On the Complexity of Managing Probabilistic XML Data

Pierre Senellart Serge Abiteboul

Principles Of Database Systems, 13th June 2007

Outline

Imprecise Data and Imprecise Tasks

Observations

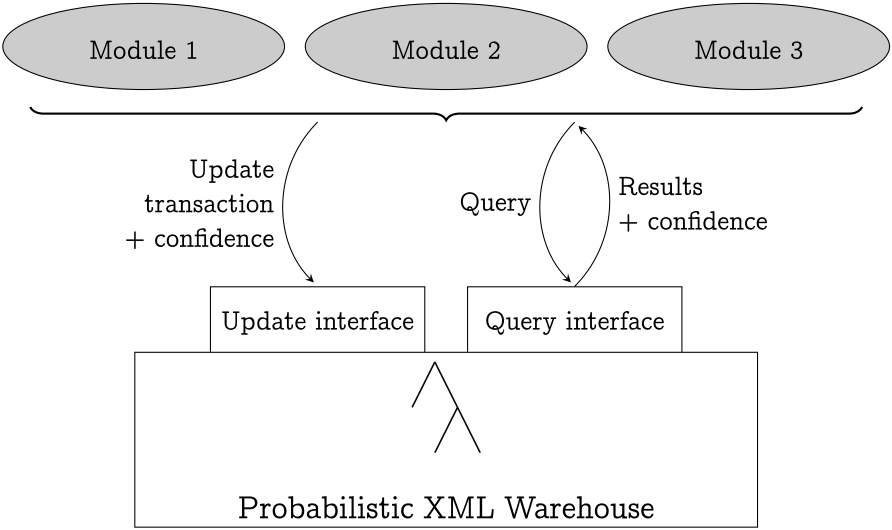
- Many tasks generate imprecise data, with some confidence value.
- Need for a way to manage this imprecision, to work with it throughout an entire complex process.

Imprecise Data and Imprecise Tasks

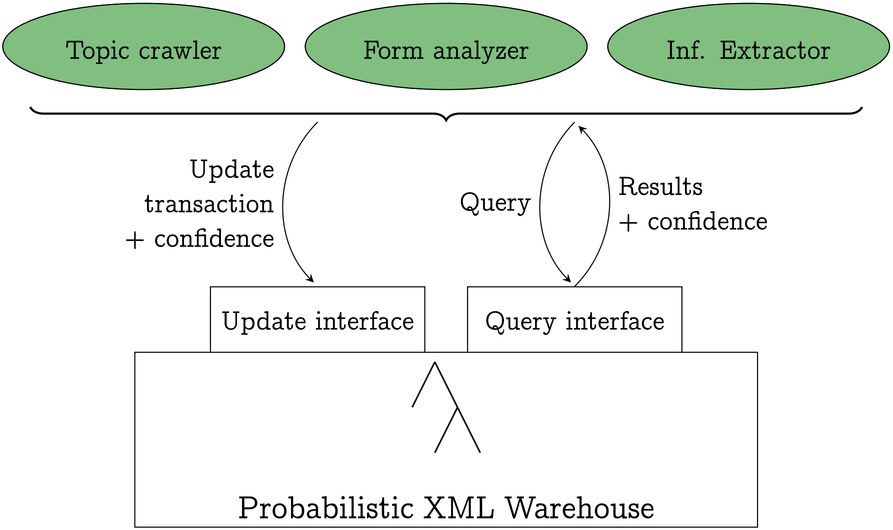
Observations

- Many tasks generate imprecise data, with some confidence value.
- Need for a way to manage this imprecision, to work with it throughout an entire complex process.

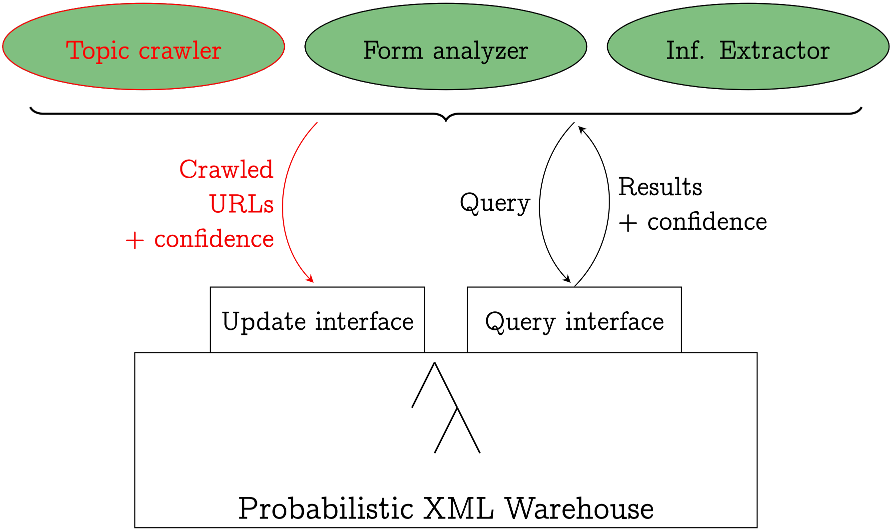
A Probabilistic XML Warehouse



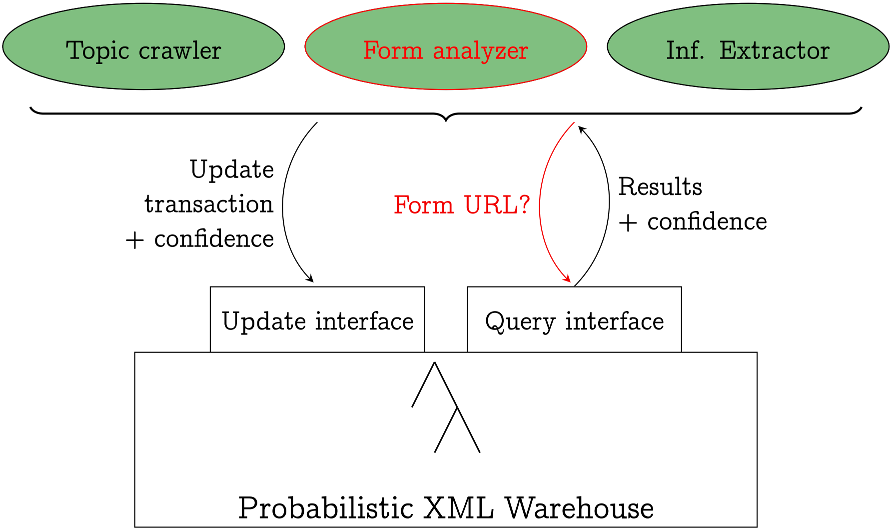
A Probabilistic XML Warehouse (Hidden Web)



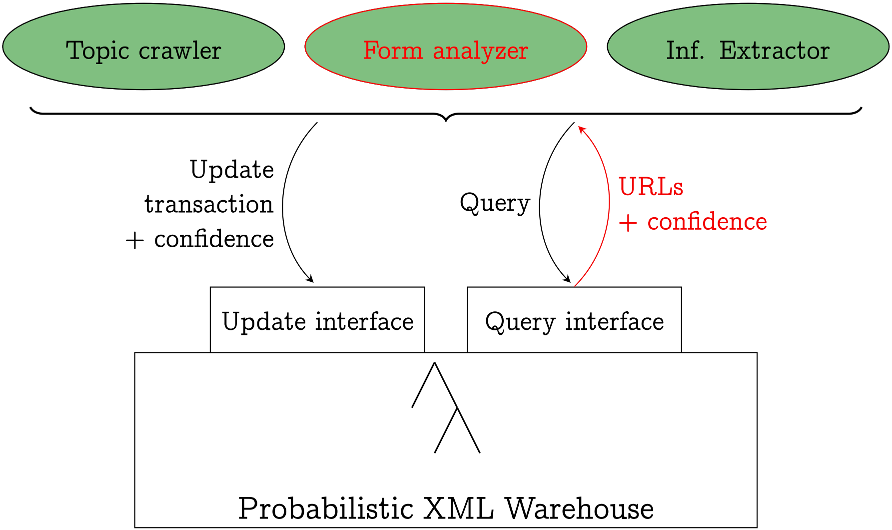
A Probabilistic XML Warehouse (Hidden Web)



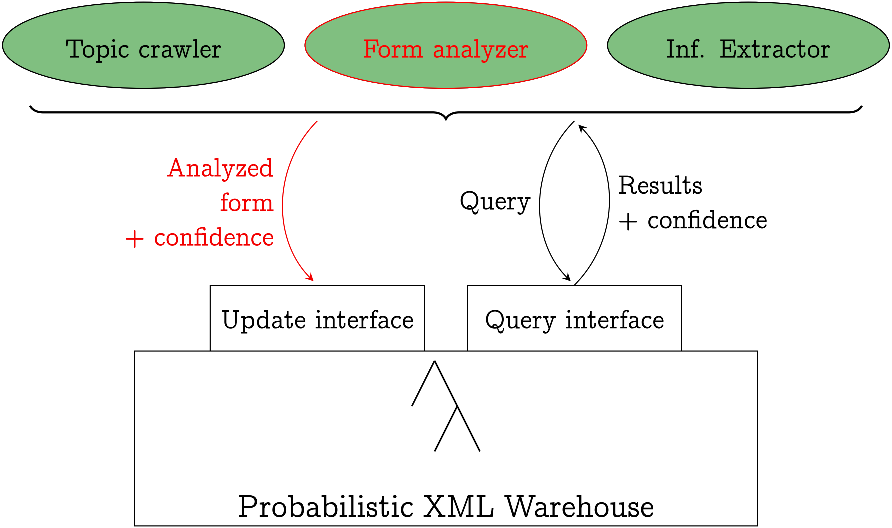
A Probabilistic XML Warehouse (Hidden Web)



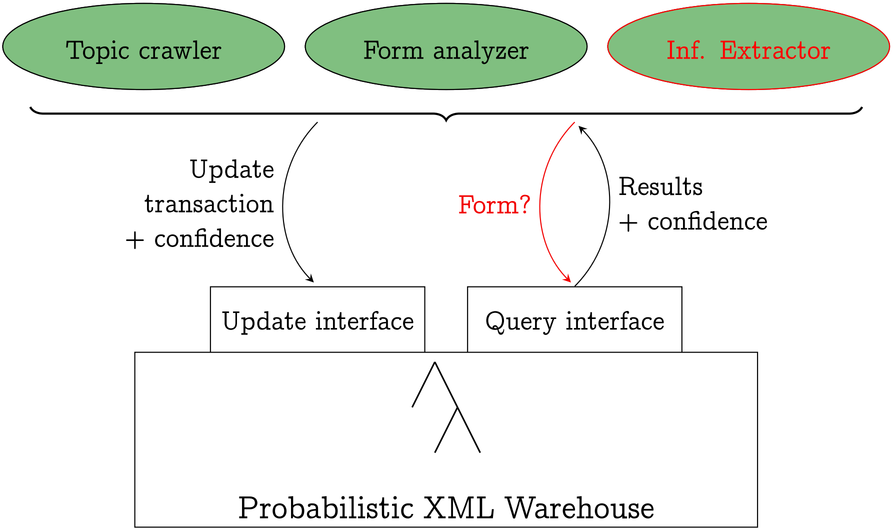
A Probabilistic XML Warehouse (Hidden Web)



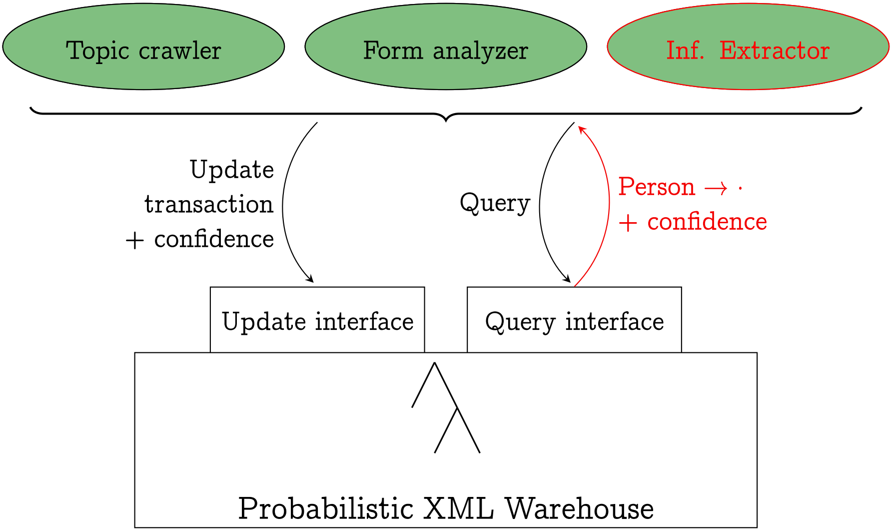
A Probabilistic XML Warehouse (Hidden Web)



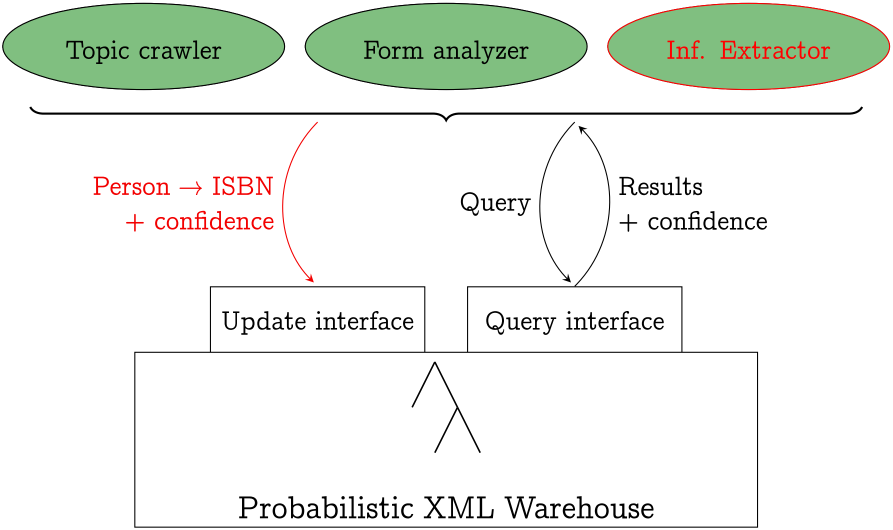
A Probabilistic XML Warehouse (Hidden Web)



A Probabilistic XML Warehouse (Hidden Web)



A Probabilistic XML Warehouse (Hidden Web)



Probabilistic Trees

Framework

- Unordered data trees
- Details: no attributes, no mixed content...

(multiset semantics)

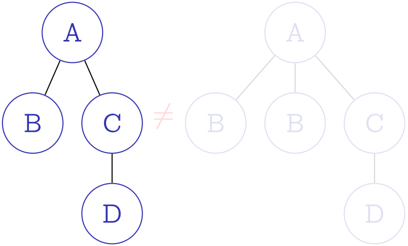
Sample space: Set of all such data trees.

Probabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.

Probabilistic Trees

Framework

- Unordered data trees
- Details: no attributes, no mixed content...



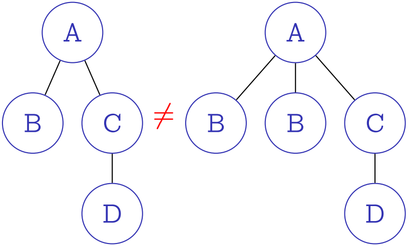
(multiset semantics)

Sample space: Set of all such data trees.

Probabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.

Probabilistic Trees

Framework
- Unordered data trees
- Details: no attributes, no mixed content...



(multiset semantics)

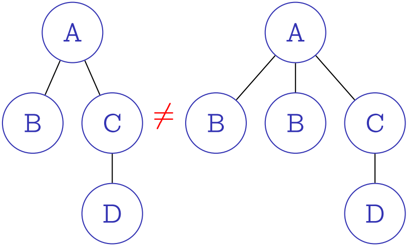
Sample space: Set of all such data trees.

Probabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.

Probabilistic Trees

Framework

- Unordered data trees
- Details: no attributes, no mixed content...



(multiset semantics)

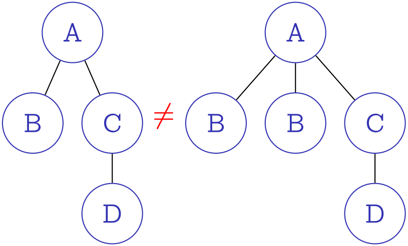
Sample space: Set of all such data trees.

Probabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.

Probabilistic Trees

Framework

- Unordered data trees
- Details: no attributes, no mixed content. . .



(multiset semantics)

Sample space: Set of all such data trees.

Probabilistic tree (prob-tree): Representation of a discrete probability distribution over this sample space.

Complexity Issues

Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?

Complexity Issues

Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?

Complexity Issues

Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?

Complexity Issues

Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?

Complexity Issues

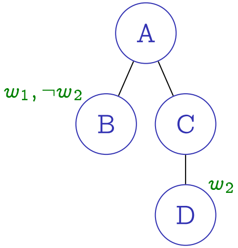
Prob-tree model defined in [Abiteboul & Senellart 2006]. Here, we tackle complexity questions about it:

- What is the complexity of queries and updates?
- Is this complexity inherent to the problem of managing tree-like probabilistic information?
- How can we check if two prob-trees are equivalent?
- Can we compute efficiently restrictions of prob-trees (e.g., by a DTD)?

Outline

The Prob-Tree Model

- Data tree with event conditions (conjunction of probabilistic events or negations of probabilistic events) assigned to each node.
- Probabilistic events are boolean random variables, assumed to be independent, with their own probability distribution.
- Representation *à la* [Imieliński & Lipksi 1984].



Event	Prob.
w_1	0.8
w_2	0.7

Semantics of Prob-Trees

Semantics of a Prob-Tree T: Set of Possible Worlds $[\![T]\!]$ (probability distribution over the set of data trees).

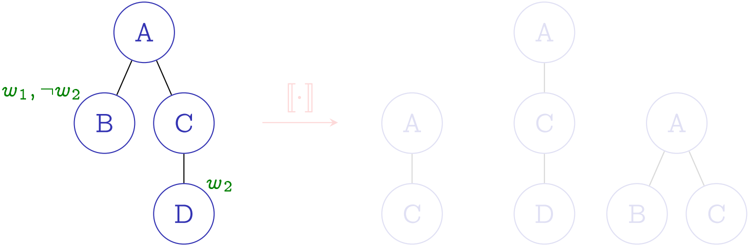
$p_1 = 0.06$ $p_2 = 0.70$ $p_3 = 0.24$

Event	Prob.
w_1	0.8
w_2	0.7

Actually, fully expressive.

Semantics of Prob-Trees

Semantics of a Prob-Tree T: Set of Possible Worlds $[\![T]\!]$ (probability distribution over the set of data trees).



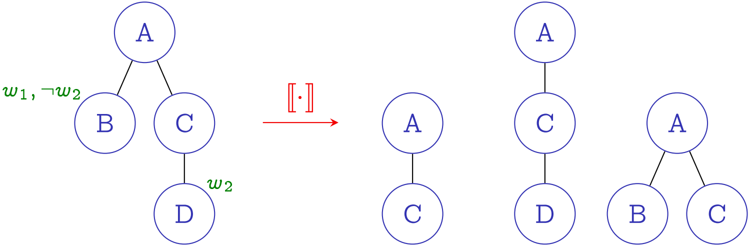
$p_1 = 0.06$ $p_2 = 0.70$ $p_3 = 0.24$

Event	Prob.
w_1	0.8
w_2	0.7

Actually, fully expressive.

Semantics of Prob-Trees

Semantics of a Prob-Tree T: Set of Possible Worlds $[\![T]\!]$ (probability distribution over the set of data trees).



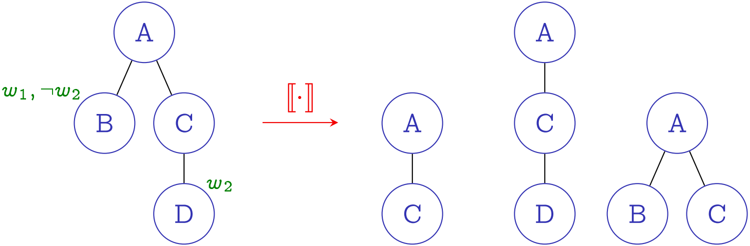
$p_1 = 0.06$ $p_2 = 0.70$ $p_3 = 0.24$

Event	Prob.
w_1	0.8
w_2	0.7

Actually, fully expressive.

Semantics of Prob-Trees

Semantics of a Prob-Tree T: Set of Possible Worlds $[\![T]\!]$ (probability distribution over the set of data trees).



$p_1 = 0.06$ $p_2 = 0.70$ $p_3 = 0.24$

Event	Prob.
w_1	0.8
w_2	0.7

Actually, fully expressive.

Locally Monotone Queries

Query: function that maps a data tree t to a set of subtrees of t containing its root.

Definition

A query Q is locally monotone if, for any data trees u, t' and t such that $u \leqslant t' \leqslant t$, $u \in Q(t) \iff u \in Q(t')$.

Examples

- Tree-pattern queries with joins are locally monotone.
- "Return the root node if it has no A child, nothing otherwise." is not locally monotone.

Locally Monotone Queries

Query: function that maps a data tree t to a set of subtrees of t containing its root.

Definition

A query Q is locally monotone if, for any data trees u, t' and t such that $u \leqslant t' \leqslant t$, $u \in Q(t) \iff u \in Q(t')$.

Examples

- Tree-pattern queries with joins are locally monotone.
- "Return the root node if it has no A child, nothing otherwise." is not locally monotone.

Locally Monotone Queries

Query: function that maps a data tree t to a set of subtrees of t containing its root.

Definition

A query Q is locally monotone if, for any data trees u, t' and t such that $u \leqslant t' \leqslant t$, $u \in Q(t) \iff u \in Q(t')$.

Examples

- Tree-pattern queries with joins are locally monotone.
- "Return the root node if it has no A child, nothing otherwise." is not locally monotone.

Locally Monotone Queries

Query: function that maps a data tree t to a set of subtrees of t containing its root.

Definition

A query Q is locally monotone if, for any data trees u, t' and t such that $u \leqslant t' \leqslant t$, $u \in Q(t) \iff u \in Q(t')$.

Examples

- Tree-pattern queries with joins are locally monotone.
- "Return the root node if it has no A child, nothing otherwise." is not locally monotone.

Queries on Prob-Trees

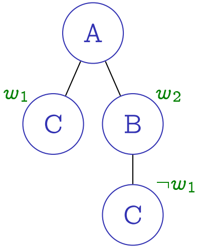
Illustration of how to query prob-trees on an example.

Query: //C

Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

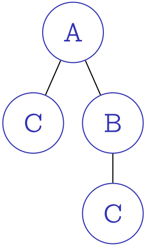
Query: //C



Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

Query: //C

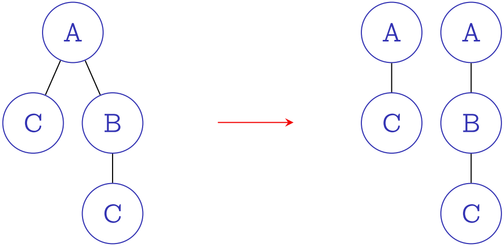


Underlying data tree.

Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

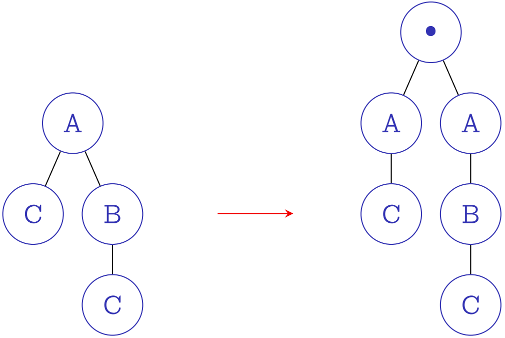
Query: //C



Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

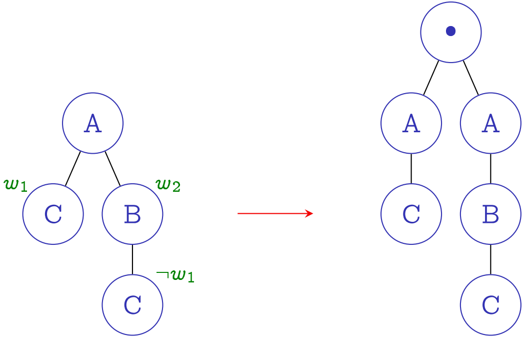
Query: //C



Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

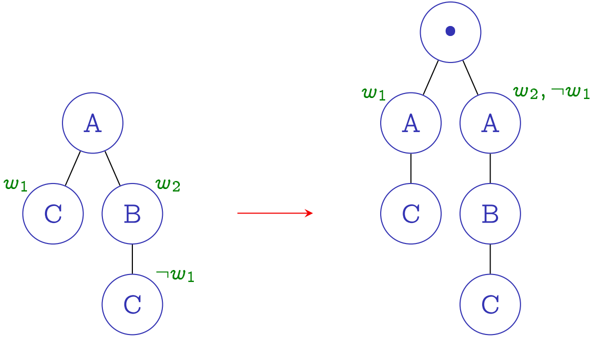
Query: //C



Queries on Prob-Trees

Illustration of how to query prob-trees on an example.

Query: //C



Consistence of Queries on Prob-Trees

Theorem

$$
\begin{tikzcd}
\textit{Prob-Tree} \arrow[r, "query"] \arrow[d, "semantics"] & \textit{Prob-Tree} \arrow[d, "semantics"] \\
\textit{Possible Worlds} \arrow[r, "query"] & \textit{Possible Worlds}
\end{tikzcd}
$$

What about updates?

- We consider sets of elementary insertions and deletions.
- Defined with respect to a query (mapping between nodes of the query and nodes to insert/delete).
- More involved definitions. . .
- . . . but a similar result:

Theorem

$$\begin{array}{ccc} \textit{Prob-Tree} & \xrightarrow{\textit{update}} & \textit{Prob-Tree} \\ \downarrow \textit{semantics} & & \downarrow \textit{semantics} \\ \textit{Possible Worlds} & \xrightarrow{\textit{update}} & \textit{Possible Worlds} \end{array}$$

What about updates?

- We consider sets of elementary insertions and deletions.
- Defined with respect to a query (mapping between nodes of the query and nodes to insert/delete).
- More involved definitions. . .
- . . . but a similar result:

Theorem

$$\begin{array}{ccc} \textit{Prob-Tree} & \xrightarrow{\textit{update}} & \textit{Prob-Tree} \\ \downarrow{\scriptstyle \textit{semantics}} & & \downarrow{\scriptstyle \textit{semantics}} \\ \textit{Possible Worlds} & \xrightarrow{\textit{update}} & \textit{Possible Worlds} \end{array}$$

What about updates?

- We consider sets of elementary insertions and deletions.
- Defined with respect to a query (mapping between nodes of the query and nodes to insert/delete).
- More involved definitions. . .
- . . . but a similar result:

Theorem

$$\begin{tikzcd} \textit{Prob-Tree} \arrow[r, "\textit{update}"] \arrow[d, "\textit{semantics}"] & \textit{Prob-Tree} \arrow[d, "\textit{semantics}"] \\ \textit{Possible Worlds} \arrow[r, "\textit{update}"] & \textit{Possible Worlds} \end{tikzcd}$$

What about updates?

- We consider sets of elementary insertions and deletions.
- Defined with respect to a query (mapping between nodes of the query and nodes to insert/delete).
- More involved definitions...
- ... but a similar result:

Theorem

$$\begin{tikzcd} \textit{Prob-Tree} \arrow[r, "\textit{update}"] \arrow[d, "\textit{semantics}"] & \textit{Prob-Tree} \arrow[d, "\textit{semantics}"] \\ \textit{Possible Worlds} \arrow[r, "\textit{update}"] & \textit{Possible Worlds} \end{tikzcd}$$

Complexity Results

T: prob-tree with underlying data tree t.
$\mathbf{time}(Q(t))$: complexity of the query Q over the data tree t.

Upper bounds for operations on T:

Operation	Complexity
Query	$\mathbf{time}(Q(t))$ + polynomial in the size of T, $Q(t)$
Insertion	$\mathbf{time}(Q(t))$ + polynomial in the size of T, $Q(t)$
Deletion	$\mathbf{time}(Q(t))$ + exponential in the size of T, $Q(t)$

Proposition

If the query language is not trivial, the result of a deletion may necessarily be exponential.

Go to proof

Complexity Results

T: prob-tree with underlying data tree t.

$\mathbf{time}(Q(t))$: complexity of the query Q over the data tree t.

Upper bounds for operations on T:

Operation	Complexity
Query	$\mathbf{time}(Q(t))$ + polynomial in the size of T, $Q(t)$
Insertion	$\mathbf{time}(Q(t))$ + polynomial in the size of T, $Q(t)$
Deletion	$\mathbf{time}(Q(t))$ + exponential in the size of T, $Q(t)$

Proposition

If the query language is not trivial, the result of a deletion may necessarily be exponential.

Go to proof

Complexity Results

T: prob-tree with underlying data tree t.

time$(Q(t))$: complexity of the query Q over the data tree t.

Upper bounds for operations on T:

Operation	Complexity
Query	**time**$(Q(t))$ + polynomial in the size of T, $Q(t)$
Insertion	**time**$(Q(t))$ + polynomial in the size of T, $Q(t)$
Deletion	**time**$(Q(t))$ + exponential in the size of T, $Q(t)$

Proposition

If the query language is not trivial, the result of a deletion may necessarily be exponential.

Go to proof

Complexity Results

T: prob-tree with underlying data tree t.
$\mathbf{time}(Q(t))$: complexity of the query Q over the data tree t.

Upper bounds for operations on T:

Operation	Complexity
Query	$\mathbf{time}(Q(t))$ + polynomial in the size of T, $Q(t)$
Insertion	$\mathbf{time}(Q(t))$ + polynomial in the size of T, $Q(t)$
Deletion	$\mathbf{time}(Q(t))$ + exponential in the size of T, $Q(t)$

Proposition

If the query language is not trivial, the result of a deletion may necessarily be exponential.

Go to proof

Complexity Results

T: prob-tree with underlying data tree t.

$\mathbf{time}(Q(t))$: complexity of the query Q over the data tree t.

Upper bounds for operations on T:

Operation	Complexity
Query	$\mathbf{time}(Q(t))$ + polynomial in the size of T, $Q(t)$
Insertion	$\mathbf{time}(Q(t))$ + polynomial in the size of T, $Q(t)$
Deletion	$\mathbf{time}(Q(t))$ + exponential in the size of T, $Q(t)$

Proposition

If the query language is not trivial, the result of a deletion may necessarily be exponential.

Go to proof

Outline

Two Notions of Equivalence

What does it mean for two prob-trees to represent the same information?

Two different notions:

Structural Equivalence: we keep the same event variables.

Semantic Equivalence: we only consider the possible worlds semantics.

Complexity results? Relation between these two notions?

Two Notions of Equivalence

What does it mean for two prob-trees to represent the same information?

Two different notions:

Structural Equivalence: we keep the same event variables.

Semantic Equivalence: we only consider the possible worlds semantics.

Complexity results? Relation between these two notions?

Two Notions of Equivalence

What does it mean for two prob-trees to represent the same information?

Two different notions:

Structural Equivalence: we keep the same event variables.

Semantic Equivalence: we only consider the possible worlds semantics.

Complexity results? Relation between these two notions?

Two Notions of Equivalence

What does it mean for two prob-trees to represent the same information?

Two different notions:

Structural Equivalence: we keep the same event variables.

Semantic Equivalence: we only consider the possible worlds semantics.

Complexity results? Relation between these two notions?

Structural Equivalence

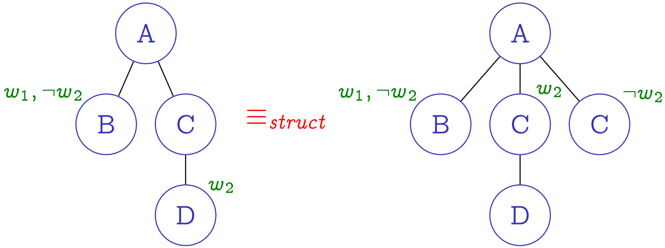
Definition

Two prob-trees T and T' are structurally equivalent ($T \equiv_{struct} T'$) if they have the same event variables, the same probability distribution, and if they define the same possible world for every valuation of the event variables.

Structural Equivalence

Definition

Two prob-trees T and T' are structurally equivalent ($T \equiv_{struct} T'$) if they have the same event variables, the same probability distribution, and if they define the same possible world for every valuation of the event variables.



Complexity of Structural Equivalence

Theorem

Structural Equivalence is a ***coRP*** *problem: there exists a randomized polynomial-time algorithm that returns* `true` *if two prob-trees are equivalent, and* `false` *with probability* $\geqslant 1/2$ *otherwise.*

Based on the notion of count-equivalence:

Definition

Two propositional formulas ψ, ψ' in DNF are count-equivalent ($\psi \stackrel{\pm}{\equiv} \psi'$) if, for every valuation of the variables of ψ and ψ', the same number of disjuncts of ψ and ψ' are satisfied.

$$A \equiv A \vee (A \wedge B) \qquad \text{but} \qquad A \stackrel{\pm}{\not\equiv} A \vee (A \wedge B)$$

Complexity of Structural Equivalence

Theorem

Structural Equivalence is a coRP problem: there exists a randomized polynomial-time algorithm that returns true if two prob-trees are equivalent, and false with probability $\geqslant 1/2$ otherwise.

Based on the notion of count-equivalence:

Definition

Two propositional formulas ψ, ψ' in DNF are count-equivalent ($\psi \overset{+}{\equiv} \psi'$) if, for every valuation of the variables of ψ and ψ', the same number of disjuncts of ψ and ψ' are satisfied.

$$A \equiv A \vee (A \wedge B) \quad \text{but} \quad A \overset{+}{\not\equiv} A \vee (A \wedge B)$$

Complexity of Structural Equivalence

Theorem

*Structural Equivalence is a **coRP** problem: there exists a randomized polynomial-time algorithm that returns true if two prob-trees are equivalent, and* false *with probability* $\geqslant 1/2$ *otherwise.*

Based on the notion of count-equivalence:

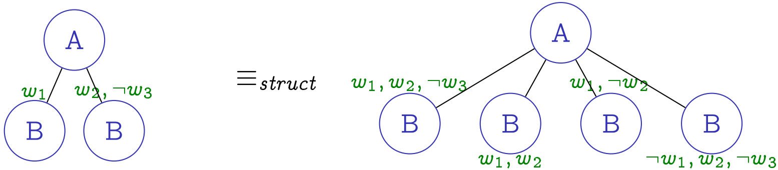
Definition

Two propositional formulas ψ, ψ' in DNF are count-equivalent ($\psi \stackrel{+}{\equiv} \psi'$) if, for every valuation of the variables of ψ and ψ', the same number of disjuncts of ψ and ψ' are satisfied.

$$A \equiv A \vee (A \wedge B) \qquad \text{but} \qquad A \stackrel{+}{\not\equiv} A \vee (A \wedge B)$$

Idea behind the Probabilistic Algorithm

In a very simple case:



$$\Longleftrightarrow \quad w_1 \lor (w_2 \land \neg w_3) \quad \stackrel{+}{\equiv} \quad (w_1 \land w_2 \land \neg w_3) \lor (w_1 \land w_2) \lor (w_1 \land \neg w_2) \lor (\neg w_1 \land w_2 \land \neg w_3)$$

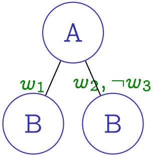
$$\Longleftrightarrow \quad X_1 + X_2(1 - X_3) \quad = \quad X_1 X_2 (1 - X_3) + X_1 X_2 + X_1(1 - X_2) + (1 - X_1) X_2 (1 - X_3)$$

(see [Green, Karvounarakis & Tannen 2007]).

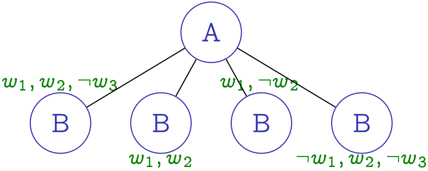
Polynomial-time randomized algorithm for determining if a multivariate polynomial is zero [Schwartz 1980].

Idea behind the Probabilistic Algorithm

In a very simple case:



$\equiv_{struct}$



$$\iff \quad w_1 \lor (w_2 \land \neg w_3) \quad \stackrel{+}{\equiv} \quad (w_1 \land w_2 \land \neg w_3) \lor (w_1 \land w_2) \lor (w_1 \land \neg w_2) \lor (\neg w_1 \land w_2 \land \neg w_3)$$

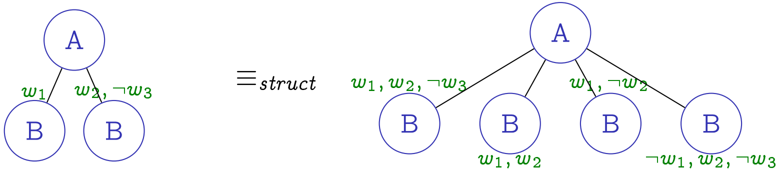
$$\iff \quad X_1 + X_2(1 - X_3) \quad = \quad X_1 X_2 (1 - X_3) + X_1 X_2 + X_1(1 - X_2) + (1 - X_1) X_2 (1 - X_3)$$

(see [Green, Karvounarakis & Tannen 2007]).

Polynomial-time randomized algorithm for determining if a multivariate polynomial is zero [Schwartz 1980].

Idea behind the Probabilistic Algorithm

In a very simple case:



$$\iff \quad w_1 \lor (w_2 \land \neg w_3) \quad \stackrel{+}{\equiv} \quad (w_1 \land w_2 \land \neg w_3) \lor (w_1 \land w_2) \lor (w_1 \land \neg w_2) \lor (\neg w_1 \land w_2 \land \neg w_3)$$

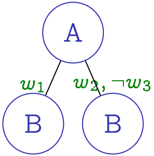
$$\iff \quad X_1 + X_2(1 - X_3) \quad = \quad X_1X_2(1 - X_3) + X_1X_2 + X_1(1 - X_2) + (1 - X_1)X_2(1 - X_3)$$

(see [Green, Karvounarakis & Tannen 2007]).

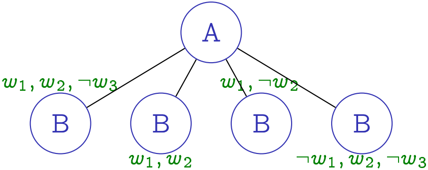
Polynomial-time randomized algorithm for determining if a multivariate polynomial is zero [Schwartz 1980].

Idea behind the Probabilistic Algorithm

In a very simple case:



$\equiv_{struct}$



$$\Longleftrightarrow \quad w_1 \lor (w_2 \land \neg w_3) \quad \stackrel{+}{\equiv} \quad (w_1 \land w_2 \land \neg w_3) \lor (w_1 \land w_2) \lor (w_1 \land \neg w_2) \lor (\neg w_1 \land w_2 \land \neg w_3)$$

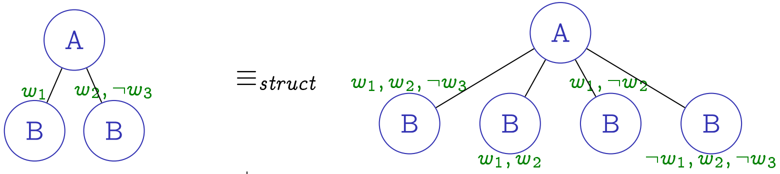
$$\Longleftrightarrow \quad X_1 + X_2(1 - X_3) \quad = \quad X_1 X_2 (1 - X_3) + X_1 X_2 + X_1(1 - X_2) + (1 - X_1) X_2 (1 - X_3)$$

(see [Green, Karvounarakis & Tannen 2007]).

Polynomial-time randomized algorithm for determining if a multivariate polynomial is zero [Schwartz 1980].

Idea behind the Probabilistic Algorithm

In a very simple case:



$$\iff \quad w_1 \vee (w_2 \wedge \neg w_3) \quad \stackrel{+}{\equiv} \quad (w_1 \wedge w_2 \wedge \neg w_3) \vee (w_1 \wedge w_2) \vee (w_1 \wedge \neg w_2) \vee (\neg w_1 \wedge w_2 \wedge \neg w_3)$$

$$\iff \quad X_1 + X_2(1 - X_3) \quad = \quad X_1 X_2 (1 - X_3) + X_1 X_2 + X_1(1 - X_2) + (1 - X_1) X_2 (1 - X_3)$$

(see [Green, Karvounarakis & Tannen 2007]).

Polynomial-time randomized algorithm for determining if a multivariate polynomial is zero [Schwartz 1980].

Semantic Equivalence

Definition

Two prob-trees T and T' are semantically equivalent ($T \equiv_{sem} T'$) if $[\![T]\!] = [\![T']\!]$.

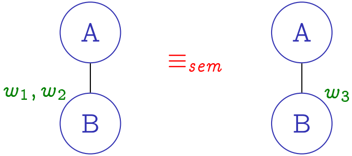
Event	Prob.
w_1	0.5
w_2	0.8
w_3	0.4

Semantic Equivalence

Definition

Two prob-trees T and T' are semantically equivalent ($T \equiv_{sem} T'$) if $[\![T]\!] = [\![T']\!]$.

Event	Prob.
w_1	0.5
w_2	0.8
w_3	0.4

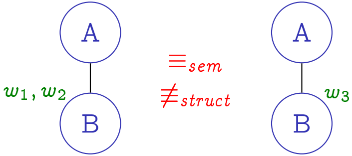


Semantic Equivalence

Definition

Two prob-trees T and T' are semantically equivalent ($T \equiv_{sem} T'$) if $[\![T]\!] = [\![T']\!]$.

Event	Prob.
w_1	0.5
w_2	0.8
w_3	0.4



Semantic and Structural Equivalence

Facts

1. *If* $T \equiv_{struct} T'$, *then* $T \equiv_{sem} T'$
2. *If* $T \equiv_{sem} T'$ *for every possible probability distribution, then* $T \equiv_{struct} T'$.

Complexity of semantic equivalence: open issue. Easy **EXPTIME** upper bound.

Semantic and Structural Equivalence

Facts

1. *If* $T \equiv_{struct} T'$, *then* $T \equiv_{sem} T'$
2. *If* $T \equiv_{sem} T'$ *for every possible probability distribution, then* $T \equiv_{struct} T'$.

Complexity of semantic equivalence: open issue. Easy **EXPTIME** upper bound.

Semantic and Structural Equivalence

Facts

1. *If* $T \equiv_{struct} T'$, *then* $T \equiv_{sem} T'$
2. *If* $T \equiv_{sem} T'$ *for every possible probability distribution, then* $T \equiv_{struct} T'$.

Complexity of semantic equivalence: open issue. Easy **EXPTIME** upper bound.

Outline

Restriction to a Probability Threshold

- Is it possible to remove from a prob-tree least probable worlds?
- $[\![T]\!]_{|\geqslant p}$: set of possible worlds in $[\![T]\!]$ whose probabilities are greater than p.

Proposition

The prob-tree representation of $[\![T]\!]_{|\geqslant p}$ is sometimes necessarily exponential.

Restriction to a Probability Threshold

- Is it possible to remove from a prob-tree least probable worlds?
- $[\![T]\!]_{|\geqslant p}$: set of possible worlds in $[\![T]\!]$ whose probabilities are greater than p.

Proposition

The prob-tree representation of $[\![T]\!]_{|\geqslant p}$ is sometimes necessarily exponential.

Restriction to a Probability Threshold

- Is it possible to remove from a prob-tree least probable worlds?
- $[\![T]\!]_{|\geqslant p}$: set of possible worlds in $[\![T]\!]$ whose probabilities are greater than p.

Proposition

The prob-tree representation of $[\![T]\!]_{|\geqslant p}$ is sometimes necessarily exponential.

DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- *Deciding if, given a prob-tree, there exists a possible world valid against a DTD is NP-complete.*
- *Deciding if, given a prob-tree, all possible worlds are valid against a DTD is coNP-complete.*
- *In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.*

DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- *Deciding if, given a prob-tree, there exists a possible world valid against a DTD is NP-complete.*
- *Deciding if, given a prob-tree, all possible worlds are valid against a DTD is coNP-complete.*
- *In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.*

DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- *Deciding if, given a prob-tree, there exists a possible world valid against a DTD is* **NP***-complete.*
- *Deciding if, given a prob-tree, all possible worlds are valid against a DTD is* **coNP***-complete.*
- *In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.*

DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- *Deciding if, given a prob-tree, there exists a possible world valid against a DTD is **NP**-complete.*
- *Deciding if, given a prob-tree, all possible worlds are valid against a DTD is **coNP**-complete.*
- *In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.*

DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- *Deciding if, given a prob-tree, there exists a possible world valid against a DTD is* **NP***-complete.*
- *Deciding if, given a prob-tree, all possible worlds are valid against a DTD is* **coNP***-complete.*
- *In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.*

DTD Validation

- Is it possible to compute the restriction of a prob-tree to worlds valid against a given DTD?
- DTD definition adapted to the case of unordered trees, and without disjunction.

Proposition

- *Deciding if, given a prob-tree, there exists a possible world valid against a DTD is* **NP**-*complete.*
- *Deciding if, given a prob-tree, all possible worlds are valid against a DTD is* **coNP**-*complete.*
- *In some cases, the prob-tree representation of the restriction of a prob-tree to a given DTD is of exponential size.*

Outline

Summary

- A model for representing probabilistic information in semi-structured databases.
- Polynomial complexity for queries and insertions.
- Unavoidable exponential complexity for deletions.
- Characterization of the complexity of key problems.
- Structural equivalence: randomized polynomial algorithm.

Summary

- A model for representing probabilistic information in semi-structured databases.
- Polynomial complexity for queries and insertions.
- Unavoidable exponential complexity for deletions.
- Characterization of the complexity of key problems.
- Structural equivalence: randomized polynomial algorithm.

Summary

- A model for representing probabilistic information in semi-structured databases.
- Polynomial complexity for queries and insertions.
- Unavoidable exponential complexity for deletions.
- Characterization of the complexity of key problems.
- Structural equivalence: randomized polynomial algorithm.

Summary

- A model for representing probabilistic information in semi-structured databases.
- Polynomial complexity for queries and insertions.
- Unavoidable exponential complexity for deletions.
- Characterization of the complexity of key problems.
- Structural equivalence: randomized polynomial algorithm.

Summary

- A model for representing probabilistic information in semi-structured databases.
- Polynomial complexity for queries and insertions.
- Unavoidable exponential complexity for deletions.
- Characterization of the complexity of key problems.
- Structural equivalence: randomized polynomial algorithm.

Perspectives

- Complexity of semantic equivalence.
- Prob-tree simplification.
- Top-k possible worlds from a prob-tree.
- Aggregate functions in queries.

Perspectives

- Complexity of semantic equivalence.
- Prob-tree simplification.
- Top-k possible worlds from a prob-tree.
- Aggregate functions in queries.

Perspectives

- Complexity of semantic equivalence.
- Prob-tree simplification.
- Top-k possible worlds from a prob-tree.
- Aggregate functions in queries.

Perspectives

- Complexity of semantic equivalence.
- Prob-tree simplification.
- Top-k possible worlds from a prob-tree.
- Aggregate functions in queries.

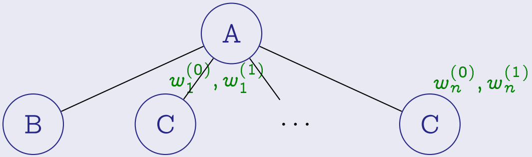
Merci.

Proof of the Exponential Complexity of Deletion

Proof.

Deletion d: "If the root has a C-child, then delete all B-children of the root."

$T =$



$\forall i, \pi(w_i) = 1/2$

Then, it can be shown that if $T' \equiv_{struct} d(T)$, at least 2^n literals appear in T'.

$\square$

◂ Return to theorem

References I

Tomasz Imieliński and Witold Lipski.
Incomplete information in relational databases.
Journal of the ACM, 31(4):761–791, 1984.

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

Serge Abiteboul and Pierre Senellart.
Querying and updating probabilistic information in XML.
In *Extending DataBase Technology*, Munich, Germany, March 2006.

References II

Todd J. Green, Grigoris Karvounarakis and Val Tannen.
Provenance semirings.
In *Principles of DataBase Systems*, Beijing, China, June 2007.