On the Complexity of Managing Probabilistic XML Data

Pierre Senellart, Serge Abiteboul

On the Complexity of Managing Probabilistic XML Data

Pierre Senellart*
INRIA Futurs & Université Paris-Sud
Orsay, France
pierre@senellart.com

Serge Abiteboul
INRIA Futurs & Université Paris-Sud
Orsay, France
serge.abiteboul@inria.fr

ABSTRACT

In [3], we introduced a framework for querying and updating probabilistic information over unordered labeled trees, the probabilistic tree model. The data model is based on trees where nodes are annotated with conjunctions of probabilistic event variables. We briefly described an implementation and scenarios of usage. We develop here a mathematical foundation for this model. In particular, we present complexity results. We identify a very large class of queries for which simple variations of querying and updating algorithms from [3] compute the correct answer. A main contribution is a full complexity analysis of queries and updates. We also exhibit a decision procedure for the equivalence of probabilistic trees and prove it is in co-RP. Furthermore, we study the issue of removing less probable possible worlds, and that of validating a probabilistic tree against a DTD. We show that these two problems are intractable in the most general case.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; H.2.1 [**Database Management**]: Logical Design

General Terms

Algorithms, Theory

Keywords

XML, probabilistic databases, semi-structured databases, complexity

1. INTRODUCTION

Many automatic tasks generate *imprecise* data, e.g., information extraction, natural language processing, data mining. Moreover, in many of these tasks, information is represented in a semi-structured way, either because of an inherent tree-like structure of the original information, or because it is natural to represent the derived knowledge in a hierarchical manner. We thus need the means to manage imprecise tree information gathered by the system during its entire life, and in particular evaluate queries and imprecise updates over such data. In [3], we introduced a probabilistic tree model for managing imprecise tree data[1]. A main issue is the tractability of such a model. In this paper, we discuss theoretical aspects of the probabilistic tree model, focusing on complexity issues.

The original motivation of the present work will best illustrate how such issues naturally arise and the nature of the problem. We are interested in discovering resources in the Web and more particularly in the Hidden Web. When we discover a new data source, we have to understand its semantics for future use. This leads to some analysis of the source (classification, extraction, semantic tagging, linguistic tools, etc.) that is by nature imprecise. We represent the information (knowledge) we extract in an XML warehouse. The various tools interact with the warehouse via updates and queries. The updates introduce imprecision. The interested reader can find in [2] a short description of the project. Updating and querying imprecise data is at its core, which motivated the present paper.

To model imprecise data, we use the probabilistic tree model introduced in [3]. The purpose was to design a model for storing imprecise information, which was both *expressive* and *concise*. Probabilistic trees are unordered trees whose nodes are annotated by conjunctions of (possibly negated) *event variables*, in the style of conditions in [12]. Each event variable is assigned a probability value. In particular, every probabilistic update introduces a new event variable (independent from the previous ones) that captures the belief the system has in this particular update. The description of an implementation and scenarios of usage of such a model can be found in [3], where we also show that it is as expressive as the extensive description of all possible worlds.

We identify a large class of queries for which simple variations of querying and updating algorithms from [3] compute the correct answer, by using evaluation algorithms developed for precise data. A main contribution of the paper is a precise complexity analysis of queries and updates for probabilistic trees. A large class of queries can be evaluated

*This work has been partially supported by the ANR projects WebContent and DocFlow.

PODS'07, June 11–14, 2007, Beijing, China.

[1] In [3], we referred to it as the *fuzzy tree* model; we changed the terminology here at the request of the reviewers, in order to avoid confusion with works on *fuzzy databases*.

in PTIME. For updates, deletion may be intractable. (Observe that in settings we are interested in, based on tools gathering knowledge, deletions are rare.) We also propose a theoretical foundation for the probabilistic tree model. In particular, we obtain results on the equivalence of probabilistic trees, which can be determined in polynomial time with a probabilistic algorithm. We also study the issue of removing less probable possible worlds, and that of validating a probabilistic tree against a DTD. We show that these two problems are intractable in the most general case.

Section 2 presents the probabilistic tree model, recalling definitions and results from [3]. It also introduces new material, in particular about the complexity of queries and updates. In Section 3, a notion of equivalence between two probabilistic trees is introduced, and its complexity is investigated. Other issues are investigated in Section 4. Variants of the probabilistic tree model are discussed in Section 5. Finally, the conclusion in Section 6 includes some related works. Appendix A contains technical details about updates that are not needed to follow the paper.

2. THE PROBABILISTIC TREE MODEL

In this section, we present the basics of the probabilistic tree model. Most of the definitions are from [3], with some minor changes of notation. Some new material is also included. In particular, Propositions 1 and 2 are new; Theorem 1 is an extension of Theorem 2 from [3] to a much more general class of queries.

We first introduce a tree data model and next, the probabilistic tree model.

DEFINITION 1. A *data tree* t is a 4-tuple $t = (A, E, r, \varphi)$ where A is a finite set of nodes, $E \subseteq A^2$ a tree rooted in $r \in A$ and φ associates a *label* from some countable set (say, the set of character strings) to each node in A.

Let $t = (A, E, r, \varphi)$ and $t' = (A', E', r', \varphi')$ be two data trees. We say that t and t' are *isomorphic* (denoted $t \sim t'$) if there is a bijection $\psi : A \to A'$ such that:

(i) For $s_1, s_2 \in A$, $(s1, s2) \in E \Leftrightarrow (\psi(s_1), \psi(s_2)) \in E'$;
(ii) $\psi(r) = r'$;
(iii) $\forall s \in A$, $\varphi'(\psi(s)) = \varphi(s)$.

The simple data model we use is inspired by XML but ignores a number of XML features such as the ordering, the distinction between attributes, labels and text. It should be observed that it adopts a multi-set semantics. To see that, consider for instance a data tree with a root node and two children with the same label. We see it essentially as different from a data tree with a root node and a single child with the same label. A model based on a pure set semantics is briefly considered in Section 5.

Syntax of Probabilistic Trees.

We next present the *probabilistic tree model* for representing probabilistic semi-structured information, that is based on annotating the nodes of a tree with probabilistic conditions in the style of the conditions in [12].

We assume the existence of a countable set $\mathcal{W}$ of *event variables*. Let W be a finite set of event variables. A *condition* over W is a (possibly empty) set of *atomic conditions* of the form w or $\neg w$ (for w in W). This set can also be seen as a conjunction of these atomic conditions. A *probability distribution* π for W assigns probabilities, i.e., values in $]0; 1]$, to the different event variables in W. We choose not to allow zero probabilities so that, in particular, updates with a zero probability will not be performed at all. But this is only a convention and could be changed without altering the results presented here. Formally, we have:

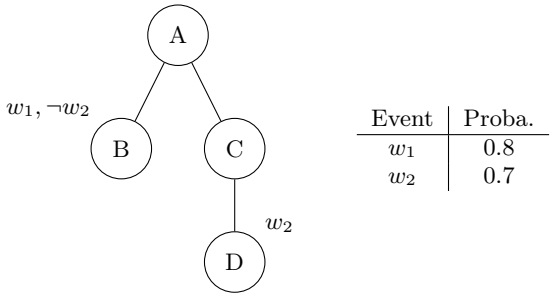


Figure 1: Example probabilistic tree

DEFINITION 2. A *probabilistic tree* (abbreviated as *prob-tree*) T is a 4-tuple (t, W, π, γ) where $t = (A, E, r, \varphi)$ is a data tree, $W \subseteq \mathcal{W}$ is finite, π is some probability distribution over W, and γ assigns conditions over W to nodes in $A - \{r\}$.

An example of prob-tree is given in Figure 1. We now define the *semantics* of a prob-tree, introducing to do that, the notion of *Possible World set*.

Semantics of Probabilistic Trees.

The real world with some uncertainty is modeled by a set of possible worlds, each with its own probability. More precisely, a *possible world (PW) set* S is a finite set of pairs (t_i, p_i) where (i) the t_i are data trees with the same root label, and (ii) each p_i is a positive real number with $\sum_{i=1}^{n} p_i = 1$. An example of a PW set is shown in Figure 2.

As different PW sets may represent the same abstract possible worlds, we need a notion of isomorphism between possible world sets. Let $S = \{(t_1, p_1) \dots (t_n, p_n)\}$ and $S' = \{(t'_1, p'_1) \dots (t'_m, p'_m)\}$ be two possible world sets. We say that S and S' are *isomorphic* (denoted $S \sim S'$) if, for all data tree t appearing either in S or in S':

$$\sum_{\substack{1 \leqslant i \leqslant n \\ t_i \sim t}} p_i = \sum_{\substack{1 \leqslant j \leqslant m \\ t'_j \sim t}} p'_j$$

This allows defining the notion of *normalization* of PW sets: A PW is normalized if it does not contain two possible worlds with isomorphic data trees. Every PW set can be *normalized* by assigning as the probability of each possible world the sum of the probabilities of possible worlds with isomorphic data trees.

We sometimes want to study *subsets* of PW sets. Observe that in such a subset the sum of the probabilities is not one. Such a subset arises naturally if we start from a PW set and introduce some additional integrity constraints that rule out some of the possible worlds. Another way to think about it is to cumulate the probabilities that were lost (those of the trees violating the constraint) and assign them to the tree consisting simply of a root. In other words, this comes down to interpreting the root-tree as *inconsistent*. With this in mind, we can see a subset of a possible world set as a possible world set as follows:

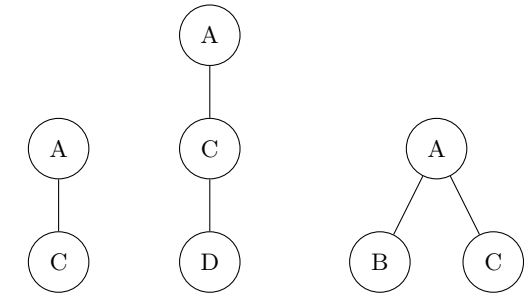


Figure 2: Example Possible World set

DEFINITION 3. Let $S = \{(t_1, p_1) \dots (t_n, p_n)\}$ be a strict subset of a PW set. Let $p = \sum_{1 \leqslant i \leqslant n} p_i < 1$ and t the data tree consisting of a single node with the same label as root nodes in t_i. By extension, we say that S is *isomorphic* to the possible world set $S \cup \{(t, 1-p)\}$, and we note $S \sim_{sub} S \cup \{(t, 1-p)\}$.

We are now ready to define the semantics of probabilistic trees in terms of possible worlds:

DEFINITION 4. Let $T = (t, W, \pi, \gamma)$ be a prob-tree. For $V \subseteq W$, the *value of T in the world V*, denoted $V(T)$, is the subtree of t where all nodes conditioned by a '$\neg w$' atom for $w \in V$ or by a 'w' atom for $w \notin V$ have been removed (as well as their descendants). The *possible world semantics* of T, denoted $[\![T]\!]$, is the PW set defined by:

$$[\![T]\!] = \bigcup_{V \subseteq W} \left\{ \left(V(T), \prod_{w \in V} \pi(w) \prod_{w \in W - V} (1 - \pi(w))\right)\right\}$$

In particular, the PW set shown in Figure 2 is (up to isomorphism) the semantics of the prob-tree of Figure 1. A result from [3] is that the probabilistic tree model has the same expressive power as the possible worlds model. More precisely, for each PW set S, there exists a prob-tree T such that $S \sim [\![T]\!]$. The (quite straightforward) construction of this prob-tree uses as many event variables as there are possible worlds in S. Thus, the size of the resulting prob-tree is essentially the size of the original PW set. One could clearly hope to find more compact representations.

In order to guarantee conciseness of the probabilistic tree model, we may want to have a polynomial bound on the size of prob-trees whose semantics only involve data trees of bounded size (and with probabilities of bounded precision). A model with such a polynomial bound, the so-called *simple probabilistic* model, is presented in [3], but it is shown to be less expressive than the PW model. Actually, the following new result shows that neither the probabilistic tree model, nor any other model as expressive as the PW model, can guarantee such a bound:

PROPOSITION 1. *Let M be a one-to-one mapping sending every normalized possible world set (with probabilities of bounded precision) to some integer (say, a binary representation of an element of a model). Then, the average size of $M(S)$ (that is, $\log M(S)$) for PW sets S in which every possible world has at most n nodes is at least exponential in n.*

PROOF. This results from a simple counting of the number of possible world sets involving only possible worlds with at most n nodes. Let us call this number σ_n. If we forget about the values of the probabilities and the labels of the nodes, we get that σ_n must be greater than the number of sets of unordered, unlabeled, rooted trees with at most n nodes. We have the following equality about the number a_n of unordered, unlabeled, rooted trees with exactly n nodes [15, 13]:

$$a_n = \frac{\alpha^{n-1}}{n} \sqrt{\beta / 2\pi n} + O(n^{-5/2} \alpha^n)$$

where $\alpha > 2$ and β are two constants. We have therefore:

$$\sigma_n \geqslant 2^{\sum_{i=1}^{n} a_n} \geqslant 2^{a_n} \geqslant \Omega(2^{2^n})$$

Since σ_n is doubly exponential in n, an element of $M(S)$ cannot be identified on average with less than $\Omega(2^n)$ bits. □

Queries and Updates.

We now look at the way to perform both queries and updates on prob-trees. The first step is to define more precisely the type of queries we consider. The goal is to be able to evaluate efficiently query answers. Indeed, in practice, one would ideally like to rely on a standard query processor to do most of the work. In [3], some efficient query processing was exhibited for the class of tree-pattern queries with joins. After extending these results in a number of directions, we realized that a similar approach can be followed for the following very large class of queries, namely the *locally monotone* queries. To define it, we need the auxiliary notion of sub-datatree. Note that we only consider subtrees which have the same root as the original trees, obtained by pruning some of its branches.

DEFINITION 5. Let $t = (A, E, r, \varphi)$, $t' = (A', E', r', \varphi')$ be two data trees. t' is a *sub-datatree* of t (denoted $t' \leqslant t$) if: (i) $A' \subseteq A$; (ii) if $n_1 \in A'$ and $(n_2, n_1) \in E$, $n_2 \in A'$; (iii) $E' = E \cap A'^2$; (iv) $r' = r$; (v) $\varphi' = \varphi_{|A'}$. The set of all sub-datatrees of a data tree t is denoted $Sub(t)$.

The sub-datatree relation is clearly a partial order, which justifies the notation $t' \leqslant t$.

The queries we consider return subtrees of the data tree. This greatly simplifies the management of probabilities: Intuitively, we return pieces of the original tree, but always keep the path from these pieces to the root. This notion is defined formally next, together with the large class of queries for which we will be able to generalize the query evaluation algorithm of [3].

DEFINITION 6. A *query* Q is a function over the set of data trees, such that for each data tree t, $Q(t)$ is a (possibly empty) set of sub-datatrees of t.

A query Q is *locally monotone* if either of the following two equivalent conditions holds:

(i) for any three data trees $u \leqslant t' \leqslant t$, $u \in Q(t) \iff u \in Q(t')$
(ii) for any two data trees $t' \leqslant t$, $Q(t') = Q(t) \cap Sub(t')$

Locally monotone queries are precisely the queries for which, *given an algorithm to compute the query on data trees*, we can compute easily the answers on a prob-tree.

A large class of queries are locally monotone, including tree-pattern queries with joins (which was the framework of [3]), but excluding negative queries. We now present the way queries are defined on PW sets and prob-trees, and state that these two definitions are consistent.

DEFINITION 7. Let Q be a query and $S = \{(t_i, p_i)\}$ a PW set. The *result* of Q for S, denoted $Q(S)$, is

$$\bigcup_{(t_i,p_i)\in S} \bigcup_{t\in Q(t_i)} \{(t, p_i)\}$$

Observe that the answer to a query is not strictly speaking a possible world set, since the probabilities do not sum to 1. To obtain one, one might group all the answers for the same t_i under a common root node.

DEFINITION 8. Let Q be a locally monotone query. The result of Q on a prob-tree $T = (t, W, \pi, \gamma)$, denoted $Q(T)$, is

$$\bigcup_{u\in Q(t)} \{(u, eval(\bigcup_{n \text{ node of } u} \gamma(n)))\}$$

where $eval(cond)$ returns 0 if there is an event w such that both 'w' and '$\neg w$' are in $cond$, and is otherwise defined as:

$$\prod_{w\in cond} \pi(w) \cdot \prod_{\neg w\in cond} (1 - \pi(w))$$

The following result states the consistence between the way queries are performed on prob-trees and the possible world semantics.

THEOREM 1. *Let T be a prob-tree and Q be a locally monotone query. Then, with a little abuse of notation since the probabilities in $Q(T)$ and $Q(\llbracket T \rrbracket)$ do not sum to 1, we have $Q(T) \sim Q(\llbracket T \rrbracket)$.*

PROOF. Let $Q(T) = \{(u_1, p_1) \dots (u_n, p_n)\}$ and $Q(T') = \{(u'_1, p'_1) \dots (u'_m, p'_m)\}$. For a data tree u, the proof is done in two steps:

1. If u appears in $Q(T)$, u appears in $Q(\llbracket T \rrbracket)$ and

$$\sum_{\substack{1\leqslant i\leqslant n \\ u_i\sim u}} p_i \leqslant \sum_{\substack{1\leqslant j\leqslant m \\ u'_j\sim u}} p'_j$$

2. If u appears in $Q(\llbracket T \rrbracket)$, u appears in $Q(T)$ and

$$\sum_{\substack{1\leqslant i\leqslant n \\ u_i\sim u}} p_i \geqslant \sum_{\substack{1\leqslant j\leqslant m \\ u'_j\sim u}} p'_j$$

Both steps are technical but not complicated. □

We have similar results for updates. For simplicity, we only consider here elementary operations (insertions and deletions); see [3] for the extension to simultaneous arbitrary sets of insertions and deletions. Typically, one wants to perform an update operation based on the result of a query. An insertion will add to a data tree some subtree at positions specified by a query. A deletion will delete from a tree all nodes at positions specified by a query. It is easy enough to extend this notion of insertions and deletions to PW sets: The *probabilistic updates*, built from an update operation and a probability value, which is the confidence in the update operation, are simply performed on each possible world. We showed in [3] how to perform updates on prob-trees in a consistent way with the possible world semantics. To facilitate the reading of the paper, technical details about updates are relegated to Appendix A.

In what follows, we assume given a locally monotone query language $\mathcal{L}_q$ and an algorithm to answer queries over trees that are "lifted" to queries/updates over prob-trees. We next analyze the complexity of the algorithms for querying and updating prob-trees. Observe that in the following proposition, the complexity of the operations on prob-trees is stated in terms of the complexity of the corresponding operation on data trees. So, for instance, since the data complexity of tree-pattern queries with join is PTIME, an immediate consequence of the proposition is that over prob-trees, it is also PTIME. More precisely, let $|\cdot|$ denote the size (number of nodes, of literals) of a prob-tree (or of a set of possible worlds), and **time** denotes the time it takes to evaluate the query or operation. Then the complexity of the algorithms presented for querying and updating, that is, an upper bound on the complexity of the associated problems (based on an algorithm to answer $\mathcal{L}_q$ queries), is as follows:

PROPOSITION 2. *Let T be a probabilistic tree with underlying data tree t. Let Q be a query over T, and i_Q and d_Q be respectively an insertion and deletion on T, with Q as defining query.*

$$\begin{array}{lcl}
\mathbf{time}(Q(T)) & \leqslant & \mathbf{time}(Q(t)) + O(|Q(t)| \cdot |T|) \\
\mathbf{time}(i_Q(T)) & \leqslant & \mathbf{time}(Q(T)) + O(|Q(t)| \cdot |T|) \\
|i_Q(T)| & \leqslant & |T| + O(|Q(t)| \cdot |T|) \\
\mathbf{time}(d_Q(T)) & \leqslant & \mathbf{time}(Q(T)) + O(|Q(t) \cdot 2^{|T|}) \\
|d_Q(T)| & \leqslant & |T| + O(|Q(t)| \cdot 2^{|T|})
\end{array}$$

PROOF. These follow from the analysis of the evaluation algorithms and from the definitions. The combinatorial explosion of deletions happens when a query has multiple results (essentially because, in this case, we need to express the negation of a disjunction of conjunctions in terms of a disjunction of conjunctions), and as we shall see in Theorem 3, this complexity is inherent to the problem of deletion in prob-trees. □

3. EQUIVALENCE OF PROB-TREES

One defines the notion of equivalence between prob-trees directly based on data tree isomorphism. It essentially states that two prob-trees talk about the same event variables and that for each assignment of values to the event variables, they define the same possible world.

DEFINITION 9. Let $T = (t, W, \pi, \gamma)$, $T' = (t', W, \pi, \gamma')$ be two prob-trees (over the same event variables and distribution). Then T and T' are *structurally equivalent* (denoted $T \equiv_{struct} T'$) if for each $V \subseteq W, V(T) \sim V(T')$.

Note that an alternative definition of equivalence of prob-trees, based on their possible world semantics, is discussed in Section 5. We have a simple complexity upper bound about structural equivalence:

PROPOSITION 3. *Determining if two prob-trees (over the same event variables and distribution) are structurally equivalent is co-*NP.

PROOF. The complement of this problem can be solved with the following NP algorithm:

INPUT: two prob-trees T_1 and T_2 on the same event variable set W and with the same probability distribution π
OUTPUT: `true` if $T_1 \not\equiv_{struct} T_2$
(a) Guess a subset V of W.
(b) Compute $V(T_1)$ and $V(T_2)$ in linear time.
(c) If $V(T_1) \not\sim V(T_2)$, return `true`. (Isomorphism of labeled unordered data trees can be determined in linear time, cf [4] and the algorithm in the proof of Theorem 2.)

□

We will also show a more precise result in Theorem 2, that this problem is co-RP [16]. To do it, we will use some bridge to (i) the number of disjuncts satisfied by valuations of DNF formulas and (ii) multivariate polynomials.

DEFINITION 10. Let ψ and ψ' be two propositional formulas in disjunctive normal form. We say that ψ and ψ' are *count-equivalent*, denoted $\psi \stackrel{+}{\equiv} \psi'$, if, for any valuation ν of the variables appearing in ψ and ψ', the same number of disjuncts is satisfied by ν in ψ and in ψ'.

We note that this is a stronger notion than simple propositional formula equivalence. For instance, the formulas $A \vee (A \wedge B)$ and A are equivalent but not count-equivalent. We indicate next how we can relate count-equivalence of formulas in DNF with equality of multivariate polynomials.

DEFINITION 11. Let ψ be a propositional formula in disjunctive normal form, over variables $X_1 \dots X_n$. Let ψ' be a formula in DNF obtained from ψ by removing every disjunct containing incompatible atomic conditions or **False**, and by removing duplicate atomic conditions from each disjunct and **True** from conjunctions with other literals. The *characteristic polynomial of* ψ, denoted P_ψ, is the multivariate polynomial in $X_1 \dots X_n$ with integer coefficients, obtained from ψ' in the following manner: (i) Positive literals X_i are left as is. (ii) Negative literals $\neg X_i$ are replaced by $(1 - X_i)$. (iii) Disjunction is replaced by addition. (iv) Conjunction is replaced by multiplication. (v) **True** is replaced by 1 and **False** by 0.

LEMMA 1. *Let ψ and ψ' be two propositional formulas in disjunctive normal form. Then, $\psi \stackrel{+}{\equiv} \psi'$ if and only if $P_\psi = P_{\psi'}$.*

PROOF. One direction is obvious, i.e., if $P_\psi = P_{\psi'}$ then $\psi \stackrel{+}{\equiv} \psi'$. For that, just observe that the number of conjuncts satisfied by some valuation ν in ψ is the value of P_ψ for this valuation. Now to consider the converse, first observe that P_ψ and $P_{\psi'}$ are polynomials with degree at most 1 in every variable (this comes from the normalization of the formula in DNF used in Definition 11). Suppose that $\psi \stackrel{+}{\equiv} \psi'$. Consider the development of P_ψ:

$$P_\psi(X_1 \dots X_n) = \sum_{V \subseteq [\![1;n]\!]} \alpha_V \prod_{i \in V} X_i$$

and similarly for $P_{\psi'}$ with coefficients α'_V.

We have, for each tuple $(x_1 \dots x_n)$ of $\{0,1\}^n$,

$$P_\psi(x_1 \dots x_n) = P_{\psi'}(x_1 \dots x_n),$$

that is to say,

$$\sum_{U \subseteq \{i | x_i = 1\}} \alpha_U = \sum_{U \subseteq \{i | x_i = 1\}} \alpha'_U.$$

We can then prove by induction on the cardinality of V that this implies that $\forall V \subseteq [\![1;n]\!], \alpha_V = \alpha'_V$, which means that $P_\psi = P_{\psi'}$. □

One can "clean" a probabilistic tree by removing (in linear time) superfluous atomic conditions, i.e., conditions implied by some condition on an ancestor; and pruning nodes with inconsistent conditions, i.e., conditions that are intrinsically inconsistent or that contradict condition imposed by an ancestor. We call such trees *clean* prob-trees. The following result gives an inductive definition of structural equivalence on clean prob-trees; the proof is straightforward.

LEMMA 2. *Let $T = (t, W, \pi, \gamma)$ and $T' = (t', W, \pi, \gamma')$ be two clean prob-trees (over the same event variables and probability distribution).*

Let $u_1 \dots u_n$ be representative elements of the n equivalence classes implied by structural equivalence over the subtrees of T and T' rooted at each child node of the root of T and T', the condition on the root of which has been removed. For $1 \leqslant i \leqslant n$, let ψ_i be the disjunction of the conditions attached to the children of the root of T whose subtree is structurally equivalent to u_i, and let ψ'_i be the same for T'.

Then, $T \equiv_{struct} T'$ if and only if $\varphi(r) = \varphi(r')$ and, for each $1 \leqslant i \leqslant n$, $\psi_i \stackrel{+}{\equiv} \psi'_i$.

This leads to our main result on structural equivalence:

THEOREM 2. *There is a PTIME algorithm, that, given two prob-trees, always returns `true` if the prob-trees are structurally equivalent and returns `false` if the prob-trees are not structurally equivalent with probability at least $1/2$ (that is, determining if two prob-trees are equivalent is a co-RP problem [16]).*

PROOF. The algorithm relies on Lemmas 1 and 2, and uses an algorithm derived from a classical algorithm for labeled tree isomorphism from [4]. Moreover, we use the Schwartz-Zippel Lemma [17, 20], which states that the probability that a multivariate polynomial of degree d is zero on a point each coordinate of which is randomly chosen in some finite set S is $d/|S|$.

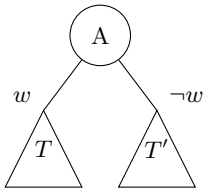
The algorithm is presented in Figure 3. We have the following lower bound for the probability that this algorithm is correct when it returns `false`: $\left(1 - \left(\frac{N_l}{|S|}\right)^m\right)^{N_n^3}$, where N_l is the number of literals of T and T', and N_n the number of nodes. This probability is greater than $1/2$ as soon as m and S are chosen such that $|S| \geqslant \frac{N_l}{\sqrt[m]{1-(1/2)^{1/N_n^3}}}$. □

Observe that determining whether a prob-tree is independent of some event variable is actually computationally as complex as deciding equivalence between prob-trees. Indeed, if T and T' are two prob-trees, determining if T is structurally equivalent to T' can be done by determining if the following tree is independent of w (a fresh variable):

INPUT: two prob-trees T and T', a finite set S of integers, a positive integer m
OUTPUT: `true` if $T \equiv_{struct} T'$; `false` if $T \not\equiv_{struct} T'$ with probability $1/2$

(a) Clean T and T'.
(b) Assign to all leaves of T and T' with the same label a fresh integer i (and do not assign the same integer to two leaves with different labels).
(c) Assume inductively that all nodes of T and T' at a distance at most k from the leaves have been assigned an integer. For each pair (n, n') where n (resp. n') is a node of T (resp. T') at distance $k+1$ from the leaves, and n and n' have the same label:
  (i) Compute the sets A and A' of the integers assigned to the children of n and n'.
  (ii) If $A = A'$, for each element i of A, compute the formulas in DNF ψ_i and ψ'_i corresponding to the conditions on the children of n and n' assigned with i. Choose at random m points of S^p, where p is the number of variables of $P_{\psi_i} - P_{\psi'_i}$, and evaluate this polynomial in these points. If all these evaluations return 0 for each i of A, assign the same integer value to n and n' (if one of them is already assigned an integer, take this integer as the assigned value for the other one, possibly doing some merging of values; otherwise, take a fresh integer).
(d) Assign fresh integers to nodes at distance $k+1$ from the leaves with no assigned integer, and go to the previous step with the next value of k.
(e) If the roots have been been assigned the same integer, then return `true` else return `false`.

Figure 3: Probabilistic algorithm for structural equivalence



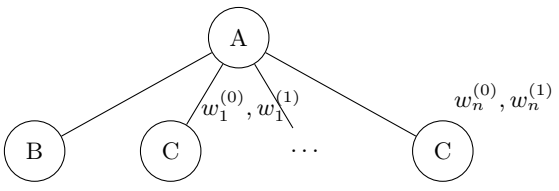
4. OTHER ISSUES ABOUT PROB-TREES

In this section, we consider three natural problems about probabilistic trees that all highlight some inherent complexity in dealing with imprecise data. First we show that some deletion may cause a combinatorial explosion. We then prove that similar phenomena arise when we try to restrict the possible worlds (i) to have at least a threshold probability and (ii) to be valid with respect to some DTD. The proofs are very similar.

Deletions. First we consider deletion. In this part, we will assume that our query language is expressive enough to express the following deletion: d_0 = "If the root has a C-child, then delete all B-children of the root." (this is not a strong assumption, since it is for instance the case with simple tree pattern queries).

THEOREM 3. *For all $n \in \mathbb{N}$, there exists a prob-tree T, of size $O(n)$, such that for each prob-tree T' such that $T' \equiv_{struct} d_0(T)$, the size of T' is $\Omega(2^n)$.*

PROOF. Consider the following prob-tree T, which has $n+2$ nodes and $2n$ event variables, each appearing only once (we take an arbitrary probability distribution π, say $\pi(n) = 1/2$ for all n):



Let T' be a prob-tree such that $T' \equiv_{struct} d_0(T)$. We assume that the deletion has a confidence of 1 (that is, it does not introduce a new event variable).

T' is necessarily some prob-tree of height 1 with root node A, and with a number of B and C children. Let Ψ be the set of conditions annotating nodes labeled by B. Observe that for all $\psi \in \Psi$, and for all $1 \leqslant k \leqslant n$, either $\neg w_k^{(0)}$ or $\neg w_k^{(1)}$ appears in ψ (otherwise, there is a possible world for T' where both B and C nodes appear, which is a contradiction with the definition of d_0).

Let now $\{b_1 \dots b_n\}$ be an arbitrary element of $\{0,1\}^n$. Let ν be the valuation of the event variables such that $\forall 1 \leqslant k \leqslant n$, $\nu(w_k^{(b_k)}) = 0$ and $\nu(w_k^{(1-b_k)}) = 1$. $\nu(T)$ is the subtree of T with only two nodes labeled by A and B. Therefore, $\nu(d_0(T)) = d_0(\nu(T)) = \nu(T)$. This means that there exists $\psi_{b_1 \dots b_n} \in \Psi$ such that $\nu \models \psi_{b_1 \dots b_n}$.

Assume now by contradiction there exist $b_1 \dots b_n$, $b'_1 \dots b'_n$ and $1 \leqslant k \leqslant n$, such that $\psi_{b_1 \dots b_n} = \psi_{b'_1 \dots b'_n} = \psi$ and $b_k \neq b'_k$. But we have already noted that ψ contains either $\neg w_k^{(0)}$ or $\neg w_k^{(1)}$. In the former case, we cannot have either $b_k = 0$ or $b'_k = 0$; in the latter, we cannot have $b_k = 1$ or $b'_k = 1$. This leads to a contradiction, which means that to each element of $\{0,1\}^n$ corresponds a different element of Ψ. T' has then more than 2^n different literals, which concludes the proof. □

Threshold Probability. We consider next what happens when some probability threshold is imposed on a probabilistic tree.

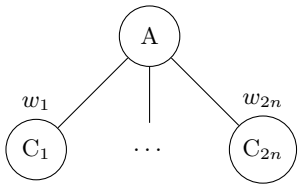
Given a prob-tree, one may want to eliminate the possible worlds that are too improbable. More precisely, consider a prob-tree T with $[\![T]\!] = \{(t_1, p_1) \dots (t_n, p_n)\}$. Let us further assume that it is normalized, i.e., that there are no i, j distinct with $t_i \sim t_j$. Suppose we fix some p for a minimum threshold on probability. Then we define

$$[\![T]\!]_{\geqslant p} = \{(t_i, p_i) \in [\![T]\!] | p_i \geqslant p\}$$

Unfortunately, there are cases where there is no compact prob-tree to represent $[\![T]\!]_{\geqslant p}$:

THEOREM 4. *For all $n \in \mathbb{N}$, there exist a prob-tree T of size $O(n)$ and a probability threshold p such that for each prob-tree T' such that $[\![T]\!]_{\geqslant p} \sim_{sub} [\![T']\!]$, the size of T' is $\Omega(2^n)$.*

PROOF. We use the following prob-tree, with $2n+1$ nodes and $2n$ event variables, each appearing once; we take a uniform probability distribution $\pi(w_i) = 1/2n$ and a probability threshold $p = 1/2$:



The proof is quite similar to that of Theorem 3, and uses the fact that $\binom{2n}{n} = \Omega(2^n)$. □

Validation. Finally we consider validity with respect to a DTD.

A Document Type Definition for an XML document defines the constraints applying on the children of a node using a *sequence* operator (`(A,B)`), a *disjunction* operator (`(A|B)`) and several *repetition* operators (`(A*)`, `(A+)`, `(A?)`). As we consider only unordered trees, we do not consider the sequence operators. To simplify, we will not consider the disjunction operator either. The following definition of DTDs then simply states that a DTD gives a lower and upper bound for the number of occurrences of nodes with a given label n' as children of some node labeled by n.

DEFINITION 12. A *Document Type Definition* (*DTD*) D is a function over some finite subset N' of the set of labels N such that for all $n \in N'$, $D(n)$ is a finite set of elements of $N \times [\![0; +\infty[\![\times [\![1; +\infty]\!]$ and, if $(n_1, p_1, q_1) \in D(n)$ and $(n_2, p_2, q_2) \in D(n)$, either $n_1 \neq n_2$ or $(n_1, p_1, q_1) = (n_2, p_2, q_2)$.

We use the following notation, for $n \in N'$: $D_-(n)(n')$ and $D_+(n)(n')$ are respectively the unique p and q such that $(n', p, q) \in D(n)$ if such p and q exist; otherwise, we note $D_-(n)(n') = 0$ and $D_+(n)(n') = 0$.

DEFINITION 13. Let D be a DTD and $t = (A, E, r, \varphi)$ a data tree. Let N' be the domain of D. We say that t *satisfies* D (denoted $t \models D$) if, for each $s \in A$ such that $\varphi(s) \in N'$, and for each $n' \in N$:

$$\begin{aligned} D_-(\varphi(s))(n') &\leqslant \left|\{s' \in A \,|\, \varphi(s') = n' \wedge (s, s') \in E\}\right| \\ D_+(\varphi(s))(n') &\geqslant \left|\{s' \in A \,|\, \varphi(s') = n' \wedge (s, s') \in E\}\right| \end{aligned}$$

Note that we do not impose any condition on nodes of t whose label is not in the domain of the DTD. Given a prob-tree T and a DTD D, three natural questions naturally arise:

1. (*DTD Satisfiability*) $\{(t, p) \in [\![T]\!] \,|\, t \models D\} \stackrel{?}{\neq} \varnothing$
2. (*DTD Validity*) $\{(t, p) \in [\![T]\!] \,|\, t \models D\} \stackrel{?}{\sim} [\![T]\!]$
3. (*DTD Restriction*) How to compute a prob-tree T' such that $\{(t, p) \in [\![T]\!] \,|\, t \models D\} \sim_{sub} [\![T']\!]$.

We have the following complexity results about these questions:

THEOREM 5.

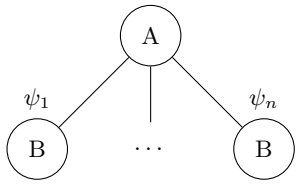
1. *The DTD Satisfiability problem is* NP*-complete in the number of event variables (and linear in the number of nodes in the tree).*
2. *The DTD Validity problem is co-*NP*-complete in the number of event variables (and linear in the number of nodes in the tree).*
3. *There are instances of the DTD Restriction problem in which the solution of the DTD Restriction problem is necessarily exponential in the size of the input.*

PROOF. The third part is proved in the same way as Theorem 4, with a DTD requiring that the node A has at most n children labeled by C. The C_i nodes are replaced by C nodes with a D_i child in order to give them the same label while keeping them distinguishable.

For the first two parts, we use a reduction of `SAT`. The beginning of the construction is the same in both cases.

Let θ be a propositional logic formula, in conjunctive normal form (i.e., an input to the `SAT` problem). Let $\psi_1 \dots \psi_n$ be the terms of $\neg\theta$ in disjunctive normal form (the DNF of $\neg\theta$ is computed in a linear time from θ which is in CNF).

Let T be the following prob-tree:



1. Consider the DTD D: $D(\mathrm{A}) = \{(\mathrm{B}, 0, 0)\}$.

$$\begin{aligned} & \{(t, p) \in [\![T]\!] \,|\, t \models D\} \neq \varnothing \\ \iff\ & \psi_1 \vee \dots \vee \psi_n \text{ not a tautology} \\ \iff\ & \theta \text{ is satisfiable} \end{aligned}$$

   Since the construction of the reduction is linear in the size of θ, this proves that the DTD satisfiability problem is NP-hard.

   Moreover, here is a NP algorithm for the DTD satisfiability problem, which concludes the proof of its NP-completeness: Guess a valuation ν of the event variables of T, and return `true` if $\nu(T)$ satisfies the DTD (which can be checked in linear time).

2. Consider the DTD D: $D(\mathrm{A}) = \{(\mathrm{B}, 1, +\infty)\}$.

$$\begin{aligned} & \{(t, p) \in [\![T]\!] \,|\, t \models D\} \sim [\![T]\!] \\ \iff\ & \psi_1 \vee \dots \vee \psi_n \text{ tautology} \\ \iff\ & \theta \text{ is not satisfiable} \end{aligned}$$

   Since the construction of the reduction is linear in the size of θ, this proves that the validity problem is co-NP-hard.

   Moreover, here is a NP algorithm for the complement of the validity problem, which concludes the proof of its co-NP-completeness: Guess a valuation ν of the event variables of T and return `true` if $\nu(T)$ does not satisfy the DTD.

Observe that the DTDs we used in the proof are all of constant size. □

5. VARIANTS

In this section, we briefly consider variants of the probabilistic tree model presented up to here, and discuss their complexity. Namely, we consider (i) a tree model with set semantics, instead of our *multi-set* semantics; (ii) the notion of *semantic* equivalence (in place of structural equivalence); (iii) a probabilistic tree model where nodes are assigned arbitrary propositional formula (and not simply conjunctions) as conditions; and (iv) ordered trees.

Set Semantics. In this paper, we use a data model with a *multi-set* (or *bag*) semantics. One can consider instead a *set semantics*. One just has to redefine isomorphism between data trees inductively as follows. Let t, t' be two trees. They are isomorphic if their roots have the same label and if each subtree of the root of t is isomorphic to some subtree of the root of t', and symmetrically. Most definitions of this paper can then be applied as is, relying on this new version of data tree isomorphism. The results about queries and updates remain, including the exponential complexity of deletions from Theorem 3 (the proofs are almost unchanged). An important difference, however, is for structural equivalence, for which there is now a simple way of proving co-NP-completeness: Just observe that we no longer deal with *count-equivalence*, but with classical equivalence of propositional formulas.

Semantic Equivalence. Structural equivalence is only relevant for prob-trees that share the same event variables. If we want to compare prob-trees with different sets of events, we can define another kind of equivalence, through their possible world semantics: T and T' are *semantically equivalent* (denoted $T \equiv_{sem} T'$) if $[\![T]\!] \sim [\![T']\!]$. The first natural question is that of the relation between structural and semantic equivalence.

PROPOSITION 4. *Let $T = (t, W, \pi, \gamma)$, $T' = (t', W', \pi', \gamma')$ be two prob-trees, with $W = W'$ and $\pi = \pi'$. Then*

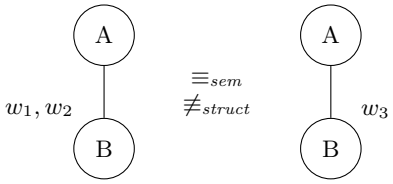
(i) If $T \equiv_{struct} T'$, then $T \equiv_{sem} T'$;

(ii) $T \equiv_{struct} T'$ if and only if, for each probability distribution π'' over W, $(t, W, \pi'', \gamma) \equiv_{sem} (t', W, \pi'', \gamma')$.

PROOF. (i) is obvious. Now suppose that for each π'' over W, $(t, W, \pi'', \gamma) \equiv_{sem} (t', W, \pi'', \gamma')$. To conclude the proof, it clearly suffices to show that $T \equiv_{struct} T'$. For each $V \subseteq W$, let π_V be the probability distribution that maps $w \in V$ to 1 and $w \in W - V$ to 0. Then, if T_V and T'_V denote respectively the prob-trees obtained from T and T' by exchanging the original probability distribution π with π_V, $[\![T_V]\!] = \{(V(T), 1)\}$ and $[\![T'_V]\!] = \{(V(T'), 1)\}$ and we have thus $V(T) = V(T')$.

Note that stricty speaking, we disallowed variables with 0 probability. So to be precise, we should use ε instead of 0 and $1 - \varepsilon$ instead of 1. If ε is chosen so that $(1 - \varepsilon)^n > 2^n \varepsilon$ (which is always possible for a sufficiently low value of ε), $V(T)$ and $V(T')$ will be the elements of highest probability of, respectively, $[\![T]\!]$ and $[\![T']\!]$. □

Note that $T \equiv_{sem} T'$ does not imply $T \equiv_{struct} T'$. For instance, if w_1, w_2, w_3 verify $\pi(w_3) = \pi(w_1) \cdot \pi(w_2)$, we have :



Clearly, there is an EXPTIME algorithm for determining if two prob-trees are semantically equivalent (just compute the possible world sets, normalize them, and check if they are isomorphic, which can be decided in quadratic time in the number of possible worlds). It is open whether the problem also belongs to a lower complexity class. Similarly, it is open whether Theorem 3 on the complexity of deletions still holds for semantic equivalence.

Arbitrary Propositional Formula. In prob-trees, the conditions we use are conjunctions of literals. A natural extension is to allow any propositional formula (including disjunctions) as conditions. A question is how this is affecting the complexity. First, one can show that the evaluation of boolean queries is NP-complete (assuming the underlying query language over data tree is in PTIME and includes, say, tree pattern queries). The fact that it is NP is obvious, and there is a linear-time reduction of `SAT` to this problem. Then, the cost of an update operation is now PTIME (again assuming the underlying language on data trees is PTIME). Indeed, we can now simply annotate inserted or deleted nodes by complex formulas. In particular, Theorem 3 is no longer valid. So this model privileges updates (that are cheap) against queries (that are expensive). It is not adapted to the applications that motivated our work.

Order Semantics. By considering ordered trees, we would move closer to standard XML. The situation is more intricate and would require totally different techniques. The complexity is higher because of the inherent combinatorics that is introduced.

6. CONCLUSION

The topic of probabilistic databases has been intensively studied, see for instance [8, 6, 5, 10], and [7, 19] for more recent works. The idea of associating probabilistic formulas to data elements comes from the *conditional tables* of [12]. A work close in spirit to this one, but in the context of relational databases, is [1]; the tree structure and multi-set semantics we use have for consequence that the complexity results on tables of [1] do not apply to our model.

A relatively small number of works have dealt with the representation of probabilistic semi-structured data. In [9], a semi-structured database is used to store complex probability distributions of data which is essentially relational. Works closer to ours are [14, 11, 18]. Nierman et al. [14] describe a very simple model, which does not have full expressive power, and present strategies for efficient evaluations of logical queries. In [11], a complex model, based on directed acyclic graphs, is developed, along with an algebraic query language. Finally, Keulen et al. [18] present an approach to data integration using probabilistic trees; their model is derived from the PW model, and allows both extensive descriptions of the possible worlds and node-based factorization. Querying and the way to present data inte-

gration results on this model are also shown. None of these works touch upon the question of updates.

We have presented a theoretical foundation for the probabilistic tree model, a model for representing probabilistic semi-structured data. We have provided a complexity analysis of updates and queries over probabilistic trees, as well as a probabilistic decision procedure for prob-tree equivalence. We have shown that other operations on prob-trees are intractable, highlighting the inherent complexity of the model. Finally, we have discussed how variations in the model affect the complexity of the various problems.

The present work may be pursued in a number of directions. A first one is prob-tree simplification. One would often like to approximate a prob-tree to get a more compact representation, perhaps ignoring less probable worlds and some of the probabilistic events (some of the provenance/history). Also, probabilities can be used to rank results. It would be useful to have algorithms obtaining the most probable results first. Finally, it would be interesting to also handle aggregate functions. We believe the use of multi-sets simplifies this last issue.

7. ACKNOWLEDGMENTS

We want to thank Luc Segoufin for his comments on this paper.

8. REFERENCES

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):158–187, 1991.

[2] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. Technical Report 435, GEMO, Inria Futurs, Orsay, France, Dec. 2005.

[3] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *Extending DataBase Technology*, Munich, Germany, Mar. 2006.

[4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, USA, 1974.

[5] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, 1992.

[6] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Very Large Data Bases*, 1987.

[7] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Very Large Data Bases*, Hong Kong, China, Sept. 2004.

[8] M. de Rougemont. The reliability of queries. In *Principles Of Database Systems*, San Jose, United States, 1995.

[9] A. Dekhtyar, J. Goldsmith, and S. R. Hawkes. Semistructured probabilistic databases. In *Statistical and Scientific Database Management*, Tokyo, Japan, July 2001.

[10] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1), 1997.

[11] E. Hung, L. Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *International Conference on Data Engineering*, Bangalore, India, Mar. 2003.

[12] T. Imieliński and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[13] D. E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, Boston, USA, third edition, 1997.

[14] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Very Large Data Bases*, Hong Kong, China, Aug. 2002.

[15] R. Otter. The number of trees. *Annals of Mathematics*, 49(3):583–599, July 1948.

[16] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Pub. Co., Reading, USA, 1994.

[17] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[18] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *International Conference on Data Engineering*, Apr. 2005.

[19] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Biennal Conference on Innovative Data Systems Research*, Pacific Grove, USA, Jan. 2005.

[20] R. Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Computation*, Marseille, France, 1979.

APPENDIX

A. UPDATES IN PROBABILISTIC TREES

In this section, we will present technical definitions on the kind of updates dealt with, and the way they are performed in probabilistic trees.

We will assume that a query Q defines, for each data tree t, and for each $t' \in Q(t)$, a mapping μ_Q from some finite set N_Q to the nodes of t'. If we consider the language of tree pattern queries, for instance, N_Q will be the set of nodes of the query tree, and μ_Q will map a node of the query tree to the corresponding node in the result tree.

DEFINITION 14. An (elementary) *update operation* is a pair $\tau = (Q, v)$ where Q is a locally monotone query and v is either:

1. an *insertion* on N_Q, that is, an expression $i(n, t')$ where $n \in N_Q$ and t' is a tree to insert (as a child of the node mapped by n);
2. or a *deletion* on N_Q, that is, an expression $d(n)$ where $n \in N_Q$ (indicating the node to delete).

Queries are used to select the nodes of the trees where insertions or deletions are made. Intuitively, when one applies an update operation (say, a deletion) on a data tree t, it results in the deletion of a sub-datatree for each valuation of Q.

DEFINITION 15. Let $\tau = (Q, v)$ be an update operation. Let t be a tree matched by Q, and let $\mu_{Q_1} \dots \mu_{Q_p}$ the mappings of Q for t and each element of $Q(t)$. Let n be the node of N_Q appearing in v. The result of the operation τ on t, denoted $\tau(t)$, is:

1. if $v = i(n, t')$, the result of the insertion of t' as a child of all $\mu_{Q_k}(n)$ for $1 \leqslant k \leqslant p$ (possibly inserting t' multiple times at the same place);
2. if $v = d(n)$, the result of the deletion of all $\mu'_{Q_k}(n)$ for $1 \leqslant k \leqslant p$.

A *probabilistic update operation* is a pair (τ, c) where τ is an update operation and $c \in]0; 1]$ is the *confidence* we have in the operation.

Definition 16. Let $S = \{(t_i, p_i)\}$ be a PW set, (τ, c) a probabilistic update operation, $\tau = (Q, v)$. The result of (τ, c) on S, denoted $(\tau, c)(S)$, is the PW set:

$$\begin{array}{ll} & \{(t,p) \in T \mid t \text{ is not selected by Q}\} \\ \bigcup & \{(\tau(t), p \cdot c) \mid t \text{ is selected by Q}\} \\ \bigcup & \{(t, p \cdot (1-c)) \mid t \text{ is selected by Q}\} \end{array}$$

We can define the result of an update operation on a prob-tree $T = (t, W, \pi, \gamma)$. Consider the case where $|Q(T)| = 1$, that is, where the position of update operations is uniquely defined (the extension when $|Q(T)| > 1$ is straightforward and detailed in [2]). Let u be the unique element of $Q(T)$ and $cond = \bigcup_{n \text{ node of } u} \gamma(n)$; *cond* is the set of conditions to be applied to the inserted and deleted nodes. Let μ_Q be the mapping defined by Q for t and u, and n the element of N_Q appearing in u. The result of (τ, c) on T, denoted $(\tau, c)(T)$, is the prob-tree obtained from t by applying the insertion or deletion of τ in the following way.

Insertions are performed at the position $\mu_Q(n)$. If we denote $cond_{\text{ancestors}}$ the union of the conditions on the (strict) ancestors of n, t' is inserted and its root is assigned the condition $\{w\} \cup \big(cond - (\gamma(\mu_Q(n)) \cup cond_{\text{ancestors}})\big)$.

Deletions are performed at the position mapped by Q on t. Let $cond_{\text{ancestors}}$ be the union of the conditions on the (strict) ancestors of $\mu_Q(n)$. Let $cond_{new} = \{w\} \cup \big(cond - (\gamma(\mu_Q(n)) \cup cond_{\text{ancestors}})\big)$. The original $\mu_Q(n)$ node is replaced by as many copies as elements of $cond_{new}$. Let now $a_1 \dots a_p$ be the p elements of $cond_{new}$. The first copy of $\mu_Q(n)$ is annotated with condition $\gamma(\mu_Q(n)) \cup \{\neg a_1\}$. The second copy of $\mu_Q(n)$ is annotated with condition $\gamma(\mu_Q(n)) \cup \{a_1, \neg a_2\} \dots$ The last copy of $\mu_Q(n)$ is annotated with conditions $\gamma(\mu_Q(n)) \cup \{a_1 \dots a_{n-1}, \neg a_n\}$.

Then, a result similar to Theorem 1 states that, for a probabilistic update operation (τ, c) and a prob-tree T, we have $[\![(\tau, c)(T)]\!] \sim (\tau, c)([\![T]\!])$, that is, the algorithm presented to perform updates on prob-trees is consistent with the possible world semantics.