THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Certain Answers over Incomplete XML Documents: Extending Tractability Boundary

Citation for published version:
Gheerbrant, A & Libkin, L 2014, 'Certain Answers over Incomplete XML Documents: Extending Tractability Boundary', *Theory of Computing Systems*, vol. 57, no. 4, pp. 1-35. https://doi.org/10.1007/s00224-014-9596-y

Digital Object Identifier (DOI):
10.1007/s00224-014-9596-y

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Theory of Computing Systems

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 07. Jul. 2020

Noname manuscript No.
(will be inserted by the editor)

Certain Answers over Incomplete XML Documents: Extending Tractability Boundary

Amélie Gheerbrant · Leonid Libkin

the date of receipt and acceptance should be inserted later

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 detailed 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 a gentle way of introducing negation via Boolean combinations of existential 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. In the process, we also establish a new result about relational query answering over incomplete databases, showing that for Boolean combinations of conjunctive queries, certain answers can be found in polynomial time.

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 [**?**] scenarios. Different types and models of incompleteness have been studied too, such as classical instances of missing information, uncertain databases [5], and probabilistic databases [26]. 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 [11,12] expressed incompleteness by means of description logic theories, and [20] surveyed incorporating probabilities into XML.

A. Gheerbrant
University of Edinburgh and Université Paris–Diderot
E-mail: amelie@liafa.univ-paris-diderot.fr

L. Libkin
University of Edinburgh
E-mail: libkin@inf.ed.ac.uk

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 [8]. 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.

In the model of [8], 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.

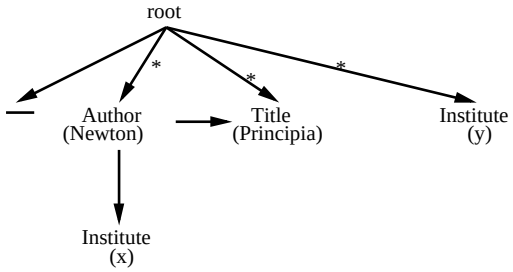

Fig. 1 An incomplete XML document

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, 23], since it leaves a complete document open to adding new nodes.

As the class of queries to study, [8] used XML analogs of *unions of conjunctive queries*, or UCQs. In XML, conjunctive queries are normally modeled via *tree patterns* [7, 9, 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, under the open world assumption this is the largest class of relational calculus queries for which such evaluation computes certain answers to queries [19, 22].

It was shown in [8] 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 [8] 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?

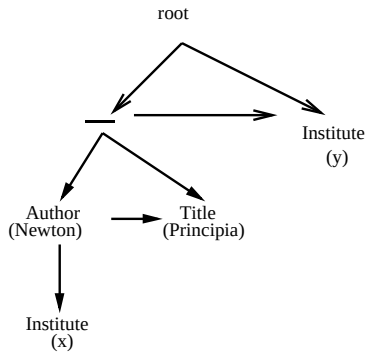

Fig. 2 A rigid incomplete XML document

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.

Boolean combinations of CQs 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 need to look at intermediate ways of adding negation without immediately jumping all the way up to an XML analog of FO. In fact we want to find fragments for which data complexity would be low, and combined complexity would be manageable. This rules out FO, and even such a standard extension as CQs with inequality [2,3, 27].

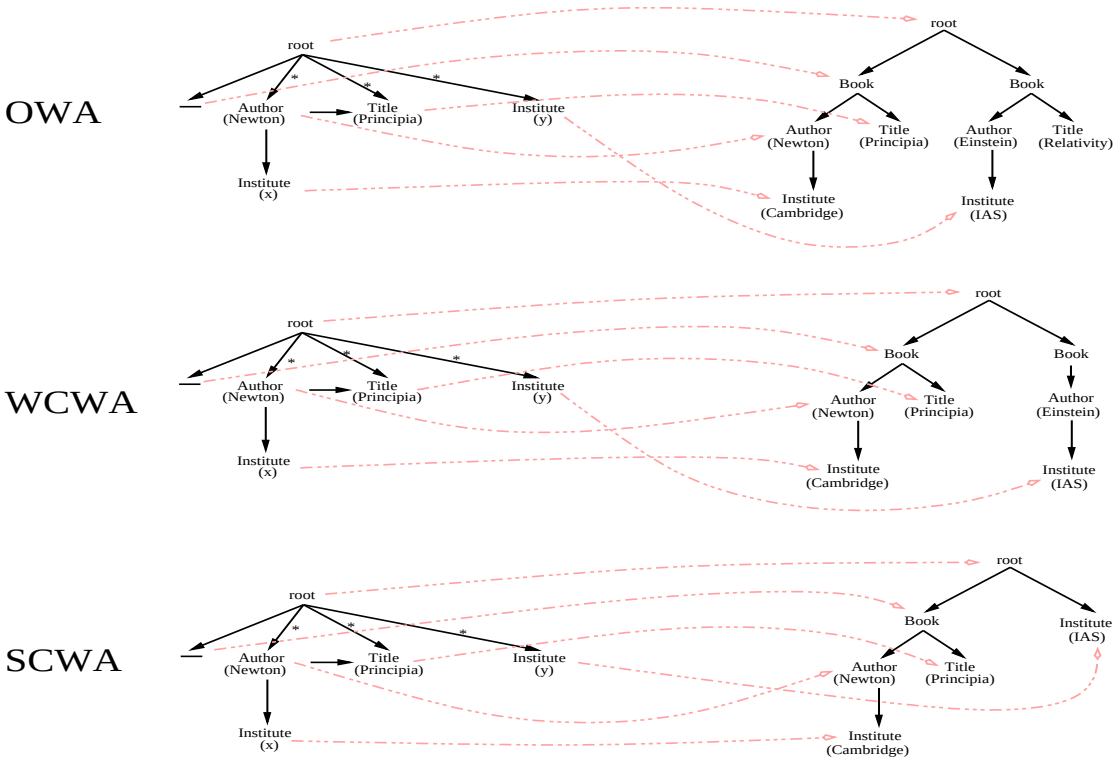

Fig. 3 Open and (weak and strong) closed world semantics of incomplete XML

Instead, we permit arbitrary Boolean combinations of previously defined queries. That is, we look at the class BCCQ of Boolean combination of conjunctive queries, i.e., the closure of conjunctive queries under operations $q \cap q'$, $q \cup q'$, and $q - q'$.

Curiously enough, there is no result in the relational literature that tells us the complexity of finding certain answers for BCCQs. We shall show that it is in fact tractable, under both OWA and CWA. Note that previous tractability results for certain-answers computation were obtained by applying naive evaluation, but we know [22] that such strategy will not work for BCCQs. In fact we have new (and different, under OWA and CWA) algorithms for producing certain answers for relational BCCQs.

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 (just explained above) 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 show that the combined complexity is only marginally higher than data complexity (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. Pattern-based languages are described in Section 4. In Section 5 we establish results on query answering over arbitrary incomplete trees, for all the languages considered here, and in Section 6 we do the same for rigid trees. Final remarks and conclusions are in Section 7.

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.

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

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

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 $\mathrm{dom}(T)$, and to A as the *active domain* (of data values) of T and denote it by $\mathrm{adom}(T)$. 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 [8], 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

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

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

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 with structural incompleteness, query answering will be in coNP (or higher), and [9] 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, [8] 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 [8], 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,

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

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 $[\![t]\!]_{\text{CWA}}$.

3 Query answering and incompleteness: 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*:

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

	data complexity		combined complexity	
	CWA	OWA	CWA	OWA
UCQ	PTIME		NP-complete	
BCCQ			Π_2^p-complete	
FO	coNP-complete	undecidable	PSPACE-complete	undecidable

Fig. 4 Complexity of computing certain answers: relational case

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 $\bar{s} \in certain_*(Q, D)$.
- *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 $\bar{s} \in certain_*(Q, D)$.

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, under OWA 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), 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, and quantification:

- UCQ unions of conjunctive queries;
- BCCQ Boolean combinations of conjunctive queries. In other words, starting with conjunctive queries $q_1(\bar{x}), \ldots, q_m(\bar{x})$, 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 relational* BCCQ*s is in* PTIME*, while the combined complexity is Π_2^p-complete. These bounds hold under both* OWA *and* CWA*.*

PROOF.

Combined complexity results follow easily from [25] which showed Π_2^p-completeness of BCCQ containment. Indeed, to check whether $certain_{\text{OWA}}(q, D)$ is true, for a Boolean query q, we just need to check that $q_D \subseteq q$, where q_D is the canonical query of the database D, and this is in Π_2^p if q is a BCCQ [25]. Under the CWA,

the straightforward algorithm for checking whether $certain_{\text{CWA}}(q, D)$ is false runs in Σ_2^p. Indeed, we guess an instance D' (which is at most of the size of D under CWA), queries $q_1, \ldots, q_k$ in the Boolean combination that will evaluate to true on D' (together with the homomorphisms witnessing that), and queries $q'_1, \ldots, q'_r$ in the Boolean combination that will evaluate to false. We then use the universal step to check that $q'_1, \ldots, q'_r$ indeed evaluate to false, and finally run a polynomial algorithm checking that with $q_1, \ldots, q_k$ evaluating to true and $q'_1, \ldots, q'_r$ evaluating to false, the Boolean combination evaluates to false.

As for hardness, this easily follows from [25]: analyzing the proof of hardness of containment, one can see that it holds even for a containment of a simple conjunctive query essentially coding a database containing two values (true and false) in another query (as they use a $\forall\exists$3CNF coding).

So we now concentrate on proving the PTIME data complexity bound.

Let $Q \in$ BCCQ be a Boolean query and let D be an incomplete relational database (assume w.l.o.g. that the conjunctive queries $q_1, \ldots, q_k$ in Q share no variable in common). We first define an algorithm which determines in polynomial time in the size of D whether $\bot \in certain_{\text{OWA}}(Q, D)$. We begin by listing all possible assignments of the Boolean conjunctive queries in Q to Boolean values for which Q evaluates to $\bot$ (Q being fixed, the number of such assignments does not matter and it is polynomial in the size of D). Now for every such assignment v, we let:

$$\alpha_v := \bigwedge_{i \in I} q_i \text{ where } I = \{i \mid q_i \text{ occurs in } Q \text{ and } v(q_i) = \top\}$$

$$\beta_v := \bigvee_{j \in J} q_j \text{ where } J = \{j \mid q_j \text{ occurs in } Q \text{ and } v(q_j) = \bot\}$$

Remark that every conjunctive query Q that does not use inequalities can also be represented as an incomplete database $Tab(Q)$ by converting Q into its tableau. Also, every incomplete database D can be represented as a conjunctive query $Qu(D)$. This can be done by replacing every distinct null in D by one distinct variable, then forming the conjunction of all the so-obtained tuples and finally prefixing it with one existential quantifier binding each distinct variable in the conjunction. For simplicity, we assume here that nulls and variables can be put in one to one correspondence, so that $Tab(Qu(D)) = D$ and $Qu(Tab(Q)) = D$. Now assume the set of nulls occurring in D and $Tab(\alpha_v)$ to be disjoint (rename nulls if needed) and let:

$$D^v := Tab((Qu(D) \wedge \alpha_v)).$$

In order to check whether $certain_{\text{OWA}}(Q, D) = \bot$, we claim that it is enough to check for every such v whether $D^v \not\models \beta_v$. In each case it is known that this can be determined in polynomial time using naive evaluation. Hence, if $D^v \not\models \beta_v$ for some v, then we simply stop and return $\bot$, otherwise we return $\top$.

We now show that:

$$certain_{\text{OWA}}(Q, D) = \bot \text{ if and only if there exists } v \text{ such that } D^v \not\models \beta_v.$$

For the "if" direction, assume $certain_{\text{OWA}}(Q, D) = \bot$, i.e., there exists a complete database D' such that there is a homomorphism $h : D \to D'$ and $D' \not\models Q$. Then there is a valuation v of the conjunctive queries in Q such that $D' \models \alpha_v \wedge \neg\beta_v$. It follows that there is a homomorphism $h' : Tab(\alpha_v) \to D'$ and that $h \cup h' : D^v \to D'$ is also a homomorphism. Now assume $D^v \models \beta_v$. Conjunctive queries being preserved via homomorphism, this entails $D' \models \beta_v$, which is a contradiction. Hence $D^v \not\models \beta_v$.

For the converse direction, assume there exists v such that $D^v \not\models \beta_v$. We let f be a one to one mapping from the set of nulls occurring in D^v to a set of fresh data values occurring neither in D^v, nor in β_v. We now construct a complete database D' from D^v by replacing every null $\bot_i$ in D^v by $f(\bot_i)$. As $f : D^v \to D'$ is a homomorphism, it follows that $D' \models \alpha_v$ and $D' \in [\![D]\!]_{\text{OWA}}$. Now assume $D' \models \beta^v$. Then there is a homomorphism $h : Tab(\beta^v) \to D'$. It follows that $f^{-1} \circ h : Tab(\beta_v) \to D^v$ is also a homomorphism and hence $D^v \models \beta^v$, which is a contradiction. Hence $D' \not\models \beta^v$, so $D' \not\models Q$ and $certain_{\text{OWA}}(Q, D) = \bot$.

The closed world algorithm relies on the open world algorithm just described. The only difference is that every time a valuation v such that $D^v \not\models \beta_v$ is found, instead of returning $\bot$ we consider a "small" set $\mathbb{D}^v$ (i.e., which size is polynomially bounded in the size of D) of new incomplete databases where for every $D' \in \mathbb{D}^v$, there is a strong onto-homomorphism from D to D' and $D' \models \alpha_v$. For every $D' \in \mathbb{D}^v$, we then determine in polynomial time using naive evaluation whether $D' \not\models \beta_v$. If this is the case for some $D' \in \mathbb{D}^v$, we stop and return $\bot$. If not, we iterate the procedure, i.e., for every v such that $D^v \not\models \beta_v$, we similarly consider every $D' \in \mathbb{D}^v$. Once done, if we did not find any v and $D' \in \mathbb{D}^v$ such that $D^v \not\models \beta_v$ and $D' \not\models \beta_v$, then we return $\top$.

We now explain how to construct the set $\mathbb{D}^v$. We first need a few general definitions.

Let D, D' be two relational databases over the same schema, where $\mathrm{adom}_\bot(D) \cap \mathrm{adom}_\bot(D') = \emptyset$. Now let f be a mapping from the set of D-tuples to the set of D'-tuples, such that for every relation name R in the schema and for every tuple $R(\bar{x}) \in D$, $f(R(\bar{x})) = R(\bar{y})$ for some $R(\bar{y}) \in D'$. Such a mapping f yields an equivalence relation $\sim_f$ over $\mathrm{adom}(D) \cup \mathrm{adom}(D')$ which we define inductively as follows:

- $x \sim_f y$ whenever there exists $R(x_1, \ldots, x_n) \in D$, $R(y_1, \ldots, y_n) \in D'$ and $1 \leq i \leq n$, such that $f(R(x_1, \ldots, x_n)) = R(y_1, \ldots, y_n)$, $x = x_i$ and $y = y_i$;
- $\sim_f$ is reflexive, symmetric and transitive.

We say that D is f-compatible with D' whenever for every $a, b \in \mathcal{C}$, if $a \neq b$, then $a \not\sim_f b$.

Observe that $\sim_f$ yields a set of equivalence classes over $\mathrm{adom}(D) \cup \mathrm{adom}(D')$ as follows:

$$x \in [y]_f \text{ if and only if } x \sim_f y$$

Consider the quotient set of $\mathrm{adom}(D) \cup \mathrm{adom}(D')$ under $\sim_f$:

$$\mathrm{adom}(D) \cup \mathrm{adom}(D')/\sim_f := \{eq_1, \ldots, eq_m\}.$$

Given D, D' and f such that D is f-compatible with D', we now define a new incomplete database D'_f. For every $eq_i \in \mathrm{adom}(D) \cup \mathrm{adom}(D')/\sim_f$, we first define the representative $r_f(eq_i)$ of eq_i by letting:

- $r_f(eq_i) = a$ whenever there exists $a \in eq_i$ such that $a \in \mathcal{C}$,
- $r_f(eq_i) = \bot_i$ otherwise,
  (note that r_f is a one to one mapping).

We then construct D'_f by replacing in D' every occurrence of $x \in \mathrm{adom}_\bot(D')$ by $r_f([x]_f)$. It is immediate that there is a strict onto-homomorphism from D to D'_f and that there is a homomorphism from D' to D'_f:

- $h_f : D \to D'_f$ is a strict onto-homomorphism, where $h_f(x) = r_f([x]_f)$ for every $x \in \mathrm{adom}_\bot(D)$,
- $h'_f : D' \to D'_f$ is a homomorphism, where $h'_f(x) = r_f([x]_f)$ for every $x \in \mathrm{adom}_\bot(D')$.

Given some database D^v as described previously, we can finally let:

$$\mathbb{D}^v = \{D_f \mid Tab(\alpha_v) \text{ is } f\text{-compatible with } D\}$$

Let n be the number of tuples in D, k the number of tuples in $Tab(\alpha_v)$. Observe that there are at most n^k different mappings f such that $Tab(\alpha_v)$ is f-compatible with D. Hence, the size of $\mathbb{D}^v$ is polynomially bounded in the size of D and so the closed-world algorithm runs in polynomial time in the size of D.

In order to show the correctness of the algorithm, we now show that $\mathit{certain}_{\mathrm{CWA}}(Q, D) = \bot$ if and only if there exists a valuation v of the conjunctive queries in Q such that $D^v \not\models \beta_v$ and there exists $D_f \in \mathbb{D}^v$ such that $D_f \not\models \beta_v$.

For the "if" direction, assume $\mathit{certain}_{\mathrm{CWA}}(Q, D) = \bot$, i.e., there exists a complete database $D' \in [\![D]\!]_{\mathrm{CWA}}$ such that there is a strict onto-homomorphism $h : D \to D'$ and $D' \not\models Q$. Then there is a falsifying valuation v of the conjunctive queries in Q such that $D' \models \alpha_v \wedge \neg\beta_v$ and so there is a homomorphism $h' : Tab(\alpha_v) \to D'$, from which it follows that $h \cup h' : D^v \to D'$ is also a homomorphism. Now assume $D^v \models \beta^v$. Conjunctive

queries being preserved by homomorphisms, it follows that $D' \models \beta_v$, which is a contradiction. Hence $D^v \not\models \beta_v$.

Now it remains to show that there is some $D_f \in \mathbb{D}^v$ such that $D_f \not\models \beta_v$. We define the desired f out of two mappings f_h and $f_{h'}$. As h is a homomorphism, each tuple in $Tab(\alpha_v)$ is witnessed by some tuple in D' and we define a mapping f_h from tuples in $Tab(\alpha_v)$ to tuples in D' as follows:

$$f_h(R(\bar{x})) = R(h(\bar{x})).$$

We now show that for every $x, y \in \text{adom}(Tab(\alpha_v)) \cup \text{adom}(D')$, if $x \sim_{f_h} y$ and $x, y \in \mathcal{C}$, then $x = y$, from which it follows that $Tab(\alpha_v)$ is f_h-compatible with D'. So let $x, y \in \text{adom}(Tab(\alpha_v)) \cup \text{adom}(D')$ such that $x \sim_{f_h} y$. It means that there is a sequence $x_0, \ldots, x_n \in \text{adom}(Tab(\alpha_v)) \cup \text{adom}(D')$ such that $x = x_1$ and $y = x_n$ and for every x_i, x_{i+1} with $0 \leq i < n$, the following holds:

$$\exists R(y_1, \ldots, y_m) \in Tab(\alpha_v), R(z_1, \ldots, z_m) \in D' \text{ and } 1 \leq k \leq m$$
$$\text{such that } f(R(y_1, \ldots, y_m)) = R(z_1, \ldots, z_m) \text{ and,}$$
$$\text{either } x_i = y_k \text{ and } x_{i+1} = z_k, \text{ or } x_i = z_k \text{ and } x_{i+1} = y_k.$$

We show the property (i.e., if $x_0, x_n \in \mathcal{C}$, then $x_0 = x_n$) by induction on the length of n. So consider a sequence of length n as described and assume the property holds for all $k < n - 1$ (i.e., if $x_0, x_k \in \mathcal{C}$, then $x_0 = x_k$) . Now consider x_{n-1} and x_n, by assumption we have:

$$\exists R(y_1, \ldots, y_m) \in Tab(\alpha_v), R(z_1, \ldots, z_m) \in D' \text{ and } 1 \leq k \leq m$$
$$\text{such that } f(R(y_1, \ldots, y_m)) = R(z_1, \ldots, z_m) \text{ and,}$$
$$\text{either } x_{n-1} = y_k \text{ and } x_n = z_k, \text{ or } x_{n-1} = z_k \text{ and } x_n = y_k.$$

By definition of f_h, it follows that either $h(x_{n-1}) = x_n$, or $h(x_n) = x_{n-1}$. In both cases, it follows from the fact that h is a homomorphism that if $x_{n-1}, x_n \in \mathcal{C}$, then $x_{n-1} = x_n$. The property follows by induction hypothesis.

We now construct $f_{h'}$. The homomorphism h' being strict onto, every tuple in D' witnesses at least one tuple in D. We define a mapping $f_{h'}$ from tuples in D' to tuples in D by arbitrarily picking one such witness for each tuple in D':

$$f_{h'}(R(\bar{x})) = R(\bar{y}) \text{ for some } R(\bar{y}) \in D', \text{ where } h'(\bar{y}) = \bar{x}.$$

It similarly follows from the fact that h' is a homomorphism that D' is $f_{h'}$-compatible with D.

We finally consider the mapping $f_{h'} \circ f_h$ from tuples in $Tab(\alpha_v)$ to tuples in D. Again, it follows by a similar argument that $Tab(\alpha_v)$ is $f_{h'} \circ f_h$-compatible with D and so $D_{f_{h'} \circ f_h} \in \mathbb{D}^v$, where

- $h_{f_{h'} \circ f_h} : Tab(\alpha_v) \to D_{f_{h'} \circ f_h}$ is a homomorphism,
- $h'_{f_{h'} \circ f_h} : D \to D_{f_{h'} \circ f_h}$ is a strict onto-homomorphism,
- $h'' : D_{f_{h'} \circ f_h} \to D'$ is a homomorphism, given by $h''(r_{f_{h'} \circ f_h}[x]_{f_{h'} \circ f_h}) = h'(x)$.

Now assume $D_{f_{h'} \circ f_h} \models \beta^v$. Conjunctive queries being preserved by homomorphisms, the existence of h'' implies that $D' \models \beta^v$, which is a contradiction. Hence $D_{f_{h'} \circ f_h} \not\models \beta^v$.

The converse direction can be shown using a similar argument as the one used in the open world case, by replacing nulls in D_f by fresh data values.

□

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] and the above theorem.

With respect to one of the items in the table, we note that there is a rather persistent confusion in the literature regarding *data* complexity of certain answers of FO queries. The problem is undecidable, and it is very common to attribute such undecidability to Trakhtenbrot's theorem. The argument usually goes as follows: take an empty database, so that every database is possible under the OWA semantics. Then let the FO sentence say that either the complete database is not a proper encoding of a Turing machine run, or it is, but the run is not halting. Then the certain answer problem becomes the (complement of the) halting problem. But this commonly used argument is fine for proving a bound for *combined* complexity, but not *data* complexity.

To clear this confusion, we present a a short and self-contained proof of undecidability of data complexity of FO on naïve tables.

Theorem 2 *There exists a query $Q \in$ FO such that the problem of determining, for an input incomplete relational database D, whether $certain_{\mathrm{OWA}}(Q, D) = \top$, is undecidable.*

Proof. We proceed by reduction from a tiling problem which is known to be undecidable. An instance of this problem is a triple $I = (Tiles, H, V)$, where $Tiles = \{t_1, \ldots, t_k\}$ is a finite set of tiles and $H, V \subseteq Tiles^2$ are binary relations over $Tiles$. A solution for I is given by a pair (n, f) where $n > 0$ is an integer and $f : \{0, \ldots, n\} \times \{0, \ldots, n\} \mapsto Tiles$ is a function such that:

- (horizontal compatibility) for each $i = 0, \ldots, n-1$ and $j = 0, \ldots, n$,
$$(f(i, j), f(i+1, j)) \in H;$$
- (vertical compatibility) for each $i = 0, \ldots, n$ and $j = 0, \ldots, n-1$,
$$(f(i, j), f(i, j+1)) \in V.$$

The integer n is called an *index solution* for I and the function f is called a *tiling function* for I. Given as input an instance I, the problem of deciding whether I has a solution is known to be undecidable.

We show that there exists a fixed Boolean query $Q \in$ FO such that for every instance I of this tiling problem, we can construct a relational database D_I such that

$$I \quad \text{has a solution} \qquad \text{if and only if} \qquad certain_{\mathrm{OWA}}(Q, D_I) = \bot.$$

Given an instance $I = (Tiles, H, V)$, where $Tiles = \{t_1, \ldots, t_k\}$, we construct D_I over schema

$$\{Tiles, H, V, <_H, <_V, Suc_H, Suc_V, Min_H, Min_V, Max_H, Max_V, R, <\},$$

where $Tiles$ is unary, $H, V, Min_H, Min_V, Max_H, Max_V, <$ are binary, R is ternary and $<_H, <_V, Suc_H, Suc_V$ are of arity 4. We treat elements in $Tiles$ as pairwise distinct data values $t_1, \ldots, t_k$ and construct D_I by interpreting each relation name in the schema by the following relations:

- $Tiles = \{t_1, \ldots, t_k\}$;
- $H, V \subseteq Tiles^2$ where H and V are the horizontal and vertical compatibility relations in I, respectively;
- $<_H, <_V \subseteq Tiles^4$ where $<_H$ and $<_V$ are some arbitrarily chosen linear orderings over tuples in H and V, respectively;
- $S_H, S_V \subseteq Tiles^4$ are the successor functions associated with $<_H$ and $<_V$, respectively;
- $Max_H, Max_V \subseteq Tiles^2$ are singletons containing the unique maximal element in $<_H$ and $<_V$, respectively (note that such an element is a pair of tiles);
- $Min_H, Min_V \subseteq Tiles^2$ are singletons containing the unique minimal element in $<_H$ and $<_V$, respectively;
- R and $<$ are both empty.

Now Q is a FO sentence over the same schema given by:

$$Q := \neg instance \vee \neg tiling$$

where for every database D_I associated to some instance I of the tiling problem, the following holds:

– for every $D'_I \in [\![D_I]\!]_{\text{OWA}}$, $D'_I \models \mathit{instance}$ whenever D'_I does not contain any more H and V tuples than D, more precisely, *instance* states that $<_H$ and $<_V$ are linear orders over the set of pairs of H and V tuples, respectively, where S_H, S_V are the associated successor functions and Min_H, Max_H, Min_V, Max_V the associated unique minimal and maximal elements in $<_H$ and $<_V$ respectively ;
– for every $D'_I \in [\![D_I]\!]_{\text{OWA}}$, $D'_I \models \mathit{tiling}$ whenever $<$ is a linear order over some arbitrary set of data values $d_0 < \ldots < d_n$ such that (n, f) is a solution for I, where f is given by:

$$f(i,j) = t_m \text{ whenever } D'_I \models R(d_i, d_j, t_m);$$

these conditions are enforced in *tiling* by first requiring that $<$ is a linear order which contains at least two elements. Additionally, the third projection of R is assumed to range over data values in $Tiles$, while its first and second projection range over the $<$-ordered d_i's, where for each pair d_i, d_j there exists exactly one t_m such that $D'_I \models R(d_i, d_j, t_m)$. Finally, for every two tuples $R(d_i, d_j, t_k)$, $R(d_{i+1}, d_j, t_{k'})$ (respectively, $R(d_i, d_j, t_k)$, $R(d_i, d_{j+1}, t_{k'})$) in D'_I, *tiling* states that $D'_I \models H(t_k, t_{k'})$ (respectively, $D'_I \models V(t_k, t_{k'})$).

Observe that all the conditions above can be formulated in FO. We now show that I has a solution (n, f) if and only if $\mathit{certain}(Q, D_I) = \bot$.

For the "if" direction of the equivalence, let I be an instance for which there exists a solution (n, f). We construct $D^f_I \in [\![D_I]\!]_{\text{OWA}}$ such that $Q(D^f_I) = \bot$ (which immediately entails $\mathit{certain}(Q, D_I) = \bot$). In addition to the data values already used to encode the elements of $Tiles$ in D_I, we assume a different data value for each natural number $0 \leq m \leq n$ and form D^f_I by adding the following tuples to D_I:

– for each $0 \leq m < m' \leq n$, we add the tuple (m, m') to the interpretation of $<$
– for each $0 \leq m, m' \leq n$, we add the tuple $(m, m', f(m, m'))$ to the interpretation of R

As *instance* is a query over schema

$$\{Tiles, H, V, <_H, <_V, Suc_H, Suc_V, Min_H, Min_V, Max_H, Max_V\},$$

$D_I \models \mathit{instance}$ and the $\{Tiles, H, V, <_H, <_V, Suc_H, Suc_V, Min_H, Min_V, Max_H, Max_V\}$-reduct of D^f_I is precisely D_I, it is immediate that $D^f_I \models \mathit{instance}$. By construction of D^f_I, it is also immediate that $D^f_I \models \mathit{tiling}$.

Now for the "only if" direction assume $\mathit{certain}(Q, D_I) = \bot$. It follows that there is $D'_I \in [\![D_I]\!]_{\text{OWA}}$ such that $Q(D'_I) = \bot$, i.e., $D'_I \models \mathit{instance} \wedge \mathit{tiling}$. By $D'_I \models \mathit{instance}$, D'_I does not contain any more H and V tuple than D_I. Now let n be the number of different data values in the interpretation of $<$ in D'_I. As $D'_I \models \mathit{tiling}$, the pair (n, f) is a solution for I, where the tiling function f is defined by $f(i,j) = t_m$ whenever $D'_I \models R(d_i, d_j, t_m)$. □

4 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 [7,8], 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

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

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 which has a following sibling which is a γ-node with data value x.

Formally, patterns are given by the grammar:

$$\pi := \alpha(\bar{z})/[\mu, \ldots, \mu]//[\mu, \ldots, \mu]$$
$$\mu := \pi \rightsquigarrow \ldots \rightsquigarrow \pi$$

where each $\rightsquigarrow$ is either $\rightarrow$ or $\Rightarrow$, 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 $T = \langle D, A, \downarrow, \rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$ and a valuation ν for variables $\bar{x}$ in $\mathcal{C}$:

- $(T, w, \nu) \models \alpha(\bar{z})/[\mu_1, \ldots, \mu_n]//[\mu'_1, \ldots, \mu'_k]$ 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 $\Rightarrow$.

Note that numbers n, k, m could all be zero; for instance, just a node description $\alpha(\bar{z})$ is a pattern.

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 [7,9,17]): 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 $i \leq n$.

Now we define the languages we deal with.

$\underline{\text{UCQ}_{\text{XML}}}$ Queries in this language are of the form $q_1(\bar{x}) \cup \ldots \cup q_m(\bar{x})$, where each q_i is a conjunctive query.
$\underline{\text{BCCQ}_{\text{XML}}}$ Queries in this language are obtained by closing conjunctive queries under operations $q \cup q'$, $q \cap q'$ and $q - q'$.
$\underline{\text{FO}_{\text{XML}}}$ 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 UCQ_{XML} query:

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

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). The following is an example of a BCCQ_{XML} query:

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

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. Finally, we give an example of an FO_{XML} query:

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

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

Fig. 5 Complexity of computing certain answers over arbitrary incomplete trees

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.

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.

5 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 [8]. We now complete the study, and present results on both data and combined complexity for all three 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 [10, 24]. Another alternative is to move from a query Q to a query $\neg\texttt{tree} \vee Q$, where `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 [15] for $\mathrm{BCCQ}_{\mathrm{XML}}$ and its sublanguages. As we shall see, we can obtain better tight complexity bounds working directly on XML trees.

5.1 Query answering under OWA

As mentioned earlier, data complexity of $\mathrm{UCQ}_{\mathrm{XML}}$ is known to be coNP-complete [8], while precise combined complexity was never stated. The proof of [8] 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 3 *Over arbitrary incomplete trees under* OWA*, for both of the languages* $\mathrm{UCQ}_{\mathrm{XML}}, \mathrm{BCCQ}_{\mathrm{XML}}$*, 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.

Proof. The coNP-hardness of $\mathrm{UCQ}_{\mathrm{XML}}$ is known from [8] for data complexity, while undecidability for $\mathrm{FO}_{\mathrm{XML}}$ follows from Theorem 2. So we first establish a Π_2^p combined complexity upper bound for $\mathrm{BCCQ}_{\mathrm{XML}}$, from which a coNP upper bound for data complexity follows. We then provide a Π_2^p-hardness reduction for $\mathrm{UCQ}_{\mathrm{XML}}$ by coding QSAT with a $\forall^*\exists^*$ quantifier prefix.

We start with the $\mathrm{BCCQ}_{\mathrm{XML}}$ algorithm and assume for now that a query is Boolean, i.e. does not have free variables. We do so for keeping the notation simple. The proof with free variables is essentially the same, and we shall explain the minor changes that need to be made to incorporate free variables at the end.

Given an incomplete tree t and a query $Q \in \mathrm{BCCQ}_{\mathrm{XML}}$ we provide an algorithm deciding $\bot \in \mathit{certain}_{\mathrm{OWA}}(Q,t)$ in Σ_2^p. Towards such an algorithm, assume that we are given an incomplete tree t and a query Q which is a Boolean combination of queries $Q_1,\ldots,Q_m$ such that $\bot \in \mathit{certain}_{\mathrm{OWA}}(Q,t)$. So there exists a function $\chi : \{1,\ldots,m\} \to \{0,1\}$ and a tree T so that:

1. there exists a homomorphism $h : t \to T$,
2. $T \models Q_i$ for each i with $\chi(i) = 1$,
3. $T \models \neg Q_j$ for each j with $\chi(j) = 0$, and
4. setting each Q_i with $\chi(i) = 1$ to true and Q_j with $\chi(j) = 0$ to false makes the Boolean combination evaluate to false.

The function χ will actually become a part of the existential guess in our Σ_2^p algorithm, but notice for now that nothing prevents the size of T from being not polynomial in $|t|$ and $|Q|$. So we will proceed in two stages. Borrowing from techniques developed in [7] and [8], we will first produce another complete tree T_0 satisfying conditions 1 to 4 above. However, the size of T_0 will still possibly be exponential in $|Q|$. So as a last step we will produce a polynomial-size data structure that encodes T_0 in such a way that all the steps of the algorithm, after the existential guess, can be done in polynomial time.

First observe that any collection of conjunctive queries can be represented as an incomplete tree. So we can reformulate conditions 1 and 2 as stating the existence of a homomorphism $h_\chi : t_{\bar{Q},\chi} \to T$, where $t_{\bar{Q},\chi}$ extends t with some incomplete tree representation of the set of all Q_is with $\chi(i) = 1$. Towards the construction of T_0, we now define a subset $sk(T)$ of the nodes in T, which we call the *skeleton with respect to* h_χ, as follows: (1) if a node s is the root of T or belongs to the image of h_χ, then s belongs to $sk(T)$; and (2) if the nodes s_1 and s_2 of T belong to $sk(T)$, then so does their largest common ancestor ("largest" in the order by the tree, i.e., the closest to the nodes). Note that the size of $sk(T)$ is polynomial (in fact, linear) in the size of $t_{\bar{Q},\chi}$. Technically, we should write $sk(T, h_\chi)$ but h_χ will be always clear from the context.

The notion of skeleton was first used in [7], where its size was bounded to obtain complexity results for query answering in data exchange. We now adapt the bounds in [7] to get bounds on the size of T_0. Let $\beta_1,\ldots,\beta_m$ enumerate all the tree-pattern formulae and subformulae that occur in Q and we set $M = 2^m + 1$. We construct T_0 in two stages as follows. We first relabel every node $s \notin Sk(T)$ using one new fresh label (available due to the infinite alphabet assumption) and pairwise distinct fresh constants, so that the arity of the new label does not match any of the arities used in Q (including arities corresponding to the wildcard). We then remove from the so obtained tree all nodes which are neither in $sk(T)$, nor in a vertical or horizontal path in between two nodes in $sk(T)$. Thus, the only nodes in the tree that are not in $sk(T)$ are those on vertical and horizontal path between nodes in $sk(T)$. We finally obtain T_0 by shortening every such vertical or horizontal path to length $M + 5$ if a path was of greater length. Then exact same argument as given in the proof of Lemma 5.7 in [7] shows that $T_0 \not\models Q$. (We note in passing that in that proof, the tree had to conform to a DTD which resulted in a slightly bigger bound, namely multiplied by the size of the alphabet. Since here we relabel the paths with a single new label, such a factor is not needed). Nevertheless, the size of T_0 is still too big to serve for our existential guess.

To overcome this problem, we now turn T_0 into a polynomial-size data structure called $\mathit{code}(T_0)$ as follows. Recall that the only nodes in T_0 that are not in $sk(T)$ are nodes on vertical or horizontal path to length at most $M + 5$ between nodes in $sk(T)$. We call endpoints of such paths *special nodes*. Let N be the set of special nodes. Then $\mathit{code}(T_0)$ contains all the information about T_0 except it does not keep the paths between special nodes. Rather, it keeps the length of such a path in binary, which is bounded by $\log(2^m+6)$,

and thus is polynomial in m. More precisely, for every two special nodes n, n', the structure $code(T_0)$ records the following: whether n' is a descendant or a following-sibling of n such that there is no other special node on the unique path between (i.e., they are consecutive special nodes, in the vertical or horizontal ordering), and in that case, the length of the path between n and n' (encoded in binary). Clearly, given the above observation, the the size of the entire $code(T_0)$ now becomes polynomial in Q.

We next need to show that we can use $code(T_0)$ to verify whether $T_0 \not\models Q$, so that with this exponentially more succinct representation the complexity of this checking is in Π_2^p. Normally one would need to check that there is a homomorphism from t_{Q_i} to T_0 and that there is no homomorphism from the incomplete tree representations t_{Q_j} of the Q_js to T_0. Instead, for $t' \in \{t_{Q_i}, t_{Q_j}\}$ we define a *semi-homomorphism* $h : t' \to code(T_0)$ just as a homomorphism, except that each node of t' is mapped into either:

- a node of $code(T_0)$, or
- a pair of special nodes n, n' such that n' is either a descendant of n or a following sibling of n with no other special nodes between them, and a number k, represented in binary, bounded by $M + 5$.

Such a map h naturally gives rise to a map $h' : t' \to T_0$ as follows. In the second case, the node is mapped by h' into the kth successor of n on the unique – vertical or horizontal – path from n to n'. We then call h a semi-homomorphism iff the map h' is a usual homomorphism from t' to T_0. A key observation is that for a map $h : t' \to code(T_0)$ one can check if it is a semi-homomorphism in polynomial time. Indeed, the information on nodes and offsets is sufficient for checking all the relations $\downarrow, \to, \downarrow^*, \to^*$, and since data values on the exponential paths have been changed, we know that any data value on such a path is different from any other data value in the document.

To sum up, for checking whether $\bot \in certain_{\text{OWA}}(Q, t)$ returns false, it suffices to find a counterexample T_0 which can be encoded by a polynomial-size data structure $code(T_0)$, and then for all Q_js check that there is *no* semi-homomorphism from t_{Q_j} into $code(T_0)$. At the same time, we have to check that there is a semi-homomorphism from t_{Q_i} to $code(T_0)$.

Putting this together, we have a Σ_2^p algorithm for BCCQ certain answer:

1. In the existential step, we guess:
   - a map $\chi : \{1, \ldots, m\} \to \{0, 1\}$;
   - a data structure S of the form $code(T_0)$ of polynomial size;
   - a semi-homomorphism $h_i : t_{q_i} \to S$.
2. In the universal step, we consider all
   - semi-homomorphisms $g_j : t_{q_j} \to S$.
3. Then, in polynomial time, we check:
   - that S is of the form $code(T_0)$ (that is, it is a tree structure, and an offset is associated with every pair of consecutive special nodes in the horizontal or vertical ordering);
   - that all the h_is and g_js are semi-homomorphisms (which we know can be done in polynomial time).

In order to show Π_2^p-hardness of combined complexity of UCQ_{XML}, we proceed by reduction from $\forall\exists$3CNF. An instance of this problem is given as follows by a fully quantified Boolean formula φ in prenex conjunctive normal form:

$$\varphi := \forall p_1 \ldots \forall p_l \exists q_1 \ldots \exists q_m \bigwedge_{1 \le i \le n} (l_{i1} \vee l_{i2} \vee l_{i3}),$$

where each l_{ij} is a literal over the p_i's, q_i's (i.e., an atom or a negation of atom).

Given as input such a formula φ, the problem of deciding whether there exist for each truth assignment of the p_i's, a truth assignment of the q_i's which makes the Boolean formula $\bigwedge_{1 \le i \le n}(l_{i1} \vee l_{i2} \vee l_{i3})$ true, is known to be Π_2^p-complete.

We show that for each $\forall\exists$3CNF-instance φ, there exist an incomplete tree t_φ and a query $q_\varphi^{\text{UCQ}} \in \text{UCQ}_{\text{XML}}$ such that:

$$\varphi \quad \text{is true} \qquad \text{if and only if} \qquad certain(q_\varphi^{\text{UCQ}}, t_\varphi) = \top.$$

We construct t_φ and q_φ^{UCQ} over the following alphabet of node labels:

$$\{root, V, Val, Disj, Lit, Q, P\}.$$

We construct t_φ as follows. The root of t_φ, labeled $root$, has $n+1$ children. The first one is labeled V and has $l+2$ children. The relative sibling order of these children is not specified (i.e., they are given as a simple union). One of these children is labeled $Val(0)$, another one is labeled $Val(1)$ (where 0 and 1 are data values) and for each $1 \leq i \leq l$, there is a child labeled $Val(\bot_{p_i})$. Now the n remaining children of the root form a sequence of siblings where the i^{th} one is labeled $Disj(d_i)$. Moreover, for each node labeled $Disj(d_i)$, we add a child labeled $Lit(l_{ij})$ for the j^{th} literal occurring in the i^{th} conjunct of φ (where each d_i and l_{ij} is a data value occurring nowhere else in the tree). Let l_{ij} denote the j^{th} literal of the i^{th} conjunct, we add descendants to each $Lit(l_{ij})$ labeled node in the following way.

1. Whenever the variable p_i occurring in l_{ij} is universally quantified in φ, we do the following:
   - for every $1 \leq j \leq m$ we add a child labeled $Q(q_j, 1)$ and a child labeled $Q(q_j, 0)$
   - whenever $l_{ij} := p_i$, we add a child labeled $P(\bot p_i, 1)$ and a child labeled $P(0,0)$
   - whenever $l_{ij} := \neg p_i$, we add a child labeled $P(\bot p_i, 0)$ and a child labeled $P(1,1)$
2. Whenever the variable q_i occurring in l_{ij} is existentially quantified in φ, we do the following:
   - we add a child labeled $P(1,1)$ and a child labeled $P(0,0)$
   - whenever $l_{ij} := q_i$, we add two children labeled $Q(q_i, 1)$ and for every $1 \leq j \leq m$ such that $j \neq i$, we add a child labeled $Q(q_j, 1)$ and a child labeled $Q(q_j, 0)$
   - whenever $l_{ij} := \neg q_i$, we add two children labeled $Q(q_i, 0)$ and for every $1 \leq j \leq m$ such that $j \neq i$, we add a child labeled $Q(q_j, 1)$ and a child labeled $Q(q_j, 0)$

Now recall that l is the number of universally quantified propositional variables, m the number of existentially quantified propositional variables and n the number of conjuncts in φ and consider the following conjunctive query:

$$\begin{aligned} q_\varphi := \exists x_1 \ldots \exists x_m (& \bigwedge_{1 \leq i \leq m} (root/V/Val(x_i)) \\ & \wedge \\ & \bigwedge_{1 \leq j \leq n} \exists y (root/Disj(d_j)/Lit(y)/P(0,0) \\ & \wedge \\ & root/Disj(d_j)/Lit(y)/P(1,1) \\ & \wedge \\ & \bigwedge_{1 \leq k \leq l} (root/Disj(d_j)/Lit(y)/Q(q_k, x_k))) \end{aligned}$$

We define:

$$q_\varphi^{\text{UCQ}} := q_\varphi \vee \exists x \exists y \exists z (root/V[Val(x) \to Val(y) \to Val(z)])$$

and now show that given some $\forall\exists$3CNF-instance $\varphi := \forall \bar{p} \exists \bar{q} \bigwedge_{1 \leq i \leq n} (l_{i1} \vee l_{i2} \vee l_{i3})$:

$$\varphi \quad \text{is true} \qquad \text{if and only if} \qquad \text{certain}(q_\varphi^{UCQ}, t_\varphi) = \top.$$

For the first direction assume φ is true. First observe that, without loss of generality, we can restrict to checking the satisfaction of q_φ on structures in $[\![t_\varphi]\!]_{\text{SCWA}}$ where each $\bot_{p_i}$'s has been assigned to either 0 or 1. This is so because q_φ^{UCQ} is monotone and its second disjunct already holds in every structure in $[\![t_\varphi]\!]_{\text{OWA}}$ where a node labeled $Val(\bot_{p_i})$ has not been merged with either the node labeled $Val(0)$ or the node labeled

$Val(1)$. So consider now some $T \in [\![t_\varphi]\!]_{\text{SCWA}}$ where each $\bot_{p_i}$'s has been assigned to either 0 or 1. Such a tree can naturally be associated to a valuation v of the p_i's such that $v(p_i) = 0$ whenever $\bot_{p_i}$ has been assigned to 0 in T and $v(p_i) = 1$ otherwise. As φ is true, the valuation v can be extended to a valuation v^+ of all the propositional variables in φ such that the quantifier-free part of φ is true whenever the p_i's and q_i's are interpreted according to v^+. We interpret each existentially quantified variable x_i in q_φ by $v^+(q_i)$ and show that T satisfies q_φ with these values. Firstly, notice that the value assigned by v^+ to each q_i is either 0 or 1 and consequently for every $1 \leq i \leq m$, $T \models root/V/Val(v^+(q_i))$. Hence, the first big conjunction in q_φ holds for the values of the x_i's that we considered. Secondly, let us pick one j such that $1 \leq j \leq n$. Consider now the j^{th} conjunct of φ. By assumption there is a literal in this conjunct which is true for the valuation v^+. Assume it is the k^{th} literal. We will show that T satisfies the remaining conjuncts of q_φ (where y is interpreted by l_{jk} and each x_i is interpreted in T by $v^+(q_i)$). Observe that the truth value of these conjuncts can be evaluated by restricting to the root and to the subtree rooted at its child labeled $lit(l_{jk})$. This subtree was constructed as an encoding of the k^{th} literal of the j^{th} conjunct of φ in the way explained earlier. By construction of t_φ, there were two cases (again, we let l_{jk} stand for that literal):

1. the variable p_r in l_{jk} was universally quantified in φ:
   - $\bigwedge_{1\leq k\leq l}(root/Disj(d_j)/Lit(l_{jk})/Q(q_k, x_k)))$ holds because whatever is the value $v^+(q_i)$ assigned to each x_i (that is, either 0 or 1), by construction of t_φ, there is a child of the node labeled $lit(l_{jk})$ which is labeled $Q(q_i, v^+(q_i))$
   - whenever $l_{jk} := p_r$, $T \models root/Disj(d_j)/Lit(l_{jk})/P(0,0)$ holds by construction of t_φ. Also $T \models root/Disj(d_j)/lit(l_{jk})/P(1,1)$ because $v^+(\bot_{p_i}) = 1$
   - whenever $l_{jk} := \neg p_r$, $T \models root/Disj(d_j)/Lit(l_{jk})/P(1,1)$ holds by construction of t_φ. Also $T \models root/Disj(d_j)/lit(l_{jk})/P(0,0)$ because $v^+(\bot_{p_i}) = 0$
2. the variable q_r in l was existentially quantified in φ:
   - $T \models root/Disj(d_j)/Lit(l_{jk})/P(0,0)$ holds by construction of t_φ
   - $T \models root/Disj(d_j)/Lit(l_{jk})/P(1,1)$ holds by construction of t_φ.
   - $\bigwedge_{1\leq k\leq l}(root/Disj(d_j)/Lit(l_{jk})/Q(q_k, x_k)))$ holds because whatever is the value $v^+(q_i)$ assigned to each x_i, by construction of t_φ, there is a child of the node labeled $lit(l_{jk})$ which is labeled $Q(q_i, v^+(q_i))$. This holds for every q_i with $i \neq r$ because for every such i there are both a node labeled $Q(q_i, 0)$ and a node labeled $Q(q_i, 1)$. Whenever $l_{jk} := q_r$, this holds in the case of q_r because as $v^+(q_r) = 1$, we also assumed that x_r is evaluated by 1. Finally whenever $l_{jk} := \neg q_r$, this holds because as $v^+(q_r) = 0$, we also assumed that x_r is evaluated by 0

Now for the converse direction assume $\varphi := \forall \bar{p} \exists \bar{q} \bigwedge_{1\leq i\leq n}(l_{i1} \vee l_{i2} \vee l_{i3})$ is false, then there is a valuation v of the p_i's such that for every extension v^+ of v to the q_i's, the formula $\bigwedge_{1\leq i\leq n}(l_{i1} \vee l_{i2} \vee l_{i3})$ evaluates to $\bot$ whenever the p_i's, q_i's are interpreted according to v^+. Let $T \in [\![t_\varphi]\!]_{\text{SCWA}}$ where for every p_i, $\bot_{p_i}$ is assigned to 0 whenever $v(p_i) = 0$ and to 1 otherwise. Also, the V-labeled node has only two children, so that the second disjunct of q_φ^{UCQ} does not hold on T. Now assume $T \models q_\varphi^{\text{UCQ}}$, so $T \models q_\varphi$ and there is a mapping of the existentially quantified variables $x_1, \ldots, x_m$ to $\{0, 1\}$ (which are the only two data values d such that $T \models root/V/Val(d)$) such that the big conjunction with n conjuncts is true for this valuation of the x_i's. This means that for each data value d_j occurring in t_φ, there exists some k such that the remaining of the formula holds on T whenever y is interpreted by l_{jk} (such l_{jk} values being the only ones to which we can set this existentially quantified variable). Let us fix one such valuation of the x_i's and notice that it corresponds to an extension v^+ of v to the q_i's. Now as φ is false, it follows that there is a conjunct of φ, let say the j^{th} one, which evaluates to $\bot$ under the valuation v^+, that is, every literal in that conjunct evaluates to $\bot$ under v^+. Now by construction of t_φ, the j^{th} conjunct is not satisfied under the valuation of the x_i's considered. As this holds for any extension v^+ of v, it follows that $T \not\models q_\varphi^{\text{UCQ}}$.

As the size of q_φ^{UCQ} and t_φ is polynomial in the size of φ, this completes the proof of the reduction. □

5.2 Query answering under CWA

We now move to the closed world assumption. The results are easily obtained as corollaries of previously established results. Recall that for arbitrary incomplete trees, there are two possible interpretations of the

	data complexity		combined complexity	
	CWA	OWA	CWA	OWA
UCQ_{XML}	PTIME		NP-complete	
BCCQ_{XML}			Π_2^p-complete	
FO_{XML}	coNP-complete	undecidable	PSPACE-complete	undecidable

Fig. 6 Complexity of computing certain answers over rigid incomplete trees

closed world assumption. 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 $\Rightarrow$ or by $\Downarrow$. Our first result is about the weak interpretation.

Corollary 1 *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 3.*

Indeed, OWA upper bounds trivially apply, and one can examine the proofs of OWA hardness results to observe that they 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 FO_{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 2 *Over arbitrary incomplete trees under* SCWA*, for each of the languages* $\text{UCQ}_{\text{XML}}, \text{BCCQ}_{\text{XML}}$*, data complexity is* coNP*-complete and combined complexity is* Π_2^p*-complete. For* FO_{XML}*, data complexity remains* coNP*-complete, while combined complexity is* PSPACE*-complete.*

For the 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 [8]. 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 FO_{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 UCQ_{XML} and BCCQ_{XML}. Combined complexity is elementary (in fact at most 2-exponential) despite previous high bounds (except for FO_{XML} of course where it is undecidable).
2. For arbitrary trees, closed world assumption – in either form – does not help bring down complexity for conjunctive queries and their relatives.

So, as in [8], this motivates looking at the restricted case of rigid trees, for all of the languages and assumptions. This is what we do next.

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

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

6.1 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 [8]. 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 $certain_{\mathrm{OWA}}(Q,t)$. By translation into relational representation, this implies NP-completeness of combined complexity. We now complete this picture.

Theorem 4 *Under* OWA*, data complexity of* $\mathrm{BCCQ}_{\mathrm{XML}}$ *queries over rigid trees is in* PTIME*, while for* $\mathrm{FO}_{\mathrm{XML}}$ *it is undecidable.*

Combined complexity is NP*-complete for* $\mathrm{UCQ}_{\mathrm{XML}}$ *and* Π_2^p*-complete for* $\mathrm{BCCQ}_{\mathrm{XML}}$.

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.

PROOF. Upper bounds for combined complexity follow from earlier results, while hardness results for combined complexity are already witnessed in the relational case [25]. Likewise, undecidability for $\mathrm{FO}_{\mathrm{XML}}$ follows from the relational case. Thus, we concentrate on proving tractability of data complexity.

A *homomorphism* $h : t \to t'$ from an incomplete tree $t = \langle N, V, \downarrow, \Downarrow, \rightarrow, \Rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$ to another incomplete tree $t' = \langle N', V', \downarrow, \Downarrow, \rightarrow, \Rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$ is a pair of maps $h = (h_1, h_2)$ where $h_1 : N \to N'$ and $h_2 : V \to V'$ such that:

- if wRw' in t, then $h_1(w)Rh_1(w')$ in t', when R is one of $\downarrow, \rightarrow, \Downarrow, \Rightarrow$;
- if $w \in P_\alpha$ in t, then $h_1(w) \in P_\alpha$ in t';
- $h_2(c) = c$ whenever $c \in \mathcal{C}$; and
- $h_2(\rho(w)) = \rho(h_1(w))$ for each $w \in N$ such that, either there is $\alpha \in \Sigma$ with $w \in P_\alpha$, or $\rho(w) \neq \emptyset$.

Our PTIME algorithm shares common features with the relational case, but we still need to adapt the definitions introduced in the proof of Theorem 1, especially the notion of f-compatibility.

In the relational case, we were concerned with "merging" different tuples. Here we will not only need to merge tuples, but also the tree nodes to which they are associated (as well as later on extending incomplete tree structures to proper tree structures). For that purpose, we first need a few definitions.

Let $t = \langle N, V, \downarrow, \Downarrow, \rightarrow, \Rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$ be an incomplete tree, we let:

$$\mathcal{H}_t = \{h \mid h : N \to N \text{ is a } \textit{root faithful} \text{ node homomorphism}\}.$$

We say that a node homomorphism h is *root faithful* whenever every node labeled with the special label *root* is mapped to one and the same image by h.

Given $h \in \mathcal{H}_t$, we inductively define a relation $\sim_h$ over $\mathrm{adom}(t)$ as follows:

- $x \sim_h y$ whenever there exist $w, w' \in N$ with $h(w) = h(w')$ and $1 \leq j \leq m$ such that $\rho(w) = (x_1, \ldots, x_m), \rho(w') = (y_1, \ldots, y_m), x = x_j$ and $y = y_j$;
- $\sim_h$ is reflexive, symmetric and transitive.

We say that t is h-consistent whenever:

- for every $a, b \in \mathcal{C}$, if $a \neq b$, then $a \not\sim_h b$,

- for every $w, w' \in N$ with $h(w) = h(w')$:
  - for every $R, R' \in \Sigma$, $w \in P_R$ and $w' \in P_{R'}$ implies $R = R'$,
  - $|\rho(w)| \neq |\rho(w')|$ implies that there is $w_i \in \{w, w'\}$ such that $|\rho(w_i)| = \emptyset$ and for all $R \in \Sigma$, $w_i \notin P_R$ (i.e., w_i is labeled with _).

As in the relational case, we observe that $\sim_h$ yields a set of equivalence classes over adom(t), as follows:

$$x \in [y]_h \text{ if and only if } x \sim_h y$$

We then consider the quotient set of adom(t) under $\sim_h$:

$$\text{adom}(t)/\sim_h := \{eq_1, \ldots, eq_r\}.$$

Now for every $eq_i \in \text{adom}(t)/\sim_h$, we define the representative $r_h(eq_i)$ of eq_i by letting:

- $r_h(eq_i) = a$ whenever there exists $a \in eq_i$ such that $a \in \mathcal{C}$,
- $r_h(eq_i) = \perp_i$ otherwise.

For every $w \in N$, we also define the representative $r_h(w)$ of w by letting $r_h(w) = r_h([y_1]_h), \ldots, r_h([y_m]_h)$, where there is some $w' \in N$ such that $h(w) = h(w')$, $\rho(w') = (y_1, \ldots, y_m)$ and for every $w'' \in N$ such that $h(w'') = h(w)$, if $\rho(w'') = (y'_1, \ldots, y'_{m'})$, then $m \geq m'$.

Let $h \in \mathcal{H}_t$ such that t is h-consistent. We define a new incomplete tree t^h as follows:

$$t^h \;=\; \langle Im(h), V, \downarrow\restriction Im(h), \Downarrow\restriction Im(h), \rightarrow\restriction Im(h), \Rightarrow\restriction Im(h), (P^h_\alpha)_{\alpha\in\Sigma}, \rho^h\rangle,$$

where

- for every $Edge \in \{\downarrow, \Downarrow, \rightarrow, \Rightarrow\}$, $Edge \restriction Im(h)$ is the restriction of the relation $Edge$ to the nodes in $Im(h)$;
- P^h_α's are disjoint subsets of $N \restriction Im(h)$ where $P^h_\alpha = \{w \in Im(h) \mid \exists w' \in N$ such that $h(w) = h(w')$ and $w' \in P_\alpha\}$; and
- $\rho^h : N \restriction Im(h) \to \bigcup_{k\geq 0} V^k$ is defined is defined by $\rho(w) = r_h(w)$.

Observe that for every $h \in \mathcal{H}_t$ such that t is h-consistent, there is a homomorphism $\mathbf{h} : t \to t^h$ with $\mathbf{h} = (\mathbf{h}_1, \mathbf{h}_2)$, where $\mathbf{h}_1(w) = h(w)$ and $\mathbf{h}_2(x) = r_h([x]_h)$.

Observe that nothing prevents the incomplete trees t_hs defined above to be not only non rigid, but also disconnected. As we are eventually interested in generating a set of rigid trees on which we know from [8] that naïve evaluation can be applied safely, we finally provide one further definition. Intuitively, we will use it to "represent" every complete tree in the OWA semantics of t_h up to a certain size (bounded by $|Q|$)).

Let $t = \langle N, V, \downarrow, \Downarrow, \rightarrow, \Rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$ be an incomplete tree and let n be a natural number. We define the set of *n-rigidifications of t* as follows:

$$t' = \langle N^+, V, \downarrow^+, \Downarrow^+, \rightarrow^+, \Rightarrow^+, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle \in \mathcal{R}_n(t) \text{ if and only if}$$

- $\langle N^+, \downarrow^+, \Downarrow^+, \rightarrow^+, \Rightarrow^+\rangle$ is a tree structure ;
- there is an embedding $f : N \to N'$ that preserves:
  - the tree structure, i.e., for all $R \in \{\downarrow, \Downarrow, \rightarrow, \Rightarrow\}$, wRw' implies $f(w)R^+f(w')$,
  - the function ρ, i.e., $\rho(w) = \rho(f(w))$,
  - labeling, i.e., for all $w \in N$, $P_\alpha(w) \Leftrightarrow P_\alpha(f(w))$;
- define $sk(f(N))$ as the closure of $f(N)$ under largest common ancestor, then:
  - $w \in sk(f(N))$ and $w \notin f(N)$ implies that w is labeled with wildcard and has no data value,
  - for every node $w \in N' - sk(f(N))$, w is at the vertical or horizontal distance $\leq n$ from a node in $sk(f(N))$, w is labeled with wildcard and has no data value.

We are now ready to define our OWA algorithm. Let $Q \in \text{BCCQ}_{\text{XML}}$ be a Boolean query and let t be a rigid incomplete tree (assume w.l.o.g. that the conjunctive queries in Q share no variable in common). We will define an algorithm which determines in polynomial time in the size of t whether $\bot \in \mathit{certain}_{\text{OWA}}(Q, t)$. As in the relational case, we begin by listing all possible assignments of the Boolean conjunctive queries in Q to Boolean values for which Q evaluates to $\bot$. Now for every such assignment v, we let:

$$\alpha_v := \bigwedge_{i \in I} q_i \text{ where } I = \{i \mid q_i \text{ occurs in } Q \text{ and } v(q_i) = \top\}$$

$$\beta_v := \bigvee_{j \in J} q_j \text{ where } J = \{j \mid q_j \text{ occurs in } Q \text{ and } v(q_j) = \bot\}$$

Every Boolean conjunctive query $Q := \exists \bar{x}\ \pi_1(\bar{x}) \wedge \ldots \wedge \pi_n(\bar{x})$ can also be represented as an incomplete tree $Tr(Q)$. Without loss of generality assume that the set of nodes and the set of nulls occurring in t and $Tr(\alpha_v)$ are disjoint. Now form a new incomplete tree t_v by taking the union of these two structures and by merging in it all nodes labeled with the special label *root*, or by adding one new *root*-labeled node if there was no occurrence of this label.

We can now define:

$$\mathcal{T}^v_{\text{OWA}} := \{(t^v)^h \mid h \in \mathcal{H}_{t^v} \text{ and } t^v \text{ is } h\text{-consistent}\}.$$

The open world algorithm proceeds as follows. First, let $\beta_1, \ldots, \beta_m$ enumerate all the tree-pattern formulae and subformulae that occur in Q and set $M = 2^m + 6$. Now pick a falsifying valuation v of the conjunctive queries in Q. For every $(t^v)^h \in \mathcal{T}^v_{\text{OWA}}$, consider each $t' \in \mathcal{R}_M((t^v)^h)$ and determine using naive evaluation whether $t' \not\models \beta_v$. If this is the case for some t', then stop and return $\bot$. If not, iterate the procedure for every falsifying valuation v. Once done, if no valuation v and trees $(t^v)^h \in \mathcal{T}^v_{\text{OWA}}$, $t' \in \mathcal{R}_m((t^v)^h)$ such that $t' \not\models \beta_v$ were found, then return $\top$.

Before we prove the correctness of the algorithm, let us first show that it runs in polynomial time in $|t|$. So let n be the number of nodes in t and k the number of nodes in t^v that did not come from t. Note that k is bounded by the size of the query. Also observe that there are at most $(n+k)^k$ structures in $\mathcal{T}^v_{\text{OWA}}$; thus, the number of such structures is polynomial in the size of t, and they all can be enumerated in polynomial time, assuming the query is fixed.

As the next step in the complexity analysis, we have to show that the number of M-rigidifications of each structure $(t^v)^h$ is polynomial in t, and that they can all be constructed in polynomial time. This will give us the desired polynomial bound.

We can assume without loss of generality that h is the identity on the nodes of t, due to rigidity. Let p be the number of nodes in all tree representations of all patterns added in t^v. We now over count the number of rigidification by assuming one just adds p such nodes (the actual number will be smaller as there are constraints on how these p nodes can be added, due to the structure of patterns in the query).

We construct a rigidification in p steps, having a rigid tree in each of those steps. Let the trees be $T_0, \ldots, T_p$ with T_0 being t itself. In each step i we pick the ith node x_i to be added and do the following:

1. Pick a node y in T_{i-1} that x_i will be the descendant of, so that x_i is not a descendant of any other node that is a descendant of y.
2. Choose the length of the path from y to x_i; it must be at most M.
3. Consider y', the child of y in the direction of x_i. We know it could not have been in T_{i-1}. Decide whether it is on the left or on the right of other children of y, and the length of the path from y' to either the first child of y or the last child of y; the length must be at most M.
4. Label all newly introduced nodes with wildcard.

In each such step we add at most $2M$ nodes to a tree, so the size of all the T_is is bounded by $2Mnp$, where n is the size of t. The number of possible trees in the $i+1$st step is obtained by multiplying the number of possible trees in the ith step by:

– the size of T_i;
– M to account for the length of the vertical path;
– 2 to account for the choice in item 3) above;
– M again to account for the length of the horizontal path.

Thus if τ_i is the number of such possible trees, then we have the condition $\tau_0 = 1$ and $\tau_{i+1} \leq \tau_i \cdot 2Mnp \cdot 2M^2$. Hence $\tau_p \leq (4M^3np)^p$ which is still polynomial in the size of the tree, since M and p depend on the query only.

The process of calculating this bound also shows how to enumerate all rigidifications in polynomial time: the only difference is that we do addition by trees, rather than individual nodes (so the number would actually be smaller).

In order to show the correctness of this algorithm, we finally show that $\mathit{certain}_{\mathrm{OWA}}(Q,t) = \bot$ if and only if there exists a valuation v of the conjunctive queries in Q and trees $(t^v)^h \in \mathcal{T}^v_{\mathrm{OWA}}$, $t' \in \mathcal{R}_M((t^v)^h)$ such that $t' \not\models \beta_v$.

The right to left direction easily follows from [8]: it is enough to remark that $t' \models \alpha_v$, there is a homomorphism from t to t' and t' is rigid. So we focus on the left to right direction. Assume $\mathit{certain}_{\mathrm{OWA}}(Q,t) = \bot$, i.e., there exists a complete tree $T \in [\![t]\!]_{\mathrm{OWA}}$ such that $T \not\models Q$. By the proof of Theorem 3 we can assume that there is a falsifying valuation v of the conjunctive queries in Q such that:

– $T \models \alpha_v \wedge \neg\beta_v$,
– there is a homomorphism $h : t^v \to T$, where $h = (h_1, h_2)$,
– the only nodes in T that are not in $sk(T)$ are those in between vertical and horizontal paths between nodes in $sk(T)$; additionally every node in T which is not in $h(T)$ satisfies similar labeling conditions as the tree T_0 constructed in the proof of Theorem 3.

We first show that t^v is h_1-consistent, which entails the existence of some incomplete tree $(t^v)^{h_1}$. We finally observe that there is some $t' \in \mathcal{R}_M((t^v)^{h_1})$ such that $t' \not\models \beta_v$.

Recall that t^v is h_1-consistent whenever:

– for every $a, b \in \mathcal{C}$, if $a \neq b$, then $a \not\sim_{h_1} b$,
– for every $w, w' \in N$ with $h_1(w) = h_1(w')$:
  – for every $R, R' \in \Sigma$, $w \in P_R$ and $w' \in P_{R'}$ implies $R = R'$,
  – $|\rho(w)| \neq |\rho(w')|$ implies that there is $w_i \in \{w, w'\}$ such that $|\rho(w_i)| = \emptyset$ and for all $R \in \Sigma$, $w_i \notin P_R$ (i.e., w_i is labeled with $\_$).

The second condition immediately follows from the fact that h is a homomorphism. For the first condition, we show that for every $x, y \in \mathrm{adom}(t^v)$, $x \sim_{h_1} y$ implies that $h_2(x) = h_2(y)$, from which it follows immediately that if $x, y \in \mathcal{C}$, then $x = y$ (as h is a homomorphism). So let $x, y \in \mathrm{adom}(t^v)$ such that $x \sim_{h_1} y$. This means that there is a sequence $x_0, \ldots, x_n \in \mathrm{adom}(t^v)$ such that $x = x_0$, $y = x_n$ and for every x_i, x_{i+1} with $0 \leq i \leq n$, the following holds:

$$\exists w, w' \in \mathit{dom}(t^v),\ \exists k \leq m \text{ such that,}$$

$$h_1(w) = h_1(w'), \rho(w) = (y_1, \ldots, y_m), \rho(w') = (z_1, \ldots, z_m), x_i = y_k \text{ and } x_{i+1} = z_k$$

We show the property (i.e., $h_2(x_0) = h_2(x_n)$) by induction on the length of n. So consider a sequence of length n as described and assume $h_2(x_0) = h_2(x_k)$ holds for all $k < n-1$. Now consider x_{n-1} and x_n, by assumption we have:

$$\exists w, w' \in \mathit{dom}(t^v),\ \exists k \leq m \text{ such that,}$$

$$h_1(w) = h_1(w'), \rho(w) = (y_1, \ldots, y_m), \rho(w') = (z_1, \ldots, z_m), x_{n-1} = y_k \text{ and } x_n = z_k$$

By $h_1(w) = h_1(w')$ and by h being a homomorphism, it follows that $h_2(\rho(w)) = h_2(\rho(w'))$ and so $h_2(x_{n-1}) = h_2(x_n)$. By induction hypothesis, we know that $h_2(x_0) = h_2(x_{n-1})$, from which it follows that $h_2(x_0) = h_2(x_n)$.

As t^v is h_1-consistent, this entails the existence of a corresponding incomplete tree $(t^v)^{h_1}$. Observe that there is a homomorphism $h' : (t^v)^{h_1} \to T$, where $h' = (h'_1, h'_2)$ is given by $h'_1(w) = h_1(w)$ and $h'_2(r_{h_1}[x]_{h_1}) = h_2(x)$. Now assume $(t^v)^{h_1} \models \beta_v$. Conjunctive queries being preserved by homomorphism, it follows that $T \models \beta_v$, which is a contradiction. Hence $(t^v)^{h_1} \not\models \beta_v$.

We finally derive from the fact that $h' : (t^v)^{h_1} \to T$ is a homomorphism, that we can construct out of T a rigid incomplete tree $t'_m \in \mathcal{R}_m((t^v)^{h_1})$ such that $t' \not\models \beta_v$. The construction is achieved very simply by removing all data values and labels attached to each of the nodes which do not belong to $h(T)$, replacing removed labels with wildcard occurrences. □

6.2 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.

Corollary 3 *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.*

The proof proceeds exactly like in the previous case; we only need to notice that due to CWA, there are fewer trees to consider, and thus the complexity does not go up. All the lower bounds, including those for $\mathrm{FO}_{\mathrm{XML}}$, are already witnessed by the relational case.

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]).

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 [13].

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 in terms of complexity classes characterization (except for the strongest language $\mathrm{FO}_{\mathrm{XML}}$), but the algorithm under OWA appears to be more manageable.

7 Conclusions

The results of this paper, together with [8], present a rather complete picture of both data and combined complexity of query answering over incomplete XML documents. 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 as far as complexity classes are concerned; however, even for cases when algorithms are in polynomial time, there could be added complexities in some cases.
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 considering Boolean combinations works as well. This result also filled a gap in our knowledge of relational query answering over incomplete databases.

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

We also remark that the proceedings version of this paper [16] made some claims about extensions to queries with inequalities. While all the lower bounds mentioned in [16] are correct (as follows from the results presented here), the exact complexity of finding certain answers for queries with inequalities is open at this time and left for future work. We also remark in passing that such problems become undecidable if, in addition to inequalities, very simple schema constraints are imposed (for instance, just a finite alphabet of labels) [14].

Acknowledgment We are grateful to Tony Tan for extensive discussions, and for help with pictures. Work partially supported by EPSRC grants G049165 and J015377.

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. L. Antova, T. Jansen, C. Koch, D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE'08*, pages 983–992.
6. M. Arenas, P. Barceló, L. Libkin, F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.
7. M. Arenas and L. Libkin. XML data exchange: consistency and query answering. *J. ACM*, 55 (2), 2008.
8. P. Barceló, L. Libkin, A. Poggi, C. Sirangelo. XML with incomplete information. *J. ACM*, 58:1 (2010).
9. H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. *J. Comput. Syst. Sci.* 77(3): 450-472 (2011).
10. 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.
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. C. David, L. Libkin, F. Murlak. Certain answers for XML queries. In *PODS 2010*, pages 191-202.
14. C. David, A. Gheerbrant, L. Libkin, W. Martens. Containment of pattern-based queries over data trees. In *ICDT 2013*, pages 201–212.
15. T. Eiter, G. Gottlob, H. Mannila. Disjunctive datalog. *ACM Trans. Database Syst.* 22(3):364-418 (1997).
16. A. Gheerbrant, L. Libkin, T. Tan. On the complexity of query answering over incomplete XML documents. In *ICDT 2012*, pages 169–181.
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. Reiter. On closed world databases. In *"Logic and Databases"*, H. Gallaire and J. Minker eds, Plenum Press, 1978, pages 55–76.
24. 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).
25. Y. Sagiv, M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM* 27(4): 633-655 (1980).
26. D. Suciu, D. Olteanu, C. Re, C. Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.
27. R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *J. Comput. Syst. Sci.* 54(1): 113-135 (1997).