Author manuscript, published in "International Conference on Very Large Data Bases (VLDB), Singapore (2010)"

hal-00537778, version 1 - 19 Nov 2010

Probabilistic XML via Markov Chains*

Michael Benedikt
Oxford University, UK
michael.benedikt@comlab.ox.ac.uk

Evgeny Kharlamov†
Free University of Bozen-Bolzano, Italy
kharlamov@inf.unibz.it

Dan Olteanu
Oxford University, UK
dan.olteanu@comlab.ox.ac.uk

Pierre Senellart
Institut Télécom; Télécom ParisTech; CNRS LTCI
pierre.senellart@telecom-paristech.fr

ABSTRACT

We show how Recursive Markov Chains (RMCs) and their restrictions can define probabilistic distributions over XML documents, and study tractability of querying over such models. We show that RMCs subsume several existing probabilistic XML models. In contrast to the latter, RMC models (i) capture probabilistic versions of XML schema languages such as DTDs, (ii) can be exponentially more succinct, and (iii) do not restrict the domain of probability distributions to be finite. We investigate RMC models for which tractability can be achieved, and identify several tractable fragments that subsume known tractable probabilistic XML models. We then look at the space of models between existing probabilistic XML formalisms and RMCs, giving results on the expressiveness and succinctness of RMC subclasses, both with each other and with prior formalisms.

1. INTRODUCTION

Uncertainty is inherently ubiquitous in today's data, and can take the shape, for instance, of measurement errors in scientific experiments or sensor readings [21] or typographical errors in manually entered data [4].

The clear demand has spurred much research activity around managing uncertain relational data, using a wide range of models and processing techniques, e.g., [21, 4]. Uncertain XML data has also received attention, e.g., [1, 16], albeit less than in the relational case. XML is nevertheless the *de facto* model for Web data, and uncertainty is commonplace in a Web context: It arises from unreliability of Web data sources, errors in automated Web information extraction tools, or imprecision in integration of XML data from sources of varying structures and vocabularies.

Existing models for uncertain XML have emerged as generalizations of concrete documents with additional structure to capture uncertainty. This paper discusses a different approach, adapting existing probabilistic word models to XML. We adapt Recursive Markov Chains (RMCs) [11] and restrictions thereof to define probabilistic distributions over XML documents. RMCs are extensions of the standard Markov chains, i.e., of graphs with edges labeled with probabilities and that define processes evolving via independent choices at nodes. The extension consists of a notion of subroutine and the runs of RMCs have a natural hierarchical structure, and thus are interpretable as nested words or trees.

The motivation behind our investigation is threefold.

Firstly, we argue that RMCs are a more natural formalism to model probability distributions over XML documents than existing representation systems proposed in the literature, called hereafter PrXML [1]. Previous PrXML systems are designed to represent *finite* probability spaces, where the sizes of the representation and of each represented XML document have the same order of magnitude. RMCs capture these systems, yet they can do much more. We study a space of RMC representation systems that are *wide*, i.e., they do not impose a bound on the width of represented documents, and also *deep* in that they do not restrict the depth of the documents. This ability to represent XML documents whose sizes are not bounded by the size of the representation makes RMCs more succinct and more expressive than existing PrXML systems, and is the key ingredient needed to express probabilistic versions of XML schema languages such as DTDs.

Secondly, PrXML does not represent the limit of tractable query evaluation. The largest prior class of tractable queries is the model PrXML^{exp} that has tractable evaluation for queries in the expressive query language of Monadic Second-Order logic (MSO) [7]. Though unrestricted RMCs are not known to be MSO-tractable, we show that important restrictions are tractable under fixed-cost arithmetic and others are tractable in the usual bit-cost arithmetic model. Our work thus provides models that are much more expressive and succinct than prior classes, but retain tractability.

Thirdly, by looking at RMCs we connect questions on probabilistic XML to the rich set of tools and techniques available from the study of computational issues surrounding Markov models. This allows existing results and algorithms to be applied. Techniques for learning and analyzing variants of RMCs have been developed in several communities: hidden Markov models and stochastic context-free grammars are variants studied within machine learning and computational linguistics, while probabilistic automata have been investigated from the point of view of verification.

Our aim in this work is to understand and quantify the inherent tension between three fundamental aspects of prob-

*Supported by the EU projects Webdam (FP7-ICT-226513), FOX (FP7-ICT-233599), ONTORULE (FP7-ICT-231875), French ANR DataRing, and British EPSRC EP/G004021/1.

†The author is co-affiliated with INRIA Saclay.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were presented at The 36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.

Proceedings of the VLDB Endowment, Vol. 3, No. 1

Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

hal-00537778, version 1 - 19 Nov 2010

abilistic XML representation systems: expressiveness, succinctness, and tractability of query evaluation. To this end, we provide a map of restrictions to the RMC model that exhibit different levels of expressiveness, succinctness, and tractability. At one end of this map we find PrXML models from the literature [1] that are fully tractable, yet narrow and shallow, i.e., they bound the depth and the width of the represented XML documents, and only have a finite number of possible worlds. At the other end, we have unrestricted RMCs that are wide and deep, with infinitely many possible worlds, and where tractability of query evaluation would require a fundamental breakthrough in numerical evaluation. In between, we define RMC restrictions that remain narrow and shallow, yet trade full tractability for succinctness, or remain fully tractable even if they can be exponentially more succinct than existing PrXML models. Also, we show there are fully tractable RMC restrictions that are wide and shallow, and therefore strictly more expressive than PrXML. Moreover, we find wide and deep models that preserve tractability under fixed-cost arithmetic.

To sum up, the contributions of this paper are as follows. *(i)* We first show how recursive Markov chains can be turned into a representation system for probabilistic XML that allows arbitrarily deep and wide documents, and that most existing probabilistic XML models are restrictions of RMCs (Section 3). *(ii)* We then propose restrictions of the general RMC model for which MSO tractability (with respect to data complexity) can be achieved. In particular, we show in Section 4 that the so-called hierarchical Markov chains are tractable under fixed-cost arithmetic, and tree-like Markov chains are tractable in the usual bit-cost arithmetic. The latter result subsumes early MSO tractability results given in [7]. *(iii)* We explore the space of representation systems within our tractable fragments, comparing their expressiveness and succinctness (Section 5). The yardstick particularly useful to differentiate among them is the ability to represent succinctly wide documents, deep documents, and probability spaces with worlds that have probabilities double-exponentially close to 1. *(iv)* The tractability results mentioned above are for data complexity and MSO. In Section 6, we complement them with combined complexity results for our RMC-based models and three query languages: tree-pattern queries, forward navigational XPath, and MSO.

2. PRELIMINARIES

XML documents (or documents for short) are ordered, unranked, labeled trees. Given a document d, the set of nodes and the set of child edges are denoted by $\mathcal{V}(d)$ and $\mathcal{C}(d)$ respectively. We use $\prec$ to denote the strict total order over nodes of a document, $\mathsf{root}(d)$ for the root of d, $\mathcal{L}$ for the finite set of labels, and $\mathsf{lbl}(v) \in \mathcal{L}$ for the label of a node v. We do not distinguish between labels and values here. The notions of *child*, *parent*, *leaf*, *descendant*, *ancestor* have their standard meaning.

Two documents d_1 and d_2 are *isomorphic*, denoted $d_1 \sim d_2$, if one can be obtained from the other by replacing nodes with some other nodes while preserving labels and the order of the nodes. Formally, $d_1 \sim d_2$ if there is a bijection $\varphi : \mathcal{V}(d_1) \to \mathcal{V}(d_2)$, such that for all $v_1, v_2 \in \mathcal{V}(d_1)$, (i) $\mathsf{lbl}(v_1) = \mathsf{lbl}(\varphi(v_1))$; (ii) $(v_1, v_2) \in \mathcal{C}(d_1)$ if and only if $(\varphi(v_1), \varphi(v_2)) \in \mathcal{C}(d_2)$; and (iii) $v_1 \prec v_2$ if and only if $\varphi(v_1) \prec \varphi(v_2)$.

We define a *probabilistic XML space* (or px-space for short) as a probability distribution over a set of non-isomorphic documents, called the *domain*.

A *probabilistic XML representation system*, or *probabilistic XML model*, is a pair $(\mathcal{S}, [\![\cdot]\!])$ where $\mathcal{S}$ is a set of *representations* S, (e.g. p-documents, RMCs, as presented later) and $[\![\cdot]\!]$, the *possible-world semantics of* $\mathcal{S}$, is a function mapping every element of $\mathcal{S}$ to a px-space. If there is no ambiguity on the possible-word semantics, we simply write $\mathcal{S}$.

A system $\mathcal{S}$ is *deep* if there exists $S \in \mathcal{S}$ such that for all $k \geqslant 1$, there is a document in the domain of $[\![S]\!]$ whose height (i.e., maximum distance from the root to a leaf) is at least k; otherwise, it is *shallow*. A system $\mathcal{S}$ is *wide* if there exists $S \in \mathcal{S}$ such that for all $k \geqslant 1$, there is a document in the domain of $[\![S]\!]$ whose width (i.e., number of leaves) is at least k; otherwise it is *narrow*. Assuming all representations of a system $\mathcal{S}$ have a finite set of labels (which is the case of all those discussed in this work), they have finite domain if and only if $\mathcal{S}$ is both shallow and narrow.

Let $\mathcal{S}, \mathcal{S}'$ be two systems. We say that $\mathcal{S}$ is *translatable* to $\mathcal{S}'$, denoted $\mathcal{S} \sqsubseteq \mathcal{S}'$, if, for every $S \in \mathcal{S}$, there exists $S' \in \mathcal{S}'$ such that the probability distributions given by the px-spaces $[\![S]\!]$ and $[\![S']\!]$ are the same (up to document isomorphism). If there is a polynomial-time procedure to obtain a representation in $\mathcal{S}'$ for every representation in $\mathcal{S}$, we say that $\mathcal{S}$ is *efficiently translatable* to $\mathcal{S}'$ and denote $\mathcal{S} \sqsubseteq_{\text{poly}} \mathcal{S}'$. If $\mathcal{S} \sqsubseteq \mathcal{S}'$ and $\mathcal{S}' \sqsubseteq \mathcal{S}$ then we denote $\mathcal{S} \equiv \mathcal{S}'$. Similarly, if $\mathcal{S} \sqsubseteq_{\text{poly}} \mathcal{S}'$ and $\mathcal{S}' \sqsubseteq_{\text{poly}} \mathcal{S}$, then $\mathcal{S} \equiv_{\text{poly}} \mathcal{S}'$.

Given a representation S in a system, a *query* q over S is a function mapping every document d in the domain of $[\![S]\!]$ to a Boolean. We write $d \models q$ for $q(d) = \text{true}$. The *quantitative probabilistic evaluation problem* for a query q and a representation S, is to determine the probability that q maps documents in S to true: $\mathsf{Pr}(S \models q) = \sum_{(d,p) \in [\![S]\!],\, d \models q} p$.

A *query language* L is a collection of queries. A system $\mathcal{S}$ is *tractable for* L if for any $q \in L$ there is a polynomial-time algorithm that takes as input $S \in \mathcal{S}$ and outputs the solution to the quantitative evaluation problem for q and S. Following [11], *ra-tractability* is tractability in case of fixed-cost rational arithmetic, i.e., all arithmetic operations over rationals take unit time, no matter how large the numbers.

Monadic second-order logic over documents (MSO) is the logic built up from unary predicates for the labels, the binary descendant relation Descendant, the ordering relation $\prec$, and free unary predicate variables, via Boolean operators and first and second-order quantifiers $\exists x, \exists S$. The semantics of MSO is standard [23]. MSO is a natural logic for documents, since any MSO formula can be converted into a bottom-up tree automaton that accepts a document if and only if the document satisfies the formula. It is more expressive than other XML query languages such as tree-pattern queries with Boolean operators [16] or XPath.

We want to reinterpret probabilistic models on words to get distributions over documents. To this effect, we use a standard encoding of a document d with labels in $\mathcal{L}$ as a (well-formed) string over the language $\mathsf{Tag}(\mathcal{L})$, which consists of the symbols, called *tags*, $\langle l \rangle$ and $\langle /l \rangle$ for each $l \in \mathcal{L}$.

We define the XML representation systems $\mathsf{PrXML}^{mux,det}$ and PrXML^{exp}, whose elements are called p-documents [1]. A p-document is a document with two types of nodes. *Distributional* nodes are only used to define the probabilistic process that generates random documents—they do not actually occur in the output of the process. *Ordinary* nodes have labels and may appear in the generated documents. We require the leaves and the root to be ordinary nodes.

hal-00537778, version 1 - 19 Nov 2010

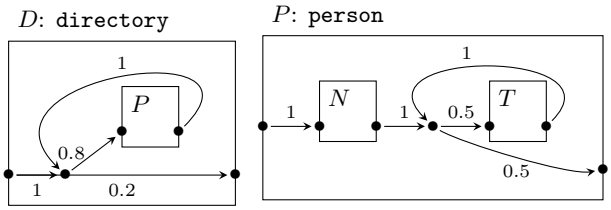


Figure 1: RMC A_{ex} of Examples 2, 4, and 5

Formally, a *p-document* is an unranked, ordered, labeled tree. Each node has a unique identifier v and a label $\mu(v)$ in $\mathcal{L} \cup \{exp(\mathsf{Pr})\} \cup \{mux(\mathsf{Pr})\} \cup \{det\}$. We consider distributional nodes that define discrete probability distributions Pr over the subsets of their children. In the case of *mux* nodes, we impose that all subsets have cardinality less than or equal to 1, that is, we select either no children at all, or a single child. A *det* node deterministically chooses all its children.

The *semantics of a p-document* $\widehat{\mathcal{P}}$, denoted $[\![\widehat{\mathcal{P}}]\!]$, is the px-space obtained from the following randomized three-step process (see [1] for a more detailed presentation): *(i)* Independently for each $exp(\mathsf{Pr})$ node, we select a subset of its children (according to the corresponding probability distribution Pr) and delete the other children and the entire subtrees underneath. Independently for each $mux(\mathsf{Pr})$ node, we select either zero or one child (according to Pr) and delete the other children and the entire subtrees underneath. We do not delete any children of *det* nodes. The probability of this run of the process is defined as the product of all choices. *(ii)* We then remove each distributional node, connecting each ordinary node with its closest ordinary ancestor. *(iii)* The resulting px-space is formed of arbitrary representatives of each isomorphism class of the documents obtained after step (ii). The probability of a document is the sum of probabilities of every run that generated an isomorphic image of it. Note that because we consider all possible runs, the order of the choice made in step (i) is irrelevant.

We denote *classes of p-documents* by PrXML with the allowed types of distributional nodes as superscripts. We recall the following results from [1]: PrXML^{mux} cannot represent all finite distributions of documents, yet $\mathsf{PrXML}^{mux,det}$ can, and $\mathsf{PrXML}^{mux,det} \sqsubseteq_{\text{poly}} \mathsf{PrXML}^{exp}$. Also, PrXML^{exp} is tractable for MSO [7].

3. RECURSIVE MARKOV CHAINS

We now adapt *recursive Markov chains* from [11] to the context of document generation, and study their relationship with prior probabilistic XML models.

Definition 1. A recursive Markov chain A, is a tuple $A = (A_0, \cdots, A_k, \mu)$, where μ labels each A_i with an element from $\mathcal{L} \cup \{\varepsilon\}$ and every component A_i is a graph $A_i = (N_i, N_i^{en}, N_i^{ex}, B_i, Y_i, \delta_i)$ that consists of:

(i) A set N_i of *nodes*, a subset of *entry* nodes $N_i^{en} \subseteq N_i$, and a subset of *exit* nodes $N_i^{ex} \subseteq N_i$;

(ii) A set B_i of *boxes*, and a mapping $Y_i : B_i \to \{1, \ldots, k\}$ that assigns to every box (the index of) one of the components, $A_1, \ldots, A_k$. To each box $b \in B_i$, we associate the sets of *call ports*, $Call(b) = \{(b, en) \mid en \in N_{Y_i(b)}^{en}\}$ and *return ports*, $Return(b) = \{(b, ex) \mid ex \in N_{Y_i(b)}^{ex}\}$, that indicate, respectively, the entries and the exits of the component corresponding to b;

(iii) A *transition relation* δ_i, where *transitions* are of the form $(u, p_{u,v}, v)$ and

(a) the source u is either a non-exit node $u \in N_i \setminus N_i^{ex}$, or a return port $u = (b, ex)$ of a box $b \in B_i$,

(b) the destination v is either a non-entry node $v \in N_i \setminus N_i^{en}$, or a call port $u = (b, en)$ of a box $b \in B_i$,

(c) $p_{u,v}$ is the probability of transiting from u to v.

For each u that is neither a call port nor exit node we have $\sum_{\{v \mid (u, p_{u,v}, v) \in \delta_i\}} p_{u,v} = 1$. ■

We distinguish one component in A, say A_0, as the *initial* component, and within that component an *initial node* $a_0 \in N_0^{en}$ and a set of *final nodes* $F_0 \subseteq N_0^{ex}$. We also require that no box in the RMC is mapped to A_0, and that $\mu(A_0) \neq \varepsilon$.

RMCs can be depicted graphically as follows: The components are represented as rectangles containing Markov chains with inner rectangles corresponding to boxes. The name of the component each box is mapped to is given inside the box. In the following figures, the initial component is the one at the top-left, and it has a single initial node and a single final node. The name of a component is given above the rectangle, along with its label.

Example 2. Figure 1 partially shows an RMC A_{ex} with four components D, P, N, and T. For instance, the label of D is $\mu(D) =$ `directory`. D either calls P with probability 0.8 or exits with probability 0.2, and this choice can occur again after returning from the call. The components N: `name` and T: `phone` are not depicted; both have a single edge going from the entrance to the exit with probability 1.

The δ-transitions for D are: $(a_0, 1, u_1)$, $(u_1, 0.2, t)$, also $(u_1, 0.8, (P, en))$, and $((P, ex), 1, u_1)$, where t is the exit node, u_1 is the only node pointed to by a_0, (P, en) is the call port for box P, and (P, ex) is the return port for box P. ■

Intuitively, a run of an RMC generates a document d in a top-down fashion where a call of a box (corresponding to a component) labeled l inside another box (corresponding to a component) labeled l' generates a node l in d that is a child of l'. If a box is labeled ε, then it generates nothing, though calls within its component may still generate labels. We next formalize this via an alternative description of RMCs.

A *vertex* of A_i is either a node in A_i, a call port, or a return port. Let V_i denote the set of all vertices of A_i. Thus, the transition relation δ_i is a set of probability-weighted directed edges on V_i. Let $V = \bigcup_i V_i$, and N, B, Y, and δ be the unions of the corresponding sets. We denote by $q_{u,ex}$ the probability that starting with u one eventually reaches an exit ex in the same component as u.

Definition 3. An RMC A defines a *global* (denumerable) *Markov chain* $M_A = (\mathsf{St}, \Delta)$ as follows:

The *global* states $\mathsf{St} \subseteq B^* \times V \times (\mathsf{Tag}(\mathcal{L}) \cup \{\varepsilon\})$ of M_A are triples of the form (β, u, α), where β is a (possibly empty) sequence of boxes from B that represents the stack of pending recursive calls, u is the current vertex of A, and α is an optional tag.

The function Δ defines probability-weighted directed edges between global states $\Delta : (\mathsf{St} \times \mathsf{St}) \to [0, 1]$:

(i) If $(u, p_{u,v}, v) \in \delta$, then for every sequence of boxes β and $\alpha \in \mathsf{Tag}(\mathcal{L}) \cup \{\varepsilon\}$ there is a Δ-transition from (β, u, α) to (β, v, ε) with probability $p_{u,v}$.

(ii) If $(b, en) \in Call(b)$ and $\mu(A_{Y(b)}) = l \in \mathcal{L}$, then for every β and α there is a Δ-transition from $(\beta, (b, en), \alpha)$ to $(\beta b, en, \langle l \rangle)$ with probability 1.

(iii) If $(b, ex) \in Return(b)$, and $\mu(A_{Y(b)}) = l \in \mathcal{L}$, then for every β and α there is a Δ-transition from $(\beta b, ex, \alpha)$ to $(\beta, (b, ex), \langle / l \rangle)$ with probability 1.

hal-00537778, version 1 - 19 Nov 2010

The initial global state is $\mathsf{st}_0 = (A_0, a_0, \langle\mu(A_0)\rangle)$ and the final ones are $(A_0, q_f, \langle/\mu(A_0)\rangle)$ for each $q_f \in F_0$. ■

M_A can be seen as an "unfolding" of A. This construction indeed forms a Markov chain in the usual sense: The sum of outgoing probabilities from each global state is 1.

EXAMPLE 4. The δ-transitions of A_{ex} induce, for instance, these global Δ-transitions: $(\varepsilon, a_0, \langle\texttt{directory}\rangle) \xrightarrow{1} (D, u_1, \varepsilon)$, $(D, u_1, \varepsilon) \xrightarrow{0.8} (D, (P, en), \varepsilon)$, $(D, (P, en), \varepsilon) \xrightarrow{1} (DP, en, \langle\texttt{person}\rangle)$, $(DP, ex, \varepsilon) \xrightarrow{1} (D, (P, ex), \langle/\texttt{person}\rangle)$. ■

The notions of *path* between two states of M_A and *state reachability* are standard. Since the choices at any two nodes are independent of each other, the *probability* $\mathsf{Pr}(p)$ of a path p is the product of all transition probabilities along p. With every global state $g = (\beta, (b, ex), \alpha)$, we set $\mathsf{LabOf}(g) = \alpha$. Given a path p, we let $\mathsf{LabOf}(p) = p_0 \dots p_n$ be the string in $\mathsf{Tag}(\mathcal{L})^*$ formed by concatenating each $\mathsf{LabOf}(p_i)$, treating ε as the empty string. Given two global states st, st' the probability of transitioning from st to st' is defined as:

$$\mathsf{Pr}(\mathsf{st}, \mathsf{st}') = \sum_{\text{path } p \text{ from } \mathsf{st} \text{ to } \mathsf{st}'} \mathsf{Pr}(p).$$

The probability of a string $\alpha \in \mathsf{Tag}(\mathcal{L})^*$ is

$$\mathsf{Pr}(\alpha) = \sum_{\text{final global state } \mathsf{st}} \left(\sum_{\substack{\text{path } p \text{ from } \mathsf{st}_0 \text{ to } \mathsf{st} \\ \text{with } \mathsf{LabOf}(p)=\alpha}} \mathsf{Pr}(p) \right).$$

The total probability of all finite strings that reach a final state may be strictly less than one: this can happen because strings reach states from which it is impossible to reach a final state, or because with non-zero probability the computation of the Markov chain never returns to a global state with no pending boxes. As shown in [11], one can compute in PSPACE the probability that an RMC generates a finite string. In the case that this is non-zero, we normalize the probability of any string by this number, thus gaining a probability space on strings. For the tractable classes of RMCs we will study later in this work, the computation of this probability is also tractable.

For a given RMC A, let us denote by $L(A)$ the set of strings that have non-zero probability. It is easy to see that any $\alpha \in L(A)$ is a well-formed encoding of a document. Consequently, the set of all documents corresponding to $L(A)$ together with the corresponding probabilities define a px-space associated to A that we denote by $[\![A]\!]$. The corresponding probabilistic XML model is denoted RMC.

EXAMPLE 5. Consider an encoding of the XML skeleton of a directory with two persons, each with two phone numbers:

$\langle\texttt{directory}\rangle\langle\texttt{person}\rangle\langle\texttt{name}\rangle\langle/\texttt{name}\rangle\langle\texttt{phone}\rangle\langle/\texttt{phone}\rangle$
$\langle\texttt{phone}\rangle\langle/\texttt{phone}\rangle\langle/\texttt{person}\rangle\langle\texttt{person}\rangle\langle\texttt{name}\rangle\langle/\texttt{name}\rangle$
$\langle\texttt{phone}\rangle\langle/\texttt{phone}\rangle\langle\texttt{phone}\rangle\langle/\texttt{phone}\rangle\langle/\texttt{person}\rangle\langle/\texttt{directory}\rangle$

This string can be generated by A_{ex} with probability $(0.8 \times (0.5 \times 0.5 \times 0.5))^2 \times 0.2 = 0.2\%$. Note A_{ex} generates skeletons of documents without any actual data. One can account for data by straightforwardly defining N and T with regular expressions specifying permissible names and phone numbers, respectively. Since every document d in $[\![A_{ex}]\!]$ has depth at most two, A_{ex} is shallow. Moreover, since the number of persons rooted under the directory node and the number of telephones per person is arbitrary, A_{ex} is wide. ■

Existing probabilistic XML models such as $\mathsf{PrXML}^{mux,det}$ can be efficiently translated to RMCs. Section 5 discusses efficient translations of other PrXML models into RMC.

PROPOSITION 6. $\mathsf{PrXML}^{mux,det} \sqsubseteq_{\text{poly}} \mathsf{RMC}$.

It is known that verifying MSO properties of RMCs over strings is in PSPACE (this follows from [10]). Results later in the paper show that the same holds for verifying MSO properties of random documents represented by RMCs.

Unfortunately, even for basic problems, RMCs are not known to be tractable. Computing reachability probabilities in RMCs is as difficult as the *square-root sum problem*, a well-known problem in numerical computation [11]. SQRT-SUM asks, given natural numbers k and $d_1, \cdots, d_n$, whether $\sum_i \sqrt{d_i} \geqslant k$. SQRT-SUM is known to be in PSPACE, but its containment even in NP is unknown and has been a long-standing open problem. In [11] it is shown even *approximating* the probability of termination of an RMC to within a fixed rational is as hard as the square-root sum problem. Hence tractable query evaluation for this model would require a breakthrough in numerical computation.

4. TRACTABLE RMC FRAGMENTS

In this section, we propose restrictions of the RMC probabilistic XML model for which some form of tractability of MSO can be achieved. We first show ra-tractability for the class of hierarchical RMCs [11], which are wide and shallow. We then show full tractability for tree-like RMCs, which are also wide and shallow. The latter generalizes an existing result on tractability of the $\mathsf{PrXML}^{mux,det}$ model [7], which is narrow and shallow and less expressive than tree-like RMCs.

Hierarchical RMCs.

DEFINITION 7. An RMC A is *hierarchical* if none of its components can eventually call itself, i.e., in M_A all states of the form $(\beta b \gamma b, u, \alpha)$ (where b is a box, β and γ are sequences of boxes) are unreachable for any u and α. ■

For example, A_{ex} is hierarchical. We denote by HMC the probabilistic XML representation system given by hierarchical RMCs. By forbidding recursion across components, HMC is shallow. However, it can be wide, since it allows paths from return ports to call ports within components.

THEOREM 8. HMC *is ra-tractable for MSO.*

This is a non-trivial extension of a result from [11] which states that computing transition probabilities in HMCs is ra-tractable. The algorithm, detailed in Appendix C, uses the construction of a product automaton of a pushdown automaton equivalent to the HMC and a streaming tree automaton representing the MSO query. We even show a stronger result: ra-tractability for the more expressive, deep representation system PLRMC of piecewise-linear RMCs introduced in [11] (see Appendix C for details).

Tree-like Markov chains.

How can we move from tractability in a unit-cost arithmetic model to full tractability? For our representation systems, this move corresponds to a transition to exponentially less succinct systems. This is the case of HMC, which is ra-tractable, and a fully tractable yet less succinct subclass of it called TLMC. In short, the latter can be obtained from the former by forbidding call sharing. Recall that the call graphs of hierarchical Markov chains are acyclic. Call sharing means that several boxes can call (i.e., be mapped to) the same component.

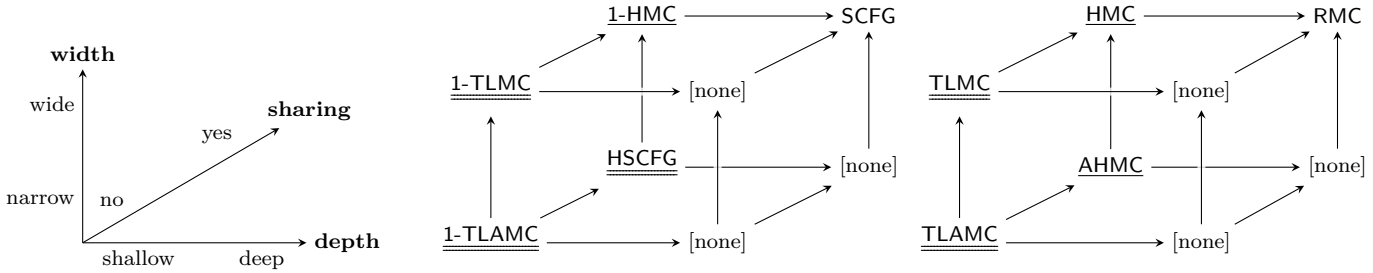


Figure 2: Space of models considered in this paper. Considered dimensions (left), 1-exit models (middle), multi-exit models (right). Tractability of MSO queries: MODEL: tractable, MODEL: ra-tractable, MODEL: SQRT-SUM is reducible to query evaluation for some fixed MSO query.

DEFINITION 9. A hierarchical Markov chain A is *tree-like* if no two boxes are mapped to the same component. ■

It follows from the definition that there is no recursion. This defines a shallow and wide probabilistic model, TLMC, that is a restriction of HMC. The RMC A_{ex} is a tree-like RMC. Note also that the translation of $\mathsf{PrXML}^{mux,det}$ into RMC used in the proof of Proposition 6 (see the appendix) produces a tree-like Markov chain, whose components are moreover acyclic. An example of RMC that is *not* tree-like is given in Theorem 3.2, part (4) of [11] (see also Figure 5 in the appendix).

The global Markov chain defined by a tree-like Markov chain A is finite, and its size is linear in the size of A. A consequence of this is that all reachability probabilities have polynomial size and can be computed in polynomial time as for standard Markov chains. The same automata-based algorithm as in the proof of Theorem 8 reduces calculating the probability of MSO queries over TLMC to calculating reachability probabilities; by combining these two insights, we get an algorithm for efficiently calculating query probabilities:

THEOREM 10. TLMC *is tractable for MSO.*

Unordered models. The existing literature on probabilistic XML models usually assumes unordered documents. We can see RMC, and all other models presented in this paper, as unordered models. Specifically, let $A \in \mathsf{RMC}$ and $L(A)$ be the set of generated strings; the unordered interpretation of A is the set of unordered documents d (with probability $\Pr(d)$) for which there exists $\alpha \in L(A)$ a possible encoding of d under *any* ordering ($\Pr(d)$ is then the sum of probabilities of all such α's). We note that the data complexity results obtained in the ordered setting extend to the unordered one:

PROPOSITION 11. *Let $\mathcal{S}$ be any ordered probabilistic XML representation system, and q any MSO query that does not make use of the order predicate $\prec$. The complexity of the quantitative evaluation problem for q and $S \in \mathcal{S}$ is the same as the complexity of the quantitative evaluation problem for q and the unordered interpretation of S.*

5. BETWEEN P-DOCUMENTS AND RMCS

In the previous section we have shown the existence of tractable restrictions of RMCs. We now explore the space of models in between PrXML and RMC. RMC models extend PrXML in expressiveness in several dimensions. Clearly RMC is deep while PrXML is shallow, it is wide while PrXML is narrow. In addition, in RMCs a component may be called from multiple places, while in PrXML the topology is a tree: a node has a single parent. Finally, by allowing multiple exits in components, RMC allows a state to be passed upward from child to parent, while in PrXML the processing is completely top-down (this is analogous to the distinction in attribute grammars between synthesized and inherited attributes). Summing up, we focus on four dimensions: 1. *depth*, 2. *width*, 3. *sharing*, 4. *number of exits*.

These dimensions, as well as RMC restrictions that are distinguished thereby, are shown in Figure 2. The left part shows the first three dimensions. In terms of depth the models can be either wide or shallow, where shallowness corresponds syntactically to being hierarchical. In terms of width the models can be either wide or narrow, and narrowness can be syntactically enforced by requiring the RMC to be *acyclic*, i.e., every component is an acyclic graph. In the following we denote the acyclic subclasses of HMC and TLMC as AHMC and TLAMC, respectively. In terms of sharing the models can either allow reuse of components in multiple boxes or not. Finally, the models can be either 1-exit or multi-exit, where an RMC is *1-exit* if each of its components has one exit node, and *multi-exit* otherwise. We denote the fact that all the RMCs in a class are 1-exit with the 1- prefix. Observe that the ability to pass information up corresponds to being multi-exit. We use two cubes to represent the fourth dimension: the central cube in Figure 2 covers 1-exit classes and the rightmost one covers the corresponding multi-exit classes.

Note the empty corners of the cubes, labeled `[none]`: Deep models without sharing cannot be obtained as RMC restrictions, and deep models can also generate wide documents, since RMC components can be labeled with ε. The arrows in Figure 2 indicate that the pointing model is a restriction of the pointed one.

The main message of the figure is the insights into the trade-off between succinctness and tractability of MSO, at least for the models considered:

(i) Tractability degrades to ra-tractability by adding sharing. As seen further, this amounts to a gain in succinctness by being able to represent worlds with probabilities doubly exponentially small.

(ii) Neither the width nor the number of exits influences tractability. As we see further in this section (and in Figure 3), for some classes of RMCs multiple exits exponentially increase succinctness.

(iii) Going from shallow to deep affects tractability.

We now investigate in detail the relationship of the classes in the cubes with each other, and with known PrXML models.

hal-00537778, version 1 - 19 Nov 2010

hal-00537778, version 1 - 19 Nov 2010

1-exit vs. Multi-exit Models. One question is whether multi-exit models can be more succinct than 1-exit ones. As we state next, it is the case, except for tree-like RMCs.

PROPOSITION 12.
1. TLAMC $\equiv_{\text{poly}}$ 1-TLAMC *and* TLMC $\equiv_{\text{poly}}$ 1-TLMC.
2. *Let* $\mathcal{S}$ *be one of* AHMC, HMC *or* RMC. *Then* $\mathcal{S} \equiv$ 1-$\mathcal{S}$ *but* $\mathcal{S} \equiv_{\text{poly}}$ 1-$\mathcal{S}$ *does not hold.*

In the appendix we give an efficient algorithm for transforming multi-exit RMCs to single-exit ones that preserves acyclicity: the idea is roughly to replace one multi-exit component with several copies, one for each exit, with the probability of going into the original component distributed among the copies. There is no blow-up in the size of the structures, the only issue is to be able to calculate the probability of entering each copy—this can be done efficiently for TLMC using the efficient model-checking algorithm for TLMC, but cannot be done for the less tractable HMC models.

We note that 1-exit RMCs are important due to their close relationship with stochastic context-free grammars (SCFGs) [11], which have been much studied in connection with computational linguistics. Parse trees of a SCFG together with their probabilities can be seen as a probabilistic space of XML documents, which gives a way to adopt SCFGs as a probabilistic XML representation system. Due to lack of space we do not present the SCFGs formally, and refer the reader to [11]. As shown there, there are linear transformations from SCFGs to 1-exit RMCs and vice-versa. These results carry over to SCFG as probabilistic XML representation systems. That is: SCFG $\equiv_{\text{poly}}$ 1-RMC. This extends to 1-AHMC $\equiv_{\text{poly}}$ HSCFG, where HSCFG denotes the shallow (i.e. non-recursive) variant of SCFGs.

Tree-Like vs. Shared. While multiple exits do not always increase succinctness, the situation is different for sharing. We first discuss sharing for RMCs and then proceed to PrXML. Note that sharing in the former case is different in nature from the one in the latter case.

For RMCs, sharing means that a component can be called from multiple places and this adds succinctness:

PROPOSITION 13. TLMC $\sqsubseteq_{\text{poly}}$ 1-HMC, *but it does not hold that* 1-HMC $\sqsubseteq_{\text{poly}}$ TLMC.

PROOF. 1-HMC can express px-spaces with doubly exponentially small probabilities of documents that are not computable in polynomial time (a p-document from Example 6.1 in [7] can be translated into such a one-exit hierarchical Markov Chain). At the same time, TLMC expresses px-spaces where the probability of every document is computable in polynomial time. The reason is any TLMC can be expanded in polynomial time into a regular Markov chain, where reachability probabilities correspond to probabilities of documents generated by the original TLMC. Moreover, reachability probabilities in regular Markov chains can be computed by solving a system of linear equations, hence in polynomial time. Therefore, 1-HMC is at least exponentially more succinct than TLMC, while both models have the same expressive power since it is possible to expand a hierarchical Markov chain into a tree-like one by duplicating components used more than once as boxes. □

Similarly (it can be shown with the same example from [7]), 1-AHMC is exponentially more succinct than TLAMC.

For PrXML, sharing comes from a generalization of PrXML^{exp} and PrXML^{mux,det} models (Section 2) and consists in introducing *sharing* of nodes in the trees. We now investigate their place in the hierarchy of RMC subclasses that we have introduced.

P-documents with sharing, PrDAG, are defined as rooted directed acyclic graphs (DAGs) with the possibility of having multiple edges between two given nodes and a total ordering on the edges outgoing from a given node, that make use of both *ordinary* and *distributional* nodes, as with regular p-documents. The semantics of a p-document with sharing is defined as the semantics of the regular p-document obtained by repeatedly duplicating subtrees whose root has more than one incoming edge until a tree is obtained. That is, for any node that has n incoming edges in the DAG we introduce n copies of it in the corresponding p-document, one per edge. As with p-documents, we denote with, for example, PrDAG^{mux,det} the class of p-documents with sharing that use only *mux* and *det* distributional nodes. It turns out that PrDAG^{exp} is more succinct than PrXML^{exp}.

PROPOSITION 14. PrXML$^{exp} \sqsubseteq_{\text{poly}}$ PrDAG^{exp}, *but it does not hold that* PrDAG$^{exp} \sqsubseteq_{\text{poly}}$ PrXML^{exp}.

This already follows from [7], where a partial form of sharing (allowing duplicated edges) has been noted, in particular, to make the probability of a possible world potentially doubly exponentially small, which requires an exponential size to be represented.

The following shows that all shallow and narrow models with sharing and no passing of information bottom up are equally expressive and succinct.

PROPOSITION 15. PrDAG$^{mux,det} \equiv_{\text{poly}}$ PrDAG$^{exp} \equiv_{\text{poly}}$ HSCFG.

Note that all these models occupy the back, bottom, left corner in the middle cube of Figure 2. Translation between these models is relatively straightforward and can be done in linear time.

PrXML *vs. subclasses of* RMC. We finally give a precise characterization of the PrXML^{mux,det} position in the hierarchy of models. Given that PrXML^{mux,det} models are shallow, narrow, and tree-like, one might think they occupy the lowest point in the lattice, and are hence essentially the same as TLAMC. Surprisingly, this is not the case:

THEOREM 16. PrXML$^{mux,det} \sqsubseteq_{\text{poly}}$ 1-TLAMC, *but it does not hold that* 1-TLAMC $\sqsubseteq_{\text{poly}}$ PrXML^{exp}.

The key is that TLAMC processes can still pass a limited amount of information over long distances in a tree, while PrXML cannot.

Figure 3 summarizes the expressive power and relative succinctness of the various models introduced in this article. The solid (resp., dashed) arrows indicate there is a polynomial-time (resp., only exponential-time) translation from the pointing model to the pointed one. The left-hand side shows if (polynomial-time) translations exist between the narrow and shallow representation systems, i.e., it zooms on the bottom-left edge of the cubes of Figure 2. An open problem is whether PrXML^{exp} can be efficiently translated into PrXML^{mux,det} (already stated as open in [1]) or into TLMC. The top-right of Figure 3 presents expressiveness and conciseness results for shallow and wide models, i.e., those of the top-left edge of the cubes. Finally, the bottom-right

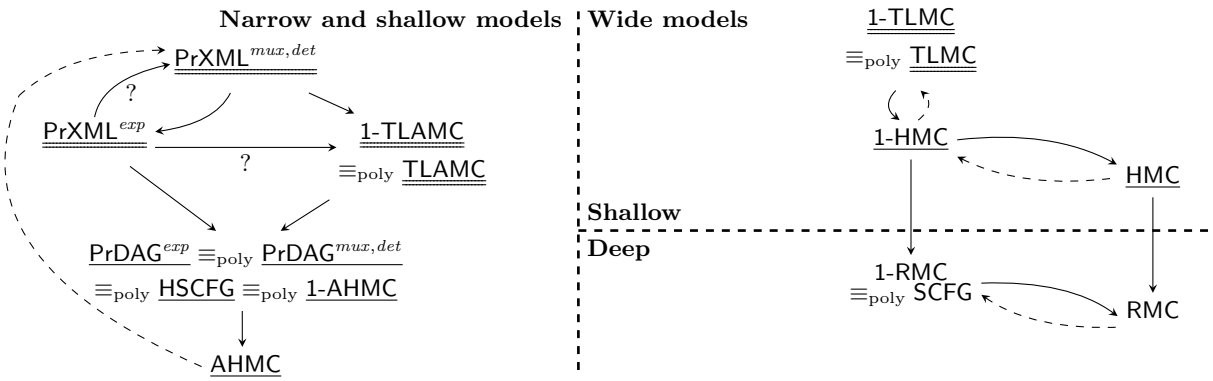


Figure 3: Translations between probabilistic XML representation systems.
" ⟶ ": polynomial translation, " ⇢ ": existence of a translation, "?": no efficient translation known.

of Figure 3 compares deep and wide models, the ones on the back top-right vertex of the cubes. For readability, translations across narrow and wide models are not shown, but they are straightforward: TLAMC, 1-AHMC, and AHMC can be translated in polynomial time to, respectively, TLMC, 1-HMC and HMC (and no translation exists in the other direction). Apart from these and the open questions discussed above (shown with a "?"), the figure is complete in the following sense: there is a (solid) path between any two models if and only if there is a (polynomial-time) translation.

6. COMBINED COMPLEXITY

In Section 4 we have discussed data complexity of query evaluation for a very rich query language, MSO. We now comment on the combined complexity of query evaluation. We focus here only on the models that we have shown ratractable. For the most general model, RMC, the best we can say about the complexity of even simple queries (e.g., tree patterns) is that it is in PSPACE, via reduction to the existential theory of the reals (by iterating over an exponential number of automata whose union is equivalent to the query, and then applying the PSPACE algorithm of [11] for computing reachability probabilities in RMCs).

Lowest combined complexity: tree-pattern queries. For *tree patterns*, the complexity of querying any of our tractable models is in $\mathsf{FP}^{\#\mathsf{P}}$ (the class of computation problems that can be solved in polynomial time using a #P oracle). Tree patterns can be converted to a disjoint union of polynomial-size deterministic tree automata (by guessing an ordering of siblings), where each tree automaton is of linear size in the pattern. #P-hardness of tree-pattern evaluation holds even for $\mathsf{PrXML}^{mux,det}$ [16].

Highest Combined Complexity: MSO. For full MSO the complexity of satisfiability over words is known to be non-elementary [22]. Hence, determining whether an MSO query has positive probability is non-elementary, in any of our wide models (e.g., HMC, TLMC).

If we turn to shallow and narrow models, we note that the combined complexity of MSO query evaluation is PSPACE-complete for on $\mathsf{PrXML}^{mux,det}$. Membership in PSPACE follows because each particular isomorphism type of a tree is of size linear in the representation, and we can evaluate an MSO query in PSPACE combined complexity over any fixed tree [18]. Thus we can determine the probability of a given query by simply generating every possible world d, recording the probability p of generation, evaluating the query in PSPACE on d, and then adding p to the running total if the query is satisfied. PSPACE-hardness is clear because the evaluation of MSO (indeed, even first-order logic) over a fixed tree is PSPACE-hard [18]. The same PSPACE-completeness result holds for any of our representation systems such that each representation S has a domain consisting only of documents of size polynomial in S. In particular, the same holds for PrXML^{exp} and TLAMC.

For AHMC, representations S may generate documents at most exponential in the size of S. Thus, MSO probability evaluation is in EXPSPACE, again simply by iterating over documents and updating the probability. A lower bound can be obtained using techniques from [12] where it is shown that the combined complexity of MSO evaluation over compressed trees is NEXPTIME-hard. The result holds even for the restricted fragment of existential MSO. By adapting the argument of [12] and taking into account the fact that the probability evaluation problem is essentially a counting problem, we can show it is #EXP-hard:

PROPOSITION 17. *Given an existential MSO formula q and $S \in$ AHMC, computing $\Pr(q \models S)$ is #EXP-hard.*

In between: Forward Navigational XPath. An intermediate language consists of forward navigational XPath (or FNXPath) filters, where *navigational* indicates that data joins are excluded [19], and *forward* that we only consider the child and descendant axes. We refer to [19] for detailed syntax and semantics.

We can consider each filter to define a Boolean query over documents; the query checks whether the root of the document satisfies the filter. It is known from [14] that FNXPath can be evaluated in polynomial time on XML documents. From this it follows that FNXPath can be evaluated in $\mathsf{FP}^{\#\mathsf{P}}$ on $\mathsf{PrXML}^{mux,det}$ models (the argument is similar to the one given in [15] for computing the probability of a logical formula). Hence, it is $\mathsf{FP}^{\#\mathsf{P}}$-complete. The same argument shows that FNXPath is in $\mathsf{FP}^{\#\mathsf{P}}$ for any formalism in which representations generate only trees of polynomial size.

Because satisfiability of FNXPath is PSPACE-hard [5] even on trees of fixed depth, the evaluation problem for FNXPath is PSPACE-hard for any of our unbounded models. In addition, [12] shows that evaluation of FNXPath is PSPACE-hard on *compressed documents*, which are subsumed by AHMC and thus by HMC. Thus query evaluation for these formalisms is PSPACE-hard. We provide in the appendix an algorithm for FNXPath that is in PSPACE with respect to query complexity on AHMC.

hal-00537778, version 1 - 19 Nov 2010

hal-00537778, version 1 - 19 Nov 2010

THEOREM 18. *There is a query evaluation algorithm for FNXPath on* AHMC *that runs in* PSPACE *in the query and uses a number of arithmetic operations that is polynomial in the model size.*

The key to the algorithm is that we do not calculate the probability of every tree, but only the probability of every "tree type", where a type abstracts trees according to which subexpressions of the the FNXPath expression it satisfies. The algorithm calculates the probability of the type by traversing the AHMC top-down, calculating probabilities for all compatible types of child subcomponents and combining them using the transition probabilities.

PSPACE combined complexity is arguably a sign that the tractability results in terms of data complexity of Section 4 apply in practice to non-trivial queries. The algorithm of Theorem 18 subsumes the well-studied case of tree-pattern queries, and applies to the most general narrow and shallow probabilistic XML representation system presented here.

7. DISCUSSION

Contribution. This work departs from the mainstream approach to probabilistic XML models, seen as XML documents with additional features. We propose new models based on variations of Markov chains, which go beyond the state-of-the-art in their ability to represent deep or wide documents, or worlds with probabilities double-exponentially small. This shift also gives new insights into tractability and expressiveness of the existing PrXML models.

The following models stand out in particular from the picture given in Figure 2: *(i)* TLMC because of its tractability in the usual bit cost arithmetic model together with its ability to represent wide trees; *(ii)* AHMC, a narrow and shallow model that is tractable in unit-cost arithmetic and that has PSPACE combined complexity; *(iii)* HMC, a wide model that is tractable in unit-cost arithmetic which is deep and also ra-tractable.

Applications. RMC can be seen as a probabilistic version of DTDs (or XML Schemas) and HMC as non-recursive DTDs. For example, consider the following DTD fragment describing an XML phone directory, with persons, each having a name and phone numbers:

```
<!ELEMENT directory (person*)>
<!ELEMENT person (name,phone*)>
```

A probabilistic version of this DTD is A_{ex} of Figure 1. An intriguing question is whether these probabilistic schemas can be applied to obtain more fine-grained models of collections of real-world documents which may not satisfy any non-trivial traditional (i.e., deterministic) schema. The learning algorithms available for SCFGs (e.g. [20]) can be applied here, although it remains to examine whether they can be tailored to produce output within particular subclasses that we have identified as interesting (e.g. tree-like).

One can also use our results to apply XML querying formalisms to the probabilistic trees given by SCFGs. The idea has been applied to querying the possible parses of a given string in [6]. Our results show that it is possible to query the (possibly infinite) collection of parse trees that can be produced by an SCFG language model using a powerful query language like MSO. One could, for instance, compute probabilities that a phrase having a specific structure appears in a corpus generated by the grammar.

8. REFERENCES

[1] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB Journal*, 18(5), 2009.

[2] S. P. Abney, D. A. McAllester, and F. Pereira. Relating probabilistic grammars and automata. In *ACL*, 1999.

[3] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *TOPLAS*, 27(4), 2005.

[4] L. Antova, C. Koch, and D. Olteanu. "10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information". *VLDB Journal*, 18(5), 2009.

[5] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *JACM*, 55(2), 2008.

[6] S. Cohen and B. Kimelfeld. Querying parse trees of stochastic context-free grammars. In *ICDT*, 2010.

[7] S. Cohen, B. Kimelfeld, and Y. Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

[8] J. Esparza, A. Kucera, and R. Mayr. Model checking probabilistic pushdown automata. In *LICS*, 2004.

[9] K. Etessami, D. Wojtczak, and M. Yannakakis. Quasi-birth-death processes, tree-like QBDs, probabilistic 1-counter automata, and pushdown systems. Technical Report 1249, U. Edinburgh, 2008.

[10] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic state machines. In *TACAS*, 2005.

[11] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *JACM*, 56(1), 2009.

[12] M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees. In *LICS*, 2003.

[13] O. Gauwin, J. Niehren, and Y. Roos. Streaming tree automata. *Inf. Process. Lett.*, 109(1), 2008.

[14] G. Gottlob and C. Koch. Monadic Datalog and the expressive power of Web information extraction languages. *JACM*, **51**(1), 2004.

[15] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, 1998.

[16] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query evaluation over probabilistic XML. *VLDB Journal*, 18(5), 2009.

[17] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic XML. In *VLDB*, 2007.

[18] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[19] M. Marx. Semantic characterizations of navigational XPath. *SIGMOD Record*, 34, 2005.

[20] S. Petrov, L. Barrett, R. Thibaux, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *ACL*, 2006.

[21] C. Ré, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD*, 2008.

[22] L. Stockmeyer. *The Complexity of Decision Problems in Automata and Logic*. PhD thesis, MIT, 1974.

[23] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*. Springer-Verlag, 1997.

hal-00537778, version 1 - 19 Nov 2010

APPENDIX

A. RELATED WORK

A unified view of existing probabilistic XML models, in the form of p-documents, has been proposed in [1]. Translations between families of p-documents, depending on the kind of distributional nodes allowed, are studied in detail in [1]. Traditionally, probabilistic XML models have been unordered, shallow, and narrow. The first efficient algorithm for evaluating tree-pattern queries in $\mathsf{PrXML}^{mux,det}$ has been proposed in [17]; this has been extended to PrXML^{exp} and Boolean combinations of tree-pattern queries in [16], and extended further to MSO queries in [7] where Cohen et al. show how to run a deterministic bottom-up automaton on an ordered PrXML^{exp} document. The same work introduces a limited form of sharing in p-documents (the possibility of re-using multiple times a given child), that is a restriction of the PrDAG^{exp} model. Recent work [6] uses the techniques of probabilistic XML to compute the probability of a tree-pattern query over the set of parse trees of a given string in a stochastic context-free grammar. If the grammar has a specific property, a generalization of Chomsky normal form, it is possible to compute these probabilities in polynomial time. We stress, however, that the problem studied in [6] is fundamentally different from the work presented here: the probabilistic XML model is not the SCFG, but the SCFG together with an input string. And all possible worlds share the same set of leaf nodes, since possible worlds are all parse trees of this string.

Recursive Markov chains were introduced by Etessami and Yannakakis in [11] as probabilistic versions of recursive state machines [3], used for the static analysis of programs and systems. An equivalent formalism, probabilistic pushdown systems, was introduced independently in [8]. Some other stochastic formalisms from the literature (notably, tree-structured quasi-birth-death processes) have been shown to be equivalent to RMCs [9], while other still (quasi-birth-death processes, probabilistic 1-counter automata) are a restricted class of RMCs [9]. Stochastic (or probabilistic) context-free grammars are another example of widely used model (notably in natural language processing) that can be seen as a particular case of RMCs, specifically, 1-exit RMCs. While context-free grammars are as succinct as pushdown automata, probabilistic context-free grammars, though they generate the same probabilistic distributions as probabilistic pushdown automata, are exponentially less succinct [2]. Algorithms for computing reachability probabilities in recursive Markov chains are presented in [11]. This is extended in [10] to probabilities of verifying a given specification, through the construction of a product automaton, similarly to the construction we use to prove Theorem 8.

B. MATERIAL FOR SECTION 3

Proof of Proposition 6. Let $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{mux,det}$. We construct the corresponding RMC A as follows. The RMC contains one component A_v for every node v of $\widehat{\mathcal{P}}$. The component A_0 corresponds to the root of $\widehat{\mathcal{P}}$. Consider cases. (i) v is a regular (resp., *det*) node labeled a with $m \geqslant 0$ children labeled $a_1, \cdots, a_m$. The corresponding component A_v is given in Figure 4 (left), where $\mu(A_v) = a$ (resp., $\mu(A_v) = \varepsilon$). (ii) v is a *mux* node with m children $v_1, \cdots, v_m$ and probabilities $p_1, \cdots, p_m$. This corresponds to the component A'_v in Figure 4, where $\mu(A'_v) = \varepsilon$.

This construction produces an RMC of a very restricted form, called 1-exit tree-like Markov chain (see Definition 9). □

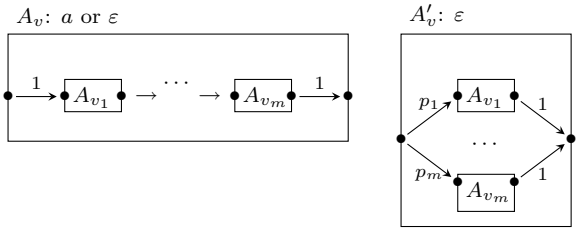


Figure 4: Translation from $\mathsf{PrXML}^{mux,det}$ to RMC. Left component A_v for a $\mathsf{PrXML}^{mux,det}$ node $a(v_1, \ldots, v_m)$ or $det(v_1, \ldots, v_m)$. Right component A'_v for a $\mathsf{PrXML}^{mux,det}$ node $mux(v_1, \ldots, v_m)$.

C. MATERIAL FOR SECTION 4

We show the ra-tractability of the class of piecewise-linear RMCs [11], PLRMC. In order to introduce this class of models, we need the notion of call graph for RMCs, that is similar to call graphs in procedural programs. The *call graph* CG_A of an RMC A has a node i for each component A_i of A and a directed edge from node i to node j if there are mappings from boxes of A_i to the component A_j.

Definition 19. An RMC A is *piecewise-linear* if in every component A_i, for every path from an entry node to an exit node, there is at most one box which is mapped to a component in the same strongly connected subgraph of the call graph CG_A as A_i. ■

PLRMC is deep, since it allows for recursion (for instance, the box within a component can be mapped to that same component), and also wide, since within a component there may be paths with several boxes from different strongly connected subgraphs of the call graph. The notion of piecewise-linearity is a generalization of the standard notion of linearity of context-free grammars, which allows at most one nonterminal on the right-hand side of each grammar rule; when adapted to RMCs, this constraint states that for every path from an entry node to an exit node, there is at most one box. It is easy to see that PLRMC also generalizes HMC.

Theorem 20 (Generalization of Theorem 8). PLRMC *is ra-tractable for MSO.*

To prove this result, we first introduce an alternate representation for RMCs in the form of probabilistic pushdown automata [2]. We then show how to construct a product of this automaton with a tree automaton encoding the MSO formula, such that reachability of some state in the resulting automaton corresponds to the probability of the query. This is similar in spirit to what is done in [10] for model-checking RMCs with respect to a Büchi automaton.

Definition 21. A *probabilistic pushdown automaton* (or pPDA) is a tuple $P = (Q, \Gamma, q_0, \Delta, \nu)$ that consists of a finite *set of control states* Q, a finite *stack alphabet* Γ, *initial* state $q_0 \in Q$, a *probabilistic transition relation* Δ with transitions of the form:

$$(q, \gamma) \xrightarrow{p, push(\gamma')} q', \text{ or } \quad (q, \gamma') \xrightarrow{p, pop} q', \quad \text{or} \quad (q, \gamma) \xrightarrow{p} q',$$

hal-00537778, version 1 - 19 Nov 2010

where $q, q' \in Q$, $\gamma \in \Gamma \cup \{\bot\}$, $\gamma' \in \Gamma$, and $p \in [0,1]$ is the probability of the transition, and a functions ν mapping symbols of Γ to labels in $\mathcal{L} \cup \{\varepsilon\}$. For any state (q,γ) with outgoing transitions, transition probabilities sum up to 1. ■

Probabilistic pushdown automata are seen as a probabilistic XML model pPDA in the following way. Intuitively, a run of a pPDA generates a document d in a top-down fashion: if $\nu(\gamma) = a$, $\nu(\gamma') = a'$, and the top element of the stack is γ, then a push of γ' on top of γ, by means of the instruction $push(\gamma')$, generates a node in d labeled a' as a child of the node labeled a. When the last child of a is generated, or when a is a leaf, the automaton *pops* γ from the stack and proceeds to generate siblings of a. If $\nu(\gamma) = \varepsilon$, then neither pushing nor popping γ affects document generation. We consider the generation finished when a *pop* operation empties the stack. A pPDA assigns probabilities to runs, and thus to documents. We can also assign probabilities to runs that do not terminate on an empty stack. Thus for any state $q \in Q$, we can also talk about the probability of generating a *partial document* (d,n) in state q: by this, we mean the probability of generating a string ending in the closing tag of n in state q, such that the completion of this string with closing tags is the document d.

As shown in [11], RMCs and pPDAs are essentially equivalent in the sense that there are linear translations from RMCs to pPDAs and the other way around such that there is some mapping from the states of the global Markov chain defined by each representation, with the same transition probabilities between states (see [11] for a more formal statement). This equivalence very naturally extends to the probabilistic XML models defined by these representations.

PROPOSITION 22. $\mathsf{RMC} \equiv_{\text{poly}} \mathsf{pPDA}$.

It is well known that MSO queries can be converted to bottom-up deterministic tree automata [23]. In contrast, pPDAs work probabilistically, but generate input top-down. To prove Theorem 20, it will be more useful to convert the MSO query into an automaton of the same "form"—one that traverses the input in pre-order. A deterministic streaming tree automaton (DSTA) is an (accepting) pushdown automaton on strings $(Q, \mathsf{Tag}(\Sigma), \Gamma, q_0, F, \delta)$ where the input alphabet is the collection of begin and end tags over some alphabet Σ, Γ is the stack alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ are the accepting states, δ is the transition function, and

- on input $\langle a \rangle$ there are only push transitions;
- on input $\langle /a \rangle$ there are only pop transitions.

We run a DSTA on a document d by running it on its string representation in the usual sense. We can also run the DSTA on any prefix of this document string. By a run of a DSTA on d up until node $n \in d$, we mean on the substring up until the closing tag of n. We assume that our DSTAs have total transition functions, and hence for every DSTA A, document d and node n, there is a unique run of the automaton on d up until n, and we can talk about the state of A when run on (d,n). We use the following result from the literature on nested word automata and streaming tree automata.

FACT 23. *[13] Every bottom-up tree automaton on ordered trees can be converted to an equivalent DSTA.*

Given a pPDA $A = (Q, \Gamma, q_0, \Delta, \nu)$ and a DSTA $A' = (Q', tag(\Gamma), \Gamma', q_0', \delta')$ we can form the product $A \times A'$, which is a pPDA with:

(i) states $Q \times Q'$;
(ii) stack alphabet $\Gamma \times \Gamma'$;
(iii) initial state (q_0, q_0');
(iv) for every transition $(q_1,\gamma) \xrightarrow{p,push(\gamma')} q_2$, and for every state $q_1' \in Q'$ with transition $(q_1', \langle\gamma'\rangle) \xrightarrow{push(\gamma'')} q_2'$, a transition $((q_1,q_1'),\gamma) \xrightarrow{p,push(\gamma',\gamma'')} (q_2,q_2')$;
(v) for every transition $(q_1,\gamma) \xrightarrow{p,pop} q_2$, and for every state $q_1' \in Q'$ with transition $(q_1', \langle/\gamma\rangle) \xrightarrow{pop} q_2'$, a transition $((q_1,q_1'),\gamma) \xrightarrow{p,pop} (q_2,q_2')$;
(vi) for every transition $(q_1,\gamma) \xrightarrow{p} q_2$, and for every state $q' \in Q'$, a transition $((q_1,q_1'),\gamma) \xrightarrow{p,pop} (q_2,q_2')$;
(vii) labeling function ν'' with $\nu''(q,q') = \nu(q)$.

This product has the following important property:

LEMMA 24. *Let A be a pPDA and A' a DSTA. The probability that $A \times A'$ generates a partial document (d,n) in state (q,q') is the probability that A generates a partial document (d,n) in state q and that the unique run of A' on d until n ends in state q'. In particular, determining the probability that A generates a document accepted by A' reduces to calculating reachability probability of states in $A \times A'$.*

PROOF. This is proven by a direct transformation of runs in $A \times A'$ to pairs of runs of A and A', where the transformation preserves probabilities. □

We now give the intuition of our proof of Theorem 20. Given a piecewise-linear RMC M and an MSO formula F, we (i) construct (in linear time) a pPDA A equivalent to M (Proposition 22), (ii) translate F into a MSO formula F' over the stack alphabet of A, (iii) convert F' into a DSTA A' (Fact 23) and (iv) construct the product $A \times A'$. It can be shown that the product is still a piecewise-linear RMC. We conclude by Lemma 24 and the ra-tractability of reachability probability computation in piecewise-linear RMCs [11]. More formally:

PROOF OF THEOREM 20. Let F be an MSO formula over documents labeled in $\mathcal{L}$ and M a piecewise-linear RMC. First construct from M (in linear time) an essentially equivalent pPDA $(Q, \Gamma, q_0, \Delta, \nu)$ as described in [11]. We must now rewrite F so that the automaton constructed from it can directly be composed with this pPDA, even though some of the push transitions of the automaton do not produce any labels. To do this, we transform F into an MSO formula over documents labeled in Γ in the following way:

- Every subformula $\forall x \varphi$ is replaced with $\forall x \neg(\gamma_1(x) \vee \cdots \vee \gamma_k(x)) \rightarrow \varphi$, where $\gamma_1 \ldots \gamma_k \in \Gamma$ are all stack symbols such that $\nu(\gamma) = \varepsilon$.
- Every subformula $\exists x \varphi$ is replaced with $\exists x \neg(\gamma_1(x) \vee \cdots \vee \gamma_k(x)) \wedge \varphi$ where $\gamma_1 \ldots \gamma_k \in \Gamma$ are all stack symbols such that $\nu(\gamma) = \varepsilon$.
- Every occurrence of a label predicate $l(x)$ is replaced with the formula $\gamma_1(x) \vee \cdots \vee \gamma_k(x)$ where $\gamma_1 \ldots \gamma_k \in \Gamma$ are all stack symbols such that $\nu(\gamma) = l$.

Observe that the probability that F satisfies a document generated by M is equal to the probability that F' satisfies a document generated by $A = (Q, \Gamma, q_0, \Delta, \mathrm{id})$ where id is the identity function. Then we convert F' into a DSTA A'. Lemma 24 shows that the probability that F satisfies a document generated by M is the probability of reaching the empty stack with a pop operation in $A \times A'$ (we can see

hal-00537778, version 1 - 19 Nov 2010

this as the probability of reaching a state q_f by adding a transition $(\bot, q) \xrightarrow{p} q_f$ for every state $q \neq q_0$). Let M' be an RMC that is equivalent to $A \times A'$.

Remark that if M is hierarchical, then M' is also hierarchical (if Q can be partitioned into $\bigcup_i Q_i$ such that a push transition always goes from a state of Q_k to a state of $Q_k + 1$, then $Q \times Q'$ can be stratified into $\bigcup_i Q_i \times Q'$, with the same property). Similarly, it can be checked that if M is piecewise-linear, then M' is also piecewise-linear. Informally, piecewise linearity means that when the RMC has recursion, then this recursion is linear. This property is preserved through the equivalence to pPDAs. Furthermore, the product construction neither add non-linearity nor recursion.

Since reachability probabilities are ra-tractable in piecewise-linear RMCs, the probability that F satisfies M is computable in PTIME assuming unit-cost rational arithmetic. We also need to normalize by the probability of termination of the RMC M, which is just another reachability probability. □

We now prove the tractability of tree-like Markov chains.

Proof of Theorem 10. We apply the same construction as in the proof of Theorem 20. If M is tree-like then M' is also tree-like. The tree-like property can be seen in pPDAs as a condition that the push and pop transition graph is a forest of connected components of states, and this is preserved by the product construction. Reachability probabilities in tree-like Markov chains are just reachability probabilities in regular Markov chains, and they are computable in polynomial time. □

D. MATERIAL FOR SECTION 5

1-exit vs. Multi-exit Models. We prove Proposition 12 with the help of the two following lemmas: item 1 is a direct consequence of Lemma 25 and item 2 is obtained by combining Lemmas 25 and 26.

Lemma 25. *Let $\mathcal{S}$ be one of* TLAMC, AHMC, TLMC, HMC, PLRMC, RMC. *Then $\mathcal{S} \equiv$ 1-$\mathcal{S}$. Moreover, if reachability probabilities can be computed in polynomial time in $\mathcal{S}$ (with bit-cost arithmetic), then $\mathcal{S} \equiv_{\text{poly}}$ 1-$\mathcal{S}$.*

Proof. For $\mathcal{S} =$ RMC, we use a result from [2] that shows that every pPDA can be (inefficiently) translated into a SCFG that generates the same string language (recall that pPDA $\equiv_{\text{poly}}$ RMC).

The other cases are subclasses of PLRMC, for which reachability probabilities are rational [11]. Let $S \in \mathcal{S}$. Since any RMC can be transformed into a 1-entry equivalent RMC in polynomial time [11], we assume that all components of S are 1-entry. Let A_i be a component of S with m exits. We substitute A_i with m components $A_i^1, \ldots, A_i^m$, that are 1-exit copies of A_i where only the jth exit has been kept. We also remove from A_i^j all nodes from which the exit node is not reachable and renormalize the probabilities to obtain an Markov chain in which the probability of reaching the exit from the entry is 1, provided that all boxes terminate with probability 1. This can be done in polynomial time. Then one replaces every box b calling A_i with m boxes b_j, one for each A_i^j. Finally one substitutes the transition $(v, p, (b, en))$ with m transitions $(v, p \cdot p_j, (b_j, en))$, where p_j is the probability of reaching the jth exit of A_i starting from its entrance (we assumed this is a rational probability). The resulting RMC generates the same px-space, and the construction preserves the hierarchical, tree-like, and acyclic properties. □

Lemma 26. *Consider the query $q = \exists x\ c(x)$. For all $n \geqslant 0$, there is an* AHMC *H_n of size $O(n)$ such that $\Pr(q \models H_n) = 1 - 2^{-2^n}$. In contrast, there is no* 1-RMC *of size polynomial in n with the same query probability.*

Proof. A hierarchical acyclic Markov chain that has the required property is given in Figure 5. The non-existence of a 1-exit RMC of size polynomial in n with a reachability probability doubly exponentially close to 1 (but not equal to 1) was proved in [11], Lemma 29, and it directly yields the second part of the lemma. □

Piecewise-linearity. We compare the piecewise linearity condition with the other classes. A direct consequence of Theorem 8.11 and Theorem 3.2, Case (1) of [11] is that RMC is strictly more expressive than PLRMC. The former model can express irrational reachability probabilities, while the latter cannot. In turn, PLRMC is more expressive than HMC since HMC is not a deep model. Moreover it follows from [11] that PLSCFG $\equiv_{\text{poly}}$ 1-PLRMC.

E. MATERIAL FOR SECTION 6

We explain the algorithm behind Theorem 18 first in the one-exit case. In this case, we can assume without loss of generality that every component has at most two boxes in it, since every one-exit machine can be rewritten in this form.

The closure of an FNXPath filter F, denoted $cl(F)$, is the set of its subformulas. An F-type is a subset of $cl(F)$. A node in a tree satisfies an F-type if it satisfies exactly the formulas in the type. The idea of our algorithm will be to work top-down on the components, guessing types for every box.

We need the following properties of the closure of F: (i) it consists only of FNXPath filters; (ii) it consists of only a polynomial number of filters (in size of F). Most importantly, it satisfies a "composition lemma", given below. A 2-*pointed tree* is a tree with two distinguished leaves. If p is a two-pointed tree and t_1, t_2 are trees, $p_2[t_1, t_2]$ is the tree formed from plugging in t_1 in to the first port of p (that is, identifying the root of t_1 with the port) and t_2 into the second port. The composition lemma states:

Lemma 27. *Suppose p is a 2-pointed tree with neither of its ports equal to its root, and suppose t_1 and t_1' are two trees whose roots satisfy the same filters in $cl(F)$, and similarly for t_2, t_2'. Then for any filter in $cl(F)$,*
$(p[t_1, t_2], \mathsf{root}(p[t_1, t_2])) \models F \Leftrightarrow (p[t_1', t_2'], \mathsf{root}(p[t_1, t_2])) \models F$

Proof. Simultaneous induction on the height of p and F. For atomic expressions this is clear, while for Boolean operators it follows immediately by induction.

Suppose $(p[t_1, t_2], root(p[t_1, t_2])) \models \mathsf{child}[G]$. There is a child c of the root that witnesses this. If this child is the root of t_1, then we know $(t_1, root(t_1)) \models G$, and thus the same holds for $(t_1', root(t_1'))$ by assumption. But then $(p[t_1', t_2'], root(p[t_1, t_2])) \models \mathsf{child}[G]$, with the root of t_1' as the witness. We argue symmetrically if c is the root of t_2.

If c is not in t_1 or t_2, then we let p' be the subtree of root rooted at c, and we conclude that $p'[t_1', t_2']$ satisfies G by induction on height, and thus that $p[t_1', t_2']$ satisfies F. □

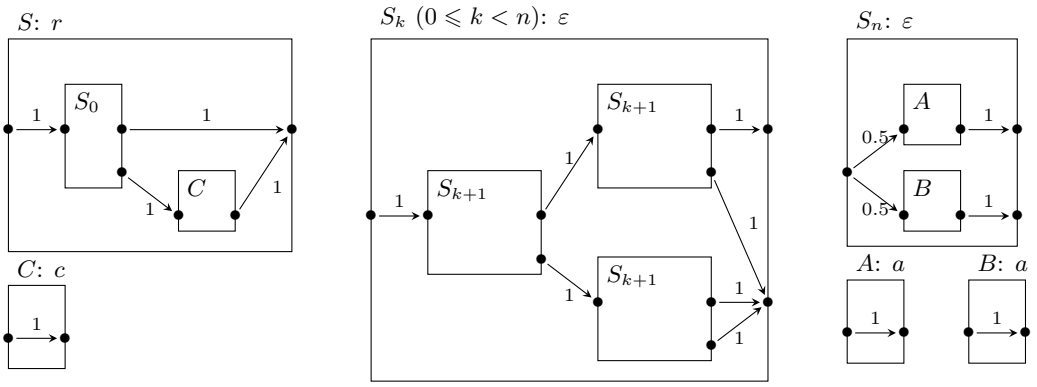


Figure 5: Acyclic hierarchical Markov chain of size $O(n)$ such that $\exists x\, c(x)$ has probability $1 - 2^{-2^n}$.

That is, for fixed p, the type of $p[t_1, t_2]$ depends only on the types of t_1 and t_2. We also need the following:

LEMMA 28. *Given a filter F, three F-types tp_1, tp_2 and tp_3 and a two-pointed tree $p[x, y]$ one can check in* PSPACE *whether there are trees t_2 and t_3 with the root of $p[t_2, t_3]$ satisfying tp_1 in $p[t_2, t_3]$, the root of t_2 satisfying tp_2 in t_2, and the root of t_3 satisfying tp_3 in t_3.*

PROOF. We need to know if there is a tree q with distinguished nodes m_2, m_3 (given, e.g., as distinguished labels), such that the subtree of the root minus the subtrees of m_2 and m_3 is isomorphic to p, and the nodes $m_1, m_2, root(q)$ satisfy the types tp_1, tp_2, tp_3 within the appropriate trees.

Any isomorphism type of an ordered tree can be described in FNXPath, hence this can be expressed as an FNXPath expression of polynomial size, using an enhanced label alphabet telling whether a particular node is m_2 or m_3. The result now follows, because satisfiability of FNXPath on trees of fixed depth d is in PSPACE in the size of the expression and the depth [5]. □

Call a 4-tuple (tp_1, tp_2, tp_3, p) *consistent* if the above holds. Note that by Lemma 27, if (tp_1, tp_2, tp_3, p) is consistent then for *every* t_2, t_3 whose roots satisfy tp_2, tp_3, respectively, the tree $p[t_2, t_3]$ satisfies tp_1 at its root. We are now ready for the algorithm.

PROOF OF THEOREM 18 (SINGLE-EXIT HAMCS). We fix a single-exit HAMC M. We know that in every component, there are only a polynomial number of paths from the entry node to the exit node. We also assume that every component has at most two boxes. An arbitrary AHMC can be normalized to achieve this. By acyclicity and the single-exit property each path hits each box at most once. Note that over a HMC, the axis descendant can be (efficiently) translated away, since the depth of the documents is bounded. Similarly, FNXPath expressions can be rewritten to explicitly skip over ε-nodes, using the bound on the depth of the tree. For example $\mathsf{child}[G]$ is rewritten to $\bigvee_{i \leqslant h} (\mathsf{child}[\varepsilon]/)^i / \mathsf{child}[G]$.

Our algorithm $Gen(tp, b)$ takes as input F-type tp and box b of M, computing the probability that b generates a tree satisfying tp at the root. Applying this to the root of the top-level box, and iterating over every type tp that contains F, will give us the final probability that we want.

$Gen(tp, b)$ works as follows. Let b point to component C that is associated with label $l \in \mathcal{L}$, and b_1, b_2 be the boxes in C. Let $p_1 \dots p_n$ be all paths from the entry node to the exit node in C, possibly going through both boxes. For each path p_i, let $l[p_i]$ be the two-pointed tree that has a root labeled L and the path p_i as a subforest, with ports corresponding to any occurrence of b_1 and b_2 in p_i.

The algorithm guesses types tp_1 and tp_2 and then recursively calculates $Gen(tp_1, b_1)$ and $Gen(tp_2, b_2)$. It then iterates over each path p_i, and checks whether the 4-tuple $(tp_1, tp_2, tp_3, l[p_i])$ is consistent; if they are multiples the probabilities returned by $Gen(tp_1, b_1)$ and $Gen(tp_2, b_2)$ with the product of probabilities of the edges in p_i, adding this to the running probability. It returns the sum total of these probabilities over all p_i.

The correctness of the algorithm follows directly from Lemma 27. The algorithm runs in PSPACE in the query, since we can implement it with a call stack that stores a sequence of pairs, box and type, of height at most the height of M. For any fixed query it will be ra-tractable, since the number of arithmetic operations used is a polynomial (depending on the query) in the AHMC. □

Let us now extend this to multiple-exit HAMCs.

PROOF OF THEOREM 18 (MULTIPLE-EXIT HAMCS). We explain the extension to the general AHMC model of the proof given in the main text. In this case, there may be j boxes per component, where j is bounded by the size of the AHMC. We thus need to deal with j-pointed trees; the extension of Lemma 27 to j-pointed trees is straightforward. In a given component, there may be exponentially many paths p. However, each path is of size linear in the chain and we only need to deal with one at a time (we can record the last path we have visited using a polynomial amount of space). So the algorithm $Gen(tp_1, b)$ will guess a path in the component pointed to by b, in addition to choosing j types, where tp_i is chosen for box b_i in the component, and will then make a recursive call to $Gen(tp_i, b_i)$ to compute the probability. Of course, we also need to verify type consistency, which requires a modification of Lemma 28 to the case where p is not of fixed sized but is part of the input. The extension follows because the FNXPath expression expressing the isomorphism type of the path is of size linear in the path. □

hal-00537778, version 1 - 19 Nov 2010