On the Complexity of Query Answering over Incomplete XML Documents

Amélie Gheerbrant
University of Edinburgh
agheerbr@inf.ed.ac.uk

Leonid Libkin
University of Edinburgh
libkin@inf.ed.ac.uk

Tony Tan
University of Edinburgh
ttan@inf.ed.ac.uk

ABSTRACT

Previous studies of incomplete XML documents have identified three main sources of incompleteness – in structural information, data values, and labeling – and addressed data complexity of answering analogs of unions of conjunctive queries under the open world assumption. It is known that structural incompleteness leads to intractability, while incompleteness in data values and labeling still permits efficient computation of certain answers.

The goal of this paper is to provide a complete picture of the complexity of query answering over incomplete XML documents. We look at more expressive languages, at other semantic assumptions, and at both data and combined complexity of query answering, to see whether some well-behaving tractable classes have been missed. To incorporate non-positive features into query languages, we look at gentle ways of introducing negation via inequalities and/or Boolean combinations of positive queries, as well as the analog of relational calculus. We also look at the closed world assumption which, due to the hierarchical structure of XML, has two variations. For all combinations of languages and semantics of incompleteness we determine data and combined complexity of computing certain answers. We show that structural incompleteness leads to intractability under all assumptions, while by dropping it we can recover efficient evaluation algorithms for some queries that go beyond those previously studied.

Categories and Subject Descriptors

F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata*; H.2.1 [**Database Management**]: Logical Design—*Data Models*; H.2.3 [**Database management**]: Languages—*Query Languages*; I.7.2 [**Document and Text Processing**]: Document Preparation—*Markup languages*

General Terms

Algorithms, Theory, Languages

Keywords

XML, incomplete information, query answering, certain answers, open-world semantics, closed-world semantics

1. INTRODUCTION

The need to deal with incomplete information has increased dramatically over the past decade, due to large amounts of data on the Web [1] (which tend to be more prone to errors than data stored in traditional relational DBMSs) as well as the need to move data between different applications as, for example, in data integration [21] and exchange [7] scenarios. Different types and models of incompleteness have been studied too, such as classical instances of missing information, uncertain databases [6], and probabilistic databases [27]. While most investigations deal with relational data, several recent papers have attempted to model and analyze incompleteness in XML. For example, [4] showed how to handle incompleteness in a dynamic setting when document's structure is revealed by a sequence of queries, while [12, 13] expressed incompleteness by means of description logic theories, and [20] surveyed incorporating probabilities into XML.

An attempt to reconstruct the classical relational theory of incompleteness [3, 19, 18] (in particular, issues such as semantics of incompleteness and the complexity of the main computational problems associated with it) was done in [9]. That paper presented a very general model of XML documents with incomplete information, and studied several computational problems, such as consistency of incomplete specifications, representability of complete documents by incomplete ones, and query answering.

ICDT 2012, March 26–30, 2012, Berlin, Germany.

In the model of [9], there are three main sources of incompleteness:

- Incompleteness at the level of data values. This is the same as in the relational case: nodes in XML trees may carry attribute values, and some of those values may not be known (i.e., nulls).
- Structural incompleteness. Some of the hierarchical structure of an XML document may not be known. For example, we may only know that a node w is a descendant of a node w' without knowing the precise path between them.
- Labeling incompleteness. Labels of some nodes may not be known and replaced by wildcards.

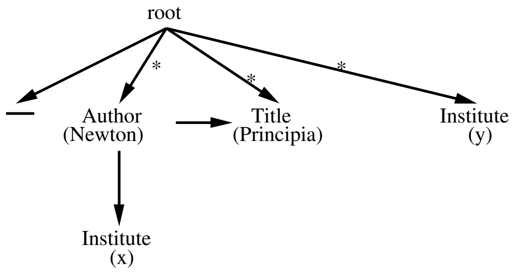

Figure 1: An incomplete XML document

Figure 1 gives an instance of an incomplete XML document. In this document we have two nodes labeled *Author* and *Title*, and we know their attribute values ("Newton" and "Principia"), as well as that the latter is next-sibling of the former. However, we do not know what the common parent of these nodes is: it may be the root, or another node, as the edges from the root to those nodes are labeled $*$, meaning descendant. We also have an *Institute* node, with an unknown attribute value y, as well as another *Institute* node which is a child of the *Author* node; its attribute is another unknown value x. Furthermore, there is a child of the root, but we know neither its label (indicated by wildcard _) nor its attribute value.

The semantics of such incomplete documents was given by homomorphisms into complete XML trees; this will be illustrated shortly and properly defined in the next section. Such semantics corresponds to *open world assumption* [19, 24], since it leaves a complete document open to adding new nodes.

As the class of queries to study, [9] used XML analogs of *unions of conjunctive queries*, or UCQs. In XML, conjunctive queries are normally modeled via *tree patterns* [8, 10, 17]. The choice of this class is not arbitrary: in the relational world, UCQs can be answered over incomplete tables by using the standard relational evaluation of queries; this is usually referred to as naïve evaluation [19]. In fact, this is the largest class of relational calculus queries for which such evaluation computes certain answers to queries [19, 22].

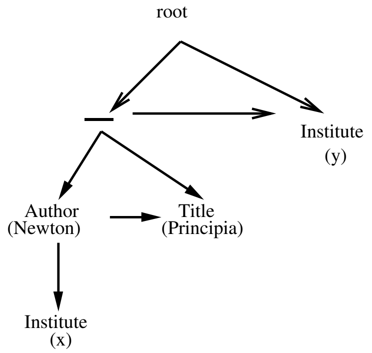

Figure 2: A rigid incomplete XML document

It was shown in [9] that data complexity of evaluating UCQs over XML documents is always in coNP, and is almost invariably coNP-complete as long as structural incompleteness is present. There are no known bounds on combined complexity; proofs in [9] only give nonelementary complexity, but we shall see that this can be significantly improved.

When the structure is fully known, i.e., only data values and labels of documents could be missing, evaluation of UCQs becomes tractable and can be done using naïve evaluation (such incomplete trees were called *rigid*; an example is shown Figure 2).

However, the picture is rather incomplete, and several natural questions arise.

1. Can the complexity of query evaluation over arbitrary incomplete documents be lowered by using a semantics based on *closed*, rather than open world assumption?
2. Can we extend the language of unions of conjunctive queries to obtain tractable query evaluation (under both open and closed world assumptions)?
3. What can be said about combined complexity of computing certain answers?

The main goal of the paper is to answer these questions. To do so, we need to explain what we mean by closed world assumption in XML, and define languages extending UCQs that we want to study. We now informally introduce these.

Closed world semantics in XML In the case of relations, closed world semantics is typically defined by having an *onto* (surjective) mapping (homomorphism) from an incomplete database to a complete one. We shall follow the same approach, but there is one issue that arises when we use transitive closures of axes, e.g., descendant relationships. Say we have just two nodes w and w', and we know that w' is a descendant of w. Any surjective mapping from such an incomplete description will produce a document with at most two nodes. Does it mean that under the closed world assumption we are then forced to reduce descendant relationship to child? On the one hand, this agrees with the intuition of not introducing new nodes; on the other hand, it seems to infer new child relationship which does not correspond to closed world assumption. So which alternative should we choose?

We believe that both in fact are reasonable, and we answer all the questions for both interpretations of closed world assumptions. More precisely, we consider three different semantics, which are shown in Figure 3, and are informally described below.

In Figure 3, we show documents that can be denoted by the incomplete document from Figure 1 under three different assumptions. Dashed lines show homomorphisms from the nodes of incomplete documents to the nodes of complete ones.

- Under the open world assumption (OWA), we permit any homomorphism (that preserves relationships between nodes and their attributes) from an incomplete document *into* a complete one.
- Under the weak closed world assumption (WCWA), we insist that the homomorphism be surjective (*onto*) except when nodes are in a relationship such as descendant: then we allow the introduction of new nodes, but only on a path between nodes that exist in an incomplete description. In the example in the picture, root is mapped to the root, and the *Institute* node with unknown value y into IAS. This lets us introduce a path to it that has a book node with a descendant author (Einstein); note however that we cannot introduce a node for book title (which was possible under OWA) as it will not be on the path to the IAS node.
- Under the strong closed world assumption (SCWA), we insist that the homomorphism be surjective.

Extensions of UCQs for XML Relational UCQs correspond to the positive fragment of relational algebra. Thus, extending them means introducing some form of negation. While we can just add it in an unrestricted way (like relational algebra does, to capture full power of first-order logic, FO), we look at intermediate ways of adding negation without immediately jumping all the way up to an XML analog of FO. There are two such standard ways:

- we can add inequalities $\neq$ as atomic formulae; or
- we can permit arbitrary Boolean combinations of previously defined queries.

For example, adding $\neq$ to UCQs we get a class UCQ$^{\neq}$ of unions of conjunctive queries with inequalities. By looking at Boolean combinations of those we get a class BCCQ of Boolean combination of conjunctive queries, i.e., the closure of conjunctive queries under operations $q \cap q'$, $q \cup q'$, and $q - q'$. Combining these, we shall obtain five languages that we study here: UCQ, UCQ$^{\neq}$, BCCQ, BCCQ$^{\neq}$, FO, and their XML analogs.

Results After formally defining XML with incomplete information and query languages, we review what is known for relational databases. In addition to recalling known (and sometimes folklore but not explicitly proven) results, we show a new result that for BCCQs, certain answers can be computed in polynomial time.

After that, we switch to XML. We show that for arbitrary incomplete documents that permit structural incompleteness, under all assumptions, and for all the languages, data complexity is intractable. We also establish combined complexity that in most cases is only marginally higher than data complexity (most commonly just one level up in the polynomial hierarchy).

We then switch to rigid trees. For them, we show that the complexity of all the query answering tasks is the same as for relations. While lower bounds can be inferred from the relational case, upper bounds require work as we are dealing with more complex tree structure (we know, for instance, that they need not hold in general with structural incompleteness).

In particular, over rigid trees, analogs of UCQs can be answered in polynomial time, by naïve evaluation, under both open and closed world assumptions, which implies efficient evaluation of queries. For analogs of BCCQs, we demonstrate a tractable query evaluation algorithm too, with combined complexity a bit higher (one level in the polynomial hierarchy) than for UCQs. We then conclude by discussing practical implications of these results.

Organization Incomplete XML documents are defined in Section 2; query answering over incomplete relational and XML databases is discussed in Section 3. In Section 4 we establish results on query answering over arbitrary incomplete trees, for all the languages considered here, and in Section 5 we do the same for rigid trees. Final remarks and conclusions are in Section 6.

2. INCOMPLETENESS IN XML

XML trees

To describe XML trees, we assume

- a countably infinite set $\mathcal{C}$ of possible data values (notation $\mathcal{C}$ stands for "constants", as opposed to nulls), and
- a countably infinite set $\mathcal{L}$ of node labels (element types). We shall normally denote labels by lowercase Greek letters.

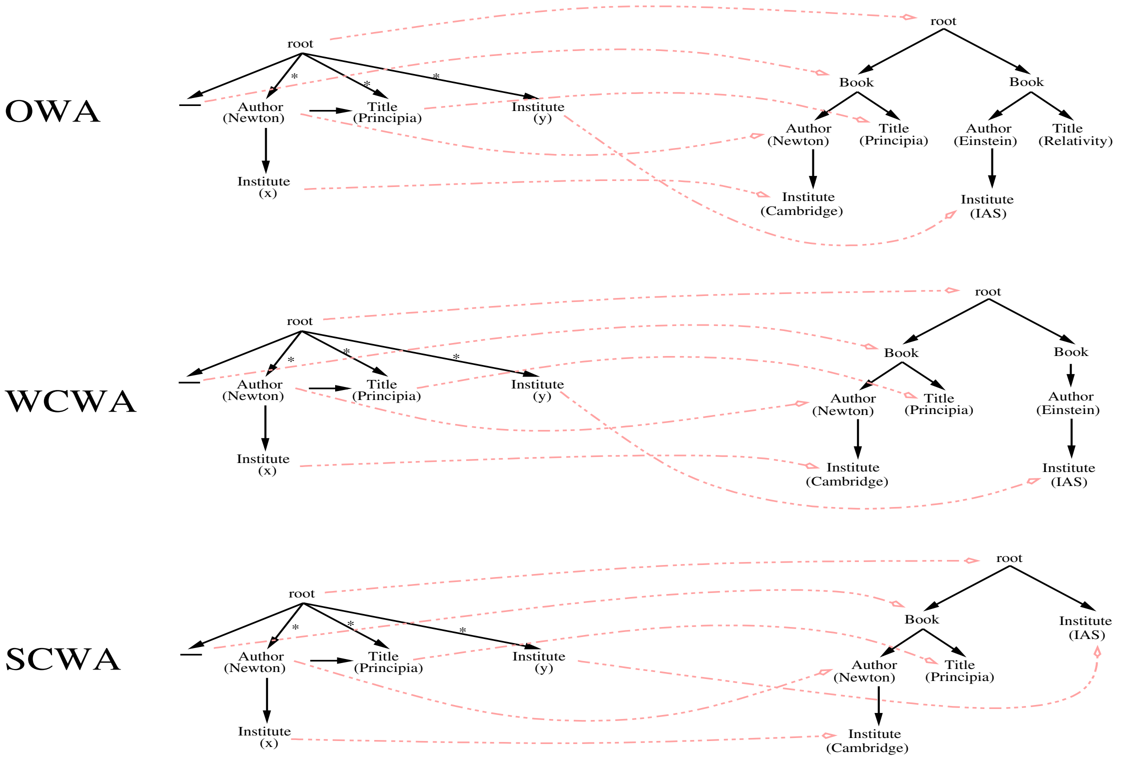

Figure 3: Open and (weak and strong) closed world semantics of incomplete XML

An XML tree over a finite alphabet $\Sigma \subset \mathcal{L}$ is a 2-sorted structure

$$T \;=\; \langle D, A, \downarrow, \rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle, \qquad (1)$$

where

- D is an unranked tree domain, i.e. a prefix-closed subset of $\mathbb{N}^*$ such that $w \cdot i \in D$ implies $w \cdot j \in D$ for $j < i$;
- $\downarrow$ and $\rightarrow$ are the child and next-sibling relations, for which we shall use, as is common, the infix notation: $w \downarrow w \cdot i$ whenever $w \cdot i \in D$, and $w \cdot i \rightarrow w \cdot (i + 1)$ whenever $w \cdot (i + 1) \in D$;
- each P_α is the set of elements of D labeled α (of course we require that these partition D);
- $A \subset \mathcal{C}$ is a finite set of data values; and
- $\rho : D \rightarrow \bigcup_{k\geq 0} A^k$ assigns to each node $w \in D$ a k-tuple of data values for some $k \geq 0$.

We refer to D as the *domain* of T, and denote it by $\text{dom}(T)$, and to A as the *active domain* (of data values) of T and denote it by $\text{adom}(T)$. We always assume that A has precisely the elements of $\mathcal{C}$ used in T, i.e., if $v \in A$ then there is a node w such that v occurs in $\rho(w)$.

We shall usually assume that for nodes w, w' with the same label, the arities of $\rho(w)$ and $\rho(w')$ are the same; this is customary for abstractions of XML documents although not technically necessary for our results.

We shall denote the transitive closure of $\downarrow$ by $\Downarrow$ and the transitive closure of $\rightarrow$ by $\Rightarrow$.

Incomplete XML trees

To define incomplete XML documents, we assume a countably infinite supply of null values (or variables) $\mathcal{V}$. Following [9], incompleteness can appear in documents in the following ways:

- Data-values incompleteness. This is the same as incompleteness in relational models: some data values could be replaced by nulls.
- Labeling incompleteness: instead of a known label, some nodes can be labeled with a wildcard.
- Structural incompleteness. Some of the structure of the document may not be known (e.g., we can use descendant edges in addition to child edges, or following-sibling edges instead of next-sibling).

This can be captured as follows. An *incomplete tree* over Σ is a 2-sorted structure

$$t \;=\; \langle N, V, \downarrow, \Downarrow, \rightarrow, \Rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle, \qquad (2)$$

where

- N is a set of nodes, and V is a set of values from $\mathcal{C} \cup \mathcal{V}$;
- $\downarrow, \Downarrow, \rightarrow, \Rightarrow$ are binary relations on N;
- P_α's are disjoint subsets of N; and
- ρ is a function from N to $\bigcup_{k \geq 0} V^k$.

As before, $\text{dom}(t)$ refers to N, and $\text{adom}(t)$ to V. We now distinguish between $\text{adom}_c(t)$, which refers to elements of $\mathcal{C}$ in $\text{adom}(t)$, and $\text{adom}_\perp(t)$, which refers to elements of $\mathcal{V}$ in $\text{adom}(t)$.

These represent incompleteness in XML as follows:

- elements of $\mathcal{V}$ are the usual null values;
- P_α's do not necessarily cover all of N; those nodes in N not assigned a label can be thought of as labeled with a wildcard;
- structural incompleteness is captured by relations $\downarrow$, $\rightarrow$, $\Downarrow$, $\Rightarrow$ which could be arbitrary. For example, we may know that $w \Downarrow w'$ without knowing anything about the path between the two.

Semantics As is common with incomplete information, we define semantics via homomorphisms. A *homomorphism* $h : t \rightarrow T$ from an incomplete tree $t = \langle N, V, \downarrow, \Downarrow, \rightarrow, \Rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$ to a complete XML tree $T = \langle D, A, \downarrow, \rightarrow, (P_\alpha)_{\alpha\in\Sigma'}, \rho\rangle$, where $\Sigma \subseteq \Sigma'$, is a pair of maps $h = (h_1, h_2)$ where $h_1 : N \rightarrow D$ and $h_2 : V \rightarrow A$ such that:

- if wRw' in t, then $h_1(w)Rh_1(w')$ in T, when R is one of $\downarrow, \rightarrow, \Downarrow, \Rightarrow$ (recall that $\Downarrow$ and $\Rightarrow$ are interpreted as descendant and following-sibling in complete XML trees);
- if $w \in P_\alpha$ in t, then $h_1(w) \in P_\alpha$ in T, for each $\alpha \in \Sigma$;
- $h_2(c) = c$ whenever $c \in \mathcal{C}$; and
- $h_2(\rho(w)) = \rho(h_1(w))$ for each $w \in N$.

The semantics of an incomplete tree t is the set of all complete trees T that it has a homomorphism into:

$$[\![t]\!]_{\text{OWA}} = \{T \mid \text{exists a homomorphism } t \rightarrow T\}.$$

The superscript OWA means that this is the semantics under the open world assumption; this will be explained in detail shortly. A homomorphism shows how missing features of t are interpreted in a complete document T.

Remark An incomplete tree may be inconsistent in the sense that $[\![t]\!]_{\text{OWA}} = \emptyset$. This however will not affect any results we prove about query answering: as we shall see, over incomplete trees, query answering will be in coNP (or higher), and [10] showed that checking inconsistency can be done in coNP. Thus we can always assume that the input is first checked for being inconsistent (in which case certain answers are vacuously true).

Rigid trees

As we already mentioned, [9] showed that query answering becomes tractable and can be achieved by naïve evaluation of rigid trees. These are trees in which no structural information is missing; that is, the only types of missing information are nulls and wildcards. A rigid tree is defined just as an XML tree (1), i.e. $t = \langle D, A, \downarrow, \rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$, with only two differences:

- A is a subset of $\mathcal{C} \cup \mathcal{V}$ rather than just $\mathcal{C}$ (i.e., nulls are permitted), and
- the union of P_α's need not be the entire D (some nodes may be labeled with wildcards).

Note that the problem of inconsistency, mentioned above, does not arise with rigid trees.

Open and closed world assumptions

Open world assumption (OWA) states that a database, or a document, is open to adding new facts (e.g., tuples, nodes, associations between nodes). This is the semantics adopted in [9], and defined above, for XML. In the relational world it is normally expressed by having a homomorphism from an incomplete instance *into* a complete instance.

On the other hand, closed world assumption (CWA) states that a database or a document is closed for adding new facts. In the relational case, this is usually formalized by having a homomorphism from an incomplete instance *onto* a complete instance. For XML, the situation is a bit more involved however, due to the presence of transitive closures of the child and next-sibling axes, as was explained informally in the introduction. We now define the notions of weak and strong closed world assumptions formally.

Of course we can adopt the relational notion of having an onto-homomorphism. We call this a *strong closed world assumption*, or SCWA. More precisely,

$$[\![t]\!]_{\text{SCWA}} = \{T \mid \text{exists an onto homomorphism } t \rightarrow T\}.$$

A homomorphism $h = (h_1, h_2)$ is an onto homomorphism if both h_1 and h_2 are onto (surjective) maps. (The reader may notice that it suffices to require that only h_1 be surjective.) Equivalently, we can say that $[\![t]\!]_{\text{SCWA}} = \{h(T) \mid h \text{ is a homomorphism }\}$.

But this assumption may be too strong if we deal with transitive closure axes. Consider, for example, an incomplete tree with two nodes v and v' such that $v \Downarrow v'$. Under SCWA, it can only be mapped into 2-node trees, while the interpretation of $\Downarrow$ says that a path between v and v' of length greater than 1 may be allowed. We thus weaken the SCWA, by allowing paths between nodes for which only $\Downarrow$ or $\Rightarrow$ associations exist.

Formally, we define the *weak closed world assumption*, or WCWA, as follows. A homomorphism $h = (h_1, h_2)$:

$t \to T$ is called an WCWA-*homomorphism* if, for every node w of T that is not in the image of h_1 (i.e., not an image of a node of t), there exist two nodes v, v' of t such that either

- $v \Downarrow v'$ holds in t and $h(v) \Downarrow w \Downarrow h(v')$ holds in T; or
- $v \Rightarrow v'$ holds in t and $h(v) \Rightarrow w \Rightarrow h(v')$ holds in T.

That is, the homomorphism h may not be surjective, but if a node is not in the image of h, then it must be on a horizontal or a vertical path between two nodes that are in the image of h.

We then define

$$[\![t]\!]_{\text{WCWA}} \; = \; \{T \mid \text{exists a WCWA-homomorphism } t \to T\}.$$

Clearly each surjective homomorphism is a WCWA-homomorphism, and thus $[\![t]\!]_{\text{SCWA}} \subseteq [\![t]\!]_{\text{WCWA}}$. Also in the absence of transitive closure axes, as in rigid trees, there is no difference between the two semantics, in which case we refer just to CWA semantics, and write $[\![t]\!]_{\text{CWA}}$.

Fixed vs nonfixed alphabets We have made an assumption that labels come from an infinite set $\mathcal{L}$. A similar, and also a reasonable assumption, is to have a finite but sufficiently large alphabet (e.g., to assume that no tree or query uses every single available label: first, such an assumption is impractical, and second, it leads to query answering abnormalities w.r.t. certain answers [9]). More precisely, in this case we assume that labels come from a fixed finite alphabet $\mathcal{L}_{\text{fin}}$, and that trees and queries do not exhaust it: for instance, we can assume that, given a tree T and a query Q, we can replace each label used in them by a new one.

It turns out that all the results are the same under both assumptions, except one: combined complexity for incomplete trees under OWA goes up by one exponent in the case of a fixed alphabet. However, we shall see that this only happens when the query uses many wildcards. All lower bounds will hold even for fixed finite alphabets. Thus, with one exception, we shall not be specifying our assumptions for the set of possible labels.

3. QUERY ANSWERING AND INCOMPLETENESS

3.1 Relational queries

We now recall the basics of query answering over databases with incomplete information. Such a database, under the naïve interpretation of nulls, is a database whose elements come from the domain of constants $\mathcal{C}$ and the domain of nulls $\mathcal{V}$. The semantics is defined via homomorphisms $h : \mathcal{V} \to \mathcal{C}$. Such a map is a homomorphism between two databases D and D' of the same schema if, for every relation R of D and every tuple $\bar{a}$ of R, the tuple $h(\bar{a})$ is in the relation of R of D'. As before, we view h as a map $\mathcal{C} \cup \mathcal{V} \to \mathcal{C}$ extended by letting $h(c) = c$ for each $c \in \mathcal{C}$.

This leads to two standard semantics:

$$[\![D]\!]_{\text{OWA}} = \{D' \mid \text{exists a homomorphism } D \to D'\}$$

and

$$[\![D]\!]_{\text{CWA}} = \{h(D) \mid h \text{ is a homomorphism}\}.$$

Given a relational query Q, its result on an incomplete database is defined by means of *certain answers*:

$$\mathit{certain}_*(Q, D) \; = \; \bigcap \{Q(D') \mid D' \in [\![D]\!]_*\}$$

where $*$ is either OWA or CWA.

Computational problems As is common, we look at *data complexity* and *combined complexity* of computing query answers (in this case, certain answers under various semantics). More precisely, the problems we deal with are as follows:

- *Combined complexity of a language L.* The input consists of a database D, a query Q in L, and a tuple of data values $\bar{s}$ of the same arity as Q; the question is whether $\bar{s} \in \mathit{certain}_*(Q, D)$.
- *Data complexity of a language L.* In this case we have a *fixed* query Q in L; the input consists of a database D and a tuple of data values $\bar{s}$ of the same arity as Q, and the question is the same: whether $\bar{s} \in \mathit{certain}_*(Q, D)$.

Here $*$ ranges over our semantic assumptions (that lead to different notions of certain answers).

Convention When we say that data complexity of a query language L is complete for a complexity class C (e.g., CONP-complete), we mean that (1) for every query Q in L, its data complexity is in C, and (2) there is a query Q_0 in L whose data complexity is C-hard.

Computing certain answers For arbitrary FO queries, the combined complexity of finding certain answers is undecidable (finite validity). For one class of queries the problem is solvable using the standard query evaluation. We define a *naïve evaluation* of a query as the standard evaluation of it followed by removing tuples containing nulls. It was shown in [19] that for *unions of conjunctive queries*, naïve evaluation computes certain answers (which are in this case the same under both OWA and CWA). In fact, the result is optimal: no larger class of queries within FO has this property [22]. Beyond unions of conjunctive queries, algorithms for finding certain answers use more complex representations, namely conditional tables [3, 19, 18].

Much of the work on complexity of query answering over relational databases with nulls concentrated on languages such as unions of conjunctive queries (UCQs),

	data complexity		combined complexity	
	CWA	OWA	CWA	OWA
UCQ	Ptime	Ptime	NP-complete	NP-complete
UCQ$^{\neq}$	coNP-complete	coNP-complete	Π_2^p-complete	Π_2^p-complete
BCCQ	Ptime	Ptime		
BCCQ$^{\neq}$	coNP-complete	coNP-complete		
FO	coNP-complete	undecidable	Pspace-complete	undecidable

Figure 4: Complexity of computing certain answers: relational case

FO, and beyond (e.g., datalog). As the gap between UCQs and FO is very large, one might look for classes between those two. Some results about such classes are known: for example, finding certain answers to UCQs with inequalities is coNP-complete [2]. On the XML side, we shall be dealing with analogs of the following relational languages, build up from conjunctive queries by using Boolean operations, quantifiers, inequalities, and quantification:

- UCQ and UCQ$^{\neq}$: unions of conjunctive queries (with inequalities, i.e. atoms $x \neq y$ are allowed);
- BCCQ and BCCQ$^{\neq}$: Boolean combinations of conjunctive queries (with inequalities). In other words, starting with conjunctive queries (with inequalities) $q_1(\bar{x}), \ldots, q_m(\bar{x})$, we can close them under operations $q \cup q'$, $q \cap q'$ and $q - q'$.
- FO, which can be viewed as closure of conjunctive queries under Boolean operations and quantification.

While some complexity bounds (both data and combined) are known for finding certain answers under both OWA and CWA, we are not aware of such results for BCCQs, and prove them for the sake of completeness.

Theorem 1. *Data complexity of finding certain answers to* BCCQ*s is in* Ptime, *while for* BCCQ$^{\neq}$ *it is* coNP*-complete. For both classes of queries the combined complexity is* Π_2^p*-complete. These bounds hold under both* OWA *and* CWA.

Proof sketch. We explain briefly the main idea behind the Ptime algorithm for BCCQs under OWA. The key case is that of a special type of query with just one negation, namely a Boolean query $Q = q \vee \neg q'$, where q is a union of conjunctive queries, and q' is a conjunctive query. Suppose we have a database D with null values. We give an algorithm for checking whether $\mathit{certain}_{\text{OWA}}(Q, D)$ is false, i.e., whether $D' \models \neg q$ and $D' \models q'$ for some $D' \in [\![D]\!]_{\text{OWA}}$. This is done as follows. First, convert q' into its tableau, $\mathrm{Tab}(q')$. Then let $D[q']$ be the "disjoint union" of D and $\mathrm{Tab}(q')$ (i.e., we rename all nulls in $\mathrm{Tab}(q')$ so that they would be different from nulls in D). Then we show that $\mathit{certain}_{\text{OWA}}(Q, D)$ is false iff $\mathit{certain}_{\text{OWA}}(q, D[q'])$ is false. Since q is a UCQ, checking the latter condition can be done by naïve evaluation, and hence in Ptime.

In the general case, when we deal with a Boolean combination Q of conjunctive queries $q_1, \ldots, q_m$ (assume they are Boolean), we look at all the assignments of $q_1, \ldots, q_m$ to true and false that make the Boolean combination false. Assume we have such a valuation that assigns q_i with $i \in I$ to true and q_j with $j \notin I$ to false. Then, checking whether $\mathit{certain}_{\text{OWA}}(Q, D) = \bot$ is witnessed by this assignment (i.e., whether there exists $D' \in [\![D]\!]_{\text{OWA}}$ so that the q_i's evaluate to true/false on it as in the assignment) amounts to checking whether $\mathit{certain}_{\text{OWA}}(q \vee \neg q', D) = \bot$ where $q = \bigvee_{j \notin I} q_j$ and $q' = \bigwedge_{i \in I} q_i$. For this we simply use the above algorithm, as the query is now in the right shape. Since Q is fixed, so is the number of assignments of the q_i's to true/false, which gives us an overall Ptime algorithm.

For CWA, however, the algorithm is more involved, as the basic reduction no longer works if we use $\mathit{certain}_{\text{CWA}}(q, D[q'])$. Indeed while always $D[q'] \in [\![D]\!]_{\text{OWA}}$, often $D[q'] \notin [\![D]\!]_{\text{CWA}}$. To overcome the problem, we consider every possible way of merging tuples in q' with tuples in D. This yields a small (i.e., polynomial in the size of D) set of new databases for which we show that $\mathit{certain}_{\text{CWA}}(Q, D)$ is false iff there is some D' in that set such that $\mathit{certain}_{\text{CWA}}(q, D')$ is false. □

Figure 4 summarizes what is known about both data and combined complexity of finding certain answers for relational queries; results are from [2, 3, 19, 28] and the above theorem.

Note that there is a rather persistent confusion in the literature regarding *data* complexity of certain answers of FO queries. It is very common to attribute such undecidability to Trakhtenbrot's theorem, which is fine for combined complexity, but not technically correct in the case of a fixed query. While the result itself seems to be folklore, we wanted to clear this confusion and we shall provide in the full version a self-contained proof of undecidability of data complexity of FO on naïve tables.

3.2 Pattern-based XML queries

We now define the analogs of the relational languages we considered in the XML setting. As is common in the scenarios when one needs to compute certain answers (by means of intersection) [8, 9], we look at queries that can only output tuples of data values.

The queries will be essentially fragments of first-order logic over XML trees; however, to avoid the clumsiness of a two-sorted presentation, we follow the standard approach and define them via *patterns*. For now, we shall look at patterns based on the child/next-sibling axes; extensions will be discussed later.

An example of a pattern is

$$\alpha(x)/[\beta(x) \to \gamma(1), \delta(y) \to \gamma(x)].$$

When evaluated on a tree T, it collects all instantiations of variables x and y so that a tree has an α-node whose data value is x, together with (1) a β-child with the same data value x whose next sibling is a γ-node with data value 1; and (2) a δ-child with data value y whose next sibling is a γ-node with data value x.

Formally, patterns are given by the grammar:

$$\begin{array}{rcl} \pi & := & \alpha(\bar{z})/[\mu, \ldots, \mu] \\ \mu & := & \pi \to \ldots \to \pi \end{array}$$

where α ranges over Σ or wildcard _, and $\bar{z}$ is a tuple of variables and constants. We write $\pi(\bar{x})$ if $\bar{x}$ is a tuple of all the variables mentioned in π. Also, to simplify notation, we shall write $\alpha(\bar{x})/\beta(\bar{y})$ instead of the more formal $\alpha(\bar{x})/[\beta(\bar{y})]$.

We define the semantics with respect to an XML tree $T = \langle D, A, \downarrow, \to, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$ and a valuation ν for variables $\bar{x}$ in $\mathcal{C}$:

- $(T, w, \nu) \models \alpha(\bar{z})[\mu_1, \ldots, \mu_n]$ if $w \in P_\alpha$ (whenever α is a Σ-letter), $\rho(w) = \nu(\bar{z})$, and there exist n children $w_1, \ldots, w_n$ of w such that $(T, w_i, \nu) \models \mu_i$ for each $i \leq n$.
- $(T, w, \nu) \models \pi_1 \to \ldots \to \pi_m$ if there is a sequence $w = w_1 \to w_2 \to \ldots \to w_m$ of nodes so that $(T, w_i, \nu) \models \pi_i$ for each $i \leq m$.

We shall write $(T, w) \models \pi(\bar{a})$ if $(T, w, \nu) \models \pi(\bar{x})$ where ν assigns values $\bar{a}$ to variables $\bar{x}$.

Classes of pattern-based XML queries We now define XML analogs of the five languages we considered in the relational case which are based on patterns. First, we need a class of *conjunctive queries* (essentially defined in [8, 10, 17]): these are obtained by closing patterns under conjunction and existential quantification of variables:

$$q(\bar{x}) \;=\; \exists \bar{y}_1 \ldots \bar{y}_n \;\pi_1(\bar{x}, \bar{y}_1) \wedge \ldots \wedge \pi_n(\bar{x}, \bar{y}_n)$$

The semantics is defined as follows. Given a tree T and a valuation $\bar{a}$ for variables $\bar{x}$, we have $T \models q(\bar{a})$ if there exist tuples $\bar{b}_1, \ldots, \bar{b}_n$ of data values and nodes $w_1, \ldots, w_n$ in T so that $(T, w_i) \models \pi_i(\bar{a}, \bar{b}_i)$ for every $i \leq n$.

If inequality atoms ($u \neq v$) are allowed too, in addition to patterns, we talk about conjunctive queries with inequalities. The semantics extends in the natural way.

Now we define the languages we deal with.

$\mathrm{UCQ}_{\mathrm{XML}}$ **and** $\mathrm{UCQ}^{\neq}_{\mathrm{XML}}$ These are defined as queries of the form $q_1(\bar{x}) \cup \ldots \cup q_m(\bar{x})$, where each q_i is a conjunctive query (with inequalities, respectively).

$\mathrm{BCCQ}_{\mathrm{XML}}$ **and** $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$ These are obtained by closing conjunctive queries (with inequalities) under operations $q \cup q'$, $q \cap q'$ and $q - q'$.

$\mathrm{FO}_{\mathrm{XML}}$ These are obtained by closing patterns and equality atoms under the Boolean operations and both universal and existential quantification.

Examples We start with an example of a $\mathrm{UCQ}_{\mathrm{XML}}$ query:

$$\begin{array}{rcl} q_1(x) & := & \exists y, z \;\; \alpha(x)/[\beta(y) \to \gamma(z)] \\ & & \vee \\ & & \exists y \;\alpha(x)/\delta(y) \end{array}$$

It selects data values x found in α-labeled nodes which either have two consecutive children labeled β and γ (with some data values attached to them), or a child labeled δ (also with a data value in it). If in addition we want to impose a requirement that the data values in the β-node (or δ-node) be different from x, we use a $\mathrm{UCQ}^{\neq}_{\mathrm{XML}}$ query:

$$\begin{array}{rcl} q_2(x) & := & \exists y, z \;\big(\alpha(x)/[\beta(y) \to \gamma(z)] \;\wedge\; x \neq y\big) \\ & & \vee \\ & & \exists y \;\big(\alpha(x)/\delta(y) \;\wedge\; x \neq y\big) \end{array}$$

The following is an example of a $\mathrm{BCCQ}_{\mathrm{XML}}$ query:

$$\begin{array}{rcl} q_3(x, y) & := & \neg\exists z \;\Big(\alpha(z)/\big[\gamma(x) \to \beta(y)\big]\Big) \\ & & \vee \\ & & \exists z \;\big(\alpha(x)/\gamma(z)/\beta(y)\big) \end{array}$$

It selects tuples (x, y) of data values that are either *not* found in two consecutive children labeled with γ and β of an α-node, or found on a path labeled $\alpha - \gamma - \beta$, in α and β-nodes. To require in addition that $x \neq y$, we use a $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$ query:

$$\begin{array}{rcl} q_4(x, y) & := & \neg\exists z \;\Big(\alpha(z)/\big[\gamma(x) \to \beta(y)\big] \wedge x \neq y\Big) \\ & & \vee \\ & & \exists z \;\big(\alpha(x)/\gamma(z)/\beta(y) \wedge x \neq y\big) \end{array}$$

Finally, we give an example of an $\mathrm{FO}_{\mathrm{XML}}$ query:

$$\forall y\Big(\alpha(x)/\beta(y) \;\longrightarrow\; \exists z \;\gamma(z)/\delta(y) \;\wedge\; y \neq z\Big)$$

It selects data values x such that if they are found in α-nodes with a β-child, then the data value y of that child must also be found in a δ-node whose parent is labeled γ and has a data value different from y.

Certain answers Since queries in languages introduced above produce sets of tuples of data values, we can define the usual notion of certain answers for evaluating them over incomplete documents. That is, for a query Q and an incomplete tree t, we let

$$certain_*(Q,t) \;=\; \bigcap\{Q(T) \mid T \in [\![t]\!]_*\},$$

where $*$ ranges over OWA, SCWA, and WCWA. The problems we consider for them are the same as in the relational case: determining data and combined complexity.

4. QUERY ANSWERING OVER ARBITRARY INCOMPLETE TREES

We now look at query answering over arbitrary incomplete XML trees. One data complexity result was previously known, namely CONP-completeness for $\mathrm{UCQ}_{\mathrm{XML}}$ under OWA [9]. We now complete the study, and present results on both data and combined complexity for all five languages introduced in the previous section.

Before we embark on this study, there is one natural question we need to ask: can we obtain the desired results simply by recourse to relational query answering? After all, incomplete XML trees are relational structures. The answer is that we *cannot* meaningfully adapt relational results. The main reason is that, if we have an incomplete tree t represented as a relational database D_t, then $[\![D_t]\!]$ is *not* the set of relational representations of trees in $[\![t]\!]$ (except in some very limited cases). This is of course due to the fact that $[\![t]\!]$ only contains trees, but $[\![D_t]\!]$ may contain databases that are not translations of trees.

To apply relational results, we would need to impose an extra constraint that complete databases are trees. This is very problematic as, under OWA, already much simpler constraints lead to undecidability of query answering [11, 25]. Another alternative is to move from a query Q to a query $\neg\mathtt{tree} \vee Q$, where $\mathtt{tree}$ expresses that a relational database is a representation of an XML tree. This needs a fixpoint mechanism. Thus, expressing the above query (that also involves $\neg$ and $\vee$) puts us in the realm of disjunctive datalog. While known results do give us decidability, complexity bounds we can infer "for free" will be Π_2^p for data complexity and $\mathrm{CONEXP}^{\mathrm{NP}}$ for combined complexity [16] for $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$ and its sublanguages. As we shall see, we can obtain better (and often tight) complexity bounds working directly on XML trees.

4.1 Query answering under OWA

As mentioned earlier, data complexity of $\mathrm{UCQ}_{\mathrm{XML}}$ is known to be CONP-complete [9], while precise combined complexity was never stated. The proof of [9] only yields a nonelementary upper bound, but it turns out that the actual bound is much lower. In the result below, for combined complexity, we assume that the alphabet of labels is infinite.

THEOREM 2. *Over arbitrary incomplete trees under* OWA*, for each of the languages* $\mathrm{UCQ}_{\mathrm{XML}}, \mathrm{UCQ}^{\neq}_{\mathrm{XML}}, \mathrm{BCCQ}_{\mathrm{XML}}, \mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$*, data complexity is* CONP*-complete and combined complexity is* Π_2^p*-complete. Furthermore,* $\mathrm{FO}_{\mathrm{XML}}$ *is undecidable with respect to both data and combined complexity.*

Recall the convention regarding completeness of data complexity: stating that it is CONP-complete means that it is always in CONP, and for some queries it is CONP-hard. Likewise, undecidability means that data complexity of some fixed query is undecidable. We shall comment on the gap for combined complexity without a bound on the number of wildcards after sketching the proof.

PROOF SKETCH. Since CONP-hardness of $\mathrm{UCQ}_{\mathrm{XML}}$ is known from [9], for data complexity it suffices to establish a CONP upper bound for $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$, and undecidability for $\mathrm{FO}_{\mathrm{XML}}$. For combined complexity we need to establish Π_2^p-hardness for $\mathrm{UCQ}_{\mathrm{XML}}$ and a CONEXP upper bound for $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$.

We now explain the idea behind the upper bound for combined complexity (of $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$ and others). The standard way to obtain an upper bound for a query Q (say, Boolean for this sketch) over t is to prove that if $certain_{\mathrm{OWA}}(Q,t)$ is false, then there is $T_0 \in [\![t]\!]_{\mathrm{OWA}}$ with some specific size bounds (in t) such that $T_0 \models \neg Q$. While this was done in the past for data complexity (e.g., in [8, 9]), the bounds were extremely high in terms of the query. So we need to reduce the size of the tree much more carefully to establish combined complexity bounds.

The starting point of the proof is standard: suppose we have $T \in [\![t]\!]_{\mathrm{OWA}}$ such that $T \models \neg Q$. For the sketch, assume that $Q = q_1 \vee \ldots \vee q_n$, where each q_i is a conjunctive query; that is, $T \models \neg q_i$ for each $i \leq n$. Take a homomorphism $h : t \to T$, and add to the image of h all nodes which are least common ancestors of nodes in the image, plus the root. We call it the skeleton. Now if we have any node that is not on a vertical or a horizontal path between two nodes in the skeleton, we can throw it away, together with the subtree rooted at it, and obtain a tree T' that agrees with T on all the q_is (by monotonicity).

The problem with T' is that we may have very long horizontal or vertical paths which still need to be cut. This is done using the fact the we have an infinite supply of labels, and shorter paths can be relabeled in a way that does not satisfy any of the queries.

We then provide a Π_2^p-hardness reduction for $\mathrm{UCQ}_{\mathrm{XML}}$ by coding QSAT with a $\forall^*\exists^*$ quantifier prefix. □

The case of a fixed alphabet The combined complexity result above, for languages from $\mathrm{UCQ}_{\mathrm{XML}}$ to $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$, does not work in the case of a fixed alphabet. As we mentioned earlier, this is the only case when results are different under these assumptions. We now state the result under the assumption that labels come

	data complexity		combined complexity	
	SCWA	WCWA and OWA	SCWA	WCWA and OWA
$\mathrm{UCQ}_{\mathrm{XML}}$	coNP-complete	coNP-complete	Π_2^p-complete	Π_2^p-complete
$\mathrm{UCQ}^{\neq}_{\mathrm{XML}}$				
$\mathrm{BCCQ}_{\mathrm{XML}}$				
$\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$				
$\mathrm{FO}_{\mathrm{XML}}$	coNP-complete	undecidable	Pspace-complete	undecidable

Figure 5: Complexity of computing certain answers over arbitrary incomplete trees

from a finite but sufficiently large alphabet $\mathcal{L}_{\mathrm{fin}}$.

Theorem 3. *Under the finite alphabet assumption, the combined complexity of* $\mathrm{UCQ}_{\mathrm{XML}}, \mathrm{UCQ}^{\neq}_{\mathrm{XML}}, \mathrm{BCCQ}_{\mathrm{XML}}$, *and* $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$, *under* OWA, *is in* coNEXP *and* Π_2^p*-hard. Furthermore, if the number of wildcards in the query is fixed, then it is* Π_2^p*-complete.*

Proof sketch. The only difference in the proof is in cutting long vertical and horizontal paths. To explain how this is done under the fixed alphabet assumption, suppose we have a long horizontal path $u_1 \to \ldots \to u_l$ of nodes. It could match some of the "horizontal" query subexpressions $\pi_1 \to \ldots \to \pi_m$. Looking at the roots of each pattern, we associate each such π_i with a word over $\Sigma \cup \{\_\}$. Next we collect all words S in Σ^* that are instantiations of such patterns with wildcards that the horizontal path does *not* match (there are at most exponentially many of them in the size of the query).

We claim that the path $u_1 \to \ldots \to u_l$ can be cut to length exponential in the size of the query so that it still does not match any words in S. To see this, we use the Aho-Corasick algorithm [5] and construct a DFA $\mathcal{A}_S$ that accepts words containing one word of S as a pattern; its size is polynomial in S and hence exponential in the query. Now take the complement DFA $\overline{\mathcal{A}_S}$ which accepts the label of the path $u_1 \to \ldots \to u_l$, and use pumping to reduce it to the required size so that none of the queries q_i is satisfied (since none of the words in S will have a match). For vertical paths, we proceed similarly. Given the overall exponential size of the tree (in terms of the size of the query), we get the coNEXP bound on combined complexity. If the number of wildcards is fixed, then the set S above is of polynomial size, which results in a polynomial-size tree and a Π_2^p upper bound. The Π_2^p-hardness proof of the previous theorem applies here verbatim. □

Remark We now comment on the coNEXP-Π_2^p gap for combined complexity. It is due to the fact that the construction of the Aho-Corasick [5] automaton is polynomial in the number of patterns, but with wildcards (known as don't-cares in string pattern matching literature) the size of the set may be blown up exponentially (unless we restrict the use of wildcard in the query).

Construction of DFAs for patterns with wildcards, that we would need to lower the bound, has so far been considered in the case of a single pattern [23], and very recently for multi-patterns too [26]. While the latter yields an efficient pattern-matching algorithm, it still gives an exponential blowup if formulated in purely automata-theoretic terms. As our technique for reducing the size of the tree depends on such automata construction, at present we do not see any possibility of using our approach to reduce the bound, due to the lack of pattern matching tools we can apply (although we conjecture that combined complexity remains in Π_2^p).

4.2 Query answering under CWA

We now move to the closed world assumption. Recall that for arbitrary incomplete trees, there are two possible interpretations of it. Under the strong interpretation SCWA, we insist that each node in a complete tree correspond to a node in an incomplete tree. Under the weak interpretation WCWA, new nodes may be inserted between nodes related by ⇒ or by ⇓. Our first result is about the weak interpretation.

Theorem 4. *Under* WCWA*, data and combined complexity of query evaluation over arbitrary incomplete trees is the same as under* OWA*, i.e., as described in Theorem 2.*

Note that OWA upper bounds trivially apply, so the key observation is that OWA hardness results can be done using OWA only to extend paths (rather than insert arbitrary trees), which corresponds to WCWA.

Under the strong assumption, complexity bounds come down only for the case of $\mathrm{FO}_{\mathrm{XML}}$, and stay as they were for OWA and WCWA for other languages. Note that for arbitrary incomplete trees, we cannot yet reduce query evaluation under SCWA to the relational case, and indeed some bounds are different (Π_2^p for trees and NP for relations for UCQs).

Corollary 1. *Over arbitrary incomplete trees under* SCWA*, for each of the languages* $\mathrm{UCQ}_{\mathrm{XML}}, \mathrm{UCQ}^{\neq}_{\mathrm{XML}}, \mathrm{BCCQ}_{\mathrm{XML}}, \mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$*, data complexity is* coNP*-complete and combined complexity is* Π_2^p*-complete. For* $\mathrm{FO}_{\mathrm{XML}}$*, data complexity remains* coNP*-complete, while combined complexity is* Pspace*-complete.*

For upper bounds, one simply guesses an onto homomorphism h so that $\bar{a} \notin Q(h(t))$. This gives a coNP upper bound for data complexity for all languages, and coNP-hardness already follows from [9]. Since conjunctive queries with negation can be evaluated with NP combined complexity, this also gives a Π_2^p upper bounds for languages based on conjunctive queries (with an additional guess which queries will be true and which will be false for Boolean combinations). And since $\mathrm{FO}_{\mathrm{XML}}$ can be translated into FO over a relational representation, we get the PSPACE bound from the corresponding bound for combined complexity of FO. The lower bound is also from the relational case, by encoding relations as XML documents.

Summary Results for arbitrary incomplete trees are summarized in Figure 5. A quick look shows the following:

1. Data complexity is always intractable – unlike in the relational case, we lose polynomial data complexity of $\mathrm{UCQ}_{\mathrm{XML}}$ and $\mathrm{BCCQ}_{\mathrm{XML}}$. Combined complexity is elementary (in fact at most 2-exponential) despite previous high bounds (except for $\mathrm{FO}_{\mathrm{XML}}$ of course where it is undecidable).

2. For arbitrary trees, closed world assumption – in either form – does not help bring down complexity.

So, as in [9], this motivates looking at the restricted case of rigid trees, for all of the languages and assumptions. This is what we do next.

5. QUERY ANSWERING OVER RIGID INCOMPLETE TREES

Recall that in rigid trees we have no missing structural information. That is, they are of the form $t = \langle D, A, \downarrow, \rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$, where D is a usual unranked tree domain, $\downarrow$ and $\rightarrow$ are child and next-sibling relations, labeling predicates P_α's need not cover all of D, and ρ assigns data values from $\mathcal{C} \cup \mathcal{V}$.

In particular, in the absence of structural incompleteness, there is no difference between SCWA and WCWA, and we shall talk just about CWA. That is, $[\![t]\!]_{\mathrm{CWA}} = \{h(t) \mid h \text{ is a homomorphism}\}$.

As before, we start with the open world assumption, and then consider the closed world assumption.

5.1 Query answering under OWA

The only previously known result on the complexity of query answering over rigid trees under OWA states that data complexity of $\mathrm{UCQ}_{\mathrm{XML}}$ queries is in PTIME [9]. Moreover, query answering can be done by naïve evaluation. That is, one simply computes $Q(t)$, and throws away tuples that contain nulls, and this guarantees to produce $\mathit{certain}_{\mathrm{OWA}}(Q, t)$. By translation into relational representation, this implies NP-completeness of combined complexity. We now complete this picture.

THEOREM 5. *Under* OWA, *data complexity of* $\mathrm{BCCQ}_{\mathrm{XML}}$ *queries over rigid trees is in* PTIME, *while for queries in* $\mathrm{UCQ}^{\neq}_{\mathrm{XML}}$ *and* $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$ *it is* coNP*-complete, and for* $\mathrm{FO}_{\mathrm{XML}}$ *it is undecidable. Combined complexity is* NP*-complete for* $\mathrm{UCQ}_{\mathrm{XML}}$ *and* Π_2^p*-complete for* $\mathrm{UCQ}^{\neq}_{\mathrm{XML}}$, $\mathrm{BCCQ}_{\mathrm{XML}}$, *and* $\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$.

Thus, while for languages with inequalities we match the high bounds for arbitrary incomplete trees, for one extension of $\mathrm{UCQ}_{\mathrm{XML}}$ queries we can get a polynomial-time evaluation algorithm, namely for Boolean combination of CQs. Combined complexity bounds match the relational case, which means they cannot be improved for any reasonable class of XML documents.

We now explain the key ideas behind the proof of Theorem 5. Note that in the case of BCCQs for relations, which was tractable under both OWA and CWA, the algorithm was much simpler under OWA. To the contrary, for XML rigid trees, the algorithm under OWA is more complicated (but still tractable).

In the CWA case, we can indeed adapt the basic reduction used in the proof of Theorem 1 in a relatively straightforward way. Given a tree t and a query $Q = q \vee \neg q'$ where q is a union of conjunctive queries and q' is a conjunctive query, we first construct a tree pattern from q', adding a new root node if needed. We then form a new incomplete tree $t_{q'}$ by merging the root of this tree pattern with the root of t. As the next step, we consider every node homomorphism from $t_{q'}$ to t (that is, a homomorphism of tree structures, that disregards data values). Then for each such homomorphism, we check whether we can merge all the labels and tuples associated to nodes of $t_{q'}$ which have the same image in t. If the answer is yes, this yields a new rigid incomplete tree $t' \in [\![t]\!]_{\mathrm{CWA}}$ which satisfies q' and on which we can evaluate q naïvely. The procedure yields a small (i.e., polynomial in the size of t, for a fixed query) set of new rigid incomplete trees for which we show that $\mathit{certain}_{\mathrm{CWA}}(Q, t)$ is false iff there is some t' in that set such that $\mathit{certain}_{\mathrm{CWA}}(q, t')$ is false.

The algorithm is more involved in the OWA case because, even though we still need to consider node homomorphisms from $t_{q'}$ to t, we more generally need to consider the image of every tree-homomorphism from $t_{q'}$. The problem is that such structures are often not rigid. To overcome the problem, with each such tree we associate a set of rigid trees, whose size grows exponentially in the size of Q, which is fixed, and polynomially in the size of t. This general procedure yields a "small" (i.e., polynomial in the size of t) set of new rigid incomplete trees for which we show that $\mathit{certain}_{\mathrm{OWA}}(Q, t)$ is false iff there is some t' in that set such that $\mathit{certain}_{\mathrm{OWA}}(q, t')$ is false.

	data complexity		combined complexity	
	CWA	OWA	CWA	OWA
$\mathrm{UCQ}_{\mathrm{XML}}$	Ptime	Ptime	NP-complete	NP-complete
$\mathrm{UCQ}^{\neq}_{\mathrm{XML}}$	coNP-complete	coNP-complete	Π_2^p-complete	Π_2^p-complete
$\mathrm{BCCQ}_{\mathrm{XML}}$	Ptime	Ptime		
$\mathrm{BCCQ}^{\neq}_{\mathrm{XML}}$	coNP-complete	coNP-complete		
$\mathrm{FO}_{\mathrm{XML}}$	coNP-complete	undecidable	Pspace-complete	undecidable

Figure 6: Complexity of computing certain answers over rigid incomplete trees

5.2 Query answering under CWA

The last question is how CWA helps when we deal with rigid trees. This is the case that is very close to relations: since the structure is fixed, every completion of a relational representation of a rigid tree would be structurally a tree. However, we still cannot apply relational results directly, because even under SCWA, working with relational representations, we need to ensure that labeling predicates behave properly. But this can be done, resulting in the following.

Theorem 6. *Over rigid incomplete trees, data and combined complexity of all languages except* $\mathrm{FO}_{\mathrm{XML}}$ *are the same under* CWA *and under* OWA.
For $\mathrm{FO}_{\mathrm{XML}}$*, data complexity is* coNP*-complete, and combined complexity is* Pspace*-complete.*

For $\mathrm{UCQ}_{\mathrm{XML}}$ queries, certain answers are the same under OWA and under CWA (and thus both can be computed by naïve evaluation). For the other tractable case of $\mathrm{BCCQ}_{\mathrm{XML}}$ queries, they need not be the same, and in fact the algorithms are more complex than the naïve evaluation algorithm (as was remarked already, even in the relational case such queries cannot be evaluated naïvely to generate certain answers, unless they are equivalent to unions of conjunctive queries [22]).

5.3 Extensions

We now show that the tractable cases withstand the addition of transitive closure axes to queries. That is, we define *extended* patterns by:

$$\begin{array}{lcl} \pi & := & \alpha(\bar{z})/[\mu,\ldots,\mu]//[\mu,\ldots,\mu] \\ \mu & := & \pi \rightsquigarrow \ldots \rightsquigarrow \pi \end{array}$$

where each $\rightsquigarrow$ is either $\rightarrow$ or $\Rightarrow$. The semantics is extended as follows:

- $(T,w,\nu) \models \alpha(\bar{z})/[\mu_1,\ldots,\mu_n]//[\mu'_1,\ldots,\mu'_k]$ if $w \in P_\alpha$ (whenever α is a Σ-letter), $\rho(w) = \nu(\bar{z})$, there exist n children $w_1,\ldots,w_n$ of w such that $(T,w_i,\nu) \models \mu_i$ for each $i \leq n$, and there exist k descendants $w'_1,\ldots,w'_k$ of w such that $(T,w'_i,\nu) \models \mu'_i$ for each $i \leq n$,.
- $(T,w,\nu) \models \pi_1 \rightsquigarrow \ldots \rightsquigarrow \pi_m$ if there is a sequence $w = w_1,\ldots,w_m$ of nodes so that $(T,w_i,\nu) \models \pi_i$ for each $i \leq m$ and $w_i \rightarrow w_{i+1}$ whenever the ith $\rightsquigarrow$ is $\rightarrow$, and $w_i \Rightarrow w_{i+1}$ whenever the ith $\rightsquigarrow$ is $\Rightarrow$.

With these patterns, we define the classes of queries well as $\mathrm{UCQ}^{//,\Rightarrow}_{\mathrm{XML}}$ and $\mathrm{BCCQ}^{//,\Rightarrow}_{\mathrm{XML}}$ just as $\mathrm{UCQ}_{\mathrm{XML}}$ and $\mathrm{BCCQ}_{\mathrm{XML}}$ but based on extended patterns.

We now show that tractable cases of query evaluation continue to apply to queries based on extended patterns.

Proposition 1. *Data complexity of computing* $\mathit{certain}_{\mathrm{OWA}}(Q,t)$ *and* $\mathit{certain}_{\mathrm{CWA}}(Q,t)$ *remains polynomial for queries* Q *in* $\mathrm{UCQ}^{//,\Rightarrow}_{\mathrm{XML}}$ *and* $\mathrm{BCCQ}^{//,\Rightarrow}_{\mathrm{XML}}$ *over rigid trees.*

We also remark that over rigid trees, certain answers can be computed efficiently for an extension of $\mathrm{UCQ}_{\mathrm{XML}}$ that expresses tree-to-tree queries [15].

Summary Going to rigid trees, i.e., giving up structural incompleteness while allowing null values and wildcards, lowers the complexity of query evaluation for all the languages to that of their relational counterparts. There is at least one extension of the standard tractable class (namely Boolean combinations of CQs), but getting answers in polynomial time requires changing the algorithm.

Using CWA does not help at all, except for the strongest language ($\mathrm{FO}_{\mathrm{XML}}$), which is undecidable under OWA as it codes finite validity. In the case of CWA it lowers data complexity to that of relational calculus, but combined complexity remains intractable.

6. CONCLUSIONS

The results of this paper, together with [9], present a complete picture of both data and combined complexity of query answering over incomplete XML documents. In just one case, there is a small gap for combined complexity (which nonetheless present a significant improvement upon previously known nonelementary bounds), which seems to hinge upon currently unavailable techniques for using automata for pattern matching (or a new technique for reducing the size of the witness tree).

Overall, we can infer the following from our study.

1. Structural incompleteness always leads to intractability of query answering (and thus should not be allowed in practical scenarios).

2. Playing with the semantic assumptions, such as open and closed world assumptions, does not have a significant effect on query answering. Thus, it probably makes sense to stick with the commonly accepted OWA in practical scenarios.

3. When incompleteness is reduced to labeling and data values, efficient query answering is possible in query languages that mimic relational languages admitting efficient evaluation.

   The most common such language is union of conjunctive queries, but we showed that several extensions work as well.

So the bottom line seems to be that one should use label and data-value incompleteness only, under OWA, as this gives the best hope for efficient query answering for practically relevant languages.

Acknowledgment Work partially supported by EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599.

7. REFERENCES

[1] S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kauffman, 1999.

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

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

[4] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. *ACM TODS*, 31(1):208–254, 2006.

[5] A. V. Aho, M. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM* 18(6):333-340 (1975).

[6] L. Antova, T. Jansen, C. Koch, D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE'08*, pages 983–992.

[7] M. Arenas, P. Barceló, L. Libkin, F. Murlak. *Relational and XML Data Exchange*. Morgan & Claypool, 2010.

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

[9] P. Barceló, L. Libkin, A. Poggi, C. Sirangelo. XML with incomplete information. *J. ACM*, 58:1 (2010).

[10] H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. *J. Comput. Syst. Sci.* 77(3): 450-472 (2011).

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

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

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

[14] M. Crochemore, W. Rytter. *Jewels of Stringology: Text Algorithms*. World Scientific 2002.

[15] C. David, L. Libkin, F. Murlak. Certain answers for XML queries. In *PODS 2010*, pages 191-202.

[16] T. Eiter, G. Gottlob, H. Mannila. Disjunctive datalog. *ACM Trans. Database Syst.* 22(3):364-418 (1997).

[17] G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees. *J. ACM* 53(2):238-272, 2006.

[18] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer, 1991.

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

[20] B. Kimelfeld, Y. Sagiv. Modeling and querying probabilistic XML data. *SIGMOD Record* 37(4): 69-77 (2008).

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

[22] L. Libkin. Incomplete information and certain answers in general data models. In *PODS'11*, pages 59–70.

[23] R. Pinter. Efficient string matching with don't-care patterns. In *Combinatorial ALgorithms on Words*, NATO ASI Series, 1985.

[24] R. Reiter. On closed world databases. In *"Logic and Databases"*, H. Gallaire and J. Minker eds, Plenum Press, 1978, pages 55–76.

[25] R. Rosati. On the finite controllability of conjunctive query answering in databases under open-world assumption. *J. Comput. Syst. Sci.* 77(3):572-594 (2011).

[26] P. Silvasti, S. Sippu, E. Soisalon-Soininen. Evaluating linear XPath expressions by pattern-matching automata. *J. UCS* 16(5):833-851 (2010).

[27] D. Suciu, D. Olteanu, C. Re, C. Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

[28] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *J. Comput. Syst. Sci.* 54(1): 113-135 (1997).