XML with Incomplete Information: Models, Properties, and Query Answering

Pablo Barceló
Dept. of Computer Science, Univ. of Chile
pbarcelo@dcc.uchile.cl

Leonid Libkin
Sch. of Informatics, Univ. of Edinburgh
libkin@inf.ed.ac.uk

Antonella Poggi
DIS, Sapienza Univ. of Rome
poggi@dis.uniroma1.it

Cristina Sirangelo
LSV, ENS-Cachan, CNRS and INRIA
sirangel@lsv.ens-cachan.fr

ABSTRACT

We study models of incomplete information for XML, their computational properties, and query answering. While our approach is motivated by the study of relational incompleteness, incomplete information in XML documents may appear not only as null values but also as missing structural information. Our goal is to provide a classification of incomplete descriptions of XML documents, and separate features - or groups of features - that lead to hard computational problems from those that admit efficient algorithms. Our classification of incomplete information is based on the combination of null values with partial structural descriptions of documents. The key computational problems we consider are consistency of partial descriptions, representability of complete documents by incomplete ones, and query answering. We show how factors such as schema information, the presence of node ids, and missing structural information affect the complexity of these main computational problems, and find robust classes of incomplete XML descriptions that permit tractable query evaluation.

Categories and Subject Descriptors. H.2.1 [**Database Management**]: Logical Design—*Data Models*

General Terms. Theory, Languages, Algorithms

Keywords. XML, incomplete information, query answering, certain answers, consistency, membership

1. Introduction

The transfer and extension of relational tools to deal with XML data has been a central theme in database research over the past decade. One area that has not witnessed much activity is the handling of incomplete information in XML. And yet incomplete information is ubiquitous in XML applications, especially in exchanging and integrating web data – the key applications XML was designed for.

In the research literature, there are some papers that address the problem of incompleteness in XML, but this typically happens in some specific scenarios. For example, [3] concentrated on handling incompleteness arising in a dynamic setting in which the structure of a tree is revealed by a sequence of queries, [11, 12] looked at graph and tree data models expressed as description logic theories that could incorporate incompleteness, [21] dealt with incompleteness in query results but not inputs, and [27, 13] looked at incorporating probabilities into XML. In practice incomplete information needs to be modeled as well, most commonly by optional attributes, or tricks such as `minOccurs="0"` to introduce nulls at the level of elements.

Our goal is to provide a systematic study of incomplete information in XML that is independent of any particular application. We would like to address the same problems as the fundamental study of relational incompleteness, namely:

1. study models of incompleteness in XML and their semantics; and
2. study the key computational tasks associated with such models (e.g., query answering) with the main goal of separating features that lead to good algorithmic solutions from those that lead to intractability.

The results we obtain can be used in any application scenario, as they say for which classes of problems and models efficient solutions cannot be found, and for which classes such solutions exist.

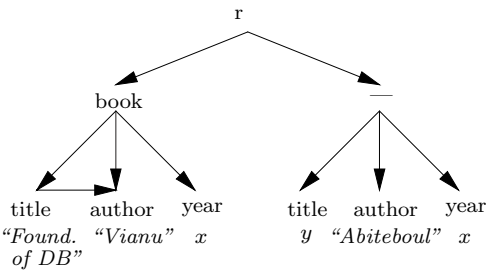
The inspiration for such a general study comes from the study of incompleteness in relational databases. There, incompleteness arises when some attribute values are unknown for a variety of reasons and are represented as nulls. The design of SQL adopted a single type of null and the (often criticized) reasoning model based on the 3-valued logic. Theoretical investigations of nulls culminated in two papers that are the foundation of the theory of relational incompleteness. The paper by Imielinski and Lipski [20] introduced the notion

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

of tables as a representation mechanism for incomplete information, and looked at types of tables that are suitable for evaluating queries from various sublanguages of relational algebra. The paper by Abiteboul, Kanellakis, and Grahne [2] studied the complexity of computational problems associated with incompleteness, and provided a clear separation between tractable and intractable cases. These results continue to be very influential. For example, the fact that unions of conjunctive queries can be evaluated in polynomial time over naïve tables (in which nulls can be repeated) is used heavily in data integration and exchange [1, 17, 23].

The structure of XML documents is much more complicated than that of relational databases, and missing information may appear not only among attribute values, but also in the *structure* itself. And in addition the way we view XML documents may lead to different representations of incomplete information.

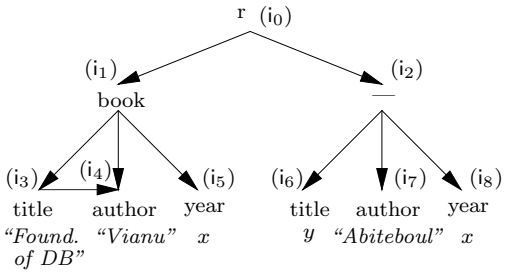
To see how incompleteness can be represented in XML, consider a document that describes books and papers, by giving their titles, authors, and years of publication. An incomplete description of such a document is presented below:



The left subtree talks about the *Foundations of Databases* book; it tells us that one of the authors is Vianu, but it does not give us precise information about the publication date (year is null, given by a variable x). The second subtree says that there is some publication by Abiteboul (we do not know if it is a book or an article since wildcard is used as a label); all we know about it is that it was published in the same year x. We also know that the author node for Vianu is an immediate successor of the book title, but no other information about sibling ordering is available.

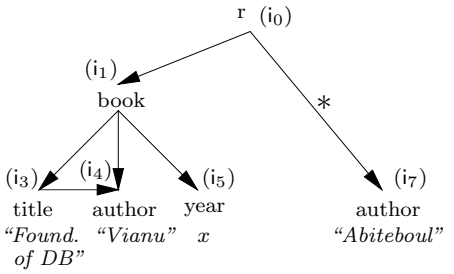
This document can represent many complete trees: one example is a description of *Foundations of Databases*. In that case we assume that the root has just one child (which is consistent with the description, since _ matches every label), with one title node, a year node with the value '1995', and three author nodes for Abiteboul, Hull, and Vianu. We are making the open world assumption and allow addition of nodes; in particular the incomplete document above does not have the knowledge that Hull is one of the authors.

We now turn to a slightly different way of modeling XML, which corresponds to the DOM interface [16]. In that case, we can access each node in a document by its id, and apply various methods that produce its parent, left and right siblings, first child, all children, etc. They key point is that a node is uniquely identified by its id. Consider now what looks like almost the same incomplete document:



The small change – we gave ids to all nodes, shown in parentheses as (i_k) – makes a big impact on the semantics. For example, it is no longer possible that the document represents a single book, as before. Indeed, we know that the two children of the root are different, since $\mathsf{i}_1 \neq \mathsf{i}_2$.

But one can still have an incomplete document description that is consistent with the document representing only information about *Foundations of Databases*, even with unique ids associated with each node. Assume that we lose *structural* information that the author-node i_7 is a grandchild of the root, and instead we only know that it is a descendant of the root, as shown below. Then it is still consistent with an incomplete description that i_7 is a child of i_1 and thus describes an author of *Foundations of Databases*.



These examples start giving us an indication of the nature of incomplete information in XML, and how various choices of parameters affect the semantics of incompleteness. In addition to the standard missing information – attribute values – we may have missing structure information such as labels (replaced by wildcards) or information about edges (in the above examples, we miss some next-sibling information or replace a precise path to a node by a single descendant edge). Furthermore, there is a choice of having node ids, which affects the semantics of incompleteness.

In comparison with relational databases, there are many more parameters to consider when we classify incomplete descriptions of XML trees. They include the nature of nulls for attributes, the exact set of axes used in descriptions, the presence of node ids. A full classification of those will give us a large number of cases, and studying all of them is certainly not our goal.

What we want to understand in this paper is the interplay between features, or groups of features, that leads to efficient algorithms (or intractability) for various computational problems associated with incomplete information. We want to find robust and naturally definable classes of incomplete descriptions that lead to efficient algorithmic solutions.

The plan of the paper is as follows. In Section 2, we review incompleteness in relational databases. In Section 3 we describe XML documents in a way that makes it easy to introduce models of incompleteness, by eliminating some of the features of complete documents. In Section 4, we introduce models of incomplete information in XML documents, their classification, and their relational representations. In Section 5 we study basic computational problems associated with incomplete information, such as consistency of incomplete descriptions (with and without schema information) and membership in the set of complete trees represented by an incomplete description. In Section 6 we study query answering; we show that even for conjunctive queries, computing certain answers could be hard (which is different from the relational case) and find a natural class of incomplete descriptions and queries for which an analog of relational naïve evaluation finds certain answers in polynomial time. In Section 7 we give an overview of restrictions that lead to tractability. Future work is outlined in Section 8.

2. Incompleteness in relational databases

We now briefly recall the basics of incomplete information in relational databases [4, 2, 20]. Incompleteness is represented by means of *tables* in which both values and variables (for nulls) can be used. For example, $T = \{(1, x), (y, 2), (x, 1)\}$ is a table. Such a table can represent complete relations, i.e. relations without nulls, that contain all the tuples in T under some valuation of nulls. Formally, a relation R is represented by T if for some valuation ν (i.e. a mapping from nulls to constants), $\nu(T) \subseteq R$. The set of such relations is usually denoted by $Rep(T)$. This definition naturally extends to databases with multiple relations. Note that we are making the open world assumption here; under the closed world assumption, $Rep(T)$ would consist only of relations $\nu(T)$.

There are different types of tables: in *Codd tables*, all variable occurrences are distinct; in *naïve* tables, the same variable can occur more than once (as in the table T above), and in conditional tables one can impose more complex conditions than just equality on variables [20].

The key computational problems related to incompleteness are membership and query answering (there are several others considered, e.g., in [2] but they are variations on these two themes). The membership problem is to check if a complete database is represented by an incomplete one, that is, whether $R \in Rep(T)$. For query answering, typically we deal with *certain answers* [20], defined as $certain(Q, T) = \bigcap\{Q(R) \mid R \in Rep(T)\}$. Key results from [20] tell us where the tractability boundary for these problems are. For example, membership is PTIME for Codd tables but NP-complete for naïve tables. Query answering over naïve tables is tractable for unions of conjunctive queries. This is done by the *naïve evaluation*. Under it, nulls are viewed as values, but only null-free tuples are kept in the output. For relational algebra, the complexity ranges from coNP-complete under the closed world assumption to undecidable under the open world assumption [2, 28].

3. XML documents

Before introducing models of incompleteness in XML, we define complete XML trees. We describe them in an exhaustive way – including information about child and next-sibling axes, their transitive closures, labels, and attributes - so that later we introduce models of incompleteness by removing features of complete documents.

We assume the following disjoint countably infinite sets:

- *Labels* of possible names of element types (that is, node labels in trees);
- $Attr$ of attribute names; we precede them with an @ to distinguish them from element types;
- $\mathcal{I}$ of node ids; and
- $\mathcal{D}$ of attribute values (e.g., strings).

We formally define trees as two-sorted relational structures over node ids and attribute values. For finite sets of labels and attributes, $\Sigma \subset Labels$ and $A \subset Attr$, define the vocabulary

$$\tau_{\Sigma,A} = \begin{pmatrix} E, NS, E^*, NS^*, (A_{@a})_{@a \in A} \\ (P_\ell)_{\ell \in \Sigma}, Root, Leaf, FC, LC \end{pmatrix}$$

where all relations in the first line are binary and all relations in the second line are unary. A tree is a 2-sorted structure of vocabulary $\tau_{\Sigma,A}$, i.e. $\langle V, D, \tau_{\Sigma,A}\rangle$, where $V \subset \mathcal{I}$ is a finite set of node ids, $D \subset \mathcal{D}$ is a finite set of data values, and

- E, NS are the child and the next-sibling relations, so that $\langle V, E, NS\rangle$ is an ordered unranked tree; E^* and NS^* are their reflexive-transitive closures (descendant or self, and younger sibling or self).
- each $A_{@a_i}$ assigns values of attribute $@a_i$ to nodes, i.e. it is a subset of $V \times D$ such that at most one pair (i, c) is present for each $\mathsf{i} \in V$;
- P_ℓ are labeling predicates: $\mathsf{i} \in V$ belongs to P_ℓ iff it is labeled ℓ; as usual, we assume that the P_ℓ's form a partition of V;
- Sets $Root, Leaf, FC, LC$ contain the root, the leaves, first (oldest) and last (youngest) children of nodes.

A DTD over a set $\Sigma \subset Labels$ of labels and $A \subset Attr$ of attributes is a triple $d = (r, \rho, \alpha)$, where $r \in \Sigma$, and ρ is a mapping from Σ to regular languages over $\Sigma - \{r\}$, and α is a mapping from Σ to subsets of A. As usual, r is the root, and in a tree T that conforms to d (written as $T \models d$), for each node s labeled ℓ, the set of labels of its children, read left-to-right, forms a string in the language of $\rho(\ell)$, and the set of attributes of s is precisely $\alpha(\ell)$. We assume, for complexity results, that regular languages are given by NFAs.

We now show how to produce complete descriptions of XML trees by means of a grammar that will guide us when we develop incomplete descriptions of trees. Trees (t) and forests (f) can be given by the following syntax:

$$t := \beta\langle f\rangle \qquad f := \varepsilon \mid tf \tag{1}$$

where β ranges over descriptions of nodes (defined below). In other words, each tree $\beta\langle f\rangle$ is given by a description of its root node β and the forest f of its children, and each forest f is either empty or a sequence of trees. Trees are *ordered*: for the tree $\beta\langle t_1 \dots t_k\rangle$ we assume that the tree t_1 is rooted at the first child of the node given by β, the tree t_2 at the second child, and so on.

A *node description* β for a node with label $\ell \in Labels$, id $\mathsf{i} \in \mathcal{I}$ and attributes $@a_1, \dots, @a_m$ with values $v_1, \dots, v_m \in \mathcal{D}$ is given by $\beta = \ell(\mathsf{i})[@a_1 = v_1, \dots, @a_m = v_m]$.

4. Models of incompleteness in XML

We start with complete tree descriptions (1) and see how missing information can be incorporated into them. In addition to missing attribute values, the following structural information can be missing too:

(a) node ids (they can be replaced by node variables);

(b) node labels (they can be replaced by wildcards _);

(c) precise vertical relationship between nodes (we can use descendant edges in addition to child edges);

(d) precise horizontal relationship between nodes (using younger-sibling edges instead of next-sibling).

In both (c) and (d), we may allow partial information to be recovered: for example, we may know that a node is a leaf, or that it is a first child.

We now represent all these types of incompleteness by means of more expressive tree/forest descriptions that those in (1). Since we deal with two-sorted structures (over nodes and attribute values), we shall need variables of two kinds to represent unknown values of those. That is, we assume that we have disjoint sets of variables $\mathcal{V}_{\rm node}$ (for node variables) and $\mathcal{V}_{\rm attr}$ (for nulls that correspond to attribute values).

Node descriptions These are of the form

$$\beta = \ell^{\mu}(x)[@a_1 = z_1, \dots, @a_m = z_m],$$

where

- $\ell \in \Sigma \cup \{\_\}$ (label or wildcard);
- μ is a *marking*: a subset (possibly empty) of *root, leaf, fc, lc.*
- $x \in \mathcal{V}_{\rm node} \cup \mathcal{I}$ is a node variable or a node id.
- $@a_1, \dots, @a_m$ are attribute names, and each z_i is a variable from $\mathcal{V}_{\rm attr}$ or a constant from $\mathcal{D}$.

Incomplete descriptions We define incomplete tree descriptions (t) and incomplete forest descriptions (f) by

$$\begin{array}{rcl} t & := & \beta\langle f\rangle\langle\langle f'\rangle\rangle \\ f, f' & := & \varepsilon \mid t_1\, \theta_1\, t_2\, \theta_2\, \dots \theta_{k-1}\, t_k \mid f \| f' \end{array} \quad (2)$$

where each θ_i is either $\rightarrow$ or $\rightarrow^*$; each t_i is an incomplete tree description.

Intuitively, node descriptions introduce nulls, wildcards, and markings. A tree description $\beta\langle f\rangle\langle\langle f'\rangle\rangle$ indicates a tree with a root node described by β so that it has a forest f of children and a forest f' of descendants. Forests could be empty, or forests of sibling trees (e.g., $t_1 \rightarrow t_2 \rightarrow^* t_3$ says that we have a forest consisting of three trees, so that the root of t_2 is the next sibling after the root of t_1, and the root of t_3 is a younger sibling than those two roots), or unions of forests ($f \| f'$).

We now describe the third tree from the introduction in our syntax. Assume that title, author, and year nodes have attributes $@t$, $@a$ and $@y$. The 6 nodes are described by:

$$\begin{array}{rcl} \beta_0 & = & r^{root}(\mathsf{i}_0) \\ \beta_1 & = & book(\mathsf{i}_1) \\ \beta_3 & = & title(\mathsf{i}_3)[@t = \text{“}Found\ of\ DB\text{”}] \\ \beta_4 & = & author(\mathsf{i}_4)[@a = \text{“}Vianu\text{”}] \\ \beta_5 & = & year(\mathsf{i}_5)[@y = x] \\ \beta_7 & = & author(\mathsf{i}_7)[@a = \text{“}Abiteboul\text{”}] \end{array}$$

Then the whole tree is described by

$$\beta_0\langle\, \beta_1\langle \beta_3 \rightarrow \beta_4 \,\|\, \beta_5\rangle \,\rangle\, \langle\langle \beta_7\rangle\rangle$$

(strictly speaking, one should write $\beta_3\langle\varepsilon\rangle \rightarrow \beta_4\langle\varepsilon\rangle \,\|\, \beta_5\langle\varepsilon\rangle$ instead of $\beta_3 \rightarrow \beta_4 \,\|\, \beta_5$, but we shall omit empty forests ε for notational convenience).

Semantics As for incomplete databases, we define $Rep(t)$ as the set of complete trees represented by an incomplete tree description t. There are two equivalent definitions: one is by stating what it means for a tree to witness an incomplete pattern (shown below), and another by an analog of the relational definition of Rep (which we present shortly, after defining the relational representation of incomplete tree descriptions).

Let $\bar{x}$ be the set of all node variables used in t and $\bar{z}$ the set of all nulls used in t. Given a valuation $\nu = (\nu_{node}, \nu_{attr})$ with $\nu_{node} : \bar{x} \rightarrow \mathcal{I}$ and $\nu_{attr} : \bar{z} \rightarrow \mathcal{D}$, and a node s of T, we use the semantic notion $(T, \nu, s) \models t$: intuitively, it means that a complete tree T matches t at node s, if node variables and nulls are interpreted according to ν. Then we define

$$Rep(t) = \{T \mid (T, \nu, s) \models t \text{ for some node } s \text{ and } \nu\}.$$

We further define $Rep_{\Sigma,A}(t)$ as the restrictions of $Rep(t)$ to $\tau_{\Sigma,A}$-trees, for $\Sigma \subset Labels$ and $A \subset Attr$.

We now define $(T, \nu, s) \models t$, as well $(T, \nu, S) \models f$ (which means that T matches f at a set S of roots of subtrees in T). We assume that ν_{node} and ν_{attr} are the identity when applied to node ids from $\mathcal{I}$ and data values from $\mathcal{D}$.

- $(T, \nu, s) \models \ell^{\mu}(x)[@a_1 = z_1, \dots, @a_m = z_m]$ iff $\nu_{node}(x) = s$, node s is labeled ℓ (if $\ell \in Labels$), all the μ-markings are correct in s, and the value of each attribute $@a_i$ of s is $\nu_{attr}(z_i)$ (i.e., $(s, \nu_{attr}(z_i)) \in A_{@a_i}$).
- $(T, \nu, s) \models \beta\langle f\rangle\langle\langle f'\rangle\rangle$ iff $(T, \nu, s) \models \beta$ and there is a set S of children of s such that $(T, \nu, S) \models f$ and a set S' of descendants of s such that $(T, \nu, S') \models f'$.
- $(T, \nu, \emptyset) \models \varepsilon$;

- $(T, \nu, \{s_1, \ldots, s_k\}) \models t_1\theta_1 t_2 \theta_2 \ldots \theta_{k-1} t_k$ iff (s_i, s_{i+1}) is in NS whenever θ_i is $\rightarrow$ and in NS^* whenever θ_i is $\rightarrow^*$, for each $i < k$, and $(T, \nu, s_i) \models t_i$ for all i.
- $(T, \nu, S) \models f_1 \| f_2$ iff $S = S_1 \cup S_2$ such that $(T, \nu, S_i) \models f_i$, for $i = 1, 2$.

Remark Note that the node s in the definition of $(T, \nu, s) \models t$ is superfluous since $s = \nu_{node}(x)$ for $t = \ell(x)[\ldots]\langle f \rangle \langle\langle f' \rangle\rangle$, but we prefer to make it explicit for notational convenience.

4.1 Classification of incomplete descriptions

There are three different groups of parameters that can vary as we define incomplete tree descriptions.

Node ids One possibility is to disregard them, as often done in the work on tree patterns [6, 8, 9, 14], i.e., assume that each node has a distinct variable for node id. In that case, we shall speak of *incomplete trees*.

At the opposite end, we have a model that corresponds to the DOM interface to XML, which assigns a constant id to each node [16, 18]. Such incomplete descriptions will be referred to as *incomplete DOM-trees*.

Structure Another parameter refers to how much of the structure of a document can be described: that is, the set of axes used (among $\downarrow, \downarrow^*, \rightarrow, \rightarrow^*$), whether the union operation $\|$ on forests is allowed and whether markings μ can be used in descriptions.

We shall precede the definition of a class of trees with this structural information.

Data values The third parameter refers to attribute values. Normally, we allow both constants and variables, i.e., an analog of naïve tables. But in some cases we look at purely structural information, with no data values. Then we talk about trees *without attributes*.

Classes of incomplete descriptions will be referred to as

$$\textit{(structure)}\text{-incomplete} \left\{ \begin{array}{l} \text{tree} \\ \text{DOM-tree} \end{array} \right\}$$

(possibly *without attributes*), where structure is a subset of $\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \|, \mu$.

To reiterate, we concentrate on the following two classes of incomplete descriptions, in combination with various structural fragments, and both with and without attributes:

Incomplete trees In those node ids are all variables and all distinct. In fact we may just omit them, writing, for example, $r\langle a \rightarrow b \| c \rangle$ instead of the more formal $r(x_1)\langle a(x_2) \rightarrow b(x_3) \| c(x_4) \rangle$. The incomplete tree description essentially enforces a tree structure for such incomplete descriptions (except possibly markings conflicting with the rest of the description).

Incomplete DOM-trees In those each node has a constant node id, which must be explicitly listed. Then non-tree-shaped descriptions are possible, e.g. $a(\mathsf{i}_0)\langle b(\mathsf{i}_1)\langle a(\mathsf{i}_0)\rangle\rangle$ saying that i_1 is a child of i_0 and i_0 is a child of i_1.

Even assuming that we always have the child axis in descriptions, these parameters give rise to 2^7 cases. Of course we shall not be attempting to classify them all; rather, our goal is to understand which combinations of parameters give us good algorithms, and which naturally lead to intractability.

Remark The treatment of node ids need not be limited to the two extremes: all distinct variable ids, or all constant ids. The model in which all ids are variables but some could be the same subsumes tree patterns of [8, 9]. Note though that most proofs of hardness results in [8, 9] are based on the assumption that variables can be repeated and thus do not apply to incomplete (DOM-)trees.

Remark The model of [3], introduced in the context of active documents, is incompatible with ours. It deals with $(\downarrow, \|)$-incomplete DOM-trees in our classification, in which at most one attribute per node is permitted, but it does not allow nulls (and, in particular, cannot model naïve features such as our model). But the model of [3] handles types of incompleteness that we do not deal with. It assumes that a portion of the document is always known (and increases as more queries are posed), and the rest is coded by a restricted form of DTDs that disregard the sibling-ordering. It can be potentially captured by an extension of our model by an analog of conditional tables, but this is beyond the scope of this work.

4.2 Relational representations

Just as complete XML trees, incomplete trees have a natural relational representation. We shall present it now, and show that the semantics of incompleteness can be described in terms of homomorphisms between relational representations of incomplete and complete trees.

With each incomplete tree description t with labels from $\Sigma \subset \mathit{Labels}$ and attributes from $A \subset \mathit{Attr}$, we associate a relational structure $\underline{rel}(t)$ of vocabulary $\tau_{\Sigma,A}$. These will be two-sorted structures, whose active domains are subsets of $\mathcal{I} \cup \mathcal{V}_{\rm node}$ and of $\mathcal{D} \cup \mathcal{V}_{\rm attr}$, defined as unions of active domains of all node descriptions. For a node description $\beta = \ell^\mu(x)[@a_1 = z_1, \ldots, @a_m = z_m]$, we let $adom_{node}(\beta) = \{x\}$ and $adom_{attr}(\beta) = \{z_1, \ldots, z_m\}$.

For a tree (t) or forest (f) description, $\underline{rel}(t)$ or $\underline{rel}(f)$ is a two-sorted structure over domains $adom_{node}(t)$ and $adom_{attr}(t)$ (or f), defined inductively (together with the notion of root nodes) as follows:

1. If $t = \beta\langle f \rangle \langle\langle f' \rangle\rangle$, where $\beta = \ell^\mu(x)[(@a_i = z_i)_{i=1}^m]$, then $\underline{rel}(t)$ includes the union of $\underline{rel}(f)$ and $\underline{rel}(f')$ and in addition it has the following: all tuples $A_{@a_i}(x, z_i)$, all tuples $E(x, y)$, where y is a root node of f, all tuples $E^*(x, y')$, where y' is a root node of f'. Furthermore, x is added to P_ℓ if $\ell \neq \_$ and to unary relations $Root$, $Leaf$, FC, LC according to the markings μ. The root node of t is x.
2. For $f = \varepsilon$, all the relations are empty;
3. For $f = t_1\ \theta_1\ \ldots\ \theta_{k-1}\ t_k$, where $x_1, \ldots, x_k$ are the root nodes of $t_1, \ldots, t_k$, we let $\underline{rel}(f)$ be the union of

all $\underline{rel}(t_i)$s, and in addition we put (x_i, x_{i+1}) in NS or NS^*, depending on whether θ_i is $\rightarrow$ or $\rightarrow^*$. We call x_i's the root nodes of f.

4. $\underline{rel}(f \| f')$ is the union of $\underline{rel}(f)$ and $\underline{rel}(f')$. We also define the root nodes of $f \| f'$ as the union of the root nodes of f and f'.

Let $h_1 : \mathcal{V}_{\text{node}} \cup \mathcal{I} \rightarrow \mathcal{V}_{\text{node}} \cup \mathcal{I}$ and $h_2 : \mathcal{V}_{\text{attr}} \cup \mathcal{D} \rightarrow \mathcal{V}_{\text{attr}} \cup \mathcal{D}$ be mappings that are constant on $\mathcal{I}$ and $\mathcal{D}$. Then $\bar{h} = (h_1, h_2)$ is a *homomorphism* of two relational structures T_1 and T_2 of vocabularies τ_{Σ_1, A_1} and τ_{Σ_2, A_2}, with $\Sigma_1 \subseteq \Sigma_2$ and $A_1 \subseteq A_2$, if for every tuple $\bar{x}$ in a relation R of τ_{Σ_1, A_1} in T_1, the tuple $\bar{h}(\bar{x})$ is in the relation R in T_2 (which must be present in T_2 since $\tau_{\Sigma_1, A_1} \subseteq \tau_{\Sigma_2, A_2}$). Of course $\bar{h}(x)$ refers to $h_1(x)$ if $x \in \mathcal{V}_{\text{node}} \cup \mathcal{I}$ and to $h_2(x)$ if $x \in \mathcal{V}_{\text{attr}} \cup \mathcal{D}$.

Proposition 4.1. *$T \in Rep(t)$ iff there is a homomorphism $\bar{h} : \underline{rel}(t) \rightarrow T$.*

5. Basic computational problems

The standard computational problems studied in connection with incomplete information in relational databases are membership (whether a complete database can be represented by an incomplete description) and query answering. Others are variations of these two (e.g., containment $Rep(R) \subseteq Rep(R')$ can be viewed as a special case of query answering). In the case of XML we have an additional problem that needs to be addressed – consistency. Due to complicated descriptions of XML documents, it is possible to provide inconsistent specifications. This is a well-recognized phenomenon, and there are many results on consistency and satisfiability for XML schemas, constraints, patterns, and queries [5, 7, 8, 9]. We already saw some examples of inconsistent descriptions: for example, under the DOM model, we can say that nodes with ids i_1 and i_2 are connected by the child edge in both directions, which is inconsistent with any tree description. With markings too inconsistency is possible, e.g., $a\langle b^{root}\rangle$ saying that a child node is marked *root*. Presence of DTDs also may lead to inconsistency. Consider a DTD $r \rightarrow bb$; $b \rightarrow \varepsilon$, where b has an attribute $@a$, and a description $r\langle b[@a = c_1] \rightarrow b[@a = c_2] \parallel b[@a = z] \rightarrow b[@a = z]\rangle$, where $c_1 \neq c_2$ are two constants from $\mathcal{D}$. This is inconsistent with the DTD.

5.1 Consistency of incomplete descriptions

We consider the following problem:

PROBLEM:	CONSISTENCY
INPUT:	an incomplete description t
QUESTION:	is $Rep(t) \neq \emptyset$?

We also look at a variation with a fixed DTD d: the problem CONSISTENCY(d) asks whether $Rep_d(t) = Rep(t) \cap \{T \mid T \models d\}$ is nonempty.

First, we get an upper bound on the complexity.

Theorem 5.1. *Both* CONSISTENCY *and* CONSISTENCY*(d) are in NP. In fact, even if both t and d are given as inputs, checking whether $Rep_d(t) \neq \emptyset$ can be done in NP.*

We want to understand which features lead to NP-hardness, and which ones allow efficient algorithms.

The consistency problem appears related to several well-studied problems – chase-based tools, constraint satisfaction, automata on trees – but techniques from those areas do not seem to provide us with a way of getting efficient algorithms. For example, some of the algorithmic techniques for checking consistency have a feel of a chase procedure that completes the relational representation $\underline{rel}(t)$. But we cannot apply chase 'as is'. The main constraint – that the resulting structure be a tree – is not even first-order expressible. Also, some constraints are disjunctive in nature: e.g., for two children s and s' of the same node, either $s \rightarrow^* s'$ or $s' \rightarrow^* s$ holds. While chase with disjunctive constraints has been considered [15], it generally yields intractable upper bounds, which we already have from Theorem 5.1.

By Proposition 4.1, consistency can be viewed as the existence of a homomorphism from $\underline{rel}(t)$ into some structure T. This suggests applicability of constraint satisfaction tools, since tractable restrictions are very well understood (cf. [22]). But Theorem 5.1 only provides an upper bound on the size of T. In particular, it is possible for T to have both long branches and high branching degree, and hence Theorem 5.1 does not give a construction for a polysize T to reduce consistency to constraint satisfaction. The problem with using automata is that data values come from an infinite domain. While some automata models have been developed for them [25, 26], they do not lead to efficient algorithms for expressive problems such as those we consider here.

We start our investigation with incomplete trees. The first result is about the consistency problem without DTDs. For incomplete trees, only markings can lead to inconsistency. For descriptions with markings we provide a full classification of tractable cases.

Theorem 5.2. *Each $(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \|)$-incomplete tree (i.e., an incomplete tree without markings) is consistent.*

With markings, CONSISTENCY *is*

- *NP-complete for the fragments $(\downarrow, \rightarrow, \star, fc, lc)$ and $(\downarrow, \downarrow^*, \star, fc, lc, leaf)$, where $\star$ is $\rightarrow^*$ or $\|$;*
- *PTIME for all other fragments containing $\downarrow$.*

With DTDs, we have intractability already for simple descriptions of incomplete trees:

Theorem 5.3. *There exist DTDs d_1, d_2, d_3 such that:*

- CONSISTENCY*(d_1) is NP-complete for $(\downarrow, \|)$-incomplete trees.*
- CONSISTENCY*(d_2) is NP-complete for $(\downarrow, \rightarrow, \|)$-incomplete trees, even without attributes.*
- CONSISTENCY*(d_3) is NP-complete for $(\downarrow, \downarrow^*, \|)$-incomplete trees, even without attributes.*

We sketch the proof of the first item, to indicate how DTDs and null values of attributes combine to lead to hardness. The DTD d_1 has three rules: $r \to CCC$; $C \to DD$; $D \to \varepsilon$ with C and D having one attribute $@c$ (color). For a graph G with n nodes and m edges $e_1, \ldots, e_m$, define n variables x_i and m trees t_e for each edge e between the ith and the jth node: $t_e = C[@c = x_i]\langle D[@c = x_j]\rangle$. We then fix three colors r, g, b, and for each color c' define $t_{c'} = C[@c = c']\langle D[@c = c_1] \| D[@c = c_2]\rangle$ where c_1, c_2 are the two colors different from c'. Finally let $t_G = r\langle t_r \| t_g \| t_b \| t_{e_1} \| \ldots \| t_{e_m}\rangle$. It is easy to see that t_G and d_1 are consistent iff G is 3-colorable.

The key feature in the above sketch is that in incomplete trees t, different subformulae can represent the same subtree of a tree in $Rep(t)$. In particular, in the proof we need to collapse multiple subtrees t_{e_i} to those describing their colorings, i.e., t_r, t_g, t_b.

This is impossible to do in the case of DOM-trees, where unique ids associated with node descriptions make such 'collapse' impossible. We now turn to incomplete DOM-trees, and show that the presence of unique ids lowers the complexity of consistency, even in the presence of DTDs. However, it makes the proofs significantly harder. First, we show:

Theorem 5.4. CONSISTENCY *can be solved in PTIME for incomplete DOM-trees.*

We can even get tractability for consistency with DTDs if we restrict to $\downarrow^*$*-free* incomplete DOM-trees that do not use the descendant relation (i.e. $\langle\!\langle \cdot \rangle\!\rangle$ cannot be used in incomplete tree descriptions).

Theorem 5.5. *For each fixed DTD d,* CONSISTENCY*(d) is solvable in* PTIME *for $\downarrow^*$-free incomplete DOM-trees.*

However, the combined complexity (when the DTD is not fixed) is intractable:

Proposition 5.6. *The problem of checking, for a DTD d and an incomplete DOM-tree t, whether $Rep_d(t)$ is nonempty, is NP-complete.*

In fact, to get NP-hardness, it suffices to look at $(\downarrow, \|)$-incomplete DOM-trees without attributes and DTDs in which every regular expression defines a finite language.

5.2 Membership test

We now consider the next basic computational problem related to incomplete information:

PROBLEM:	MEMBERSHIP
INPUT:	an incomplete tree t, a complete tree T
QUESTION:	is $T \in Rep(t)$?

To test whether $T \in Rep(t)$ one just guesses a homomorphism $h : \underline{re\ell}(t) \to T$; hence MEMBERSHIP is in NP.

Recall what is known in the relational case. The problem of checking whether R' is in $Rep(R)$ is NP-complete if R is a naïve table, and in PTIME if R is a Codd table, i.e. each variable occurs exactly once in it. We shall prove an analog of this result. We say that t is an incomplete *Codd* tree if every variable from $\mathcal{V}_{\rm attr}$ occurs at most once in t.

Theorem 5.7.
- MEMBERSHIP *for $(\downarrow, \|)$-incomplete trees is* NP*-complete.*
- *For incomplete Codd trees,* MEMBERSHIP *is solvable in PTIME.*
- *For incomplete DOM-trees,* MEMBERSHIP *is solvable in PTIME.*

The proof for the Codd case is quite different from the relational technique [2], which is based on bipartite graph matching; instead we use a technique inspired by CTL model-checking to see if $T \in Rep(t)$.

6. Query answering

For relational databases, we know that unions of conjunctive queries can be efficiently evaluated over databases with nulls. One just uses the naïve evaluation, which treats nulls as if they were simply different elements of the domain, and then discards tuples that contain nulls from the output. Naïve evaluation correctly computes certain answers [20] and has the same complexity as the usual conjunctive query evaluation. Once negation is added to queries, or the representation mechanism changes, the complexity quickly rises [2].

We want to find classes of queries and incomplete representations that admit tractable query evaluation for computing certain answers. The first obstacle is that for XML queries that produce trees as outputs, the notion of certain answers is far from clear. So for now, since our goal is to broadly outline the tractability boundary, we look at XML queries that produce tuples of values (this, of course, includes Boolean queries). Once we define a query language, we present a few results that rule out several features as immediately leading to intractability. Then we define a class of *rigid* incomplete trees and show that a natural analog of unions of conjunctive queries admits tractable naïve evaluation over them. We conclude by showing that over DOM-trees, the picture is rather different.

6.1 A simple query language

We shall use queries whose free variables range over the domain of attribute values, and thus their results are usual relations. We start with conjunctive queries over trees. These are essentially standard (see, e.g., [8, 19]). We express them in our syntax for incomplete trees, and add existential quantification over variables from $\mathcal{V}_{\rm attr}$. That is, conjunctive queries $\mathcal{CQ}$ are of the form $q(\bar{x}) = \exists \bar{y}\ t_q(\bar{x}, \bar{y})$, where t_q is an incomplete tree, and $\bar{x}, \bar{y}$ list variables from $\mathcal{V}_{\rm attr}$. Their semantics on complete trees T is defined as

$$q(T) = \left\{ \nu_{attr}(\bar{x}) \;\middle|\; \begin{array}{l} (T, \nu, s) \models t_q \text{ for some node } s \\ \text{and valuation } \nu = (\nu_{node}, \nu_{attr}) \end{array} \right\}.$$

Recall that in incomplete trees we omit node variables for notational convenience; the semantics of $q(\bar{x})$ of course assumes existential quantification over all node variables.

As our language $\mathcal{UCQ}$ we take unions of conjunctive queries:

$$q_1(\bar{x}) \cup \ldots \cup q_k(\bar{x})$$

For (unions of) conjunctive queries, we use the notation $\mathcal{UCQ}(structure)$ or $\mathcal{CQ}(structure)$, where *structure* refers to the structural information used in incomplete trees t_q. For example, $\exists y\, r\langle \ell_1[@a = x] \to \ell_2[@b = y]\rangle$ is a $\mathcal{CQ}(\downarrow, \to)$-query that returns values of the $@a$ attribute of ℓ_1-children of r that have an ℓ_2-labeled next sibling with a $@b$ attribute.

For $\mathcal{UCQ}$ queries we can define the notion of certain answers since these queries produce relations:

$$certain(q, t) \;=\; \bigcap\{q(T) \mid T \in Rep(t)\}.$$

The main computational problem we consider here is:

PROBLEM:	QUERYANSWERING(q)
INPUT:	an incomplete tree description t, a tuple $\bar{a}$
QUESTION:	is $\bar{a} \in certain(q, t)$?

We also define $certain_d(q, t)$ as $\bigcap\{q(T) \mid T \models d \text{ and } T \in Rep(t)\}$, and a problem QUERYANSWERING(q, d) (query answering with DTDs) where the question is whether $\bar{a} \in certain_d(q, t)$.

A fragment of the language, namely $\mathcal{UCQ}(\downarrow, \downarrow^*, \|)$, was considered in the study of query answering in XML data exchange [6]. We first provide an upper bound on the complexity of query answering. We show that a counterexample to $\bar{a} \in certain(q, t)$, i.e., a complete tree T so that $\bar{a} \notin q(T)$ can be chosen to be of polynomial size in t and $\bar{a}$. Thus,

Theorem 6.1. *Both* QUERYANSWERING(q) *and* QUERYANSWERING(q, d) *are in coNP for all* $q \in \mathcal{UCQ}$ *and all* d.

6.2 Intractable cases of query answering

We now show that query answering could be intractable, even for unions of conjunctive queries. This contrasts sharply with the relational case, where all unions of conjunctive queries can be evaluated in PTIME.

We can obtain several intractability results by using hardness results for consistency. Note that if we have a class of incomplete trees over which CONSISTENCY is NP-hard, and a class of queries that includes a query false in all trees, then over these classes of incomplete trees and queries, QUERYANSWERING is coNP-hard. This follows from the fact that $certain(false, t) = true$ iff $Rep(t) = \emptyset$.

With both DTDs and markings, it is easy to write unsatisfiable queries (e.g., $r\langle a\rangle$, where a cannot appear under the root according to the DTD, or $\_\langle \_^{lc} \to \_^{fc}\rangle$ without DTDs). Hence, we have

Corollary 6.2.
- *There exists a DTD* d *and a query* $q \in \mathcal{CQ}(\downarrow)$ *such that* QUERYANSWERING(q, d) *is coNP-complete over* $(\downarrow, \|)$*-incomplete trees.*
- *For the classes of* $(\downarrow, \to, \star, \mu)$*- and* $(\downarrow, \downarrow^*, \star, \mu)$*-incomplete trees (where* $\star$ *is either* $\|$ *or* $\to^*$*), there exist queries* q *that use markings such that* QUERYANSWERING(q) *is coNP-complete.*

Thus, having DTDs, or markings in trees and queries, immediately gives us coNP-hardness of query answering. But coNP-hardness can occur even without DTDs and markings.

Theorem 6.3. *There is a query* $q \in \mathcal{CQ}(\downarrow, \to)$ *such that* QUERYANSWERING(q) *is coNP-complete over* $(\downarrow, \|)$*-incomplete trees.*

We can also get coNP-hardness for $(\downarrow, \downarrow^*, \|, \mu)$-incomplete trees without attributes and a fixed $\mathcal{CQ}(\downarrow)$ query. But so far these results do not say much about the transitive-closure axes in incomplete trees. We now show that with $\downarrow^*$ or $\to^*$, answering unions of conjunctive queries is coNP-hard.

Theorem 6.4.
- *There is a query* $q \in \mathcal{UCQ}(\downarrow, \|)$ *such that* QUERYANSWERING(q) *over* $(\downarrow, \to, \downarrow^*)$*-incomplete trees is coNP-complete.*
- *There is a query* $q \in \mathcal{UCQ}(\downarrow, \to, \to^*)$ *such that* QUERYANSWERING(q) *over* $(\downarrow, \to, \to^*)$*-incomplete trees is coNP-complete.*
- *Both results hold for incomplete DOM-trees as well.*

In the presence of DTDs, we have cases of coNP-hard query answering for very simple queries over incomplete DOM-trees, as the following result shows.

Proposition 6.5. *There exists a DTD* d *and a query* $q \in \mathcal{CQ}(\downarrow, \|)$ *such that* QUERYANSWERING(q, d) *is coNP-complete for* $(\downarrow, \downarrow^*, \|)$*-incomplete DOM-trees.*

6.3 Tractable case: rigid incomplete trees

So far, we have seen that the following features quickly lead to the intractability of query answering for (unions of) conjunctive queries:

1. DTDs; and
2. structural information: transitive-closure axes $\downarrow^*$ and $\to^*$; union; and markings.

We now exclude these features and obtain a tractable class with respect to query answering. That is, we restrict ourselves to incomplete trees that have neither the transitive closures of axes nor union $\|$ nor markings. We call them *rigid incomplete trees*; they are defined by the grammar:

$$\begin{array}{rcl} t & := & \beta\langle f\rangle \\ f & := & \varepsilon \mid t \to f \end{array} \qquad (3)$$

where node ids are all distinct variables, and markings are not allowed in node descriptions β. This definition mimics (1)

except that node descriptions use variables instead of node ids, and may have nulls as values of attributes and wildcard as labels. Note that each rigid incomplete tree t is consistent.

Our goal is to show that an analog of naïve evaluation will compute certain answers for unions of conjunctive queries over such incomplete trees. We define *naïve-evaluation* as follows. First, each conjunctive query $q(\bar{x}) = \exists \bar{y}\ t_q(\bar{x}, \bar{y})$ is turned into a usual relational conjunctive query by taking $\underline{rel}(t_q)$ and viewing it as a tableau for a query, where $\bar{x}$ are distinguished variables. We shall denote this query by $\underline{rel}(q)_{\bar{x}}$. We then consider the input t, and transform $\underline{rel}(t)$ into $\underline{rel^*}(t)$ by adding transitive closures of E and NS.

Then $naïve\_eval(q, t)$ is the result of evaluating the relational conjunctive query $\underline{rel}(q)_{\bar{x}}$ on the relational database $\underline{rel^*}(t)$ naïvely, and then dropping tuples with nulls. We refer to the result as $naïve\_eval(q, t)$. This extends to unions of conjunctive queries, simply by taking $\bigcup_i naïve\_eval(q_i, t)$.

We illustrate this by an example. Suppose we have a query

$$q(x) \ = \ \exists y\ r(n_0)\langle \ell(n_1)[@a = x] \rightarrow^* \_(n_2)[@b = y]\rangle$$

asking for values of the $@a$-attributes of ℓ-children of r-nodes that have a younger sibling with the $@b$-attribute. In the tableau, we shall have tuples (n_0, n_1) and (n_0, n_2) for E, one tuple (n_1, n_2) for NS^*, node n_0 is in P_r and n_1 is in P_ℓ, and pairs $(n_1, x), (n_2, y)$ are in $A_{@a}$ and $A_{@b}$, resp. Since x is the only distinguished variable, this tableau generates a relational conjunctive query $q'(x)$:

$$\exists n_0, n_1, n_2, y\ E(n_0, n_1) \wedge E(n_0, n_2) \wedge NS^*(n_1, n_2) \wedge P_r(n_0) \wedge P_\ell(n_1) \wedge A_{@a}(n_1, x) \wedge A_{@b}(n_2, y).$$

Now suppose we have an incomplete tree

$$t \ = \ r\langle \ell[@a = 1] \rightarrow \ell[@a = u] \rightarrow \ell'[@b = v]\rangle$$

By introducing node variables n'_0 for the root and n'_1, n'_2, n'_3 for three children of the root, we create $\underline{rel}(t)$, which has pairs (n'_1, n'_2) and (n'_2, n'_3) in NS. By computing $\underline{rel^*}(t)$ we put those pairs, as well as (n'_i, n'_i) and (n'_1, n'_3) in NS^*. Evaluating q' naïvely over $\underline{rel^*}(t)$ yields $\{1, u\}$. Eliminating null u, we conclude that $naïve\_eval(q, t) = \{1\}$. In this case, it is easy to see that $\{1\}$ is the set of certain answers. This correspondence works for all rigid incomplete trees.

Theorem 6.6. *Let t be a rigid incomplete tree, and q a query from $\mathcal{UCQ}$ that does not use markings. Then*

$$certain(q, t) \ = \ naïve\_eval(q, t).$$

In particular, evaluating no-marking queries over rigid incomplete trees has DLOGSPACE data complexity.

Proof sketch. This follows from the observation that by rigidity, node variables can be replaced by distinct node ids without changing the semantics of $\mathcal{UCQ}$ queries. This replacement ensures that $Rep(\underline{rel}(t))$, under the closed world assumption, only contains representation of trees. This reduces the problem of finding $certain(q, t)$ to the problem of finding certain answers to relational unions of conjunctive queries under the closed world assumption. This problem, by [20], is solvable by the naïve relational evaluation, which could be seen to coincide with $naïve\_eval(q, t)$.

We have seen in Section 6.2 that the tractability of query answering over the class of rigid trees does not withstand the additions of union, descendant, younger-sibling, or markings. It is also easy to construct examples showing that the naïve evaluation fails with these structural additions. For example, consider $t = r\langle a \| b\rangle$ and $q = r\langle a \rightarrow^* b\rangle \cup r\langle b \rightarrow^* a\rangle$. We know that $certain(q, t) = true$ but $naïve\_eval(q, t)$ produces *false*. To see why Theorem 6.6 restricts to queries without markings, consider a Boolean query $r\langle \_^{fc}\rangle$ and $t = r\langle a\rangle$. Again naïve evaluation produces *false* but the query is true with certainty.

6.4 Query answering: node ids make a difference

Theorem 6.6 applies to *incomplete rigid DOM-trees* (defined just as rigid incomplete trees, except that node ids are now all constants). This is because the rigid structure ensures that every homomorphism $h : \underline{rel}(t) \rightarrow T$ is one-to-one. We saw that by adding union or descendant to rigid incomplete trees, we get coNP-hardness of query answering. But, similarly to the cases of consistency and membership, we can find classes of incomplete DOM-trees that have tractable query evaluation while for the corresponding class of incomplete trees, it is coNP-hard. We do it by adding union to the class of rigid incomplete trees.

Theorem 6.7. *There is a class $\mathcal{Q} \subseteq \mathcal{CQ}(\downarrow, \rightarrow, \|)$ of queries so that:*

- *For every query $q \in \mathcal{Q}$, QUERYANSWERING(q) over $(\downarrow, \rightarrow, \|)$-incomplete DOM-trees can be solved in PTIME.*
- *There is a query $q \in \mathcal{Q}$ such that QUERYANSWERING(q) over $(\downarrow, \rightarrow, \|)$-incomplete trees is coNP-hard.*

Note that the PTIME algorithm is not based on the naïve evaluation (in fact it is much more complicated, even for very simple queries). To see why, consider an incomplete DOM-tree $t = r(\mathsf{i}_0)\langle a(\mathsf{i}_1) \| a(\mathsf{i}_2)\rangle$ and a query $q = r\langle \_ \rightarrow \_\rangle$. Since $\mathsf{i}_1 \neq \mathsf{i}_2$, we know that r has at least two children, and thus $certain(q, t) = true$, but the naïve evaluation returns false. Similarly, if $t' = r(\mathsf{i}_0)\langle (a(\mathsf{i}_1) \rightarrow b(\mathsf{i}_2)) \| (a(\mathsf{i}_3) \rightarrow b(\mathsf{i}_4))\rangle$, then for the query $q' = r\langle b \rightarrow \_ \rightarrow \_\rangle$, the certain answers are true, but the naïve evaluation returns false. Note that this is caused by node ids, and the knowledge that nodes are distinct: if we replace node ids from t and t' with variables, then both $certain(q, t)$ and $certain(q', t')$ would become false.

7. Overview of tractability restrictions

We now summarize what we have learned about various models of incompleteness in XML. The key parameters were:

1. the presence of schema information;
2. the presence of markings in node descriptions;
3. structural information (i.e., $\downarrow, \downarrow^*, \rightarrow, \rightarrow^*$ and $\|$); and
4. the presence of node ids.

We have seen that the presence of DTDs, and the presence of markings, makes everything significantly more complicated. Even the simplest cases of consistency and query answering become intractable with DTDs and with markings. So it is natural to suggest that key computational problems for XML with incomplete information be considered without restriction to specific schema information.

The lack of complete structural information is another big obstacle to tractability. Introducing structural uncertainty such as transitive-closure axes and union quickly leads to intractability of both consistency and query answering (Theorems 5.2, 6.3, and 6.4). This happens even for unions of conjunctive queries – the class that is well-behaved with respect to incomplete relational databases.

To achieve tractable query answering over documents with nulls, one needs to restrict not only the class of queries to unions of conjunctive queries but also the class of structural document descriptions so that a portion of a tree is fully described with the child and next-sibling relations. These are rigid incomplete trees: incompleteness only occurs in attribute values and labelings. Then an analog of relational naïve evaluation finds certain answers.

8. Future work

There are several possible directions. First, we have only looked at models based on the open world assumption. In the relational case, both open and closed world assumptions (OWA and CWA) are considered, and in many cases the behavior under the CWA is quite different [28]. Many results presented here work for both OWA and CWA but not all. And some existing models (e.g., [3]), fall between CWA and OWA. We also would like to look at analogs of more expressive representations, such as conditional tables [4, 20] or relational representation techniques such as those in [24] to overcome intractability.

Our understanding of models with node ids is not as complete as our understanding of models without ids. And yet this is a fascinating class, because we saw that tractability boundaries can be pushed much further for it.

We would like to address a number of traditional issues related to incomplete information in the context. One example is constraints over documents with incomplete information. It is expected that in the most general form query answering and consistency analysis will be undecidable (cf. [5, 10]) but one should expect to find reasonable restrictions for decidability and tractability. Another example is using incomplete information in data integration and exchange tasks.

Acknowledgment This work started when the third and the fourth authors were at the University of Edinburgh. We acknowledge support by EPSRC grants E005039 and F028288, EU grant MEXC-CT-2005-024502, FONDECYT grant 11080011, and MIUR FIRB 2005 project TOCAI.IT.

9. References

[1] S. Abiteboul, O. Duschka. Complexity of answering queries using materialized views. In *PODS 1998*, pages 254–263.

[2] S. Abiteboul, P. Kanellakis, G. Grahne. On the representation and querying of sets of possible worlds. *TCS* 78 (1991), 158–187.

[3] S. Abiteboul, L. Segoufin, V. Vianu. Representing and querying XML with incomplete information. *ACM TODS*, 31 (2006), 208–254.

[4] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases*, Addison Wesley, 1995.

[5] M. Arenas, W. Fan, L. Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38 (2008), 841–880.

[6] M. Arenas, L. Libkin. XML data exchange: consistency and query answering. *J. ACM* 55(2): (2008).

[7] M. Benedikt, W. Fan, F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM* 55(2): (2008).

[8] H. Björklund, W. Martens, T. Schwentick. Conjunctive query containment over trees. *DBPL'07*, pages 66–80.

[9] H. Björklund, W. Martens, T. Schwentick. Optimizing conjunctive queries over trees using schema information. *MFCS'08*, pages 132–143.

[10] A. Calì, D. Lembo, R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. *PODS'03*, pages 260-271.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini. Semi-structured data with constraints and incomplete information. In *Description Logics*, 1998.

[12] D. Calvanese, G. De Giacomo, M. Lenzerini. Representing and reasoning on XML documents: a description logic approach. *J. Log. Comput.* 9 (1999), 295–318.

[13] S. Cohen, B. Kimelfeld, Y. Sagiv. Incorporating constraints in probabilistic XML. In *PODS'08*, pages 109–118.

[14] C. David. Complexity of data tree patterns over XML documents. In *MFCS'08*, pages 278–289.

[15] A. Deutsch, V. Tannen. Reformulation of XML queries and constraints. In *ICDT'03*, pages 225–241.

[16] Document Object Model (DOM). W3C Recommendation, April 2004. http://www.w3.org/TR/DOM-Level-3-Core.

[17] R. Fagin, Ph. Kolaitis, R. Miller, L. Popa. Data exchange: semantics and query answering. *TCS* 336(1): 89–124 (2005).

[18] P. Gardner, G. Smith, M. Wheelhouse, U. Zarfaty. Local Hoare reasoning about DOM. In *PODS'08*, pages 261–270.

[19] G. Gottlob, C. Koch, K. Schulz. Conjunctive queries over trees. *J. ACM* 53 (2006), 238–272.

[20] T. Imielinski, W. Lipski. Incomplete information in relational databases. *J. ACM* 31 (1984), 761–791.

[21] Y. Kanza, W. Nutt, Y. Sagiv. Querying incomplete information in semistructured data. *JCSS* 64 (2002), 655–693.

[22] P. Kolaitis and M. Vardi. A logical approach to constraint satisfaction. In *Finite Model Theory and its Applications*, Springer 2007, pages 339–370.

[23] M. Lenzerini. Data integration: a theoretical perspective. In *PODS'02*, pages 233–246.

[24] D. Olteanu, C. Koch, L. Antova. World-set decompositions: expressiveness and efficient algorithms. *TCS* 403 (2008), 265–284.

[25] T. Schwentick. A little bit infinite? On adding data to finitely labelled structures. In *STACS'08*.

[26] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL'06*, pages 41–57.

[27] P. Senellart, S. Abiteboul. On the complexity of managing probabilistic XML data. In *PODS'07*, pages 283–292.

[28] M. Vardi. Querying logical databases. *JCSS* 33 (1986), 142–160.