Running Tree Automata on Probabilistic XML

Sara Cohen
The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University
Jerusalem 91094, Israel
sara@cs.huji.ac.il

Benny Kimelfeld*†
IBM Almaden Research Center
San Jose, CA 95120, USA
kimelfeld@us.ibm.com

Yehoshua Sagiv*
The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University
Jerusalem 91094, Israel
sagiv@cs.huji.ac.il

ABSTRACT

Tree automata (specifically, bottom-up and unranked) form a powerful tool for querying and maintaining validity of XML documents. XML with uncertain data can be modeled as a probability space of labeled trees, and that space is often represented by a tree with *distributional nodes*. This paper investigates the problem of evaluating a tree automaton over such a representation, where the goal is to compute the probability that the automaton accepts a random possible world. This problem is generally intractable, but for the case where the tree automaton is deterministic (and its transitions are defined by deterministic string automata), an efficient algorithm is presented. The paper discusses the applications of this result, including the ability to sample and to evaluate queries (e.g., in monadic second-order logic) while requiring a-priori conformance to a schema (e.g., DTD). XML schemas also include attribute constraints, and the complexity of key, foreign-key and inclusion constraints are studied in the context of probabilistic XML. Finally, the paper discusses the generalization of the results to an extended data model, where distributional nodes can repeatedly sample the same subtree, thereby adding another exponent to the size of the probability space.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; H.2.1 [**Database Management**]: Logical Design—*Data models*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages

General Terms

Algorithms, Theory

*This research was supported by The Israel Science Foundation (Grant 893/05).

†Some of the work was done while this author was in The Hebrew University.

PODS'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

Keywords

Probabilistic XML, probabilistic trees, tree automata, XML schema, XML constraints, XML query evaluation

1. INTRODUCTION

Many real-world scenarios involve uncertain data. To store and manipulate such data, probabilistic databases have been studied, e.g., [1, 5–7, 15, 16, 18, 19, 26, 29, 32]. A probabilistic database is essentially a probability distribution over ordinary databases (often called *possible worlds*), and it is usually represented by annotating data items with probabilities (or probabilistic events). Those representations, together with statistical assumptions such as probabilistic independence, can usually encode exponentially large probability spaces. Much of the research effort has been on adapting traditional concepts to probabilistic databases. For example, when querying a probabilistic database, a common approach [5–7, 18, 19, 29] is to associate with each tuple t a level of confidence, which is the probability that t is in the answer when the query is applied to a random possible world. This makes query evaluation significantly harder than in ordinary databases. Even in the conceivably simplest relational model, where each tuple has an independent probability of existing, conjunctive queries are often highly intractable [7].

The situation is better for probabilistic XML, where the various proposed models[1] [1, 5, 15, 16, 18, 19, 21, 26, 29, 32] can be represented by means of a *p-document* [18], which is essentially an XML tree with special types of nodes called *distributional*. It was shown that when distributional nodes are independent, twig patterns with projection can be evaluated efficiently [18, 19], even when constraints (involving aggregate functions) are imposed on the data [5].

This paper is an important step towards the development of theoretical foundations necessary to derive a powerful database system for probabilistic XML. In particular, we explore fundamental issues that have been previously ignored. For one, all previous results on efficient query evaluation over probabilistic XML are for queries that do not consider order and have an acyclic structure (and the techniques used for obtaining these results are inherently based on this acyclicity). Does tractability carry over beyond those queries? Another issue is that of schema (e.g., DTD) validation. Given

[1]For [15, 16], this refers only to the "probabilistic instances" that are tree-structured and use point probabilities. In [5], the representation includes a set of *constraints* in addition to the p-document.

a probabilistic XML document and a schema, the following problems are important from a practical point of view. Is *every* possible world valid? Alternatively, is the probability of invalidity negligible? Is the probabilistic document even *relevant* to the schema in the sense that at least one possible world is valid? Can one impose validity on the probability space by querying under the a-priori condition that the random instance is valid?

In Section 3, we devise the pTT model of probabilistic XML, which generalizes all of the tractable models studied in [18] by allowing distributional nodes to manipulate the order of siblings. More particularly, pTTs are tree structures similar to the p-documents of [5, 18]. When using a pTT to generate a random document, each distributional node independently chooses, from a predefined distribution, a subset of its children as well as an order over this subset. In this model, the above fundamental issues reduce to the *acceptance problem*. Namely, compute the probability that a random instance of a given pTT is accepted by a given (bottom-up) tree automaton. Section 4 presents an efficient algorithm for this task. However, the algorithm is correct only if both the automaton and the description of its transitions are deterministic. Otherwise, the acceptance problem becomes intractable, but the algorithm can still solve efficiently the relevancy problem. Section 4.3 describes consequences of our algorithm, including the following. First, all Boolean queries in monadic second-order tree logic have a tractable data complexity (and they are even *fixed-parameter tractable* [9, 10]) over pTTs. Second, querying and sampling a pTT remain tractable even if a schema (given as a tree automaton) is imposed on the data.

The translation of a schema into an automaton does not capture constraints on XML attributes, such as *key*, *foreign-key* and *inclusion* constraints [11]. In Section 5, we show that the above tractability does not carry over to these constraints (even if approximations are allowed), but one can efficiently test whether *all* of the random instances of a pTT satisfy a set of constraints. Finally, in Section 6, we study the generalization of the pTT model by allowing a distributional node to repeatedly sample the same subtree. This extension adds another exponent to the size of the underlying probability space and, as a result, the acceptance problem is no longer tractable, since just representing the final answer (as a rational number) may require exponentially many bits. However, if the accuracy of the answer is limited to k-bit precision (for any k), the above algorithm can be adapted so that it remains efficient.

2. PRELIMINARIES

This section describes the preliminary concepts and notation that are used throughout the paper.

2.1 Strings and String Automata

Let S be a set. By S^* we denote the set of all finite strings of elements of S, namely, the sequences $s_1 \cdots s_n$ where $s_i \in S$ for all $1 \leq i \leq n$. A *language* over S is a subset of S^*. We use $|s|$ to denote the length n of s. The symbol ϵ denotes the empty string.

A *nondeterministic finite automaton* (*NFA*) M is a tuple $\langle \Sigma_M, S_M, s^0_M, F_M, \delta_M \rangle$ where Σ_M is a finite *alphabet*, S_M is a finite set of *states*, $s^0_M \in S_M$ is the *initial state*, $F_M \subseteq S_M$ is the set of *accepting states* and $\delta_M : S_M \times \Sigma_M \rightarrow 2^{S_M}$ (i.e., $\delta_M(s, \sigma)$ is a subset of S_M for all $s \in S_M$ and $\sigma \in \Sigma_M$) is the *transition function*. We consistently denote an NFA by M (possibly with a superscript and/or a subscript). Moreover, we implicitly assume that M is the automaton $\langle \Sigma_M, S_M, s^0_M, F_M, \delta_M \rangle$.

Let M be an NFA. A *run* of M on a string $s = \sigma_1 \cdots \sigma_n \in {\Sigma_M}^*$ is a mapping $\rho : \{1, \ldots, n\} \rightarrow S_M$, such that $\rho(1) \in \delta_M(s^0_M, \sigma_1)$ and $\rho(i) \in \delta_M(\rho(i-1), \sigma_i)$ for all $2 \leq i \leq n$. The run ρ is *accepting* if $\rho(n) \in F_M$, and in this case M *accepts* s. We denote by $\mathcal{L}(M)$ the language over Σ_M comprising the strings that are accepted by M.

A *deterministic finite automaton* (*DFA*) is an NFA M, such that the image of δ_M comprises singletons; that is, $|\delta_M(s, \sigma)| = 1$ for all $s \in S_M$ and $\sigma \in \Sigma_M$. It is well-known that NFAs and DFAs (as well as regular expressions) recognize the same class of languages, called *regular string languages*.

2.2 Trees and Hedges

We use trees that are ordered, unranked, and with labeled nodes. XML documents (and their probabilistic counterparts) are modeled as such trees. When modeling an XML document, the label corresponds either to the *tag* of an element or to a *textual value* (PCDATA). In some cases (e.g., DTD validation), one might use the same label (e.g., #PCDATA) to represent all the textual nodes. Next, we define how trees are represented as strings.

Definition 1. Let Σ be a (possibly infinite) alphabet of labels. A Σ-*tree* is inductively defined as follows.

- If $\sigma \in \Sigma$ then $\sigma()$ is a Σ-tree.
- If $\sigma \in \Sigma$ and $t_1, \ldots, t_n$ are Σ-trees, then $\sigma(t_1 \cdots t_n)$ is a Σ-tree. $\square$

We use $\mathbf{Trees}_\Sigma$ to denote the set of all Σ-trees. We often omit the parentheses from $\sigma()$ and write just σ.

A Σ-*hedge* is a sequence $h = t_1 \cdots t_n$ of Σ-trees (i.e., a member of ${\mathbf{Trees}_\Sigma}^*$). Note the following. First, every Σ-tree is a Σ-hedge. Second, if h is a Σ-hedge and $\sigma \in \Sigma$, then $\sigma(h)$ is a Σ-tree. Finally, if h_1 and h_2 are Σ-hedges, then $h_1 h_2$ is the Σ-hedge that is obtained by concatenating the two sequences of Σ-trees.

Obviously, there is a one-to-one correspondence between the notion of hedges described above and the common representation by means of nodes and edges. Each *occurrence* of a label σ in a Σ-hedge h defines a unique *node* v. We denote by $\mathcal{V}(h)$ the set of nodes of the hedge h. The label associated with the node v is denoted by $\lambda(v)$. A *leaf* of a Σ-hedge is a node without children. The *root* of a Σ-tree t, denoted $root(t)$, is the node without a parent.

Consider a Σ-hedge h. If there is a path from node v_1 to node v_2 in h, then we say that v_2 is a *descendant* of v_1, whereas v_1 is an *ancestor* of v_2. Note that every node is both a descendant and an ancestor of itself. If $v_1 \neq v_2$, then v_2 is a *proper descendant* of v_1, which in turn is a *proper ancestor* of v_2. If v is a node of h, then h^v_Δ denotes the subtree of h that is rooted at v and induced by all the descendants of v.

2.3 Tree Automata

We define a (bottom-up, unranked) *nondeterministic tree automaton* (*NTA*) A as a four-tuple $\langle \Sigma_A, Q_A, F_A, \delta_A \rangle$, where Σ_A is a finite alphabet, Q_A is a finite set of states, $F_A \subseteq Q_A$ is a set of accepting states and $\delta_A : Q_A \times \Sigma_A \rightarrow 2^{{Q_A}^*}$

is a function, such that for all $q \in Q_A$ and $\sigma \in \Sigma_A$, the set $\delta_A(q,\sigma)$ is a regular string language over the alphabet Q_A. For distinction from NFAs, we use A (possibly with a subscript and/or a superscript) to denote NTAs. Moreover, the NTA A is implicitly assumed to be the quadruple $\langle \Sigma_A, Q_A, F_A, \delta_A \rangle$.

A *run* of an NTA A on a Σ-tree t is *defined* when t is a Σ_A-tree and, in that case, it is a mapping $\rho : \mathcal{V}(t) \to Q_A$, such that for all nodes $v \in \mathcal{V}(t)$ with children $u_1 \cdots u_n$ (where $n \geq 0$), the string $\rho(u_1) \cdots \rho(u_n)$ belongs to the language $\delta_A(\rho(v), \lambda(v))$. The run ρ is *accepting* if it maps the root to an accepting state, i.e., $\rho(root(t)) \in F_A$. The language $\mathcal{L}(A)$ consists of all the Σ_A-trees that are *accepted* by A, that is, Σ_A-trees t, such that an accepting run on t exists.

An NTA A is *deterministic* if for all symbols $\sigma \in \Sigma_A$ and states $q_1, q_2 \in Q_A$, if $q_1 \neq q_2$ then $\delta_A(q_1,\sigma) \cap \delta_A(q_2,\sigma) = \emptyset$. It is then called a *deterministic tree automaton* (*DTA*).

In terms of representation, classes of NTAs (and DTAs) differ from one another in the way of specifying the regular languages $\delta_A(q,\sigma)$. We focus on specifications by means of NFAs (and, in particular, DFAs). We denote by $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ the class of all NTAs A, such that $\Sigma_A \subseteq \Sigma$ (where Σ can be infinite) and A uses an NFA (as indicated by the superscript "n") to specify each $\delta_A(q,\sigma)$. We similarly define $\mathbf{NTA}^{\mathrm{d}}_{\Sigma}$, $\mathbf{DTA}^{\mathrm{n}}_{\Sigma}$ and $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$. For example, an automaton of $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$ is a DTA A such that $\Sigma_A \subseteq \Sigma$ and each $\delta_A(q,\sigma)$ is specified by a DFA. From now on, we assume that $\delta_A(q,\sigma)$ *is* an NFA (or a DFA) that accepts the desired language, rather than the language itself. Note that $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ is syntactically the most general class, that is, each of $\mathbf{NTA}^{\mathrm{d}}_{\Sigma}$, $\mathbf{DTA}^{\mathrm{n}}_{\Sigma}$ and $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$ is a subset of $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$. Similarly, $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$ is syntactically the most restricted class. It was shown in [3] that NTAs and DTAs recognize the same languages, which are called *regular* tree languages. Thus, all four classes (and in particular $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ and $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$) are equivalent in terms of expressiveness.

We now define the notion of a size bound for tree automata. Consider a tree automaton $A \in \mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ and let N_A and N_M be two positive integers. We say that A is (N_A, N_M)-*bounded* if $|Q_A| \leq N_A$, and $|S_M| \leq N_M$ holds for all NFAs $M = \delta_A(q,\sigma)$ where $q \in Q_A$ and $\sigma \in \Sigma_A$. In other words, A has at most N_A states and each NFA used to represent a language of δ_A has at most N_M states.

2.4 Finite Probability Spaces

A *finite probability space* (abbr. *fp-space*) is a pair $F = (\Omega(F), p_F)$, such that $\Omega(F)$ is a finite set and $p_F : \Omega(F) \to (0,1]$ is a function that satisfies $\sum_{o \in \Omega(F)} p_F(o) = 1$. We say that the fp-space F is *over* a (possibly infinite) set O if $\Omega(F) \subseteq O$. In the sequel, we implicitly assume that an fp-space F is the pair $(\Omega(F), p_F)$.

3. PROBABILISTIC DATA MODEL

The notion of *probabilistic XML* refers to a finite probability space over XML documents. Formally, we consider fp-spaces over $\mathbf{Trees}_{\Sigma}$ and, more generally, over $\mathbf{Trees}_{\Sigma}{}^{*}$. The fp-space is given in terms of some *compact description* $\tilde{\mathcal{D}}$. Usually, we do not distinguish between $\tilde{\mathcal{D}}$ and the fp-space it describes. That is, we treat $\tilde{\mathcal{D}}$ as an fp-space, even though it is actually a compact description thereof. Each element of $\Omega(\tilde{\mathcal{D}})$ is a *possible world* (also called *random instance* or *sample*) of $\tilde{\mathcal{D}}$. Conventionally [1, 5, 16, 18, 19, 26, 29, 32], a compact description is a tree that specifies probabilistic junctions (alternatives) in addition to ordinary data. Thus, it describes the probabilistic process of generating a sample rather than the explicit probability space. The compact description that we use here, called Σ-*pTT*, is an *order-aware* generalization of the probabilistic XML denoted in [18] by $\mathsf{PrXML}^{\{\mathsf{exp}\}}$, which itself generalizes[2] other models in the literature (e.g., [5, 16, 19, 21, 26, 32]).

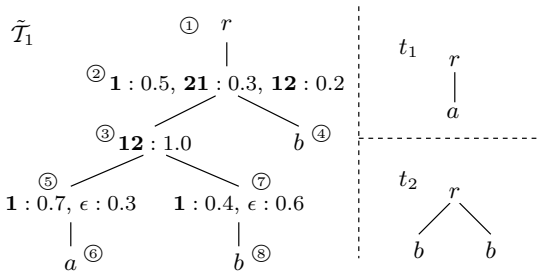

Figure 1: An $\{r, a, b\}$-pTT $\tilde{\mathcal{T}}_1$ and two samples

Below, we actually define three variants of compact descriptions, namely, Σ-*pTT*, Σ-*pTH* and Σ-*pHH*. In each acronym, the first two letters are either pT or pH, and they describe the structure of the compact description. The meaning of pT and pH is *probabilistic tree* and *probabilistic hedge*, respectively. The last letter is either T or H and it means that the compact description is over an fp-space of trees or hedges, respectively. For example, a Σ-*pTT* is a probabilistic tree that describes a space of trees. Our model of probabilistic XML uses only Σ-pTTs, but the other two variants are needed for describing our results. We define the three variants below.

Let n be a natural number. The set $\mathbf{PS}_n$ comprises all the fp-spaces π, such that the elements of π are permutations[3] on subsets of $\{1, \ldots, n\}$ (i.e., sequences of distinct numbers from $\{1, \ldots, n\}$). By $\mathbf{PS}$ we denote the union $\cup_{n \in \mathbb{N}} \mathbf{PS}_n$. We assume that a set of labels Σ is always disjoint from $\mathbf{PS}$.

Definition 2. A Σ-*pTH* is a $(\Sigma \cup \mathbf{PS})$-tree $\tilde{\mathcal{P}}$, such that for all nodes $v \in \mathcal{V}(\tilde{\mathcal{P}})$ with n children, $\lambda(v) \in \Sigma \cup \mathbf{PS}_n$. A Σ-*pHH* is a sequence $\tilde{\mathcal{H}} = \tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_k$ of Σ-pTHs. A Σ-*pTT* is a Σ-pTH $\tilde{\mathcal{T}}$, such that $\lambda(root(\tilde{\mathcal{T}})) \in \Sigma$. □

Observe that a Σ-pHH has one of the forms $\sigma(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$, $\pi(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$ or $\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n$, where $\sigma \in \Sigma$ is a label, $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_n$ are Σ-pTHs, and $\pi \in \mathbf{PS}_n$ is an fp-space. Note that every Σ-hedge is a Σ-pHH.

A node of a Σ-pHH with a label of Σ is called *ordinary*, and one with a label of $\mathbf{PS}$ is *distributional*. Note that every node is either ordinary or distributional, but not both. Also note that a Σ-pTT is a Σ-pTH with an ordinary root.

Example 1. The $\{r, a, b\}$-pTT $\tilde{\mathcal{T}}_1$ appears in Figure 1. The nodes are numbered for easy reference, and we denote the node numbered by ⓘ as v_i. The root v_1 of $\tilde{\mathcal{T}}_1$ is an ordinary node, and its only child v_2 is a distributional node. For distributional nodes, we use $s_1 : p_1, \ldots, s_k : p_k$ to denote the fp-space that assigns the probability p_i to the sequence s_i. The tree rooted at v_2 is an $\{r, a, b\}$-pTH, but not an $\{r, a, b\}$-pTT (since v_2 is distributional). □

[2] See [18] for a study of the expressiveness relationships among models of probabilistic XML.

[3] In Section 6, we consider the generalization of our results to the case where duplicates are allowed.

A Σ-pHH defines an fp-space over $\mathbf{Trees}_\Sigma{}^*$. The probabilistic process of generating a sample is similar to the one described in [5,18,19] for "p-documents," with two main differences. First, the process is naturally adapted from trees to hedges. Second, it takes into account the *order* among trees and siblings. Next, we give the exact definition.

A sample is obtained from a Σ-pHH $\tilde{\mathcal{H}}$ by the following recursive randomized process, denoted by $\mathsf{smpl}(\tilde{\mathcal{H}})$. An important point is that different executions of smpl are probabilistically independent.

1. If $\tilde{\mathcal{H}} = \tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n$, return $\mathsf{smpl}(\tilde{\mathcal{P}}_1) \cdots \mathsf{smpl}(\tilde{\mathcal{P}}_n)$.
2. If $\tilde{\mathcal{H}} = \sigma(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$, return $\sigma(\mathsf{smpl}(\tilde{\mathcal{P}}_1) \cdots \mathsf{smpl}(\tilde{\mathcal{P}}_n))$.
3. If $\tilde{\mathcal{H}} = \pi(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$, then randomly choose a string $I = j_1 \cdots j_k \in \Omega(\pi)$ with probability $p_\pi(I)$ and return $\mathsf{smpl}(\tilde{\mathcal{P}}_{j_1}) \cdots \mathsf{smpl}(\tilde{\mathcal{P}}_{j_k})$.

Example 2. The $\{r, a, b\}$-trees t_1 and t_2 in Figure 1 are possible worlds of $\tilde{\mathcal{T}}_1$. The probability of t_1 being returned in the sampling process is $0.5 \times 0.7 \times 0.6 = 0.21$, as it can be created only when the sequence $\mathbf{1}$ is chosen at v_2, and then when the sequences $\mathbf{1}$ and ϵ are chosen at v_5 and v_7, respectively. The probability of t_2 is $(0.3 \times 0.3 \times 0.4) + (0.2 \times 0.3 \times 0.4) = 0.06$ where the sum in the calculation comes from the two ways to generate t_2. Note that a positive probability for the empty string ϵ at a node v indicates that the subtree of v is optional. Also observe that the choices of children made by v_5 and v_7 are probabilistically independent of each other, because v_3 deterministically chooses both v_5 and v_7. This illustrates a general way (cf. [18]) of efficiently representing independent branches. □

Recall that a Σ-pTT $\tilde{\mathcal{T}}$ is a Σ-pTH that has an ordinary root. This means that a sample of $\tilde{\mathcal{T}}$ is always a single Σ-tree. Thus, Σ-pTTs define fp-spaces over $\mathbf{Trees}_\Sigma$, and they constitute our main model of probabilistic XML. (In Section 4.3, 5 and 6, additional models are considered.) We denote by $\mathbf{pTTs}_\Sigma$ the class of all Σ-pTTs.

In the sequel, the following conventions are used. First, we may omit Σ from the terms Σ-pTH, Σ-pHH and Σ-pTT when Σ is not necessary. For a clear distinction between the three, we consistently use $\tilde{\mathcal{P}}$, $\tilde{\mathcal{H}}$ and $\tilde{\mathcal{T}}$ to denote pTHs, pHHs and pTTs, respectively. Recall that we often do not distinguish between a compact description and the fp-space that it induces. Thus, we treat Σ-pHHs and Σ-pTTs as fp-spaces over $\mathbf{Trees}_\Sigma{}^*$ and $\mathbf{Trees}_\Sigma$, respectively. In particular, $\Omega(\tilde{\mathcal{H}})$ denotes the set of all the possible worlds of the Σ-pHH $\tilde{\mathcal{H}}$.

We always use the tilde symbol in the notation $\tilde{\mathcal{D}}$ of (a compact description of) an fp-space over $\mathbf{Trees}_\Sigma{}^*$ (note that $\tilde{\mathcal{D}}$ is not necessarily a Σ-pHH). We *omit* the tilde to denote the random variable $\mathcal{D}$ that represents a possible world chosen according to the probability distribution of $\tilde{\mathcal{D}}$. For example, $\Pr(\mathcal{T} = t)$, where t is a Σ-tree, is the probability that a randomly chosen possible world of $\tilde{\mathcal{T}}$ is t. As another example, if A is an NTA, then $\Pr(\mathcal{T} \in \mathcal{L}(A))$ is the probability that A accepts a random instance of the fp-space $\tilde{\mathcal{T}}$.

When representing a Σ-pHH, each fp-space π is given by explicitly specifying the strings I of $\Omega(\pi)$ and their probabilities. We assume that each $p_\pi(I)$ is a rational number that it is given as two integers: the numerator and the denominator.

4. RUNNING AUTOMATA ON PTTS

In this section, we consider the complexity of computing the probability of acceptance of a pTT by an NTA. We call this problem *probabilistic acceptance* (abbr. *p-acceptance*). A restricted version of this problem is that of *relevancy*, namely, deciding whether at least one possible world is accepted by the automaton. We formally define these two problems as follows.

Definition 3. Let $\mathbf{A}$ be a class of NTAs and $\mathbf{PT}$ be a class of fp-spaces over $\mathbf{Trees}_\Sigma$. *P-acceptance* of $\mathbf{PT}$ by $\mathbf{A}$ is the problem of computing $\Pr(\mathcal{D} \in \mathcal{L}(A))$, given $\tilde{\mathcal{D}} \in \mathbf{PT}$ and $A \in \mathbf{A}$. *Relevancy* of $\mathbf{PT}$ to $\mathbf{A}$ is the problem of deciding whether $\Pr(\mathcal{D} \in \mathcal{L}(A)) > 0$. □

The following theorem shows that p-acceptance is $\mathrm{FP}^{\#\mathrm{P}}$-complete in the case of $\mathbf{pTTs}_\Sigma$ and $\mathbf{NTA}^{\mathrm{n}}_\Sigma$, even for a singleton alphabet, and even when some determinism is required. Recall that $\mathrm{FP}^{\#\mathrm{P}}$ is the class of functions that are efficiently computable using an oracle to some function in #P. A function f is $\mathrm{FP}^{\#\mathrm{P}}$-hard if there is a polynomial-time *Turing reduction*[4] (or *Cook reduction*) from every function in $\mathrm{FP}^{\#\mathrm{P}}$ to f. The proofs of hardness are by reductions from the problem of determining the number of satisfying assignments of a positive 2-DNF formula, which is known to be #P-complete [28], and membership in $\mathrm{FP}^{\#\mathrm{P}}$ is shown by adapting the techniques of [13].

THEOREM 1. *Let Σ be an alphabet. The problem of p-acceptance of $\mathbf{pTTs}_\Sigma$ by $\mathbf{NTA}^{\mathrm{n}}_\Sigma$ is $\mathrm{FP}^{\#\mathrm{P}}$-complete. In the case of either $\mathbf{NTA}^{\mathrm{d}}_\Sigma$ or $\mathbf{DTA}^{\mathrm{n}}_\Sigma$, it remains $\mathrm{FP}^{\#\mathrm{P}}$-hard even when Σ is a singleton.*

In contrast to the above intractability, we next show that p-acceptance is tractable for tree automata of $\mathbf{DTA}^{\mathrm{d}}_\Sigma$.

4.1 Algorithm for Probabilistic Acceptance

We now consider the problem of p-acceptance of Σ-pTTs by tree automata of $\mathbf{DTA}^{\mathrm{d}}_\Sigma$. The *naive* approach of solving this problem, given $\tilde{\mathcal{T}}$ and A, is by enumerating all the possible worlds of $\tilde{\mathcal{T}}$, running A over each of them and summing up the probabilities of all the possible worlds that are accepted by A. Obviously, this approach is infeasible since the number of possible worlds can be exponential in the size of $\tilde{\mathcal{T}}$. Nevertheless, we show that this problem is tractable (i.e., in polynomial time) even without bounding the size of the alphabet Σ. Moreover, it can be solved by a rather fast algorithm—polynomial in A and merely linear in $\tilde{\mathcal{T}}$. We fix an infinite alphabet Σ that is used throughout this section. We start with some notation.

4.1.1 Notation

Let M be an NFA and $s, t \in S_M$ be two states. We denote by $M^{s \rightsquigarrow t}$ the NFA that accepts all the strings of $(\Sigma_M)^*$ that enable M to move from state s to state t. That is, $M^{s \rightsquigarrow t}$ is the automaton M' with $\Sigma_{M'} = \Sigma_M$, $S_{M'} = S_M$, $s^0_{M'} = s$, $F_{M'} = \{t\}$ and $\delta_{M'} = \delta_M$.

Let A be an NTA, M be an NFA over $(Q_A)^*$ and $h = t_1 \cdots t_n$ be a Σ_A-hedge. A *run* of A on h is a mapping $\rho : \mathcal{V}(h) \to Q_A$, such that the restriction of ρ to each $\mathcal{V}(t_i)$ is a run of A on t_i. The run ρ is *accepting relative to* M if $\rho(root(t_1)) \cdots \rho(root(t_n))$ is accepted by M. We use $\mathcal{L}(A \mid$

[4] Using an oracle to a #P-hard (or $\mathrm{FP}^{\#\mathrm{P}}$-hard) function, one can efficiently solve the entire polynomial hierarchy [31].

M) to denote the set of all Σ_A-hedges that are accepted by A relative to M. Intuitively, the notion of accepting relative to M implies a restriction as to the word formed by the states assigned to the roots. This will be important when devising an algorithm for p-acceptance.

Let t be a Σ-tree. *Postorder traversal* of the nodes of t means traversal in a bottom-up, left-to-right order. That is, if $t = \sigma(t_1 \cdots t_n)$, then we (recursively) traverse t_1, t_2, and so on until t_n, and then visit the root.

Let $\tilde{\mathcal{T}}$ be a Σ-pTT. The *lowest Σ-label at or above v*, denoted by $\lambda_\uparrow(v)$, is simply $\lambda(v)$ if v is ordinary; otherwise, if v is distributional, then it is the label of the lowest ordinary ancestor of v. Note that $\lambda_\uparrow(v)$ is always well-defined, since $\tilde{\mathcal{T}}$ is a Σ-pTT, and hence, every distributional node has an ordinary ancestor (since the root is an ordinary node).

4.1.2 Data Structures

The algorithm RunTA of Figure 2 gets as input a pTT $\tilde{\mathcal{T}}$ and a DTA $A \in \mathbf{DTA}^{\mathrm{d}}_\Sigma$, and returns (in Line 25) the numeric value $\Pr(\mathcal{T} \in \mathcal{L}(A))$. In this section, we describe the data structures used by the algorithm. Essentially, the algorithm incrementally constructs two arrays $\mathcal{A}^t$ and $\mathcal{A}^h$ by dynamic programming. The values of $\mathcal{A}^t$ are used for computing the values of $\mathcal{A}^h$ and vice versa. Finally, the output of the algorithm is obtained from the entries of $\mathcal{A}^t$. Next, we describe these two arrays in detail.

An index of the array $\mathcal{A}^t$ consists of a subtree $\tilde{\mathcal{T}}^v_\Delta$ of $\tilde{\mathcal{T}}$, such that v is ordinary (thus, $\tilde{\mathcal{T}}^v_\Delta$ is a pTT), and a state $q \in Q_A$. When the algorithm terminates, it holds that

$$\mathcal{A}^t[\tilde{\mathcal{T}}^v_\Delta, q] = \Pr\left(\mathcal{T}^v_\Delta \in \mathcal{L}(A_q)\right),$$

where A_q is obtained from A by replacing F_A with the singleton $\{q\}$. In other words, it is the probability that a run of A on $\mathcal{T}^v_\Delta$ maps the root to q. In particular, the value returned by the algorithm in Line 25 is the sum $\mathcal{A}^t[\tilde{\mathcal{T}}, q]$ for all accepting states q of A. We now show that this is indeed the required value. By definition,

$$\Pr\left(\mathcal{T} \in \mathcal{L}(A)\right) = \Pr\left(\bigvee_{q \in F_A} \mathcal{T} \in \mathcal{L}(A_q)\right).$$

Since A is a DTA, there is at most one run of A over a Σ-tree t. Therefore, the events $\mathcal{T} \in \mathcal{L}(A_q)$ (where $q \in F_A$) are pairwise disjoint. Thus,

$$\Pr\left(\bigvee_{q \in F_A} \mathcal{T} \in \mathcal{L}(A_q)\right) = \sum_{q \in F_A} \Pr\left(\mathcal{T} \in \mathcal{L}(A_q)\right),$$

as required.

In the second array, $\mathcal{A}^h$, an index comprises a pHH $\tilde{\mathcal{H}}$ and two states s_1 and s_2 of a DFA $M = \delta_A(q, \sigma)$, where $q \in Q_A$ and $\sigma \in \Sigma$ (see the precise definition below). We assume that distinct DFAs of A have disjoint sets of states. For all nodes v of $\tilde{\mathcal{T}}$, a *simple sub-pHH of v* is a pHH $\tilde{\mathcal{H}}$ having one of the following three forms.

1. $\tilde{\mathcal{T}}^u_\Delta$, if u is a child of v,
2. A prefix $\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n$, if $\tilde{\mathcal{T}}^v_\Delta$ is the pTT $\sigma(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$, *and*
3. A prefix of $\tilde{\mathcal{P}}_{m_1} \cdots \tilde{\mathcal{P}}_{m_k}$, when $\tilde{\mathcal{T}}^v_\Delta$ is $\pi(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$ and $m_1 \cdots m_k \in \Omega(\pi)$.

Note that in each of 2 and 3, the prefix can be the empty pHH. The entry $\mathcal{A}^h[\tilde{\mathcal{H}}, s_1, s_2]$ is defined for all pHHs $\tilde{\mathcal{H}}$ and states s_1 and s_2, such that there exists a node v of $\tilde{\mathcal{T}}$ and a state $q \in Q_A$ where $\tilde{\mathcal{H}}$ is a simple sub-pHH of v and s_1 and s_2 are states of the DFA $M = \delta_A(q, \sigma)$ for $\sigma = \lambda_\uparrow(v)$ (namely, σ is the lowest Σ-label at or above v). When the algorithm terminates, the following holds.

$$\mathcal{A}^h[\tilde{\mathcal{H}}, s_1, s_2] = \Pr\left(\mathcal{H} \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s_2})\right)$$

Note that the right-hand side is the probability that $\mathcal{H}$ is accepted by A relative to $M^{s_1 \rightsquigarrow s_2}$.

The next section describes how the values of $\mathcal{A}^t$ and $\mathcal{A}^h$ are computed.

4.1.3 The Algorithm

We now describe the algorithm RunTA. Lines 4–24 are executed for all nodes v of $\tilde{\mathcal{T}}$, in postorder, and states q of A. For each v and q, the following is done. Let σ_v be $\lambda_\uparrow(v)$. If $\sigma_v \in \Sigma_A$, then M is the DFA $\delta_A(q, \sigma_v)$. Otherwise, $\delta_A(q, \sigma_v)$ is undefined and M is the DFA over $(Q_A)^*$ that rejects every word (i.e., $\mathcal{L}(M) = \emptyset$); this DFA is denoted in Line 4 by $M^{Q_A}_\emptyset$. The values $\mathcal{A}^h[\tilde{\mathcal{H}}, s_1, s_2]$ are computed for every simple sub-pHH of v and states $s_1, s_2 \in M$. In addition, if v is ordinary, then $\mathcal{A}^t[\tilde{\mathcal{T}}^v_\Delta, q]$ is also computed. Lines 6–15 handle the case where v is ordinary, whereas a distributional v is handled in Lines 17–24. In each of these line segments, the algorithm iterates over all relevant $\tilde{\mathcal{H}}$, s_1 and s_2, and the computation of $\mathcal{A}^h[\tilde{\mathcal{H}}, s_1, s_2]$ is done by the subroutine ProcPHH($\tilde{\mathcal{H}}, M, s_1, s_2$) (the DFA M is provided since it is needed for the computation).

Lines 13–15 compute $\mathcal{A}^t[\tilde{\mathcal{T}}^v_\Delta, q]$ (for an ordinary v) for each state q of A, as follows. Suppose that $\tilde{\mathcal{T}}^v_\Delta$ is $\sigma_v(\tilde{\mathcal{H}})$. Then $\mathcal{A}^t[\tilde{\mathcal{T}}^v_\Delta, q]$ is set to the sum of all the $\mathcal{A}^h[\tilde{\mathcal{H}}, s^0_M, s]$, where s is an accepting state of M. Note that each of the components of this sum has already been computed in Lines 6–12 of the current iteration of Line 5. The correctness of this computation is explained as follows. From the definitions, we get the following equality.

$$\begin{aligned}\Pr\left(\mathcal{T}^v_\Delta \in \mathcal{L}(A_q)\right) &= \Pr\left(\mathcal{H} \in \mathcal{L}(A \mid M)\right) = \\ &= \Pr\left(\bigvee_{s \in F_M} \mathcal{H} \in \mathcal{L}(A \mid M^{s^0_M \rightsquigarrow s})\right)\end{aligned}$$

Now, since A is a DTA, there exists at most one run of A on a given Σ-hedge. Moreover, since M is a DFA, a specific string of $(Q_A)^*$ can move M to exactly one state $s \in F_M$. Consequently, the events $\mathcal{H} \in \mathcal{L}(A \mid M^{s^0_M \rightsquigarrow s})$, where $s \in F_M$, are pairwise disjoint. It thus follows that

$$\begin{aligned}&\Pr\left(\bigvee_{s \in F_M} \mathcal{H} \in \mathcal{L}(A \mid M^{s^0_M \rightsquigarrow s})\right) = \\ &= \sum_{s \in F_M} \Pr\left(\mathcal{H} \in \mathcal{L}(A \mid M^{s^0_M \rightsquigarrow s})\right).\end{aligned}$$

Next, we describe the subroutine ProcPHH. As explained above, for $s_1, s_2 \in S_M$, ProcPHH($\tilde{\mathcal{H}}, M, s_1, s_2$) computes $\mathcal{A}^h[\tilde{\mathcal{H}}, s_1, s_2]$. Let $\tilde{\mathcal{H}} = \tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n$. The case where $n = 0$ is straightforwardly handled in Lines 3–4. (Recall that M does not have empty transitions.) The case where $n = 1$ is considered in Lines 6–7, and it is actually handled by a separate subroutine ProcPTH($\tilde{\mathcal{P}}_1, M, s_1, s_2$) that is described

Algorithm $\mathsf{RunTA}(\tilde{\mathcal{T}}, A)$

1: **for all** nodes v of $\tilde{\mathcal{T}}$ in postorder traversal **do**
2: $\quad\sigma_v \leftarrow \lambda_\uparrow(v)$
3: $\quad$**for all** $q \in Q_A$ **do**
4: $\quad\quad M \leftarrow \delta_A(q, \sigma_v)$ or $M^{Q_A}_\emptyset$ if $\sigma_v \notin \Sigma_A$
5: $\quad\quad$**if** v is ordinary **then**
6: $\quad\quad\quad$let $\tilde{\mathcal{T}}^v_\Delta = \sigma_v(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$
7: $\quad\quad\quad$**for** $j = 1$ to n **do**
8: $\quad\quad\quad\quad$**for all** $s_1, s_2 \in S_M$ **do**
9: $\quad\quad\quad\quad\quad\mathsf{ProcPHH}(\tilde{\mathcal{P}}_j, M, s_1, s_2)$
10: $\quad\quad\quad$**for** $j = 0$ to n **do**
11: $\quad\quad\quad\quad$**for all** $s_1, s_2 \in S_M$ **do**
12: $\quad\quad\quad\quad\quad\mathsf{ProcPHH}(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_j, M, s_1, s_2)$
13: $\quad\quad\quad$**for all** $q \in Q_A$ **do**
14: $\quad\quad\quad\quad\tilde{\mathcal{H}} \leftarrow \tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n$
15: $\quad\quad\quad\quad\mathcal{A}^t[\tilde{\mathcal{T}}^v_\Delta, q] \leftarrow \sum_{s \in F_M} \mathcal{A}^h[\tilde{\mathcal{H}}, s^0_M, s]$
16: $\quad\quad$**else**
17: $\quad\quad\quad$let $\tilde{\mathcal{T}}^v_\Delta = \pi(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$
18: $\quad\quad\quad$**for** $j = 1$ to n **do**
19: $\quad\quad\quad\quad$**for all** $s_1, s_2 \in S_M$ **do**
20: $\quad\quad\quad\quad\quad\mathsf{ProcPHH}(\tilde{\mathcal{P}}_j, M, s_1, s_2)$
21: $\quad\quad\quad$**for all** $m_1 \cdots m_k \in \Omega(\pi)$ **do**
22: $\quad\quad\quad\quad$**for** $j = 0$ to k **do**
23: $\quad\quad\quad\quad\quad$**for all** $s_1, s_2 \in S_M$ **do**
24: $\quad\quad\quad\quad\quad\quad\mathsf{ProcPHH}(\tilde{\mathcal{P}}_{m_1} \cdots \tilde{\mathcal{P}}_{m_j}, M, s_1, s_2)$
25: **return** $\sum_{q \in F_A} \mathcal{A}^t[\tilde{\mathcal{T}}, q]$

Subroutine $\mathsf{ProcPHH}(\tilde{\mathcal{H}}, M, s_1, s_2)$

1: let $\tilde{\mathcal{H}} = \tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n$
2: **if** $n = 0$ **then**
3: $\quad\mathcal{A}^h[\tilde{\mathcal{H}}, s_1, s_2] = 1$ if $s_1 = s_2$ and 0 otherwise
4: $\quad$**return**
5: **if** $n = 1$ **then**
6: $\quad\mathsf{ProcPTH}(\tilde{\mathcal{P}}_1, M, s_1, s_2)$
7: $\quad$**return**
8: $\tilde{\mathcal{H}}^- \leftarrow \tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_{n-1}$
9: $\mathcal{A}^h[\tilde{\mathcal{H}}, s_1, s_2] \leftarrow \sum_{s \in S_M} \mathcal{A}^h[\tilde{\mathcal{H}}^-, s_1, s] \cdot \mathcal{A}^h[\tilde{\mathcal{P}}_n, s, s_2]$

Subroutine $\mathsf{ProcPTH}(\tilde{\mathcal{P}}, M, s_1, s_2)$

1: **if** *root*$(\tilde{\mathcal{P}})$ is ordinary **then**
2: $\quad Q' \leftarrow \{q \in \Sigma_M \mid s_2 \in \delta_M(s_1, q)\}$
3: $\quad\mathcal{A}^h[\tilde{\mathcal{P}}, s_1, s_2] \leftarrow \sum_{q \in Q'} \mathcal{A}^t[\tilde{\mathcal{P}}, q]$
4: **else**
5: $\quad$let $\tilde{\mathcal{P}} = \pi(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$
6: $\quad\mathcal{A}^h[\tilde{\mathcal{P}}, s_1, s_2] \leftarrow 0$
7: $\quad$**for all** $I = m_1 \cdots m_n \in \Omega(\pi)$ **do**
8: $\quad\quad\tilde{\mathcal{H}}_I \leftarrow \tilde{\mathcal{P}}_{m_1} \cdots \tilde{\mathcal{P}}_{m_n}$
9: $\quad\quad\mathcal{A}^h[\tilde{\mathcal{P}}, s_1, s_2] \mathrel{+}= p_\pi(I) \times \mathcal{A}^h[\tilde{\mathcal{H}}_I, s_1, s_2]$

Figure 2: Computing $\Pr(\mathcal{T} \in \mathcal{L}(A))$

below. The rest of the subroutine, in Lines 8–9, handles the case where $n > 2$. Let $\tilde{\mathcal{H}}^-$ be obtained from $\tilde{\mathcal{H}}$ by removing $\tilde{\mathcal{P}}_n$, namely, $\tilde{\mathcal{H}}^-$ is the pHH $\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_{n-1}$. Observe that by the flow of the algorithm RunTA, the values $\mathcal{A}^h[\tilde{\mathcal{H}}^-, s'_1, s'_2]$ and $\mathcal{A}^h[\tilde{\mathcal{P}}_n, s'_1, s'_2]$ are already computed at this point for all $s'_1, s'_2 \in S_M$. Then, in Line 9, $\mathcal{A}^h[\tilde{\mathcal{H}}, s_1, s_2]$ is set to the sum $\sum_{s \in S_M} a_{s_1,s} \times b_{s,s_2}$, where $a_{s_1,s} = \mathcal{A}^h[\tilde{\mathcal{H}}^-, s_1, s]$ and $b_{s,s_2} = \mathcal{A}^h[\tilde{\mathcal{P}}_n, s, s_2]$. The reason for this assignment is the equality below that follows from the definitions.

$$\Pr(\mathcal{H} \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s_2})) =$$
$$= \Pr\left(\bigvee_{s \in S_M} \mathcal{H}^- \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s}) \wedge \mathcal{P}_n \in \mathcal{L}(A \mid M^{s \rightsquigarrow s_2})\right)$$

Due to the arguments made above regarding the determinism of both A and M, the events $\mathcal{H}^- \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s})$ of different s are pairwise disjoint. Consequently, the above probability is equal to the following sum.

$$\sum_{s \in S_M} \Pr(\mathcal{H}^- \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s}) \wedge \mathcal{P}_n \in \mathcal{L}(A \mid M^{s \rightsquigarrow s_2}))$$

We use the fact that $\tilde{\mathcal{H}}^-$ and $\tilde{\mathcal{P}}_n$ are independent to derive that

$$\sum_{s \in S_M} \Pr(\mathcal{H}^- \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s}) \wedge \mathcal{P}_n \in \mathcal{L}(A \mid M^{s \rightsquigarrow s_2})) =$$
$$= \sum_{s \in S_M} \Pr(\mathcal{H}^- \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s})) \times$$
$$\times \Pr(\mathcal{P}_n \in \mathcal{L}(A \mid M^{s \rightsquigarrow s_2})).$$

Finally, we describe the subroutine $\mathsf{ProcPTH}$ that computes $\mathcal{A}^h[\tilde{\mathcal{P}}, s_1, s_2]$, when given a pTH $\tilde{\mathcal{P}}$, a DFA M and $s_1, s_2 \in S_M$. Lines 2–3 handle the case where $\tilde{\mathcal{P}}$ is a pTT. In this case, the random variable $\mathcal{P}$ is a single tree. Let Q' be the set of all states $q \in \Sigma_M$, such that $s_2 \in \delta_M(s_1, q)$. Then, $\mathcal{P}$ is in $\mathcal{L}(A \mid M^{s_1 \rightsquigarrow s_2})$ if and only if a run of A on $\mathcal{P}$ maps the root to a state q of Q', i.e., $\mathcal{P} \in \mathcal{L}(A_q)$. Moreover, since A is deterministic, the events of two different q are disjoint. Consequently, we get the following equality, which explains the assignment of Line 3.

$$\Pr(\mathcal{P} \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s_2})) = \Pr\left(\bigvee_{q \in Q'} \mathcal{P} \in \mathcal{L}(A_q)\right) =$$
$$= \sum_{q \in Q'} \Pr(\mathcal{P} \in \mathcal{L}(A_q))$$

The flow of RunTA implies that the entries $\mathcal{A}^t[\tilde{\mathcal{P}}, q]$ have already been computed when Line 3 is executed. To see why, suppose that $\tilde{\mathcal{P}}$ is $\tilde{\mathcal{T}}^v_\Delta$. Then $\mathsf{ProcPTH}(\tilde{\mathcal{P}}, M, s_1, s_2)$ is called when the parent of v is chosen in Line 1 of RunTA, whereas the $\mathcal{A}^t[\tilde{\mathcal{P}}, q]$ are computed when v is chosen.

Lines 5–9 handle the case where $\tilde{\mathcal{P}}$ has a distributional root (hence it is not a pTT). Suppose that $\tilde{\mathcal{P}}$ is $\pi(\tilde{\mathcal{P}}_1 \cdots \tilde{\mathcal{P}}_n)$. For a string $I \in \Omega(\pi)$, let $\tilde{\mathcal{H}}_I$ be the pHH that corresponds to I (i.e., the one of Line 8). Then, from the definition of a pTH and the law of total probability, we get the following.

$$\Pr(\mathcal{P} \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s_2})) =$$
$$= \sum_{I \in \Omega(\pi)} p_\pi(I) \times \Pr(\mathcal{H}_I \in \mathcal{L}(A \mid M^{s_1 \rightsquigarrow s_2}))$$

Observe that $\mathcal{A}^h[\tilde{\mathcal{H}}_I, s_1, s_2]$ has already been computed for all $I \in \Omega(\pi)$ since, if $\tilde{\mathcal{P}}$ is $\tilde{\mathcal{T}}^v_\Delta$, then this computation took place when v was chosen in Line 1 of RunTA (whereas the current execution is made when the parent of v is chosen).

4.1.4 Correctness and Efficiency

The next lemma states the correctness and efficiency of the algorithm RunTA. For simplicity, an arithmetic operation is assumed to have a fixed cost. Thus, our analysis takes into account the number of arithmetic operations, but not the actual cost of their execution.[5] The *size* of a pTT $\tilde{\mathcal{T}}$, denoted by $\|\tilde{\mathcal{T}}\|$, is the sum of the number of ordinary nodes of $\tilde{\mathcal{T}}$ and all the sizes of the fp-spaces $\pi \in \mathbf{PS}$ that appear in $\tilde{\mathcal{T}}$.

LEMMA 1. *For a Σ-pTT $\tilde{\mathcal{T}}$ and an (N_A, N_M)-bounded DTA $A \in \mathbf{DTA}^{\mathrm{d}}_{\Sigma}$, $\mathsf{RunTA}(\tilde{\mathcal{T}}, A)$ returns $\Pr(\mathcal{T} \in \mathcal{L}(A))$ in $O(\|\tilde{\mathcal{T}}\| \cdot N_A^2 N_M^2)$ time.*

As a result, we get the following theorem.

THEOREM 2. *The problem of p-acceptance of $\mathbf{pTTs}_{\Sigma}$ by $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$ is in polynomial time.*

4.2 Nondeterminism Revisited

Recall that Theorem 1 shows that for $\mathbf{pTTs}_{\Sigma}$ (even when Σ is fixed and very small), p-acceptance by either $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$, $\mathbf{NTA}^{\mathrm{d}}_{\Sigma}$ or $\mathbf{DTA}^{\mathrm{n}}_{\Sigma}$ is intractable. Obviously, one can determinize the given NTA and then apply Theorem 2. The running time is determined by the size of the resulting DTA. The results[6] of [22] and standard facts on determinization of binary tree automata imply that an NTA of $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ can be converted into an equivalent DTA of $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$ at an exponential cost. (Note that an exponential blowup is unavoidable, because it is already required for transforming an NFA to a DFA [14].) Formally, if a given $A \in \mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ is (N_A, N_M)-bounded, then a $(2^{N_A}, 2^{N_A N_M})$-bounded $A' \in \mathbf{DTA}^{\mathrm{d}}_{\Sigma}$ with $\mathcal{L}(A') = \mathcal{L}(A)$ is constructible in $O(2^{N_A N_M} \cdot |\Sigma_A|)$ time. Thus, we conclude the following.

COROLLARY 1. *The problem of p-acceptance of $\mathbf{pTTs}_{\Sigma}$ by automata of $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ can be solved in time that is exponential in the NTA and linear in the pTT.*

The important question of whether there are efficient relative (multiplicative) approximations[7] remains open. As the following argument shows, one way of proving their nonexistence could be by showing intractability of the relevancy problem. To see why, observe that if it is intractable to determine whether at least one possible world is accepted by the NTA, then multiplicatively approximating the probability of acceptance is intractable as well. The reduction of Theorem 1 does not show intractability of relevancy. In fact, the next theorem shows that the contrary is true for the whole class $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ and, moreover, relevancy can be solved by the algorithm RunTA.

THEOREM 3. *For $\tilde{\mathcal{T}} \in \mathbf{pTTs}_{\Sigma}$ and $A \in \mathbf{NTA}^{\mathrm{n}}_{\Sigma}$, it holds that $\mathsf{RunTA}(\tilde{\mathcal{T}}, A) \geq \Pr(\mathcal{T} \in \mathcal{L}(A))$, and the two numbers are either both zero or both nonzero. Hence, relevancy of $\mathbf{pTTs}_{\Sigma}$ to $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ is in polynomial time.*

[5]Formally, taking the actual arithmetic operations into accounts has a polynomial effect on the overall running time.

[6]In particular, we use the translation of [22] that converts an NTA of $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ into an equivalent *non-deterministic stepwise* tree automaton.

[7]Observe that a randomized additive approximation can be obtained by running the NTA over samples of the pTT (and applying, e.g., Hoeffding inequality).

Theorem 3 is important in terms of *optimization*, namely, before paying the cost of running an NTA over a pTT, one can efficiently test whether the result is nonzero. In addition, since the output of $\mathsf{RunTA}(\tilde{\mathcal{T}}, A)$ is an upper bound on $\Pr(\mathcal{T} \in \mathcal{L}(A))$, we can use the returned value to prune results with probabilities that are too low to be of interest.

4.3 Applications

We now discuss the application of the results of the previous section to some central aspects of managing probabilistic XML, namely, querying (possibly in the presence of constraints or a schema) and sampling.

4.3.1 Query Evaluation

The first application that we discuss is that of evaluating queries over probabilistic XML. A common approach in querying probabilistic data [1, 5–7, 18, 26, 29] is to assign a degree of *confidence* to each answer. More formally, the result of applying a query Q comprises pairs (a, c_a), where a is a possible answer and c_a (the confidence of a) is the probability of obtaining a when querying a random instance. In terms of *data complexity* [33], query evaluation often (e.g., [5–7, 18]) reduces to the computation of *Boolean queries*, where the goal is to find the probability that the query returns **true**. In [18], they show that Boolean *twig queries* [4] are *fixed-parameter tractable*[8] [9, 10] over their model of probabilistic XML, and that result immediately generalizes to pTTs. (For a formal definition of twig queries see, e.g., [4, 5, 18].)

A language that is by far more expressive than twig queries is that of *monadic second-order logic* (MSO) over Σ-trees[9] (the exact definition can be found in, e.g., [25]). For a (possibly infinite) alphabet Σ, we denote by $\mathbf{MSO}_{\Sigma}$ the set all MSO formulae that use only labels of Σ. Let Σ be a finite alphabet. It is known that every formula $\varphi \in \mathbf{MSO}_{\Sigma}$ can be translated into an NTA $A_{\varphi} \in \mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ (and vice versa), such that the Σ-trees that satisfy φ are precisely those that are accepted by A_{φ} [8,25,30]. Furthermore, it can be shown that there is such a translation where A_{φ} is $(K_{\varphi}, K_{\varphi})$-bounded, where K_{φ} depends only on the size of φ and *not* on the size of Σ. As a consequence, we get the following corollary of Theorem 2, where Σ may be an infinite alphabet.

COROLLARY 2. *Evaluation of $\mathbf{MSO}_{\Sigma}$ formulae over Σ-pTTs is fixed-parameter tractable.*

4.3.2 Querying under Constraints

Tree automata can have various roles in data management. As discussed above, they can represent queries. They can also represent integrity or schema *constraints*, such as a DTD or *specialized DTD* [27]. Previous work [5, 20] studied various sorts of a-priori *assertion* of constraints in probabilistic databases, namely, restricting the possible worlds to those in the sub-space consisting of all the Σ-trees that satisfy the constraints. Formally, we consider fp-spaces $\tilde{\mathcal{D}}$ over $\mathbf{Trees}_{\Sigma}$, such that $\tilde{\mathcal{D}}$ is represented by a pair $(\tilde{\mathcal{T}}, A_c)$, where $\tilde{\mathcal{T}} \in \mathbf{pTTs}_{\Sigma}$ and $A_c \in \mathbf{DTA}^{\mathrm{d}}_{\Sigma}$ (e.g., A_c is a DTD). Semantically, $\tilde{\mathcal{D}}$ is the fp-space such that $\Omega(\tilde{\mathcal{D}}) = \Omega(\tilde{\mathcal{T}}) \cap \mathcal{L}(A_c)$ and

[8]This is a stronger notion than polynomial data complexity.

[9]The twig queries in [5,18] allow for arbitrary conditions over labels, but that can be incorporated in MSO quite easily. Details are omitted due to a lack of space.

for all $t \in \Omega(\tilde{\mathcal{T}}) \cap \mathcal{L}(A_C)$, the probability $p_{\tilde{\mathcal{D}}}(t)$ is given by

$$p_{\tilde{\mathcal{D}}}(t) = \Pr\left(\mathcal{T} = t \mid \mathcal{T} \in \mathcal{L}(A_c)\right) = \frac{\Pr\left(\mathcal{T} = t\right)}{\Pr\left(\mathcal{T} \in \mathcal{L}(A_c)\right)}\,.$$

The fp-space $\tilde{\mathcal{D}}$ is well defined only if the set $\Omega(\tilde{\mathcal{T}}) \cap \mathcal{L}(A_c)$ is nonempty. We denote by $\mathbf{pTTs}_\Sigma|\mathbf{DTA}^{\mathrm{d}}_\Sigma$ the class of all such representations. The following corollary of Theorem 2 is due to the fact that given two DTAs $A_1, A_2 \in \mathbf{DTA}^{\mathrm{d}}_\Sigma$, one can efficiently construct a DTA A such that $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$.

COROLLARY 3. *For $\mathbf{pTTs}_\Sigma|\mathbf{DTA}^{\mathrm{d}}_\Sigma$, both testing well definedness and p-acceptance by $\mathbf{DTA}^{\mathrm{d}}_\Sigma$ are in time that is linear in the pTT and polynomial in the DTAs.*

Applying Corollary 3 requires the query to be defined as a DTA. As said above, this can be done for MSO. Unfortunately, translating a formula φ of $\mathbf{MSO}_\Sigma$ to a DTA is highly expensive, namely, the running time is unlikely to be bounded by an elementary function of $\|\varphi\|$ [12, 23]. But for twig queries, this translation can be bypassed. The next theorem states that there is a practically efficient evaluation of twig queries over $\mathbf{pTTs}_\Sigma|\mathbf{DTA}^{\mathrm{d}}_\Sigma$. The proof is by showing how to merge, in a single scan of the pTT, two algorithms: one is RunTA and the second is similar to the evaluation of twigs in [18].

THEOREM 4. *Let $\tilde{\mathcal{D}} \in \mathbf{pTTs}_\Sigma|\mathbf{DTA}^{\mathrm{d}}_\Sigma$ be given by a pTT $\tilde{\mathcal{T}} \in \mathbf{pTTs}_\Sigma$ and an (N_A, N_M)-bounded $A \in \mathbf{DTA}^{\mathrm{d}}_\Sigma$. Let ξ be a Boolean twig query. Computing $\Pr(\tilde{\mathcal{D}} \models \xi)$ is in $O(4^{\|\xi\|} \cdot N_A^2 N_M^2 \cdot \|\tilde{\mathcal{T}}\|)$ time.*

4.3.3 Sampling under Constraints

The final application that we consider is that of *sampling under constraints*. Let $\tilde{\mathcal{D}}$ be an fp-space over $\mathbf{Trees}_\Sigma$. A *sampling algorithm* for $\tilde{\mathcal{D}}$ is a randomized process $\mathsf{Sample}(\tilde{\mathcal{D}})$ that correctly simulates $\tilde{\mathcal{D}}$, namely, it generates a random Σ-tree so that for all $t \in \Omega(\tilde{\mathcal{D}})$, the following holds:

$$\Pr\Big(\mathsf{Sample}(\tilde{\mathcal{D}}) = t\Big) = p_{\tilde{\mathcal{D}}}(t)\,.$$

Again, we consider the class $\mathbf{pTTs}_\Sigma|\mathbf{DTA}^{\mathrm{d}}_\Sigma$. A naive algorithm would repeatedly emulate the random process defined by the given pTT until it gets a sample that is accepted by the DTA. However, it is easy to come up with examples where the expected running time of this algorithm is exponential in the pTT (even for fixed alphabet and DTA). However, by a fairly easy adaptation of the sampling algorithm given in [5], combined with the tractability result of Theorem 2, we obtain the following.

COROLLARY 4. *An fp-space of $\mathbf{pTTs}_\Sigma|\mathbf{DTA}^{\mathrm{d}}_\Sigma$ can be sampled in polynomial time.*

5. ATTRIBUTE CONSTRAINTS

As explained in the previous section, tree automata can be used for analyzing the relationship between a pTT and a DTD. More specifically, a DTD D without attribute specifications can be represented by a tree automaton. In that case, one can efficiently test the probability that a random instance of the Σ-pTT is valid w.r.t. D (and, in particular, whether each or none of the random instances is valid w.r.t. D). However, DTDs also include constraints about the attributes. In this section, we consider three kinds thereof, namely, *key*, *inclusion*, and *foreign-key* constraints—all were studied in [11]. We restrict the discussion to *unary* constraints (i.e., constraints involving a single attribute). First, we enrich our data model with attributes.

We assume that $\mathbf{V}$ is a fixed infinite set of *attribute values*. Let Σ be a set of labels. A *$\Sigma^@$-tree* is a pair $t^@ = (t, \alpha)$, where t is a Σ-tree and α is a finite partial function from $\mathcal{V}(t) \times \Sigma$ to $\mathbf{V}$. The function α specifies the attributes of a node v of t and their values as follows. We say that ξ *is an attribute of* v if $\alpha(v, \xi)$ is defined, and in this case, $\alpha(v, \xi)$ is the *value of* ξ *for* v and is also denoted by $v.\xi$.

Let Σ be a set of labels and let $t^@$ be a $\Sigma^@$-tree. A *key* constraint has the form $\sigma.\xi \to \sigma$, where σ and ξ are labels of Σ. The constraint $\sigma.\xi \to \sigma$ is satisfied by $t^@$ if distinct nodes with the label σ and the attribute ξ have different values for ξ; that is, for all $v, u \in \mathcal{V}(t)$ it holds that

$$(\lambda(v) = \lambda(u) = \sigma \wedge v.\xi = u.\xi) \to v = u\,.$$

An *inclusion* constraint has the form $\sigma_1.\xi_1 \subseteq \sigma_2.\xi_2$, and it is satisfied by $t^@$ if for all nodes v_1 with the label σ_1 and attribute ξ_1, there exists a node v_2 with the label σ_2 and attribute ξ_2, such that $v_1.\xi_1 = v_2.\xi_2$. Finally, a *foreign-key* constraint has the form $\sigma_1.\xi_1 \subseteq_{\mathrm{fk}} \sigma_2.\xi_2$, and its meaning is the conjunction of $\sigma_1.\xi_1 \subseteq \sigma_2.\xi_2$ and $\sigma_2.\xi_2 \to \sigma_2$. For a constraint c, we use $t^@ \models c$ to denote that $t^@$ satisfies c.

We now enrich pTTs with attributes. For simplifying the presentation, our model does not allow the attributes themselves to be uncertain. Formally, a $\Sigma^@$-pTT is a pair $(\tilde{\mathcal{T}}, \alpha)$, where $\tilde{\mathcal{T}}$ is a Σ-pTT and α is a finite partial function from $\mathcal{V}(\tilde{\mathcal{T}}) \times \Sigma$ to $\mathbf{V}$; moreover, α is only defined over pairs (v, ξ) such that v is an ordinary node. We naturally identify a $\Sigma^@$-pTT $(\tilde{\mathcal{T}}, \alpha)$ with the fp-space $\tilde{\mathcal{T}}^@$, such that a random instance (t, α') of $\tilde{\mathcal{T}}^@$ is obtained as follows. First, t is obtained by sampling $\tilde{\mathcal{T}}$ (in the usual way). Second, α' is the restriction of α to the ordinary nodes of $\tilde{\mathcal{T}}$ that actually appear in t. This instance is represented, as usual, by the random variable $\mathcal{T}^@$.

Let $\tilde{\mathcal{T}}^@$ be a $\Sigma^@$-pTT. We would like to understand how $\tilde{\mathcal{T}}^@$ statistically relates to a given constraint, namely, to know the probability that the constraint is satisfied. The following theorem shows that computing this probability is intractable (even for small alphabets); moreover, just testing whether it is nonzero is NP-hard. Hence, an efficient relative approximation is unlikely to exist (unless RP=NP or P=NP, depending on whether the approximation is randomized or not). NP-hardness and $\mathrm{FP}^{\#\mathrm{P}}$-hardness are proved by reductions from the problem *exact cover by 3-sets* and its counting[10] version, respectively.

THEOREM 5. *Let Σ be an alphabet of four or more symbols and let c be a key, inclusion, or foreign-key constraint. It is $\mathrm{FP}^{\#\mathrm{P}}$-complete to compute $\Pr\left(\mathcal{T}^@ \models c\right)$, where the input is a $\Sigma^@$-pTT $\tilde{\mathcal{T}}^@$. Moreover, it is NP-complete to decide whether $\Pr\left(\mathcal{T}^@ \models c\right) > 0$.*

Although one cannot efficiently compute the probability of satisfying a constraint, the following proposition shows that one can still efficiently test whether all the possible worlds satisfy that constraint.

[10]The counting version is that of computing the number of exact covers. This problem is known to be #P-complete (see, e.g., [17]).

PROPOSITION 1. *Let c be a key, inclusion, or foreign-key constraint. Deciding whether $\Pr\left(\mathcal{T}^{@} \models c\right) = 1$, given a $\Sigma^{@}$-pTT $\tilde{\mathcal{T}}^{@}$, is in polynomial time.*

A practically important consequence is the following. Suppose that we are given (as input) a $\Sigma^{@}$-pTT $\tilde{\mathcal{T}}^{@}$, a DTD D described as a DTA of $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$, and a set C of (key, inclusion and foreign-key) constraints on attributes. By Theorem 2 and Proposition 1, we can efficiently determine whether $\tilde{\mathcal{T}}^{@}$ is always valid w.r.t. D and C; that is, whether all the possible worlds of $\tilde{\mathcal{T}}^{@}$ satisfy both D and each of the constraints of C.

6. DUPLICATING PTTS

We now consider a model that extends pTTs by allowing branches of a distributional node to be sampled more than once. We show that this model offers a greater compactness than pTTs. Generalizing our results to this model is discussed rather informally, due to a lack of space.

Recall from Definition 2 that in a Σ-pTT, the label of a distributional node with n children is an element of $\mathbf{PS}_n$, namely, an fp-space over permutations (duplicate-free sequences) on subsets of $\{1,\ldots,n\}$. A *duplicating* Σ-pTT (abbr. Σ-dpTT) is defined similarly to a Σ-pTT, except that the label of a distributional node with n children can be any fp-space over $\{1,\ldots,n\}^*$. We may write just dpTT when Σ is not needed. A sample of a dpTT is obtained exactly as that of a pTT; in particular, when sampling the same subtree more than once, different samples are probabilistically independent (and, hence, they are not necessarily equal). By $\mathbf{dpTTs}_{\Sigma}$ we denote the class of all Σ-dpTTs. We similarly generalize the notion of $\Sigma^{@}$-pTT to $\Sigma^{@}$-dpTT.

Generalizing pTTs to dpTTs might seem straightforward. However, it is not! For one, the next example shows that the fp-space of a dpTT $\tilde{\mathcal{T}}$ can have (as a function of $\|\tilde{\mathcal{T}}\|$) a double-exponential number of possible worlds, where each possible world can be exponentially large.

Example 3. Let $\Sigma = \{a, b\}$ and let $\pi_{1|2}$ and π_{11} be the fp-spaces over $\{1, 2\}^*$, such that $\pi_{1|2}$ uniformly chooses between 1 and 2 (each with probability 0.5) and π_{11} always chooses 11. Let $\tilde{\mathcal{P}}_0 = \pi_{1|2}(ab)$. Inductively, for $k > 0$, let $\tilde{\mathcal{P}}_k = \pi_{11}(\tilde{\mathcal{P}}_{k-1})$. Finally, let $\tilde{\mathcal{T}}_m$ be the dpTT $a(\tilde{\mathcal{P}}_m)$. For example, $\tilde{\mathcal{T}}_2$ is $a(\pi_{11}(\pi_{11}(\pi_{1|2}(ab))))$. Clearly, $\tilde{\mathcal{T}}_m$ encodes the uniform distribution over the Σ-trees $a(s)$, where s is in $\{a, b\}^*$ and its length is 2^m. In particular, this fp-space includes 2^{2^m} samples, each with 2^m leaves and probability 2^{-2^m}. □

Consider a Σ-dpTT $\tilde{\mathcal{T}}$ and a DTA $A \in \mathbf{DTA}^{\mathrm{d}}_{\Sigma}$. Note that the algorithm RunTA (Figure 2) can be applied *as is* to $\tilde{\mathcal{T}}$ and A. This algorithm correctly computes $\Pr\left(\mathcal{T} \in \mathcal{L}(A)\right)$, even though $\tilde{\mathcal{T}}$ can represent a much larger fp-space than a pTT of a similar size. Moreover, the running time is polynomial, assuming that an arithmetic operation has a fixed cost (e.g., as in the *rational Blum-Shub-Smale* computational model [2]). Without this assumption (i.e., under bit complexity), RunTA is still efficient for pTTs, but not for dpTTs. The reason is that an exponential number of bits may be needed for representing the computed numbers (i.e., the entries of $\mathcal{A}^t$ and $\mathcal{A}^h$). This is a tight lower bound since representing the output $\Pr\left(\mathcal{T} \in \mathcal{L}(A)\right)$ could require an exponential number of bits. For instance, consider the dpTT $\tilde{\mathcal{T}}_m$ of Example 3 and let A_a be the DTA that accepts the set of all $\{a\}$-trees. Then $\Pr\left(\mathcal{T}_m \in \mathcal{L}(A_a)\right) = 2^{-2^m}$ and it requires 2^m bits in a standard representation. In particular, p-acceptance of $\mathbf{dpTTs}_{\Sigma}$ by $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ (or by $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$) is not in $\mathrm{FP}^{\#\mathrm{P}}$.

For the sake of efficiency, it is common to only require that the output will have *k-bit precision*, for a given number k. That is, an additive error of at most 2^{-k} is tolerable. Under this requirement, we can show that even for dpTTs, the acceptance problem remains tractable in the following sense. The algorithm RunTA guarantees k-bit precision if its arithmetic operations are done with $(k+q)$-bit precision, where q is polynomial in the input. The other tractability results of the previous sections can be similarly generalized[11] to dpTTs, with the only exception of Corollary 4 that is discussed below. Moreover, k-bit precision is irrelevant to the tractable *decision* problems, and they remain tractable for dpTTs. That is, the following problems are solvable in polynomial time.

- Relevancy of $\mathbf{dpTTs}_{\Sigma}$ to $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$.
- Deciding whether $\Pr\left(\mathcal{T}^{@} \models c\right) = 1$, given a $\Sigma^{@}$-dpTT $\tilde{\mathcal{T}}^{@}$ and a (key, inclusion, or foreign-key) constraint c.

One can always transform a given dpTT to an equivalent pTT (that is, the dpTT and the pTT define the same fp-space) by explicitly adding subtrees as required by the duplication. This entails (in the worst case) an exponential blowup of the size.[12] Thus, Corollary 4 implies that an fp-space of $\mathbf{dpTTs}_{\Sigma}|\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$ (which is defined similarly to $\mathbf{pTTs}_{\Sigma}|\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$) can be sampled in exponential time, which is worst-case optimal (since the sample itself can be exponentially large).

7. CONCLUSIONS

The Σ-pTT model of probabilistic XML generalizes those studied in [5,16,19,21,21,26,32] (as well as some of those investigated in [1,18]). It is the first model in which the order among siblings (and not just their existence) is probabilistic. We presented an efficient algorithm for p-acceptance of a Σ-pTT by a DTA of $\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$, and showed that this problem becomes intractable if either the tree automaton itself or the description of its transitions is non-deterministic. We discussed the applications of our algorithm to common tasks, such as validating and enforcing schema constraints (expressed as a DTA). In particular, evaluation of MSO queries is fixed-parameter tractable.

We also investigated the application of common types of attribute constraints that are not captured by tree automata, namely, key, inclusion and foreign-key constraints. Although it is intractable to compute (or even multiplicatively approximate) the probability that a constraint (of each of the three types) is satisfied, one can efficiently test whether it holds in *all* of the possible worlds. Finally, we showed how our results carry over to dpTTs, which are exponentially more compact than pTTs. In particular, our

[11] For generalizing Corollary 3 and Theorem 4, we assume that the DTA describing the constraint accepts the dpTT with a non-negligible probability (e.g., at least 2^{-k}).

[12] Note that this transformation is actually efficient if there is a fixed upper bound on the number of duplicating nodes along a path from the root to a leaf.

algorithm can be used to efficiently solve the acceptance problem when the accuracy of the output is limited to k-bit precision, for any k.

This work gives rise to various important future directions. In [24], the query language of *efficient tree logic* (ETL) is proposed. This language is as expressive as MSO, yet query evaluation is fixed-parameter tractable with only an exponential (hence, elementary) dependence of the running time on the size of the query. So, a natural question is whether ETL has a similar complexity over pTTs (and dpTTs). A second direction is that of *approximate* p-acceptance. Specifically, what is the complexity of evaluating an automaton of $\mathbf{NTA}^{\mathrm{n}}_{\Sigma}$ if one is satisfied with a multiplicative approximation (possibly a randomized one)? Another direction is regarding the complexity of sampling an fp-space of $\mathbf{dpTTs}_{\Sigma}|\mathbf{DTA}^{\mathrm{d}}_{\Sigma}$. We showed that sampling is in exponential time, which is a tight worst-case analysis, because the generated sample itself might have an exponential size. But it is yet unknown whether sampling can be done in time that is polynomial in the combined size of the input and the output, possibly while allowing the probability of generating a sample to be an approximation of the exact one. Finally, an important direction is to find natural conditions that make it possible to efficiently compute the probability that attribute constraints are satisfied by a pTT.

Acknowledgment

The authors thank the anonymous referees of PODS 2009 for valuable comments and suggestions. They also thank Ronald Fagin and C. Seshadhri for helpful discussions.

8. REFERENCES

[1] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *EDBT*, pages 1059–1068, 2006.

[2] L. Blum, M.Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP completeness, recursive functions, and universal machines. *Bull. A.M.S.*, 21:1–46, 1989.

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

[4] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *SIGMOD*, pages 310–321. ACM, 2002.

[5] S. Cohen, B. Kimelfeld, and Y. Sagiv. Incorporating constraints in probabilistic XML. In *PODS*, pages 109–118, 2008.

[6] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[7] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302. ACM, 2007.

[8] J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

[9] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[10] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[11] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3):368–406, 2002.

[12] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *LICS*, pages 215–224. IEEE Computer Society, 2002.

[13] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, pages 227–234. ACM, 1998.

[14] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[15] E. Hung, L. Getoor, and V. S. Subrahmanian. Probabilistic interval XML. In *ICDT*, pages 361–377, 2003.

[16] E. Hung, L. Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *ICDE*, pages 467–478, 2003.

[17] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, and R. E. Stearns. The complexity of planar counting problems. *SIAM J. Comput.*, 27(4):1142–1167, 1998.

[18] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query efficiency in probabilistic XML models. In *SIGMOD Conference*, pages 701–714, 2008.

[19] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic XML. In *VLDB*, pages 27–38. ACM, 2007.

[20] C. Koch. Approximating predicates and expressive queries on probabilistic databases. In *PODS*, pages 99–108. ACM, 2008.

[21] T. Li, Q. Shao, and Y. Chen. PEPX: a query-friendly probabilistic XML database. In *CIKM*, pages 848–849. ACM, 2006.

[22] W. Martens and J. Niehren. On the minimization of xml schemas and tree automata for unranked trees. *J. Comput. Syst. Sci.*, 73(4):550–583, 2007.

[23] A. R. Meyer. Weak monadic second-order theory of successor is not elementary recursive. *Logic Colloquim*, 453:132–154, 1975.

[24] F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *PODS*, pages 145–156. ACM, 2000.

[25] F. Neven and T. Schwentick. Query automata over finite trees. *Theor. Comput. Sci.*, 275(1-2):633–674, 2002.

[26] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *VLDB*, pages 646–657, 2002.

[27] Y. Papakonstantinou and V. Vianu. Incremental validation of xml documents. In *ICDT*, pages 47–63. Springer, 2003.

[28] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[29] P. Senellart and S. Abiteboul. On the complexity of managing probabilistic XML data. In *PODS*, pages 283–292, 2007.

[30] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[31] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2), 1992.

[32] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *ICDE*, pages 459–470. IEEE Computer Society, 2005.

[33] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146. ACM, 1982.