \dots, p_n) = \sum_{i=1}^{n} \alpha_i \log p_i.$$

Observe next that the maximum value of $f(p_1 \dots p_n)$ under the constraint $\sum_{i=1}^{n} p_i \leqslant 1$ is necessarily obtained when $\sum_{i=1}^{n} p_i = 1$ since this function is strictly increasing with respect to each of its argument. Therefore, $\sum_{i=1}^{n} p_i^* = 1$.

The classical Gibbs lemma (see, e.g., [24]) states that for such a function $\log f$, there exists a constant λ such that for all $1 \leqslant i \leqslant n$, $\frac{\partial(\log f)}{\partial p_i}(p_i^*) = \lambda$. This can also be shown using elementary analysis and induction on n. This means that for all i, $\frac{\alpha_i}{p_i^*} = \lambda$, since $\sum_{k=1}^{n} p_k^* = 1$, $\lambda = \sum_{k=1}^{n} \alpha_k$, and thus $p_i^* = \frac{\alpha_i}{\sum_{k=1}^{n} \alpha_k}$. $\square$

By the lemma above, q_D is defined exactly as the maximum quality for the corpus D. $\square$

Note that if we do not restrict ourselves to any particular schema, it is easy to design a generator that achieves this optimal quality: ignore any schema information, and simply randomly choose documents from the corpus, according to their relative frequency. We argue that this is not a good generator. First, if the corpus is very large, this generator will be much less compact than the ones we study, so not appropriate for explanation or query evaluation. Furthermore, this generator suffers from *over-fitting*: it cannot generate any documents other than those already in the corpus, and thus it is not appropriate for, e.g., testing. We want to generate documents that are similar to, yet different from, those in the corpus. This will be achieved by the kinds of generators we study.

4.2 An Optimal Generator

We next consider the problem of finding the optimal probabilistic generator *out of those conforming to a given schema*, in the unconstrained case. We introduce Algorithm 1 that takes a schema and a corpus as inputs and computes a

```
   Input: schema $S$, corpus $D$ of documents accepted by $S$
   Output: p-generator $G$ conforming to $S$
1  foreach transition $(q, a)$ in an automaton of $S$ do
       freq$(q, a) \longleftarrow 0$;
2  foreach $d \in$ supp$(D)$ do
       $\xi \longleftarrow$ the generation trace of $d$ by $S$;
       foreach $\langle q, a\rangle$ in $\xi$ do
3          freq$(q, a) \longleftarrow$ freq$(q, a) + D(d)$;
4  foreach state $q$ in an automaton of $S$ do
       total$(q) \longleftarrow 0$;
       out$(q) \longleftarrow 0$;
       foreach transition $(q, a)$ in an automaton of $S$ do
           out$(q) \longleftarrow out(q) + 1$;
5          total$(q) \longleftarrow$ total$(q)$ + freq$(q, a)$;
6  $G \leftarrow \langle S, \text{t-prob}\rangle$ s.t. $\forall q \in \mathcal{Q}, a \in \mathcal{L} \cup \{\$\}$ t-prob$(q, a) = \frac{1}{\text{out}(q)}$ if total$(q) = 0$,
   otherwise t-prob$(q, a) = \frac{\text{freq}(q,a)}{\text{total}(q)}$;
   return $G$;
```

Algorithm 1: Algorithm for `OPT-GEN` (no constraints)

probability for each transition, i.e., produces a probabilistic generator. The next result states that this generator is optimal. In its statement and in the remaining of the article, we denote by $|S|$, the size of the schema S and by $|D|$, the total size of the corpus D (i.e., the sum of the size of all distinct elements in D, plus a binary encoding of their multiplicity).

Theorem 2 *Given a schema S and a corpus D of documents accepted by S, Algorithm 1 computes an optimal p-generator for S and D in time $O(|S|+|D|)$; thus,* `OPT-GEN` *(without constraints) can be solved in linear time.*

Proof Algorithm 1 takes a schema as input and computes a probability for each transition. In lines 2–3 the schema is used for type-checking the corpus documents, and in the process the number of times each transition (q, a) was chosen is recorded in freq(q, a) (also considering the frequency of each document in the corpus). Then in lines 4–6 we assign as probability of each transition (q, a) the relative number of times it was chosen after reaching q. If some state was not reached during the verification phase, we give equal probabilities to all transitions from it.

By construction, Algorithm 1 outputs a generator which has the same structure as S. The normalization in line 6 enforces that the sum of probabilities of transitions with the same origin is always 1.

Lines 1, 4–5, and 6 require a time linear in S. The loop in lines 2–3 consists in running the schema on each unique $d \in D$ and therefore require a time linear in the size of D.

It is still to be shown that the output G of Algorithm 1 has maximum quality among all generators that conform to S. The quality of G is:

$$\begin{aligned}
\text{quality}(G,D) &= \prod_{d\in\text{supp}(D)} \Pr(d\,|\,G)^{D(d)} \\
&= \prod_{d\in\text{supp}(D)} \prod_{q \text{ in } S} \prod_{(q,a) \text{ in } S} \left(\frac{\text{freq}(q,a)}{\text{total}(q)}\right)^{\substack{D(d)\times\#\langle q,a\rangle \text{ in} \\ d\text{'s trace by } S}} \\
&= \prod_{q \text{ in } S} \prod_{(q,a) \text{ in } S} \left(\frac{\text{freq}(q,a)}{\sum_{(q,b) \text{ in } S} \text{freq}(q,b)}\right)^{\text{freq}(q,a)}
\end{aligned}$$

whereas, similarly, every probabilistic generator G' conforming to S has quality:

$$\text{quality}(G',D) = \prod_{q \text{ in } S} \prod_{(q,\,a) \text{ in } S} p(q,a)^{\text{freq}(q,a)}$$

for some assignment $p(q,a)$ verifying, for each state q of S, $\sum_{(q,\,a) \text{ in } S} p(q,a) = 1$. Observe that there is no constraint between transitions of different origins (q,a) and (q',b). We can then look independently for each state q which assignment of $p(q,a)$ maximizes $\prod_{(q,\,a) \text{ in } S} p(q,a)^{\text{freq}(q,a)}$ under the summing constraint. Lemma 1 shows that this is exactly the assignment made by G.[4] $\square$

To be of practical use, the generator returned by Algorithm 1 needs a guarantee of almost always termination, which is not a consequence of Theorem 2. However, we next show that our construction guarantees termination.

Theorem 3 *The generator returned by Algorithm 1 has a termination probability of* 1.

Proof For a *schema* $S = (\mathsf{r}, \mathcal{A}_\downarrow)$, we define the Context Free Grammar (CFG) $S_{\text{CFG}} = (\mathcal{L}_{\text{inner}} \cup \mathcal{Q}, \mathcal{L}_{\text{leaf}} \cup \{\$\}, R, \mathsf{r})$, where R contains the following production rules: for every $\mathsf{a} \in \mathcal{L}_{\text{inner}}$, R contains the rule $\mathsf{a} \to q$, where q is the initial state of A_a; for every transition $(q,\$)$ in S, R contains the rule $q \to \$$; finally, for every transition (q,a), $\mathsf{a} \neq \$$, R contains $q \to \mathsf{a}\ q'$, such that q' is the target state of the transition. S_{CFG} simulates the operation of S.

We can also define how to translate the XML documents in an input corpus D for S to parse trees of S_{CFG}. This is done simply by replacing every a-labeled inner node, whose generation trace by S is $\langle q_0, \mathsf{a}_1\rangle, ..., \langle q_{n-1}, \mathsf{a}_n\rangle, \langle q_n, \$\rangle$, with a sub-tree containing a as a root, q_0 as its single child and then for $0 \leqslant i < n$, the children of q_i are the subtree corresponding to a_{i+1} and q_{i+1}; finally, q_n has \$ as its single child. Let us use D' to denote the bag of trees achieved in this manner from the documents in D.

Now assume that G is the output of Algorithm 1 for S,D. Let us assign probabilities to the rules in R, according to those assigned to the transitions of S

[4] When $\text{total}(q) = 0$, the value of this term is 1 for any assignment of $p(q,a)$, and in particular for the uniform probabilities assigned by Algorithm 1.

by the algorithm. Note that by the construction of S_{CFG}, the only non-terminals that may have more than one possible production rule, are those representing non-accepting states in $\mathcal{Q}$. We will assign each such rule $q \rightarrow \mathsf{a}\ q'$ or $q \rightarrow \$$ the probability t-prob(q, a) or t-prob$(q, \$)$, respectively, and the probability 1 to every other rule.

According to the definition of Algorithm 1 and the construction of D', those probabilities reflect exactly, for every rule of the form $q \rightarrow \mathsf{a}\ q'$ or $q \rightarrow \$$, its relative frequency in D' among all production rules of q (for the rest of the production rules, this is trivial). Thus, according to [14], the probabilities we assigned to S_{CFG} are the maximum-likelihood estimator for D' and S_{CFG}, and therefore the termination probability of S_{CFG} is 1.

Let t be any parse tree produced by S_{CFG}. Note that the way we mapped documents to parse trees is reversible, thus there exists a document d corresponding to t. The probability for G to generate d is the same as the probability for S_{CFG} to produce t, since choices with the same probabilities are taken in both processes. Thus, the probability that G generates a finite document, i.e., that the generation process terminates, is also 1. □

5 The Case with Constraints

We now allow constraints, as defined in Section 3.3. We consider the computation of optimal continuation-test generators (ct-generators) and restart generators (r-generators). We start with ct-generators.

5.1 Continuation-Test Generators

We first study the complexity of continuation tests. To do that, we need to adapt some known result:

Proposition 2 (adapted from [17, 19]) *The satisfiability of an XML schema with unary key, inclusion, and domain constraints is NP-complete w.r.t. the size of the schema.*

Proof A similar claim is proved in [17], which follows, in turn, from the proof in [19]. Both models in [17, 19] are more expressive than ours (which means that NP membership carries over), but the hardness results are given even for a very simple model, a deterministic restriction of DTDs (which is less expressive than ours). One last required adaptation follows from the fact that their results are for key and inclusion constraints but not for domain constraints. To account for domain constraints, we start by reviewing the proof used in [17]. The proof there is by encoding the schema with constraints as a Presburger formula, and showing that the formula is satisfiable if and only if the schema with constraints is satisfiable. To extend the proof to also account for domain constraints in our settings, we first observe that a domain constraint on a restricts the set of valid document skeletons only if the domain is finite and there is a key constraint on a; in this case the domain constraint is expressible as an inequality specifying that the number of occurrences of a is smaller than the domain size. So, we add the relevant inequalities to the Presburger formula, and the proof technique of [17] can still be used. $\square$

We may now show the following proposition, where we test for the existence of a continuation for a partial document using a schema satisfiability test.

Proposition 3 *Let $S = \langle S^{\mathrm{u}}, C\rangle$ be a schema with constraints, ξ a partial generation trace valid for S^{u}, and (q, a) a possible next transition. Solving* $\mathtt{CONT}(S, \xi, \mathsf{a})$ *is NP-complete w.r.t. $|S|$. Moreover, we can give a decision algorithm of complexity $O\left(\mathrm{poly}(|\xi|)^{\mathrm{poly}(|S|)}\right)$ (i.e., polynomial in the size of the input partial document, if the schema is fixed).*

Proof Given $S = \langle S^{\mathrm{u}}, C\rangle$, we construct a new schema $S' = \langle S'^{\mathrm{u}}, C'\rangle$, as follows. After $\xi' = \xi, \langle q', \mathsf{a}\rangle$, a generator conforming to S^{u} will be in the process of generating children for some node v in $d_{\xi'}$ (the partial document obtained after ξ'), at some state q (assuming transition $(q', \mathsf{a}) \to q$). Let us denote by P the set of nodes in $d_{\xi'}$ for which children were not generated (i.e., all the leaves of $d_{\xi'}$, and among the inner nodes – in the cases of a BF-LTR order – the right siblings of v, the children of v, and the children of v's left siblings). Denote by $\#_{\mathsf{a}_i}$ the number of a_i-labeled nodes in P.

First, we define the schema S'^{u}: let r', *curr* be new labels. We also define for each $\mathsf{a}_i \in \mathcal{L}_{\mathrm{inner}}$ (resp., $\in \mathcal{L}_{\mathrm{leaf}}$) a new inner (resp., leaf) label a'_i, with the same value domain and an equivalent deriving automaton. This set of new labels will be used to represent nodes that already existed in $d_{\xi'}$. We set the root label of S'^{u} to be r', and $A_{\mathsf{r}'}$ to be such that $L(A_{\mathsf{r}'}) = \mathit{curr}\,{\mathsf{a}'_0}^* ... {\mathsf{a}'_n}^*$, where $\mathsf{a}_0, ..., \mathsf{a}_n$ are all the labels of leaves in P. The deriving automaton of *curr* is the same as that of $\mathrm{lbl}(v)$, but its initial state is q.

Second, we want to add slightly different constraints to C': for every a'_i, we require that the number of a'_i-labeled nodes is exactly $\#_{\mathsf{a}_i}$ (note that this constraint may apply also to inner nodes); if $\mathrm{uniq}(\mathsf{a}_i) \in C$, we add $\mathrm{uniq}(\mathsf{a}_i \cup \mathsf{a}'_i)$ to C'; if $\mathsf{a}_i \subseteq \mathsf{a}_j$ is in C, we add $(\mathsf{a}_i \cup \mathsf{a}'_i) \subseteq (\mathsf{a}_j \cup \mathsf{a}'_j)$ to C'; and if $\mathsf{a}_i \subseteq \mathrm{dom}(\mathsf{a}_i)$ is in C, we add $\mathsf{a}_i \cup \mathsf{a}'_i \subseteq \mathrm{dom}(\mathsf{a}_i)$ to C'. Again, these constraints are not allowed in our model, but the more expressive model of [17] allows encoding them.

Now we need to show that S' is satisfiable iff $\mathtt{CONT}(S, \xi, \mathsf{a}) = T$. The construction of $\mathit{curr}, \mathsf{a}'_0, ..., \mathsf{a}'_n$ as the root children and the constraints on the number of a'_i-labeled nodes ensure that every document that satisfies S' has the same continuations to the root children as the possible continuations of $\xi, \langle q', \mathsf{a}\rangle$; the changed constraints capture the fact that a_i and a'_i are treated as nodes of the same kind with respect to the constraints. Thus every document valid for S' can be translated to a continuation for (S, ξ, a), and vice versa.

For the complexity,

- Constructing S' takes time polynomial in $|S|$, $|\xi|$.
- The size of S'^{u} is linear in the size of S^{u}, and C' is of size $O(|C| + |S| \log(|\xi|))$, because it has one equivalent for every rule in C, plus the constraints on the amounts of a'_i-labeled leaves (in which numbers are encoded in binary).

- Constructing the Presburger formula, using the results of [17], takes time polynomial in $|S'|$.
- The size of the formula is $O\left(\text{poly}\left(|S| \log\left(|\xi|\right)\right)\right)$, with $O(|S|)$ variables, and solving it is NP-complete in this size, thus also NP-complete w.r.t. the size of the schema.
- As explained in [17], if there exists a solution to the formula, there exists a solution where the value assigned to each variable is bounded, in our case, by $p_1(|S|\,|\xi|)^{p_2(|S|)}$. p_1, p_2 are polynomials determined by the sizes of matrices in the formula and the values in the matrices. Thus, a brute-force algorithm for checking whether the formula is satisfiable, that checks all the possible assignments to the formula variables up to the bound, has the complexity $O\left(\text{poly}(|S|\,|\xi|)^{\text{poly}(|S|)}\right)$.
- The overall complexity of the brute-force solution is thus $O\left(\text{poly}(|\xi|)^{\text{poly}(|S|)}\right)$. This means that in particular, if the size of the schema is fixed, the brute force algorithm is polynomial in the size of the partial document.

□

Finding an optimal binary ct-generator. We assume that the schema has a particular property, namely that it is *binary*. A schema is *binary* if for each state of each automaton in the schema, there are at most two possible transitions. We will discuss the case of non-binary schemas afterwards.

Recall that FP^{NP} is the class of problems solvable by polynomial-time computation algorithms that are allowed calls to an NP oracle. We show (the complexity is with respect to the schema size, the algorithm is polynomial with respect to the corpus size):

Theorem 4 *Given a binary schema with constraints S and a corpus, finding an optimal ct-generator is in FP^{NP}.*

Proof Algorithm 2 computes the optimal ct-generator in time polynomial in the size of S, while making calls to an oracle *cont* that performs continuation tests. Generally, Algorithm 2 is very similar to Algorithm 1, except that the frequency of taking a transition is only recorded in situations where there exists another optional transition, which according to the oracle *does not lead to a dead end.* The time complexity of the algorithm follows from the complexity of Algorithm 1, and the calls to *cont* in line 3.

It is still to be shown that the output G of Algorithm 1 has maximum quality among all the ct-generators that conform to S. This proof is similar to that of Proposition 1, but this time when we maximize the term $\text{quality}(G', D) = \prod_{(q,\,a) \text{ in } S} p(q,a)^{\text{freq}(q,a)}$, $\text{freq}(q,a)$ refers to the number of times the transition (q,a) was taken when there was a second choice with continuation. In other cases every ct-generator must have chosen the only possibility with prob. 1. □

Generation time. Without constraints, it was trivially the case that a document was generated in time linear in its size and the size of the schema. However, for ct-generators the generation time depends on the complexity of the continuation

```
  Input: constrained schema $S$, corpus $D$ of documents accepted by $S$
  Output: ct-generator $G$ conforming to $S$
1 foreach transition $(q,a)$ in an automaton of $S$ do
      freq$(q,a) \longleftarrow 0$;
2 foreach $d \in$ supp$(D)$ do
      $\xi \longleftarrow$ the generation trace of $d$ by $S$;
      foreach $\langle q,a \rangle$ in $\xi$ do
          if $\exists a' \neq a$ s.t. $(q,a')$ is a transition in $S$ then
              $\xi' \longleftarrow$ the prefix of $\xi$ before $\langle q,a \rangle$ (exclusive);
3             if $cont(S,\xi',a') = T$ then
4                 freq$(q,a) \longleftarrow$ freq$(q,a) + D(d)$;
5 Compute total and out as in Algorithm 1 lines 4-5;
6 $G \leftarrow$ ct-generator based on $S$ and where $\forall q \in \mathcal{Q}, a \in \mathcal{L} \cup \{\$\}$ t-prob$(q,a) = \frac{1}{\text{out}(q)}$ if
  total$(q) = 0$, otherwise t-prob$(q,a) = \frac{\text{freq}(q,a)}{\text{total}(q)}$;
  return $G$;
```

Algorithm 2: Algorithm for `OPT-GEN` (constraints, ct-generators)

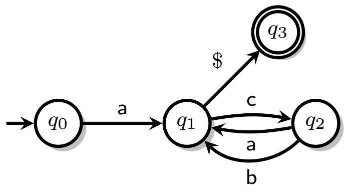

Fig. 4: The A_{r} DFA

test. This means that the generation time will be exponential in the size of the schema (unless there exists a continuation test algorithm with lower complexity, which is unlikely assuming P≠NP).

Termination probability. Unfortunately, it turns out that the constrained setting of ct-generators affects the termination guarantee that we had in the unconstrained case (Theorem 3). We can show that, even in simple cases with non-recursive schemas, termination of the optimal generator is not almost certain.

Theorem 5 *For every $\varepsilon > 0$ there exists a binary, non-recursive schema with constraints S and an input corpus D such that the optimal ct-generator G for S, D, has termination probability $\leqslant \varepsilon$.*

Proof Consider the following schema with constraints S. We have $\mathcal{L}_{\text{inner}} = \{\mathsf{r}\}$, A_{r} is the automaton depicted in Figure 4, and $C = \{\mathsf{b} \subseteq \mathsf{a}, \text{uniq}(\mathsf{b})\}$. The constraints imply, in particular, that there must be at least as many a-labeled leaves in any valid document as b-labeled leaves. Let d be a document such that $\text{lbl}_{\downarrow}(\text{root}(d)) = \mathsf{acb}\$$, and d' such that $\text{lbl}_{\downarrow}(\text{root}(d')) = \mathsf{acacb}\$$. Let D be a corpus that contains $N-1$ copies of d and one of d'. Consider a ct-generator G optimal for S, D. By the optimality of Algorithm 2, t-prob(q_2, a)

in G (when both choices from q_2 have a continuation) must be $\omega = \frac{1}{N}$. Similarly, t-prob$(q_1, \mathsf{c}) = \frac{N+1}{2N+1}$, and, in general, since every transition is encountered during the type-check of the corpus, the probability of every transition in an optimal generator is never chosen arbitrarily.

Note that during any generation process of the ct-generator, every continuation test from q_2 always succeeds. For instance, after generating n a-labeled leaves and m b-labeled leaves, it is naturally possible to generate another a, but also another b, because there exists a continuation with $\max(n, m) + 1$ a-labeled leaves. Denote by p_n the probability of generating a document with exactly n c-labeled leaves. We can give an upper bound for this probability by computing the probability of generating n a's and b's, in some order, such that at least half of them are a's (to satisfy the constraints).

$$\begin{aligned} p_n \leqslant \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{k} (1-\omega)^k \omega^{n-k} &\leqslant \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{k} \omega^{n-k} \\ &\leqslant 2^n \omega^{\lfloor \frac{n}{2} \rfloor} \leqslant 2 \times (4\omega)^{\lfloor \frac{n}{2} \rfloor} \end{aligned}$$

Observe that p_1 is the probability of generating exactly one c, i.e., that of generating a document with root label either acb\$ or aca\$; while generating these documents, the continuation tests always succeeds, so that

$$p_1 = \frac{N+1}{2N+1} \times \left(\frac{1}{N} \times \frac{N}{2N+1} + \frac{N-1}{N} \times \frac{N}{2N+1} \right) = \frac{N(N+1)}{(2N+1)^2}.$$

Now we want to compute an upper bound for p, the termination probability of G, which is the sum of probabilities for generating a finite document, that has a finite number of c-labeled leaves:

$$\begin{aligned} p = \sum_{n=0}^{\infty} p_n &= \frac{N}{2N+1} + \frac{N(N+1)}{(2N+1)^2} + \sum_{n=2}^{\infty} p_n \\ &\leqslant \frac{N}{2N+1} + \frac{N(N+1)}{(2N+1)^2} + \sum_{n=2}^{\infty} 2 \times (4\omega)^{\lfloor \frac{n}{2} \rfloor} \\ &\leqslant \frac{N}{2N+1} + \frac{N(N+1)}{(2N+1)^2} + 2 \times \sum_{n=1}^{\infty} 2(4\omega)^n \\ &\leqslant \frac{N}{2N+1} + \frac{N(N+1)}{(2N+1)^2} + 4 \times \frac{4\omega}{1-4\omega}. \end{aligned}$$

These three terms are, respectively, arbitrarily close to $\frac{1}{2}$, $\frac{1}{4}$, and 0 when N is large enough. Therefore, for any small η (say, $\eta = \frac{1}{8}$), we can choose N large enough so that $p \leqslant \frac{3}{4} + \eta$. Note that by separating p_0 and p_1 from the sum, we correct a minor mistake in the proof of the same theorem in [1].

Finally, to create a schema for which the termination probability is $\leqslant \varepsilon$, we can chain multiple occurrences of S one after the other, as required. □

Note, however, that since the probability of generating the corpus is greater than zero, the termination probability of the optimal generator is always strictly greater than zero.

There are numerous ways of dealing with the problem of non-termination. One practical such way, following a natural assumption in document sampling [7], is to restrict the size of the generated document. This upper bound on the document size must be at least that of the largest document in the input corpus (to ensure that the probability of generating the corpus is non-zero), and can be estimated based on the corpus. Such a size limit can be encoded as a constraint, by making certain changes to the schema (obtaining a new schema whose size is linear in the size of original schema and size limit encoded in binary). Thus, it directly follows that we obtain the optimal probabilities for a size-limited ct-generator. A different direction for guaranteeing almost always termination is by restricting the expressiveness of the schema. We leave as an interesting open problem the characterization of what constraints on the schema will guarantee termination, and the question of translating schemas to safe ones which are sure to terminate.

We conclude the discussion on ct-generators by a remark on non-binary choices.

Non-binary choices. We have assumed so far in this section that the schema is binary. We study here two approaches for handling the non-binary case: (1) turning the choices into binary choices, and (2) keeping probabilities for all combinations of valid choices. We present these by example.

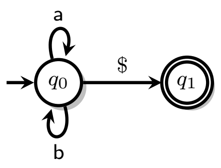

Fig. 5: A_3, a DFA with 3 choices

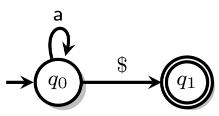

Fig. 6: The DFA A_{tradeoff}

Consider the following constrained schema. The deriving automaton A_3 of the root label $\mathsf{r} \in \mathcal{L}_{\text{inner}}$ is shown in Figure 5 with $\mathsf{a}, \mathsf{b} \in \mathcal{L}_{\text{leaf}}$. ($A_3$ accepts $(\mathsf{a} \mid \mathsf{b})^*\$$.) Observe it has a ternary choice. We also assume that b has a key constraint and domain cardinality 1.

Consider option (1) above. We show two ways of turning the ternary choice into a binary one (there is a third possibility but it is not considered here).

First, one decides whether a is produced or not and then (if an a is not produced) whether b is produced or whether we are done with the children of r. We use a probability assignment t-prob: we choose to produce a with probability t-prob(q_0, a) and to produce b (given that we have not produced a) with probability t-prob(q_0, b). As before, we use continuation tests to avoid reaching dead ends during generation, and in the probability learning, as in Algorithm 2. Alternatively, one can choose whether we are done with r first, and, if we are not done, whether we produce a or b. This yields t-prob′. Take the singleton corpus <r><a/><b/></r>. The transition probabilities are:

$$\begin{cases} \text{t-prob}(q_0, \mathsf{a}) = \frac{1}{3} & \text{t-prob}'(q_0, \$) = \frac{1}{3} \\ \text{t-prob}(q_0, \mathsf{b}) = 1 & \text{t-prob}'(q_0, \mathsf{a}) = \frac{1}{2} \end{cases}$$

Then the probability of generating the corpus is $\frac{1}{3} \times \frac{2}{3} \times 1 \times \frac{2}{3} = \frac{4}{27}$ using the first alternative, and $\frac{2}{3} \times \frac{1}{2} \times \frac{2}{3} \times \frac{1}{2} \times \frac{1}{3} = \frac{1}{27}$ using the second one: the quality of the generator depends of the way the choice has been made binary.

Now consider (2). We keep the ternary choices but assign a probability to each possible subset of the transitions of size more than 1. For the example, this yields:

a, b, \$ **are all available**	**only** a, \$ **are available**
$\text{t-prob}(q_0, \mathsf{a}) = \frac{1}{2}$	$\text{t-prob}'(q_0, \mathsf{a}) = 0$
$\text{t-prob}(q_0, \mathsf{b}) = \frac{1}{2}$	
$\text{t-prob}(q_0, \$) = 0$	$\text{t-prob}'(q_0, \$) = 1$

which gives a probability of generating the corpus of $\frac{1}{2} \times \frac{1}{2} \times 1 = \frac{1}{4}$. In both cases, we can obtain an optimal generator *for this particular class of generators.* For (1), this suffers from the inelegance of the arbitrary ordering of the transitions that is chosen and affects the outcome. For (2), this may result in a large number of parameters.

5.2 Restart Generators

We next consider r-generators. First, we show that given a generated document skeleton, we can check its validity efficiently (and if invalid, restart). Then, however, we show that the *number of restarts* may be unboundedly large; and this can hold particulary for r-generators that are optimal (i.e., best fit to the corpus). We start by defining the problem of checking validity for document skeletons.

Definition 7 Given as input (1) a schema with constraints $S = \langle S^{\mathrm{u}}, C\rangle$, (2) a skeleton d valid for S^{u}, the $\texttt{VALID}(S, d)$ *problem* is to decide whether d is valid w.r.t. S.

Proposition 4 $\texttt{VALID}(S, d)$ *can be decided in PTIME.*

Proof We consider again the schema satisfiability test from [17], which is checked via the satisfiability of a formula $\varphi \wedge \psi$. The variables $x_1, ..., x_n$ in the formula represent the numbers of occurrences of nodes labeled with $\mathsf{a}_1, ..., \mathsf{a}_n$. In this case, if we want to test the validity of a skeleton $d = (V, E)$, we take the assignment for each x_i to be $\#_i^d = |\{v \in V \mid \text{lbl}(v) = \mathsf{a}_i\}|$. Since d is valid for S^{u}, this assignment satisfies φ, which is the part of the formula expressing the validity of the document for the schema S^{u}.

It is left to find a satisfying assignment for ψ, that expresses validity with respect to the constraints in C. For that we must also assign values to the variables $y_1, ..., y_n$, which represent the number of unique values for each label. If we find such values we can be sure that there exists a valid assignment for the leaf values, for the generated document skeleton. Let us construct a directed graph $G = (V, E)$, such that there is a node $v(y_i)$ for every variable, node $v(0)$ and $v(\#_i^d)$ for $1 \leqslant i \leqslant n$, and add the edges $(v(0), v(y_i))$, $(v(y_i), v(\#_i^d))$ for each i. In G a directed edge (a, b) expresses that $a \leqslant b$. ψ connects, using $\wedge$, sub-formulas of the 4 following types:

$$(1)\ y_i \leqslant x_i \qquad (2)\ y_i = 0 \leftrightarrow x_i = 0 \qquad (3)\ y_i = x_i \qquad (4)\ y_i \leqslant y_j$$

In addition, for each domain constraint $\mathsf{a}_i \in \mathrm{dom}(\mathsf{a}_i)$ we add $y_i \leqslant |\mathrm{dom}(\mathsf{a}_i)|$ (recall that we only need to verify validity w.r.t. constraints of finite domains).

For each sub-formula, We will replace each x_i with its assigned value, and update G in the process, as follows. Sub-formulas of the first kind can be ignored, as they are already expressed in G; for sub-formulas of the second kind, if indeed $x_i = 0$, we will add the edge $(v(y_i), v(0))$; otherwise we will add $(v(1), v(y_i))$, creating a new node $v(1)$ if necessary; for $y_i = x_i$ we will add $(v(\#_i^d), v(y_i))$; for $y_i \leqslant y_j$ we will add $(v(y_i), v(y_j))$; and finally for $y_i \leqslant k$ we will add $(v(y_i), v(k))$, creating $v(k)$ if necessary. Then we will take $G^* = (V, E^*)$, the transitive closure of G.

We claim that ψ is satisfiable iff in G^* there exists no edge $(v(k), v(k'))$ s.t. $k' < k$.

For the one direction, assume that there exists no such $(v(k), v(k'))$, and let us assign to each y_i the minimal k s.t. $(v(y_i), v(k)) \in E^*$ (i.e., the lowest upper bound for y_i). By construction there must exist such a k. It is straightforward to verify that every sub-formula of ψ is satisfied. E.g., consider sub-formula of the form $y_i = x_i$. By construction, $(v(y_i), v(\#_i^d))$ and $(v(\#_i^d), v(y_i))$ are in E, E^*. Assume by contradiction that y_i is assigned $k < \#_i^d$; then $(v(y_i), v(k)) \in E^*$ and thus also $(v(\#_i^d), v(k))$, which yields a contradiction. Assigning y_i a value $k > \#_i^d$ contradicts the choice of minimal upper bounds as values.

Now, assume that there exists such $(v(k), v(k'))$. By the definition of transitive closure there is a path from $v(k)$ to $v(k')$ in E, representing a sequence of inequalities $k \leqslant z_1, z_1 \leqslant z_2, ..., z_t \leqslant k'$, which cannot all be satisfied together. Thus ψ is not satisfiable.

Finally, generating G and G^*, and checking for an edge $(v(k), v(k'))$ s.t. $k' < k$ can all be performed in time polynomial w.r.t the size of the schema and document skeleton. $\square$

Quality of an r-generator vs. restart overhead. We next examine how many times we will restart (i.e., what is the expected number of generated invalid documents). In particular, we show that there is a tradeoff between the optimality of an r-generator, and its restart overhead.

Example 6 Consider a simple schema S_{tradeoff}, which consists of a root label r, whose automaton A_{tradeoff} is depicted in Figure 6. The regular language of this automaton is $\mathsf{a}^*\$$. Let $\mathcal{L}_{\text{leaf}} = \{\mathsf{a}\}$ and let the set of constraints $C = \{\text{uniq}(\mathsf{a}), \mathsf{a} \in \{\mathsf{0}\}\}$ (a can have only one value, 0).[5] Consider a document corpus which consists only of the document d, whose root has a single child a with value 0.

The only parameter that can be chosen in an r-generator is the probability α to choose the transition from q_0 to itself. In a single invocation, the probability of generating d is $\alpha \cdot (1 - \alpha)$, the probability of generating a document with only a root is $1 - \alpha$, and the probability of generating an invalid document (and restarting) is α^2.

Now, maximizing the quality of the generator means maximizing the probability for generating d. The probability of generating d is the probability of generating it in the first invocation, in the second one, etc., that is (assuming $\alpha < 1$, if $\alpha = 1$ then the probability is 0): $\sum_{k=0}^{+\infty} \alpha(1-\alpha)(\alpha^2)^k = \alpha(1-\alpha)\frac{1}{1-\alpha^2} = \frac{\alpha}{1+\alpha}$

This function is monotonically increasing for $\alpha \in [0; 1)$. Let us choose α to be $1 - \varepsilon$, for some arbitrarily small $\varepsilon > 0$. The expected number of restarts for this generator can be computed to be $\frac{1-(1-\alpha^2)}{1-\alpha^2} = \frac{(1-\varepsilon)^2}{1-(1-\varepsilon)^2}$, which shows that the expected number of restarts tends towards $+\infty$ as $\varepsilon \to 0$ (i.e., as the generator gets closer to optimal).

Remark 3 A conclusion from the example is that maximizing the corpus likelihood may not be the best quality measure for r-generators, and finding better measures for such generators will be considered in future research.

6 Data Values

So far, we have only considered generating document skeletons. To complete the picture, we finally discuss the generation of leaf values, to be injected into such skeletons. While the ideas provided here shed light on value generation, we believe that this is not the final word on the subject, and this direction deserves to be further investigated. We start by considering the generation of values given some probabilistic distribution. Then, we consider additional information that may help us improve the quality of the value generator.

6.1 Generating Values from Distributions

We assume that for each leaf label $\mathsf{a} \in \mathcal{L}_{\text{leaf}}$ we are given some probabilistic distribution v-dist$_{\mathsf{a}}$ on values, e.g., uniform distribution on a finite domain, Zipfian, etc. We also assume that the distribution is discrete. Distributions could be, e.g., learned in practice from the corpus [13]; such a learning process is out of the scope of the present paper.

[5] We could also construct more complicated examples, where the value domains are infinite.

In the absence of constraints, value generation is rather simple: given a document skeleton, for each a-labeled leaf, randomly choose a value according to v-dist$_{\mathsf{a}}$. The difficulty comes from constraints, that we now consider.

Construction. For the domain constraints, we can simply assume that the distribution gives non-zero (zero) probability to every value in (out of) the domain. (Otherwise, as mentioned in Section 2.3, the "actual" domain of each a must be computed and this domain must also be considered in the continuation test.)

Then what remains is to verify that the value assignment satisfies the key and inclusion constraints. To that end, we propose the following algorithm. For every a_i, let y_i be a variable representing the number of unique values for a_i-labeled leaves.

1. Create a graph representing the inclusion constraints on leaf labels; split it to strong connectivity components (SCCs) and find a topological order σ on those SCCs.
2. Construct the transitive closure graph G^* representing the constraints sub-formulas as in the proof of Prop. 4.
3. Start with a label a_i from the "smallest" (i.e., only included and not including) SCC according to σ.
4. Randomly choose an a_i-labeled leaf and randomly choose a value for it according to v-dist$_{\mathsf{a}_\mathsf{i}}$. Then assign this value to some (randomly chosen) a_j-labeled leaf, for every a_j that (transitively) includes a_i, if an a_j-labeled leaf with this value does not exist yet.
5. Update the lower and upper bounds of y_j, for every a_j for which a value was generated in the previous step.
6. Treat the new lower and upper bounds as new sub-formulas and update G^* accordingly; use G^* to perform the PTIME validity test from the proof of Prop. 4, on the skeleton with partial value assignment.
7. If the partial assignment is not valid, "rollback" all the added occurrences of the value, and return to step 4.
8. Repeat for all the a_i-labeled leaves, then do the same for every other member of a_i's SCC, then move on the next SCC in σ and so on, until all leaves have values.

One can show that the algorithm is correct in the sense that it generates a valid document with respect to the constraints, and that termination of the algorithm is guaranteed.

6.2 Old vs. New Values

We note that additional information about the correlation between values can be helpful for the generation. In particular, we consider information on the likelihood of values in specific leaves to *repeat old values* that were already generated. This information could for instance be learned during the corpus type-check. We suggest here to encode this information, during the generation of the document skeleton, as additional annotations *old* or *new* for each leaf. This information indicates whether the value for this leaf should be drawn out of the values already chosen or whether a new value should be picked. Then the value generation phase follows the technique of Section 6.1, while also respecting these annotations when choosing a value.

Example 7 The new kind of skeletons with *old* and *new* annotations will be referred to as *annotated document skeletons*, exemplified next. We next give an example of an annotated document skeleton, based on the schema S_0 from Example 5. We use the XML attribute ann = "*old*/*new*" to denote the annotation in this example. Consider the following (full) XML document.

```
<Dept>
  <Head>Martha B.</Head>
  <Seniors>
    <Emp>
      <Name>Martha B.</Name>
      <Tel>123-5234</Tel>
      <Tel>123-5357</Tel>
    </Emp>
    <Emp>
      <Name>Max</Name>
      <Tel>123-5234</Tel>
      <Tel>123-5357</Tel>
    </Emp>
  </Seniors>
  <Juniors>
    <Emp>
      <Name>Martha C.</Name>
      <Tel>123-5234</Tel>
      <Tel>123-5358</Tel>
    </Emp>
  </Juniors>
</Dept>
```

The annotated skeleton of the document above is:

```
<Dept>
  <Head>Martha B.</Head>
  <Seniors>
    <Emp>
      <Name ann="new"></Name>
      <Tel  ann="new"></Tel>
      <Tel  ann="new"></Tel>
    </Emp>
    <Emp>
      <Name ann="new">Max</Name>
      <Tel  ann="old"></Tel>
      <Tel  ann="old"></Tel>
    </Emp>
  </Seniors>
  <Juniors>
    <Emp>
      <Name ann="new"></Name>
      <Tel  ann="old"></Tel>
      <Tel  ann="new"></Tel>
    </Emp>
  </Juniors>
</Dept>
```

Recall that there is a key constraint on employee names, and thus all the names have new values (and annotations accordingly). Some of the phone lines, however, are shared by two employees or more, and accordingly some of the Tel leaves are annotated with *old*.

We denote by D_{ann} the bag of annotated skeletons of all documents in D_{full}.

We next present and compare two alternative ways of generating annotated skeletons: an *offline generator*, that adds annotations to skeletons after they have been generated; and an *online generator*, that generates the skeleton along with annotations. To simplify definitions, we assume in the sequel that both generators are based on an optimal binary ct-generator (we will explain how). In both models, we associate each transition (q, a) in the schema ($\mathsf{a} \in \mathcal{L}_{\text{leaf}}$), to a probabilistic word generator $A_{(q,\mathsf{a})}$, that produces either an *old* or a *new* annotation. We denote the probability of $A_{(q,\mathsf{a})}$ to generate *new* by $\text{t-prob}_{new}(q, \mathsf{a})$. We next outline the two generators, show we can find optimal probabilities for each, and that, interestingly, each generator gives better quality for different inputs.

Offline generator. This kind of generator gets as input a document skeleton (which we assumed is generated by an optimal ct-generator), and annotates its leaves as follows. The generator traverses the leaves of the input skeleton (in a BF-LTR order), and for each leaf performs a validity test for the two possible annotations.[6] Assume this leaf was generated by the transition (q, a) of the ct-generator. If both options are valid, the offline generator uses $A_{(q,\mathsf{a})}$ to generate an annotation for the leaf; otherwise it annotates the leaf with the only valid option.

Algorithm 3 details the learning of probabilities for an offline generator. Note that it makes calls to `IsValid`, which tests for the validity of the partially annotated document represented by the document skeleton and the partial annotation list. We prove in the sequel that the algorithm learns optimal probabilities.

Online generator. In this kind of generator, we "embed" each word generator $A_{(q,\mathsf{a})}$ into its corresponding transition (q, a) in the ct-generator. This means that, during generation, after choosing some transition (q, a) (where $a \in \mathcal{L}_{\text{leaf}}$) we also invoke $A_{(q,\mathsf{a})}$ for generating an annotation to this leaf. Continuation tests are performed both before choosing (q, a) *and* before choosing the annotation in $A_{(q,\mathsf{a})}$. The key point here is that the annotations of the partial document can be encoded as constraints, and then the continuation test detailed in the proof of Proposition 3 may be used.

[6] Note that the choice of the number of new and old values determines the number of unique values for each label, thus the validity test may be done in the same way as the algorithm in the previous section.

```
Input: A schema with constraints $S = \langle S^{\mathrm{u}}, C\rangle$, a corpus of documents $D_{\mathrm{ann}}$ accepted
        by $S$
Output: An offline generator $G$ conforming to $S$
foreach transition $(q, \mathsf{a})$ in an automaton of $S$, where $\mathsf{a} \in \mathcal{L}_{\mathrm{leaf}}$ do
    $\mathrm{freq}_{old}(q, \mathsf{a}) \longleftarrow 0$;
    $\mathrm{freq}_{new}(q, \mathsf{a}) \longleftarrow 0$;
foreach $d \in \mathrm{supp}(D_{\mathrm{ann}})$ do
    $anns \longleftarrow [ann_1, ..., ann_n]$ the list of annotation of $d$ in a BF-LTR order;
    $d_{\mathrm{skel}} \longleftarrow$ the skeleton of $d$;
    foreach $ann_i$ in $anns$ do
        $(q, \mathsf{a}) \longleftarrow$ the transition that genrated the $i$-th leaf in $d$;
        if $ann_i = old$ and IsValid($d_{\mathrm{skel}}, [..., ann_{i-1}, new], C$) then
            $\mathrm{freq}_{old}(q, \mathsf{a}) \longleftarrow \mathrm{freq}_{old}(q, \mathsf{a}) + 1$
        if $ann_i = new$ and IsValid($d_{\mathrm{skel}}, [..., ann_{i-1}, old], C$) then
            $\mathrm{freq}_{new}(q, \mathsf{a}) \longleftarrow \mathrm{freq}_{new}(q, \mathsf{a}) + 1$
$G \longleftarrow$ an offline generator based on $S$ where $\forall q \in \mathcal{Q}, \mathsf{a} \in \mathcal{L}_{\mathrm{leaf}}$ t-prob$_{new}(q, \mathsf{a}) = \frac{1}{2}$ if
$\mathrm{freq}_{old}(q, \mathsf{a}) + \mathrm{freq}_{new}(q, \mathsf{a}) = 0$, otherwise t-prob$_{new}(q, \mathsf{a}) = \frac{\mathrm{freq}_{new}(q,\mathsf{a})}{\mathrm{freq}_{new}(q,\mathsf{a})+\mathrm{freq}_{old}(q,\mathsf{a})}$;
return $G$;
```

Algorithm 3: Offline algorithm

The quality of offline and online generators w.r.t. a corpus D_{ann} can be defined, in the same spirit as the optimality of skeleton generators, as the multiplication of the probabilities to generate the annotated skeletons in D_{ann}.

We next define the quality of an offline generator, then show the optimality of our construction.

Definition 8 Given an offline generator G and for every annotated document skeleton d, let $\Pr(d \,|\, G, d_{\mathrm{skel}})$ be the probability for G to generate the correct annotations of d given its skeleton d_{skel} as an input. Let D_{ann} be an annotated document skeleton corpus. The *quality* of G with respect to D_{ann}, is then

$$\mathrm{quality}(G, D_{\mathrm{ann}}) = \prod_{d \in \mathrm{supp}(D_{\mathrm{ann}})} \Pr(d \,|\, G, d_{\mathrm{skel}})^{D_{\mathrm{ann}}(d)}$$

Theorem 6 *For a given schema with constraints S and an annotated skeletons corpus D_{ann}, we can compute the optimal offline generator in PTIME in $|S|$ and $|D_{\mathrm{ann}}|$, and the optimal online generator in FP^{NP} w.r.t. $|S|$ and PTIME w.r.t. $|D_{\mathrm{ann}}|$.*

Proof Let us start with finding an optimal offline algorithm. Let G' be an offline generator for some schema with constraints S. The quality of G' w.r.t. D_{ann} can be computed as a function of the probabilities assigned to each transition annotation: in cases where the two options are possible (according to the validity test), for each *new*-annotated leaf we multiply by the probability of annotating with new p, and for each *old*-annotated leaf we multiply by 1 minus that probability, to get

$$\begin{aligned}\mathrm{quality}(G,D) &= \prod_{d\in \mathrm{supp}(D_{\mathrm{ann}})} \Pr(d\,|\,G, d_{\mathrm{skel}})^{D_{\mathrm{ann}}(d)}\\ &= \prod_{\substack{(q,\mathsf{a})\text{ in } S,\\ \mathsf{a}\in\mathcal{L}_{\mathrm{leaf}}}} \Big(p(q,\mathsf{a})^{\mathrm{freq}_{new}(q,\mathsf{a})}\cdot(1-p(q,\mathsf{a}))^{\mathrm{freq}_{old}(q,\mathsf{a})}\Big)\end{aligned}$$

Since each variable $p(q,\mathsf{a})$ is independent and following Lemma 1, this maximizes when for every (q,a), $p(q,\mathsf{a}) = \text{t-prob}_{new}(q,\mathsf{a})$. These are exactly the probabilities assigned by Algorithm 3.

To verify that Algorithm 3 is in PTIME, note that the initialization, traversal and finalization steps are linear in the size of the corpus and the schema, and that validity test is in PTIME.

Now, for the optimal online algorithm. First, let us sketch how to encode the annotations of the partial document, in order to use the same continuation test for it. For each leaf label a, let y be a variable denoting the number of unique values for a-labeled leaves, x a variable denoting the number of a-labeled leaves, and $\#_{old}$, $\#_{new}$ be the numbers of *old*-annotated and *new*-annotated a-labeled leaves respectively. Then $\#_{new}$ is a lower bound for y, and $x - \#_{old}$ is an upper bound. These bounds can be encoded as constraints in the model of [17], which we use for performing the continuation test. This is without increasing the complexity beyond the usual complexity of the continuation test, presented in Proposition 3.

Second, let us sketch the probability learning phase for this type of generator. The annotated documents in the corpus are traversed; as in a ct-generator, for every transition (q,a) taken during the traversal, we check (using the continuation test adjusted for annotated partial documents) whether there exists another valid choice; as in a ct-generator, we only take this choice into account for probability learning if there was indeed another valid option. Then, if $\mathsf{a}\in\mathcal{L}_{\mathrm{leaf}}$, we use $A_{(q,\mathsf{a})}$ as an acceptor for the annotation of the leaf. We check, using the continuation test, whether the other annotation was also possible for this leaf. If so, we take this choice into account in the computation of $\text{t-prob}_{new}(q,\mathsf{a})$ (which is, as usual, the relative frequency).

Using the same technique as in the optimality proof for offline generator above, we can show that this algorithm optimizes all the parameters of the online generator, thus yielding an optimal online generator. The complexity is the same complexity as in the learning phase of a ct-generator, i.e., $\mathrm{FP}^{\mathrm{NP}}$ in the size of the schema, since the dominant complexity factor is again the continuation test. Traversal on the corpus and the continuation tests are only polynomial w.r.t. the corpus size. $\square$

Comparing offline and online generators. We have presented two possible models for the generation of data values, and have shown algorithms for obtaining optimal generators . This raises the question of which generator is “superior”. The following proposition states that the offline and online generators are incomparable in terms of their quality.

Proposition 5 *There exist schemas with constraints S, S' and annotated skeleton corpora D_{ann}, D'_{ann}, such that the quality of the optimal offline generator w.r.t. S, D_{ann} (resp. S', D'_{ann}) is lower (resp. higher) than the quality of the optimal online generator w.r.t. the same input.*

Proof We next provide two examples, in the first the optimal online algorithm performs better than the optimal offline algorithm, and in the second the optimal offline algorithm performs better.

Consider the schema S, where as usual $\mathcal{L}_{\text{inner}} = \{\mathsf{r}\}$, r is the root label, and $A_{\mathsf{r}} = A_{\text{on-off}}$ is depicted in figure 7. The set of constraints C will be $\{\mathsf{a} \subseteq \mathsf{b}\}$ Our input corpus D will consist this time of the document

```
<r>
    <a>0</a>
    <a>1</a>
    <a>2</a>
    <b>0</b>
    <b>1</b>
    <b>2</b>
</r>
```

In the next table we describe the probabilities obtained from applying both the online and the offline learning methods on D_{ann}, the corpus of annotated skeletons for D. Each cell contains the probabilities for some transition (q, a) in the form t-prob(q, a) (t-prob$_{new}(q, a)$). In the offline column, t-prob(q, a) refers to the probability in an optimal ct-generator. $p?$ denotes a probability that was set arbitrarily due to the lack of statistics from the corpus.

	Online	**Offline**
$(\mathbf{q_0}, \mathsf{a})$	$1\ (\frac{1}{2}?)$	$1\ (\frac{1}{2}?)$
$(\mathbf{q_1}, \mathsf{a})$	$\frac{2}{3}\ (1)$	$\frac{2}{3}(1)$
$(\mathbf{q_1}, \mathsf{b})$	$\frac{1}{3}\ (\frac{1}{2}?)$	$\frac{1}{3}\ (\frac{1}{2}?)$
$(\mathbf{q_2}, \mathsf{b})$	$0\ (\frac{1}{2}?)$	$\frac{2}{3}(1)$
$(\mathbf{q_2}, \$)$	1	$\frac{1}{3}$

The quality in the online case is $1 \cdot \left(\frac{2}{3} \cdot 1\right)^2 \cdot \frac{1}{3} \cdot 1 = \frac{4}{27}$. In order to make a fair comparison, we compare this online quality to the quality of the optimal offline algorithm *times* the quality of a ct-generator optimal for D. This gives 1 (as there are no choices for the offline generator) multiplied by $1 \cdot \frac{2}{3}^2 \cdot \frac{1}{3} \cdot \frac{2}{3}^2 \cdot \frac{1}{3} = \left(\frac{4}{27}\right)^2$, which is lower.

For the second counter-example, let S' be the same as S, except for the set of constraints, which will be $\{\mathsf{b} \subseteq \mathsf{a}, \mathrm{uniq}(\mathsf{b})\}$. The input corpus D' will be

```
<r>
    <a>0</a>
    <a>0</a>
```

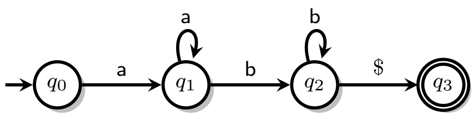

Fig. 7: The $A_{\text{on-off}}$ DFA

```
    <a>0</a>
    <a>1</a>
    <b>0</b>
    <b>1</b>
</r>
```

The probabilities learned for the optimal online and offline algorithms in this case are:

	Online	**Offline**
$(\mathbf{q_0}, \mathsf{a})$	$1\ (\frac{1}{2}?)$	$1\ (\frac{1}{2}?)$
$(\mathbf{q_1}, \mathsf{a})$	$\frac{3}{4}\ (\frac{1}{3})$	$\frac{3}{4}(0)$
$(\mathbf{q_1}, \mathsf{b})$	$\frac{1}{4}\ (\frac{1}{2}?)$	$\frac{1}{4}\ (\frac{1}{2}?)$
$(\mathbf{q_2}, \mathsf{b})$	$1\ (\frac{1}{2}?)$	$\frac{1}{2}(\frac{1}{2}?)$
$(\mathbf{q_2}, \$)$	0	$\frac{1}{2}$

Then the quality of the online algorithm is $1 \cdot \left(\frac{3}{4} \cdot \frac{2}{3}\right)^2 \cdot \frac{3}{4} \cdot \frac{1}{3} \cdot \frac{1}{4} \cdot 1 \cdot 1 \cdot 1 = \frac{1}{64} = \frac{16}{1024}$ and the quality of the offline algorithm, multiplied by that of the optimal ct-generator is $1 \cdot \left(\frac{3}{4}\right)^3 \cdot \frac{1}{3} \cdot \left(\frac{1}{2}\right)^2 = \frac{27}{1024}$, which is higher. □

7 Extension to a Typed Model

Our schema model used so far in the paper was based on DTDs, i.e., the derivation of document nodes and values was based only on the node labels. However, *typed models*, where each node is assigned an underlying type during the verification process, are very useful in practice (e.g., W3C XML Schema, aka XSD, or Relax NG). These models allow decoupling the label of a node from its function or meaning in the document, and thus extend the domain of possible document languages.

We next extend our previous model with types. As determinism is an essential property of our previous model, our typed model is also deterministic. This means that as before, every document will only have a single possible generation trace for a given schema. But moreover, the assignment of types to nodes during the verification process is also deterministic. In this our model differs from other models in literature such as Specialized DTDs [31], where the uniqueness of type assignment is not required.

*Model.*We formalize the definition of a typed schema below. $\mathcal{L}$, $\mathcal{Q}$ are used as before, and we further assume a finite domain $\mathcal{T}$ of node types.

Definition 9 (Typed schema) A typed schema S is a tuple $(\mathsf{r}, t_\mathsf{r}, \mathcal{A}_\downarrow)$, where r is the root label as in a schema, t_r is the root type, and $\mathcal{A}_\downarrow$ is a partial function mapping a type $t \in \mathcal{T}$, instead of a simple DFA, to a Mealy Machine A_t s.t., as in a schema-deriving automaton, its states are taken from $\mathcal{Q}$ and its transitions inputs are labels from $\mathcal{L}$, but in addition, each transition is annotated with an output from $\mathcal{T}$. For every document node v, type(v) returns the type associated with v by S, i.e., the output of the transition that was triggered by the label of v.

A document d is accepted by S if the label of root(d) is r, and given that we associate root(d) with the type t_r, for every inner node v, *once it is associated a type* $t \in \mathcal{T}$, the machine A_t accepts $\mathrm{lbl}_\downarrow(v)$.

We assume the same restrictions on the automata structures as we before. Note that the verification process is indeed deterministic, as each node triggers exactly one (deterministic) transition and thus may be associated with exactly one type. Also note that this model allows for different labels to be associated with the same type, and for different types to be associated with the same label (in different locations). The output on \$-annotated transitions is insignificant, since it is not associated with any node, and thus we can fix some arbitrary output for such transitions, say, t_r.

Now, if we look at a typed schema as a non-deterministic generator, then each non-deterministic transition choice is equivalent to choosing both the label and a compatible type of the next child (or to stop generating children, as before). Again, each generated document d has only one possible generation trace (using the same definition), since by the type of the root and the labels of its children we can determine the type of each child; then the same for their children, and so on. This model extends in a straightforward manner also to a typed nd-generator, typed p-generator, typed ct-generator, etc.

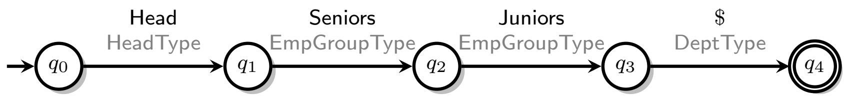

Fig. 8: The A_{DeptType} Mealy Machine

Example 8 The A_{DeptType} (depicted in Fig. 8) is similar to A_{Dept}, but has an output on each edge (marked by gray font). This allows, for instance, using the same type EmpGroupType for the nodes with different labels Seniors and Juniors.

*Results.*We next show that, although the typed schema model is strictly more expressive, all our results for the DTD-based model still hold.

Proposition 6 *Typed schema is strictly more expressive than un-typed (normal) schema.*

Proof To prove that typed schema is more expressible, we can define for any un-typed schema S an equivalent typed schema S' (accepting exactly the same documents). We can define S' such that the type on every transition is the same as the label, and the automatons and the root label are the same as in S.

For strictly more expressibility, the following example shows a typed schema S such that no un-typed schema is equivalent to S. Let $S = (\mathsf{r}, t_\mathsf{r}, \mathcal{A}_\downarrow)$. $\mathcal{L} = \mathcal{L}_{\text{inner}} = \{\mathsf{r}\}$. The language of the root type is $L(A_{t_\mathsf{r}}) = \{\mathsf{r}\$\}$. The type on the transition labeled r in this automaton is t_2, and $L(A_{t_2}) = \{\$\}$. This means that S accepts exactly one document, `<r><r></r></r>`. Using an un-typed schema, if a node labeled r can have a child node labeled r, the schema has to be recursive, and thus accept an unbounded number of documents. Hence, no un-typed schema is equivalent to S. □

The main technique we use for transferring the results to the typed model, is by translating the typed schema to a particular un-typed schema, referred to as "dual". The node labels of the dual schema are labels or pairs of labels and types from the original schema.

Definition 10 (Dual un-typed schema) Given a typed schema $S = (\mathsf{r}, t_\mathsf{r}, \mathcal{A}_\downarrow)$ we define $\text{dual}(S)$ as the schema $(\langle \mathsf{r}, t_\mathsf{r}\rangle, \text{dual}(\mathcal{A}_\downarrow))$, where $\text{dual}(\mathcal{A}_\downarrow)$ is defined as follows. If $\mathcal{A}_\downarrow$ maps a type t to an automaton A_t, for every label $\mathsf{a} \in \mathcal{L}_{\text{inner}}$, $\text{dual}(\mathcal{A}_\downarrow)$ maps $\langle \mathsf{a}, t\rangle$ to a DFA $\text{dual}(A_t)$. The start states of A_t and $\text{dual}(A_t)$ are the same. If $\mathsf{a} \in \mathcal{L}_{\text{inner}}$, for every transition $\delta(q, \mathsf{a}) = q'$ in A_t with output t', there is a transition $\delta(q, \langle \mathsf{a}, t'\rangle) = q'$ in $\text{dual}(A_t)$. Otherwise, for $\mathsf{a} \in \mathcal{L}_{\text{leaf}}$, every transition $\delta(q, \mathsf{a}) = q'$ in A_t is also in $\text{dual}(A_t)$ (without the output). Every transition $\delta(q, \$) = q'$ in A_t is also in $\text{dual}(A_t)$.

For every document skeleton d accepted by S there is exactly one dual document skeleton $\text{dual}(d)$ accepted by $\text{dual}(S)$. $\text{dual}(d)$ is identical to d, except that the labels of inner nodes also include their types. The fact that $\text{dual}(d)$ is unique follows from the type determinism of S.

The following lemma holds.

Lemma 2 $\text{dual}(S)$ *can be constructed in time linear in $|S|$, and hence the size of the result is also linear in $|S|$.*

Proof Computing the dual transition for each original transition can be done in $O(1)$. It is left only to associate the dual inner labels with the relevant automaton. In general, the size of $(\mathcal{L}_{\text{inner}} \times \mathcal{T}) \cup \mathcal{L}_{\text{leaf}}$ from which the dual labels are taken is quadratic in S. However, we only need to consider label-type combinations that actually appear in the transition function of S, which can be done in time linear in S. These combinations can be associated with the relevant automaton in time $O(1)$. □

Using the lemma above, we can now prove the main result of this section.

Proposition 7 *The equivalents of Theorems 1, 2, 3, 4 and 5, and Propositions 1, 2, 3 and 4 for typed schemas hold.*

Proof Let us now give a proof for each of the equivalents of the theorems and propositions mentioned above.

Prop. 1 can be proved as in the original proof (generation trace definition is not affected by the use of types).

Thm. 1 holds for every generator that has a fixed probability for generating each document skeleton. Thus, it holds also for typed p-generators.

The proof of Thm. 2 is based on the relative transition usage during the corpus verification process. Whether these transitions have outputs or not, does not affect the correctness of the proof.

To prove Thm. 3 for the typed case, we can construct, for the optimal typed p-generator G a dual un-typed p-generator dual(G) (conforming to dual(S)), where each dual transition in is dual(G) assigned the same probability as the original transition in G. Note that dual(G) is the same as the output of Algorithm 1 on dual(S) and the corpus of dual documents. Thus, by Thm. 3, dual(G) has termination probability 1. Since for every final dual document generated by dual(G) there is a final document generated by G with the same probability, it follows that G also has termination probability 1.

To prove Props. 2 and 3 we can construct the dual schema in linear time. Here, the fact that the dual documents have the same leaves as the original ones allows us to have the same constraints in the dual schema as in the original one. Thus, the dual schema is satisfiable iff the original schema is. Deciding on the dual schema satisfiability, according to Prop. 2, is NP-complete. Similarly, if we take the dual generation trace (that includes types), there is a continuation for the dual schema, dual trace and dual transition iff there is a continuation for the original schema, trace and transition.

The proof for Thm. 4 for the typed case only requires replacing the oracle used in the proof by one that works for typed schemas. Its complexity follows from the previous two propositions.

Thm. 5 works also for the typed case, if we take a schema where the types and labels are the same.

Finally, Prop. 4 can be proved by first computing the dual schema and dual document skeleton, in time linear in the original schema and document sizes. Then the dual document skeleton is valid w.r.t. the dual schema iff the original document is valid w.r.t. the original schema. □

The results of Section 6 also transfer to the typed case. The proofs can be used in a straightforward manner, following Prop. 7.

8 Related Work

Various models for probabilistic XML documents exist in the literature (e.g. [16, 5]); see [6] for a review of such models and a comparison of their expressiveness. The model considered here is not of a probabilistic document but rather of a probabilistic *schema*; in particular our model allows to define infinitely many documents, in contrast to the finitely many documents (*worlds*) in the models above. Probabilistic schemas were also considered in [9] that suggested the use of recursive Markov chains [18] for modeling and querying probabilistic XML. The model of [9] can be seen as a straightforward extension of p-generators where global states and labels are uncoupled. There are also various models for generation of XML documents (e.g., for testing): in [15] the author suggests a language for specifying (manually) desired constraints on generated documents and then shows how to obtain a (non-probabilistic) generator conforming to these constraints (when possible); in [8] the suggested language allows to (again manually) define a probabilistic distribution on local parts of the documents; and the recent [7] suggests a way for *uniform* sampling of documents conforming to a schema. To our knowledge, no prior work deals with learning a maximum likelihood estimator of a given example XML corpus, in contrast to the present work.

As noted in Remark 2, the different models presented in this paper, including probabilistic, and constrained generators, can also be captured by Active XML [4] and tree rewriting. For instance, in AXML a *random* function can be used to introduce probabilistic choices in the tree rewriting; however, much more complicated functions, including ones performing queries on the tree structure, may also be used. To enforce a BF-LTR order of rewriting, *guard* functions may be used; the guards may also be used enforce other, more complicated orders. This suggests a variety of interesting research questions that can be studied in future work.

The starting point of this work assumes that we are given a schema; there are many works on *schema inference* from a corpus of documents (e.g. [30, 10,12,23,28,21], and the work on key approximation in [22]). These works complement our work in two senses: first, we can use the inferred schemas as inputs; second, our results can be used to measure the quality of inferred schemas, based on the quality of the optimal generator conforming to them. There are other measurements for schema quality (see suggestions in recent work of [27,7]), and combining them with our measurement is an interesting research direction.

Our work also has strong connections with the works of [17,19]. They consider satisfiability tests for XML schemas with constraints, and prove that these tests are NP-complete; we used an adaptation of this result to show NP-completeness of the continuation tests. Note that in contrast to our work, the works of [17,19] focus on satisfiability, and thus the models used there are not probabilistic.

On the technical level, our work is also related to other (non-XML) probabilistic models. In particular, *Probabilistic Context-Free Grammars* (PCFGs) [25,14] are a common model for the probabilistic generation of strings, used heavily in natural language processing, bioinformatics, and more. We have noted that our algorithm for the non-constrained case is inspired by [14]; we are not aware of an equivalent result in the presence of constraints on strings. Applying our results to this area is an intriguing future research task.

9 Conclusions

We have studied the problem of finding an optimal probabilistic model for a given schema and corpus of XML documents. We have shown how to view the model as a probabilistic generator. We have provided elegant solutions for two cases: with and without constraints. For the former, we have studied two kinds of generators, ct-generators and r-generators, provided algorithms for finding optimal generators, and analyzed the advantages and disadvantages of both kinds. Finally, we have considered the generation of data values, to be fed into the generated XML structure.

We believe that there are still many open problems to be investigated in future research. For example, recall that a ct-generator always generates valid documents (but generation is costly), while an r-generator avoids the cost of continuation test but may restart often. This suggests combining both approaches to obtain better performing generators, that generate faster many valid documents. More possibilities for future research lie in, on the one hand, extending our model to consider more expressive constraints (such as in [17, 15]), as well as parallelism and different orders of generation, etc. On the other hand, it would be valuable to find more restricted cases that allow efficient document generation. Some of these directions may be studied by further extending our model to full Active XML. For generation of data values we intend to explore and compare other possible methods, using various kinds of information about the values distribution.

Last but not least, it would be interesting to experiment with the generators that were formally introduced here. For instance, use our model to compute the quality of schemas resulting from different inference techniques, and compare them; or test our model as a means of explaining and testing on online XML corpora (such as, e.g., the XML version of the DBLP bibliography).

Acknowledgments

We would like to thank Yann Ollivier for insightful comments, and Siqi Liu for feedback on the proof of Theorem 5. This work has been supported in part by the Advanced European Research Council grants Webdam, agreement 226513 (`http://webdam.inria.fr/`), and MoDaS, agreement 291071 (`http://www.math.tau.ac.il/~milo/projects/modas/`), by the Israel Ministry of Science, and by the US–Israel Binational Science Foundation.

References

1. S. Abiteboul, Y. Amsterdamer, D. Deutch, T. Milo, and P. Senellart. Finding optimal probabilistic generators for XML collections. In *ICDT*, 2012.
2. S. Abiteboul, Y. Amsterdamer, T. Milo, and P. Senellart. Auto-completion learning for XML. In *SIGMOD Conference*, pages 669–672, 2012. Demonstration.
3. S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *VLDB J.*, 17(5), 2008.

4. S. Abiteboul, P. Bourhis, A. Galland, and B. Marinoiu. The AXML artifact model. In *TIME*, 2009.
5. S. Abiteboul, T.-H. H. Chan, E. Kharlamov, W. Nutt, and P. Senellart. Aggregate queries for discrete and continuous probabilistic XML. In *ICDT*, 2010.
6. S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB J.*, 18(5), 2009.
7. T. Antonopoulos, F. Geerts, W. Martens, and F. Neven. Generating, sampling and counting subclasses of regular tree languages. In *ICDT*, 2011.
8. D. Barbosa, A. O. Mendelzon, J. Keenleyside, and K. A. Lyons. ToXgene: An extensible template-based data generator for XML. In *WebDB*, 2002.
9. M. Benedikt, E. Kharlamov, D. Olteanu, and P. Senellart. Probabilistic XML via Markov chains. *PVLDB*, 3(1), 2010.
10. G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. In *WWW*, 2008.
11. G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *VLDB*, 2006.
12. G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In *VLDB*, 2007.
13. C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.
14. Z. Chi and S. Geman. Estimation of probabilistic context-free grammars. *Comput. Linguist.*, 24(2), 1998.
15. S. Cohen. Generating XML structure using examples and constraints. *PVLDB*, 1(1), 2008.
16. S. Cohen, B. Kimelfeld, and Y. Sagiv. Incorporating constraints in probabilistic XML. In *PODS*, 2008.
17. C. David, L. Libkin, and T. Tan. Efficient reasoning about data trees via integer linear programming. In *ICDT*, 2011.
18. K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *JACM*, 56(1), 2009.
19. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *JACM*, 49(3), 2002.
20. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: a system for extracting document type descriptors from XML documents. In *SIGMOD*, 2000.
21. W. Gelade, T. Idziaszek, W. Martens, and F. Neven. Simplifying XML Schema: Single-type approximations of regular tree languages. In *PODS*, 2010.
22. G. Grahne and J. Zhu. Discovering approximate keys in XML data. In *CIKM*, 2002.
23. R. Kosala, H. Blockeel, M. Bruynooghe, and J. Van den Bussche. Information extraction from structured documents using *k*-testable tree automaton inference. *Data Knowl. Eng.*, 58(2), 2006.
24. K. Lange. *Optimization.* Springer-Verlag, 2004.
25. K. Lary and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algrithm. *Computer Speech and Language*, 4, 1990.
26. W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML Schema. *ACM Trans. Database Syst.*, 31(3), 2006.
27. W. Martens and J. Niehren. On the minimization of XML schemas and tree automata for unranked trees. *J. Comput. Syst. Sci.*, 73(4), 2007.
28. T. Milo and D. Suciu. Type inference for queries on semistructured data. In *PODS*, 1999.
29. M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Technol.*, 5(4), 2005.
30. S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *SIGMOD*, 1998.
31. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS*, 2000.