Static Analysis and Query Answering for Incomplete Data Trees with Constraints

Static Analysis and Query Answering for Incomplete Data Trees with Constraints

Amélie Gheerbrant[1,2], Leonid Libkin[1], and Juan Reutter[1,3]

[1] School of Informatics, University of Edinburgh
[2] LIAFA (Université Paris Diderot - Paris 7 & CNRS)
[3] Department of Computer Science, Pontificia Universidad Catolica de Chile

Abstract. Data trees serve as an abstraction of XML documents: in such trees, every node comes with a label from a finite alphabet, as well as a data value from an infinite set. Incomplete data trees model XML documents with incomplete information; they may include both structural incompleteness and incompleteness of data. Here we study two basic problems for incomplete data trees under typical constraints such as keys and foreign keys. The first problem is consistency of specifications of incomplete data trees. We show that many of recently established results on consistency of constraints and schema descriptions can be transferred to the consistency of incomplete tree specifications without any increase in complexity. After that we examine query answering over incomplete data trees under constraints, and show that tractable bounds can be recovered under key constraints, but are lost under foreign keys.

1 Introduction

In this paper we examine two basic problems about XML documents with incomplete information: namely their consistency (or satisfiability), and query answering. The first problem asks whether a description of an incomplete document is consistent, under some schema restrictions: that is, whether a completion satisfying the schema requirement exists. The second problem is to find certain answers, i.e., answers independent of a particular interpretation of missing features of the incomplete document. These are standard data management problems and they have been studied extensively, in particular in the context of incomplete XML documents. Our *main contribution* here is to study them when the schema description contains constraints commonly found in databases, such as keys and foreign keys.

Traditional XML schema descriptions, such as DTDs, can be subsumed by the power of tree automata [22]; such automata operate on trees labeled with letters from a finite alphabet. Constraints such as keys and foreign keys, on the other hand, talk about *data* in XML documents. Since data values typically come from infinite domain (e.g., numbers, or strings), they cannot be captured by traditional automata.

The interplay between finiteness of the description of the structure of XML document and the infinite domains of data such document carry has been a central theme in XML research. A typical object of investigation is the abstraction of XML documents known as *data trees*: these are finitely-labeled trees that can carry data from an infinite set. Now we apply some of the developed techniques to the study of such data trees with incomplete information. In the rest of the introduction, we explain briefly what incomplete data trees are, and the two main problems we study.

Incomplete XML documents We follow a general approach to incompleteness in XML described in [3]. In relational databases, incompleteness is usually modeled via null values, which may appear in place of constants. In XML, due to its more complex structure, two other types of incompleteness may appear:

– structural incompleteness: precise relationships between some nodes may not be known (for instance, we may know that one node is a descendant of another without knowing the full path between them);
– labeling incompleteness: some node labels can be replaced by wildcards, indicating that the exact label is not known at present.

Consistency problem For usual XML documents with incomplete information, the consistency problem asks whether such an incomplete description t can represent a complete tree T satisfying some schema constraints, typically expressed by a tree automaton $\mathcal{A}$. Most versions of this problem range from tractable to being NP-complete [3].

A different type of consistency problems often arises in the study of *static analysis* of XML. A typical formulation is as follows: we are given some schema information, say an automaton $\mathcal{A}$, and some constraints Δ involving data values (e.g., keys, foreign keys). The question is whether there is an XML document that conforms to the schema (is accepted by $\mathcal{A}$) and satisfies Δ.

Simplest versions of this problem (for Δ containing unary keys and foreign keys, for instance) are known to be NP-complete [13], but by now many other variations exist, e.g., [2, 6–8, 12, 23, 24]. Reasoning tasks can be of varying complexity, starting from NP and going up to high but elementary [2, 7] or extremely high (e.g., non-primitive-recursive [14, 15]) and even undecidable (e.g., binary keys and foreign keys [13]).

Our consistency problem is different from a pure static analysis, as it takes an incomplete data tree t as an input, together with the static information such as $\mathcal{A}$ and Δ. Our result on the consistency problem is that, under mild assumptions, the complexity of static analysis tasks for complete XML documents applies to the analysis of incomplete documents. In other words, we show how incomplete documents can be added into static analysis tasks without any increase in computational complexity. Note of course that an incomplete tree t can be encoded as an automaton, so in principle static analysis can be extended to handle data. However, an automaton encoding t may well be of exponential size in t, and to prove our result we need to find a way around this exponential blow up.

Query answering The standard approach to answering queries over databases with incomplete information is to look for *certain answers*, i.e., answers independent of how particular incomplete features are interpreted. Here we look at analogs of (unions of) conjunctive queries for XML. In the relational case, these can be evaluated in polynomial time [18], but in the XML case, their complexity can range from polynomial time to coNP-complete [3]. However, it was shown in [3] that their complexity drops back to polynomial time in the case of *rigid* trees, that do not allow any structural incompleteness. This is true without constraints, but the relational case teaches us that constraints are likely to change the complexity of query answering [10].

We show here that this is true in the XML case too: when we allow just a single unary inclusion constraint, finding certain answers over rigid trees jumps to coNP-complete, but with keys it stays in polynomial time.

Organization In Section 2 we describe data trees and integrity constraints. In Section 3 we describe the model of incomplete XML documents (or data trees). In Section 4 we study the consistency problem, and in Section 5 we present our results on query answering.

2 Preliminaries

Data trees and automata To describe data trees, we assume

– a countably infinite set $\mathcal{C}$ of possible data values (notation $\mathcal{C}$ stands for "constants"; later we shall extend data trees to domains that contain both constants and nulls), and
– a countably infinite set $\mathcal{L}$ of node labels (element types). We shall normally denote labels by lowercase Greek letters.

A data 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 a finite 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
– $\rho : D \rightarrow A$ assigns to each node $w \in D$ a data value.

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 $v = \rho(w)$. We denote by $V_\alpha(T)$ the set of all data values assigned to α-nodes by ρ. That is, $V_\alpha(T) = \{\rho(w) \mid P_\alpha(w) \text{ holds}\}$.

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

An *unranked tree automaton* [26] over Σ is a tuple $\mathcal{A} = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \to 2^{Q^*}$ is a transition function. We require all $\delta(q, \alpha)$'s to be regular languages over alphabet Q, for all $q \in Q$ and $\alpha \in \Sigma$.

A *run* of $\mathcal{A}$ over a tree T is a function $\tau_{\mathcal{A}} : \text{dom}(T) \to Q$, such that for each node w that is labeled α and has k children $w \cdot 0, \ldots, w \cdot (k-1)$, the word $\tau_{\mathcal{A}}(w \cdot 0) \cdots \tau_{\mathcal{A}}(w \cdot (k-1))$ belongs to the language of $\delta(\tau_{\mathcal{A}}(w), \alpha)$. In particular, if w is a leaf, then the empty word belongs to $\delta(\tau_{\mathcal{A}}(w), \alpha)$. A run is accepting if $\tau_{\mathcal{A}}(\epsilon) \in F$, that is, if the root of T is assigned a final state. As customary, we denote the language of all trees accepted by $\mathcal{A}$ by $L(\mathcal{A})$.

XML Integrity Constraints We consider keys, inclusion constraints and foreign keys as our basic integrity constraints. They are the most common constraints in relational databases, and are common in XML as well, as many documents are generated from databases. Moreover, these sets of constraints are similar to, but more general than XML ID/IDREF specifications, and can be used to model most of the key/keyref specifications of XML Schema used in practice [20, 19]. Here we only deal with constraints specified with element types rather than those specified by paths, as is done, for instance, in [2, 9].

Let $\Sigma \subset \mathcal{L}$. Then a *basic XML constraint* over Σ is one of the following:

– A *key* constraint $\mathit{key}(\alpha)$, where $\alpha \in \Sigma$. An XML tree T satisfies $\mathit{key}(\alpha)$, denoted by $T \models \mathit{key}(\alpha)$ iff for every distinct α-nodes n and n' in T, we have $\rho(n) \neq \rho(n')$, i.e., the data values on n and n' are different.
– An *inclusion constraint* $\alpha_1 \subseteq \alpha_2$, where $\alpha_1, \alpha_2 \in \Sigma$. This constraint is satisfied, i.e., $T \models \alpha_1 \subseteq \alpha_2$, iff $V_{\alpha_1}(T) \subseteq V_{\alpha_2}(T)$.
– A *foreign key*: A combination of an inclusion constraint and a key constraint, namely $\alpha_1 \subseteq_{\text{FK}} \alpha_2$ holds iff $\alpha_1 \subseteq \alpha_2$ and $\mathit{key}(\alpha_2)$ both hold.

3 XML with incomplete information

To define incomplete XML documents, we assume a countably infinite supply of null values (or variables) $\mathcal{V}$. Following [3, 16], 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 data tree* over Σ is a 2-sorted structure

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

where

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

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.

Rigid trees An incomplete tree $t = \langle N, V, \downarrow, \Downarrow, \rightarrow, \Rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle$ is called *rigid* [3] if its pure "structural part", i.e., $t_0 = \langle N, \downarrow, \rightarrow, (P_\alpha)_{\alpha\in\Sigma}\rangle$ is a labeled unranked tree with wildcards. That is, N is an unranked tree domain, the P_α's are disjoint subsets of N, and for nodes n, n' we have

- $n \downarrow n'$ iff $n' = n\cdot i$ for some $i \in \mathbb{N}$;
- $n \rightarrow n'$ iff $n = w\cdot i$ and $n' = w\cdot(i+1)$ for some $w \in \mathbb{N}^*$ and $i \in \mathbb{N}$.

In other words, in rigid trees we do not permit any structural incompleteness regarding the axes $\downarrow, \rightarrow, \Downarrow$, and $\Rightarrow$ (the axes $\Downarrow$ and $\Rightarrow$ will always be interpreted as the transitive closures of $\downarrow$ and $\rightarrow$ respectively). The only allowed types of incompleteness are nulls for data values, and wildcards.

Semantics As is common with incomplete information, we define semantics via homomorphisms $h : t \rightarrow T$ from an incomplete data tree t to a complete data tree T. A *homomorphism*

$$h : \langle N, V, \downarrow, \Downarrow, \rightarrow, \Rightarrow, (P_\alpha)_{\alpha\in\Sigma}, \rho\rangle \quad\longrightarrow\quad \langle D, A, \downarrow, \rightarrow, (P_\alpha)_{\alpha\in\Sigma'}, \rho\rangle,$$

where $\Sigma \subseteq \Sigma'$, is a map from $N \cup V$ to $D \cup A$ such that

- $h(n) \in D$ if $n \in N$ and $h(v) \in A$ if $v \in V$;
- if wRw' in t, then $h(w)Rh(w')$ in T, when R is one of $\downarrow, \rightarrow, \Downarrow, \Rightarrow$;
- if $w \in P_\alpha$ in t, then $h(w) \in P_\alpha$ in T, for each $\alpha \in \Sigma$;
- $h(c) = c$ whenever $c \in \mathcal{C}$; and
- $h(\rho(w)) = \rho(h(w))$ for each $w \in N$.

Not that homomorphisms are not injective, i.e. two nodes in N can be mapped to the same node in D. To allow such a situation where two nodes might represent the same information is standard in incomplete information. The semantics of an incomplete tree t is the set of all complete trees T that it has a homomorphism into:

$$[\![t]\!] \;=\; \{T \mid \text{there exists a homomorphism } h : t \to T\}.$$

4 Consistency

In this section we consider the consistency problem for XML incomplete descriptions in the presence of integrity constraints of various forms. More formally, let **I** be a class of XML integrity constraints. We consider the following problem:

PROBLEM:	INCTREE-CONSISTENCY(**I**)
INPUT:	an incomplete tree t,
	a tree automaton $\mathcal{A}$;
	a set Δ of constraints in **I**.
QUESTION:	is there a tree $T \in [\![t]\!]$ so that $T \in L(\mathcal{A})$ and $T \models \Delta$?

4.1 Consistency with respect to automata and constraints

As already mentioned, satisfiability (or consistency) questions arise often in the XML setting, and substantial progress has been made on solving pure static analysis problems, i.e., those not involving data. In fact, many decidable formalisms are known for XML schemas, specified by automata, and constraints or queries [2, 6–8, 11–13, 21, 23, 24]. The complexity of such reasoning problems usually ranges from NP-complete (since Boolean satisfiability can easily be encoded) to a stack of exponentials and beyond (e.g., non-primitive-recursive [14, 15], or even undecidable [7, 13]).

Many of such static analysis problems studied in the XML context can be abstracted as follows. Let again **I** be a class of XML integrity constraints. The problem we deal with now is:

PROBLEM:	AUTOMATA-CONSISTENCY(**I**)
INPUT:	A tree automaton $\mathcal{A}$,
	a set Δ of constraints in **I**.
QUESTION:	is there a tree T so that $T \in L(\mathcal{A})$ and $T \models \Delta$?

For constraints that we deal with here, [13] tells us that AUTOMATA-CONSISTENCY($\mathcal{K} + \mathcal{IC}$) is NP-complete, where $\mathcal{K} + \mathcal{IC}$ is the class of keys and inclusion constraints. These results have been extended to more powerful constraints (and automata as well). For instance, [12] looked at *linear data constraints* and *set constraints*, that essentially extend basic XML constraints with the full power of linear equations; these generalize keys and foreign keys. A different approach was considered in [2], where constraints involving regular expressions were studied, and results extended for constraints that do not hold in the entire XML document, but in subsets of it. Multiple papers deal with constraints provided by XPath expressions (e.g., [7, 8, 11, 24]) or more complex models of automata (e.g., [6]).

Naturally, any lower bound for Automata-Consistency($\mathbf{I}$) applies directly as a lower bound for IncTree-Consistency($\mathbf{I}$), since one can easily construct incomplete trees that represent the entire universe of XML trees (e.g., a single root-node). This implies, for example, that IncTree-Consistency($\mathcal{K} + \mathcal{IC}$) is NP-hard.

What about the upper bounds? We show in the next section that most known upper bounds for different versions of Automata-Consistency continue to hold if we add incomplete trees into the mix, as one can reason about incomplete trees for free, provided the Automata-Consistency does not trivialize. Essentially, this says that we can transfer known results on static reasoning about XML to reasoning about incomplete trees: those come for free.

4.2 General upper bound

To achieve the transfer of complexity results from Automata-Consistency to IncTree-Consistency, we need to impose some mild conditions on the former. The first is that its complexity should not be too low. The second and third state that systems of constraints must have some degree of uniformity (for instance, they should not be tied to just one alphabet, or a particular data value). And the last condition states that they should be extendable with constraints that admit modest reasoning complexity. We now formalize these requirements and demonstrate that many instances of Automata-Consistency satisfy them.

Complexity Most reasoning tasks involving schema and constraints are at least NP-hard (e.g., even the simple case of DTDs and unary keys and inclusion constraints is such [13]). Hence, we shall require that the complexity class to which Automata-Consistency belongs be closed under NP-reductions.
A complexity class $\mathbf{C}$ is *closed under* NP*-reductions* if whenever we have languages $A, B \subseteq \Gamma^*$ such that $B \in \mathbf{C}$ and there is a polynomial-time computable function $f : \Gamma^* \times \Gamma^* \to \Gamma^*$ and a polynomial p such that for each $x \in \Gamma^*$ we have $x \in A$ iff $f(y, x) \in B$ for some $y \in \Gamma^*$ of size at most $p(|x|)$, then $A \in \mathbf{C}$.
Most complexity classes above NP that allow nondterministic guesses rather trivially satisfy this condition.

Alphabet extensions A class $\mathbf{I}$ allows for *alphabet extension* if the following holds. Let Σ and Σ' be alphabets of labels, and let γ be a surjective map $\Sigma' \to \Sigma$. Then for every set Δ of constraints over Σ one can construct, in polynomial time, a set Δ' of constraints over Σ' such that a tree T' over Σ' satisfies Δ' iff its projection T to Σ satisfies Δ.

Constraint extensions Following [13, 12], we introduce set and linear constraints as follows. Fix variables x_α, V_α and $|V_\alpha|$, for each $\alpha \in \Sigma$. The interpretation of x_α is $\#_\alpha(T)$, the number of α-nodes in T; and the interpretation of V_α and $|V_\alpha|$ is, respectively, $V_\alpha(T)$ and $|V_\alpha(T)|$, the set of data values found in α nodes in T, and the cardinality of this set. We shall assume that the complexity of AUTOMATA-CONSISTENCY($\mathbf{I}$) does not change if $\mathbf{I}$ is expanded with the following: linear constraints over variables x_α and $|V_\alpha|$, and set constraints of form $V_\alpha = V_{\alpha_1} \cap \cdots \cap V_{\alpha_p}$, or $V_\alpha \cap V_{\alpha'} = \emptyset$.

We call a class of constraints *feasible* if it is generic (i.e., invariant under permutations of the domain of data values) and satisfies the alphabet-extension and the constraint-extension conditions above.

While these conditions (with the exception of the standard notion of genericity) may look restrictive, they are not: in fact, they apply to a large number of constraints. For instance, they apply to the following.

– Classes of keys, inclusion constraints, and foreign keys. Indeed, it is well known that these can be stated as linear and set constraints introduced above [13], and the complexity of such constraints is in NP [12]. In fact many other constraints could be for free added too, e.g., *denial* constraints, stating that $V_\alpha(T) \cap V_{\alpha'}(T) = \emptyset$ for $\alpha \neq \alpha'$.
– Extensions of keys and inclusion constraints specified by properties of nodes. For instance, instead of *key*(α), one can state a condition *key*(ϕ), where ϕ is a formula with one free variable over the language of unranked trees. We only requite that ϕ be definable in MSO. For instance, ϕ could be an XPath node formula. The meaning of such a constraint is that all nodes satisfying ϕ have different data values. Such constraints include many constraints considered, for instance, in [2, 9]. Their good properties easily follow from [12].
– Classes of constraints expressed by [25]. Such automata are closed under adding set and linear constraints, and they capture many existing models of constraints over data trees (e.g., those expressible in 2-variable logic).

Now we can state our transfer result.

Theorem 1. *Let* $\mathbf{C}$ *be a complexity class that is closed under* NP*-reductions, and* $\mathbf{I}$ *a feasible class of integrity constraints. If* AUTOMATA-CONSISTENCY($\mathbf{I}$) *is in* $\mathbf{C}$*, then so is* INCTREE-CONSISTENCY($\mathbf{I}$)

Proof. Let $\mathbf{I}$ and $\mathbf{C}$ be as stated in the theorem. Since by the assumption $\mathbf{C}$ is closed under NP-reductions, it suffices to show that INCTREE-CONSISTENCY($\mathbf{I}$) is NP-reducible to AUTOMATA-CONSISTENCY($\mathbf{I}$).

To that extent, let t, $\mathcal{A}$ and $\Delta \in \mathbf{I}$ be arbitrary inputs of INCTREE-CONSISTENCY($\mathbf{I}$). The basic idea behind the reduction is to construct a tree automaton $\mathcal{A}_t$ whose language is exactly $[\![t]\!]$, in which case one trivially has that AUTOMATA-CONSISTENCY($\mathbf{I}$) accepts on inputs $\mathcal{A} \times \mathcal{A}_T$ and Δ if and only if INCTREE-CONSISTENCY($\mathbf{I}$) accepts on inputs t, $\mathcal{A}$ and Δ.

Unfortunately, it is not difficult to show that $\mathcal{A}_t$ might be exponential in the size of t. In order to avoid the exponential blowup, we use the fact that we can *guess* with no cost (since $\mathbf{C}$ is closed under NP-reductions), and guess first an *intermediate* structure describing only the information about $[\![t]\!]$ that is enough to show consistency. We denote these structures as *tree skeletons*, which we define next.

Tree skeletons are defined just as XML trees, with the difference that instead of the child relation we can use either $\downarrow$ or $\downarrow^+$, and instead of next sibling relation one can use either $\rightarrow$ or $\rightarrow^+$. As expected, $\downarrow^+$ and $\rightarrow^+$ are interpreted as *strict* descendant and *strict* following sibling, respectively. More formally, we define a tree skeleton as a structure $sk = \langle D, A, \downarrow, \downarrow^+, \rightarrow, \rightarrow^+, (P_\alpha)_{\alpha \in \Sigma}, \rho \rangle$, where D is an unranked tree domain, the relations $\downarrow, \downarrow^+, \rightarrow, \rightarrow^+$, are binary, and relations P_α's are unary, and the following are satisfied:

1. Every node w in D can have at most one $\downarrow^+$-child (i.e., at most one w' such that $w \downarrow^+ w'$ holds), and
2. No node w in D can have $\downarrow$-children and $\downarrow^+$-children at the same time (i.e., there could be no nodes w', w'' so that $w \downarrow w'$ and $w \downarrow^+ w''$ hold).

We define the notion of a homomorphism h from a skeleton to a tree T so that they preserve all relations, i.e., if $w \downarrow^+ w'$ in the skeleton, then $h(w')$ is a strict descendant of $h(w)$ in T, and so on. With this, the semantics of tree skeletons is defined using homomorphisms:

$$[\![sk]\!] \;=\; \{T \mid \text{there exists a homomorphism } sk \rightarrow T\}.$$

The following lemma captures the intuition that tree skeleton, while possibly exponentially smaller than trees, carry enough information to solve the consistency problem.

Lemma 1. *There exists a polynomial p with the following properties. Let t be an incomplete tree, T a data tree in $[\![t]\!]$, and S a subset of nodes in T. Then, there exists a skeleton sk of size at most $p(|t| + |S|)$ such that*

1. *$T \in [\![sk]\!]$,*
2. *$[\![sk]\!] \subseteq [\![t]\!]$, and*
3. *for each α-node in S with data value c, there is at least one α-node in sk with data value c.*

Proof sketch: Since $T \in [\![t]\!]$, there is a homomorphism from t to T. The skeleton sk is constructed by marking in T all nodes in S, as well as all nodes in T that witness the homomorphism from t to T. From these marked nodes one can construct a tree-shaped skeleton by subsequently adding the least common ancestor of every pair of nodes that are not a direct descendant of a marked node, and adding first and last siblings if these are not already marked. It is not difficult, but rather cumbersome, to show that this construction can be done in polynomial time, and satisfies the desired properties. □

Let sk be a tree skeleton over Σ. Our next task is to define an automaton that corresponds, at least in some extent, to the set of trees represented by sk. Recall that we denote the active domain (of data values) by $\mathrm{adom}(sk)$, and let

$\perp$ be a fresh value not in $\mathcal{C}$. We now show how to construct from *sk* and $\perp$ an unranked tree automaton $A_{(sk,\perp)} = (Q, (\Sigma \times (\text{adom}(sk) \cup \{\perp\}), \{q_f\}, \delta)$, where Q and δ are defined inductively. We start with $Q = \{q_f\} \cup \{q_n \mid n \in D\}$, and the following transitions in δ:

– $(q_f, (\alpha, \perp)) \to q_f^*$, for each α in Σ

Next, for each $n \in D$, we add extra states and transitions to $A_{(sk,\perp)}$, according to the following conditions:

– If n is a leaf, add to δ the transition $(q_n, (\alpha, \rho(n))) \to q_f^*$.
– Else, if n has a (single) child $n \cdot 0$ such that $n \downarrow^+ n \cdot 0$ holds in the skeleton, then add to Q a fresh state q, and add to δ the transitions $(q_n, (\alpha, \rho(n))) \to q_f^* \cdot q \cdot q_f^*$, and $(q, (\alpha', \perp)) \to q_f^* \cdot (q \mid q_{n\cdot 0}) \cdot q_f^*$, for each α' in Σ.
– Finally, if n has k children $n \cdot 0, \ldots, n \cdot (k-1)$ under relation $\downarrow$, assume that the children are ordered as $n \cdot 0\ \theta_1\ n \cdot 1\ \theta_2\ \ldots\ \theta_{k-1}\ n \cdot (k-1)$, where each θ_i is either $\to$ or $\to^+$. Add to δ the transition

$$(q_n, (\alpha, \rho(n))) \to q_f^* \cdot q_{n\cdot 0} \cdot r_1 \cdot q_{n\cdot 1} \cdot r_2\ \cdots\ \cdot r_{k-1} \cdot q_{n\cdot(k-1)} \cdot q_f^*,$$

where each r_j is ϵ if θ_j is $\to$, or q_f^* if θ_j is $\to^+$.

The intuition is that every tree in the language of $A_{(sk,\perp)}$ represents to some extent a set of trees in $[\![sk]\!]$. The data values used in *sk* are represented with labels of the form (α, c), for some α in Σ and c in $\mathcal{C}$, and nodes in which the data value is not important to witness the membership in $[\![sk]\!]$ are labeled with $(\alpha', \perp)$, for α' in Σ.

More formally, given a tree T over $\Sigma \times (V \cup \{\perp\})$, we say that a tree T' over Σ is a *data projection* of T into Σ if T' can be formed from T by replacing each node in T labeled with (α, c) for a node labeled α with data value c, and every node labeled with $(\alpha', \perp)$ in T for a node labeled α' and a data value $a \in \mathcal{C}$. The following is straightforward from the construction:

Lemma 2. *Let sk be a tree skeleton over Σ with active domain* adom(*sk*)*, and let $\perp$ be a fresh data value not in $\mathcal{C}$. A tree T over Σ belongs to $[\![sk]\!]$ if and only if there is a tree T' over $\Sigma \times (V \cup \{\perp\})$ that is accepted by $A_{(sk,\perp)}$, and such that T is a data projection of T' over Σ.*

In other words, the set of data projections over Σ of all trees accepted by $A_{(sk,\perp)}$ corresponds precisely to the set of trees in $[\![sk]\!]$. We now have all the ingredients to state our NP-reduction for consistency.

Checking for consistency:

Consider an arbitrary incomplete tree t, a tree automaton A over Σ and a set Δ of constraints.

First, perform the following operations:

1. Guess a tree skeleton *sk* such that $[\![sk]\!] \subseteq [\![t]\!]$.

2. Let adom(*sk*) be all data values that are mentioned in *sk*, and let $\perp$ a fresh data value not used in $\mathcal{C}$.
3. Define the alphabet $\Sigma \times (\text{adom}(sk) \cup \{\perp\})$ and consider its projection to Σ. Due to feasibility, construct, in polynomial time, the set Δ' of constraints over trees over $\Sigma \times (\text{adom}(sk) \cup \{\perp\})$ so that such a tree satisfies Δ' iff its Σ-projection satisfies Δ.
4. Construct the following set of constraints Γ:
   - For each $c \in \text{adom}(sk)$ and $\alpha \in \Sigma$ such that there is at least one (α, c)-labeled node in *sk*, add the constraint $|V_{(\alpha,c)}| = 1$ to Γ.
   - For each $c \in \text{adom}(sk)$ and $\alpha \in \Sigma$, if *sk* does not contain a node labeled α with data value c, then add the constraint $V_{(\alpha,c)} = \emptyset$.
   - Moreover, for each pair of values c, c' in adom(*sk*) and each $\alpha \in \Sigma$, such that there are nodes labeled with α and α' with data value c in *sk*, add the constraints $V_{(\alpha,c)} = V_{(\alpha',c)}$ to Γ.
   - Finally, for each $\alpha, \alpha' \in \Sigma$ and distinct values $c, c' \in \text{adom}(sk) \cup \{\perp\}$, add the constraints $V_{(\alpha,c)} \cap V_{(\alpha',c')} = \emptyset$ to Γ.
5. Build an automaton $A^{(\text{adom}(sk),\perp)}$ from A by replacing every transition of form $(q, \alpha) \to L$ with the transitions $(q, (\alpha, c)) \to L$ for each $c \in \text{adom}(sk) \cup \{\perp\}$.
6. Finally, check whether $(A^{(\text{adom}(sk),\perp)} \times A_{(sk,\perp)}), (\Delta' \cup \Gamma)$ is consistent. This of course is possibile due to the feasibility assumption, with the same complexity.

Correctness and soundness

We need to prove that (A, t, Δ) is consistent if and only if there exists a skeleton *sk*, with $[\![sk]\!] \subseteq [\![t]\!]$, and such that $(A^{(\text{adom}(sk),\perp)} \times A_{(sk,\perp)}), (\Delta' \cup \Gamma)$ is consistent.

($\Rightarrow$): Let *sk* be a skeleton such that $[\![sk]\!] \subseteq [\![t]\!]$, and $(A^{(\text{adom}(sk),\perp)} \times A_{(sk,\perp)}), (\Delta' \cup \Gamma)$ is consistent, and let T be the tree over $\Sigma \times (\text{adom}(sk) \cup \{\perp\})$ that witnesses the consistency.

Let $f : \text{adom}(T) \to (\text{adom}(T) \cup \text{adom}(sk))$ be the following renaming of data values: For each data value $d \in \text{adom}(T)$, if there is an (α, c)-node in T with data value d, then $f(d) = c$; and otherwise $f(d) = d$. Notice then that f is an injection. Indeed, since T is consistent with Γ, it satisfies the constraints $V_{(\alpha,c)} = V_{(\alpha',c)}$ and $V_{(\alpha,c)} \cap V_{(\alpha',c')} = \emptyset$, and $|V_{(\alpha,c)}| = 1$, for each $\alpha, \alpha' \in \Sigma$ and distinct $c, c' \in (\text{adom}(sk) \cup \{\perp\})$. Thus, all (α, c)-nodes, for any $\alpha \in \Sigma$, share the same, single data value, which is at the same time not used anywhere else in T.

Next we prove that the data projection $f(T)'$ of $f(T)$ over Σ is consistent with Δ, A and t. From the construction of $A^{(\text{adom}(sk),\perp)}$, it is obvious that $f(T)'$ is in the language of A. Furthermore, from feasibility of $\mathbf{I}$ we have that $f(T)$ is consistent with Δ', and that its data projection $f(T)'$ is consistent with Δ. Finally, since T is in the language of $A_{(sk,\perp)}$, so is $f(T)$, and then by Lemma 2 we have that $f(T)'$ belongs to $[\![sk]\!]$, which by the assumption that $[\![sk]\!] \subseteq [\![t]\!]$ entails that $f(T)'$ belongs to $[\![t]\!]$.

($\Leftarrow$): Assume that (A, t, Δ) is consistent, and let T be a tree witnessing the consistency, with h a homomorphism from t to T. Construct a set S of nodes from T as follows. If n is in the image of h, then add n to S. Moreover, for each α-node n in S with data value c, and for each $\alpha' \in \Sigma$ such that T has at least one α'-labeled node with data value c, add one of these nodes to S. By Lemma 1, there is a skeleton *sk* containing all nodes in S, such that $T \in [\![sk]\!]$ and such that $[\![sk]\!] \subseteq [\![t]\!]$. Let $\perp$ be a value not in $\mathcal{C}$.

We now construct a tree T' over the alphabet $\Sigma \times (\mathrm{adom}(sk) \cup \{\perp\})$ that is a witness for the consistency of $(A^{(\mathrm{adom}(sk),\perp)} \times A_{(sk,\perp)}), (\Delta' \cup \Gamma)$. This is done as follows. Replace each α-node in T with data value $c \in \mathrm{adom}(sk)$ with a (α, c)-labeled node with data value c, and each α-node in T with data value not in $\mathrm{adom}(sk)$ with a $(\alpha, \perp)$-labeled node with the same original data value. By construction, it is not difficult to see that T' is consistent with Γ. Moreover, from feasibility of $\mathbf{I}$ we obtain that T' is consistent with Δ'. Third, given that T is a data projection of T' over Σ, by Lemma 2 we have that T' belongs to the language of $A_{(sk,\perp)}$. Finally, it is straightforward to see that T' is in the language of $A^{(\mathrm{adom}(sk),\perp)}$, which finishes the proof.

Membership in C: A simple inspection on the reduction reveals that steps (3), (4) and (5) can be performed in polynomial time with respect to *sk*, $\mathcal{A}$, Σ and Δ. Furthermore, from Lemma 1 and the above remarks we have that there always exists a skeleton *sk* of polynomial size with respect to t that suffices for the correctness of the reduction. This shows that the problem IncTree-Consistency($\mathbf{I}$) is NP-reducible to Automata-Consistency($\mathbf{I}$). The theorem follows from the assumption that $\mathbf{C}$ is closed under NP-reductions.

From Theorem 1 and the results from [13] we immediately obtain tight complexity bounds for IncTree-Consistency($\mathcal{K} + \mathcal{IC}$), where $\mathcal{K} + \mathcal{IC}$ is the class of basic XML constraints (keys and foreign keys).

Corollary 1. IncTree-Consistency($\mathcal{K} + \mathcal{IC}$) *is* NP-*complete.*

In fact, any class of constraints expressible with linear and set constraints (e.g., denial constraints) can be added for free, without changing the complexity bound.

4.3 A tractable case

The fact that Automata-Consistency($\mathcal{K} + \mathcal{IC}$) is already NP-hard rules out any possibility of finding tractable classes for IncTree-Consistency problem without extra restrictions. Following [4], one can look at the consistency problems without tree automata, in which case, given a set Δ of constraints and an incomplete tree t, we ask for a $T \in [\![t]\!]$ so that $T \models \Delta$. It is not difficult to adapt the results in [4] to obtain the following.

Theorem 2. *Without automata in the input,* IncTree-Consistency($\mathcal{K} + \mathcal{IC}$) *can be solved in* Ptime *for rigid incomplete data trees, but remains* NP-*hard for arbitrary incomplete data trees.*

5 Query Answering

As is common in the scenarios when one needs to compute certain answers (by means of intersection) [18, 3], we look at queries that can only output tuples of data values. The queries will be essentially unions of conjunctive queries over XML trees; however, to avoid the clumsiness of a two-sorted presentation, we follow the standard approach and define them via *patterns*.

An example of a pattern is

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

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

Formally, patterns are given by the grammar:

$$\begin{aligned}&\pi := \alpha(\bar{z}) \mid \alpha(\bar{z})/[\mu, \ldots, \mu] \mid \alpha(\bar{z})//[\mu, \ldots, \mu] \mid \alpha(\bar{z})/[\mu, \ldots, \mu]//[\mu, \ldots, \mu]\\ &\mu := \pi \mid \pi \leadsto \ldots \leadsto \pi\end{aligned}$$

where each $\leadsto$ is either $\to$ or $\Rightarrow$.

Semantics We define the semantics of a pattern with respect to an XML tree $T = \langle D, A, \downarrow, \to, (P_\alpha)_{\alpha \in \Sigma}, \rho\rangle$, a node w, 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), $\rho(w) = \nu(z)$, and there exist n children $w_1, \ldots, w_n$ of w such that $(T, w_i, \nu) \models \mu_i$ for each $i \leq n$, and there exist k descendants $w'_1, \ldots, w'_k$ of w such that $(T, w'_i, \nu) \models \mu'_i$ for each $i \leq k$.

– $(T, w, \nu) \models \pi_1 \leadsto \ldots \leadsto \pi_m$ if there is a sequence of nodes $w = w_1, \ldots, w_m$ so that $(T, w_i, \nu) \models \pi_i$ for each $i \leq m$ and $w_i \to w_{i+1}$ whenever the ith $\leadsto$ is $\to$, and $w_i \Rightarrow w_{i+1}$ whenever the ith $\leadsto$ is $\Rightarrow$.

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 $\alpha(\bar{x})/[\beta(\bar{y})]$. Finally, we write $(T, w) \models \pi(\bar{a})$ if $(T, w, \nu) \models \pi(\bar{x})$ where ν assigns values $\bar{a}$ to variables $\bar{x}$.

Pattern-based XML queries We now define XML analogs of unions of conjunctive queries based on patterns. First, we need a class of *conjunctive queries* (essentially defined in [1, 5, 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 $i \leq n$. We define $\mathrm{UCQ}_{\mathrm{XML}}$ as queries of the form $q_1(\bar{x}) \cup \ldots \cup q_m(\bar{x})$, where each q_i is a conjunctive query.

Example Consider the query

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

It selects data values x found in α-labeled nodes which either have two consecutive children labeled β and γ, or a descendant labeled δ.

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

$$\mathit{certain}^{\Delta}_{\mathcal{A}}(Q(\bar{x}), t) \;=\; \bigcap \Big\{ Q(T) \;\; \big| \;\; T \in [\![t]\!], \;\; T \in L(\mathcal{A}) \text{ and } T \models \Delta \Big\}.$$

We study data complexity of certain answers (where Q, $\mathcal{A}$ and Δ are fixed).

PROBLEM:	$\mathit{certain}^{\Delta}_{\mathcal{A}}(Q)$
INPUT:	an incomplete data tree t and a tuple $\bar{a}$ of size $\lvert\bar{x}\rvert$
QUESTION:	does $\bar{a}$ belong to $\mathit{certain}^{\Delta}_{\mathcal{A}}(Q(\bar{x}), t)$?

We also consider variations of the problem when the automaton $\mathcal{A}$, the constraints Δ, or both are missing from the parameters, referring to them as $\mathit{certain}_{\mathcal{A}}$, $\mathit{certain}^{\Delta}$, and just *certain*. Note that $\mathit{certain}(Q(\bar{x}), t) = \bigcap\{Q(T) \mid T \in [\![t]\!]\}$ is the standard notion of certain answers, without constraints and schemas.

5.1 General Upper bound

Theorem 3. *For every query $Q(\bar{x}) \in \mathrm{UCQ}_{\mathrm{XML}}$, tree automaton $\mathcal{A}$ and a set Δ of keys, foreign keys and inclusion constraints, the problem $\mathit{certain}^{\Delta}_{\mathcal{A}}(Q)$ is in* coNP.

Proof sketch: From [3], we know that $\mathit{certain}_{\mathcal{A}}(Q)$ is in coNP. We first briefly recall the idea behind this upper bound and then explain how to extend the proof to account for an additional set Δ of constraints. The standard way to obtain an upper bound for a query Q (say, Boolean for this sketch) over t is to prove that if $\mathit{certain}(Q, t)$ is false, then there is $T \in [\![t]\!]$ with some specific size bounds in t such that $T \models \neg Q$.

Our starting point here is as follows: suppose we have $T \in [\![t]\!]$ such that $T \models \neg Q$ and $T \in L(\mathcal{A})$. For the sketch, assume that $Q = q_1 \vee \ldots \vee q_n$, where each q_i is a conjunctive query; that is, $T \models \neg q_i$ for each $i \leq n$. Take a homomorphism $h : t \to T$, and add to the image of h all nodes which are least common ancestors of nodes in the image, plus the root. We call it the skeleton. Via careful renaming of some occurrences of data values in the tree and using a reasoning on types, the main argument of the proof consists of pruning long vertical and horizontal paths in T in order to obtain a tree which contains every node in the skeleton of T and hence still belongs to $[\![t]\!]$, while it also agrees with T on all the q_is, is still accepted by $\mathcal{A}$ and is of polynomial size in t.

This proof can be extended as follows. Since we only have unary constraints in Δ, we first chase their relational representation, i.e., constraints applied to unary relations U_α corresponding to labels α. The form of constraints implies that they are weakly acyclic, and hence the chase terminates in polynomial time and produces a set of tuples of the form $U_\alpha(x)$ where x is either a data value or a null. To each of these tuples, we associate a new single node pattern labeled $L(x)$. We then form the union of all these structures with the incomplete tree t and call this new structure t^Δ. Note that labels, nulls and constants might occur both in t and in some of the single nodes patterns in t^Δ. At this point if t^Δ still does not satisfy some of the constraints in Δ (e.g. some key constraints) we simply conclude that the certain answer is vacuously true. Otherwise, assume $\mathit{certain}^\Delta_\mathcal{A}(Q(\bar{x}), t)$ is false. Then there is $T \in [\![t^\Delta]\!]$ such that T is accepted by $\mathcal{A}$, T satisfies Δ and $T \models \neg Q$. The only difference with the proof for $\mathit{certain}_\mathcal{A}(Q(\bar{x}), t)$ is that instead of taking a homomorphism $h : t \to T$, we now consider a homomorphism $h : t^\Delta \to T$ and generate the corresponding skeleton. The remainder of the proof is as in [3]. □

5.2 Rigid trees

It was shown in [3] that on rigid trees, the problem $\mathit{certain}(Q)$ becomes tractable as certain answers can be computed by naïve evaluation. Recall that a rigid tree is a tree in which no structural information is missing; that is, the only types of missing information are nulls and wildcards. In the following we show that on rigid trees $\mathit{certain}_\Delta(Q)$ can become coNP-hard as soon as Δ contains even a single inclusion constraint, but remains tractable if Δ only contains keys.

coNP-hardness

Theorem 4. *There exists a query $Q(\bar{x})$ in* UCQ$_{\text{XML}}$ *and a set of constraints Δ containing one single inclusion constraint such that the problem* $\mathit{certain}_\Delta(Q)$ *is* coNP*-hard even over rigid incomplete data trees.*

Proof. The proof is by reduction from non 3-colorability. Let $G = \langle V, E\rangle$ be a directed graph, with the set of vertices $V = \{v_1, \ldots, v_n\}$ and the set of edges $E = \{e_1, \ldots, e_m\}$, where each edge e_i is a pair (v^i_1, v^i_2) of vertices from V. We show how to build a rigid incomplete data tree t from G and give a fixed Boolean query $q \in$ UCQ$_{\text{XML}}$ and a fixed inclusion constraint Δ such that $\mathit{certain}_\Delta(Q, t)$ evaluates to *true* if and only if G is not 3-colorable.

We use *root*, C, G and E as labels and *red*, *blue*, *green* and a as data values. We use $v_1, \ldots, v_n$ as null values and construct t as follows. The root, labeled $\mathit{root}(a)$ has four linearly ordered children:

– the first one labeled $C(\mathit{red})$,

- the second one labeled $C(blue)$,
- the third one labeled $C(green)$,
- the last one labeled $G(a)$.

The three first children of the root are leaves, but its last $G(a)$-labeled child has m linearly ordered $G(a)$-labeled children where for every $i \leq m$ the following holds:

- the i^{th} child has two ordered children, the first one is labeled $E(v_1^i)$ and the second one is labeled $E(v_2^i)$.

Now we let $\Delta = \{E \subseteq C\}$ and $Q = q_1 \vee q_2 \vee q_3 \vee q_4$, where:

$$\begin{aligned} q_1 &= \exists x\ root(a)/G(a)/G(a)[E(x) \to E(x)] \\ q_2 &= \exists x \exists y \exists z \exists v \exists w\ root(a)/[\_(x) \to \_(y) \to \_(z) \to \_(v) \to \_(w)] \\ q_3 &= \exists x\ root(a)/G(a)//C(x) \\ q_4 &= \exists x \exists y\ root(a)/C(x)/\_(y) \end{aligned}$$

We show that $certain_\Delta(Q, t)$ evaluates to *true* if and only if G is not 3-colorable.

Assume first that G is 3-colorable and let $c : V \to \{red, blue, green\}$ be a 3-coloring of G. We construct a complete tree $T \in [\![t]\!]$ such that $T \models \Delta$ and $T \not\models Q$ by simply replacing every null value v_i occurring in t with $c(v_i)$. Since T is a homomorphic image of t, it does not satisfy any of the queries q_2, q_3, q_4. Also, since c is a 3-coloring of G, we have that $T \not\models q_1$ and so $T \not\models Q$. It follows that $certain_\Delta(Q, t)$ evaluates to $false$.

Now assume that $certain_\Delta(Q, t)$ evaluates to $false$. Then there exists a tree $T \in [\![t]\!]$ such that $T \models \Delta$ and $T \not\models Q$. As $T \in [\![t]\!]$, there is a homomorphism $h : t \to T$. Since $T \not\models q_i$ for every $2 \leq i \leq 4$, it follows that T only contains three C-labeled nodes, which carry respectively one of the three data values *red*, *blue*, *green*. Also as T satisfies the constraint $E \subseteq C$, every E-labeled node carries one of the data values *red*, *blue*, *green*. As $T \not\models q_1$, the homomorphism h gives a 3-coloring of G.

Tractable Upper Bounds

Theorem 5. *For every query $Q \in \mathrm{UCQ}_{\mathrm{XML}}$ and set Δ of keys, the problem $certain_\Delta(Q)$ is in* PTIME*, when restricted to rigid incomplete data trees.*

The tractability of $certain_\Delta(Q)$ follows directly from the proof of tractability of $certain(Q)$ on rigid trees in [3]. There is only one additional step, in which we first check whether $t \models \Delta$. By Theorem 2, this can be done in polynomial time in the size of t. If $t \not\models \Delta$, then it is clear that Δ will not be satisfied by any completion of t and we conclude that the certain answers is vacuously *true*. Otherwise $t \models \Delta$ and we go on with evaluating Q on t using naïve evaluation, exactly as in [3].

Acknowledgment Work partially supported by EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599. It was done while all authors were at the University of Edinburgh.

References

1. M. Arenas, W. Fan, L. Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38 (2008), 841–880.
2. M. Arenas, L. Libkin. XML data exchange: consistency and query answering. *Journal of the ACM* 55:2 (2008).
3. P. Barceló, L. Libkin, A. Poggi, C. Sirangelo. XML with incomplete information. *Journal of the ACM*, 58:1 (2010).
4. P. Barceló, L. Libkin, J. Reutter. On incomplete XML documents with integrity constraints. *AMW'10* (2010).
5. H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. *J. Comput. Syst. Sci.* 77(3): 450-472 (2011).
6. M. Bojanczyk. Automata for data words and data trees. In *RTA 2010*, pages 1-4.
7. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.* 12(4): 27 (2011).
8. M. Bojanczyk, S. Lasota. An extension of data automata that captures XPath. *Logical Methods in Computer Science* 8(1): (2012).
9. P. Buneman, S. Davidson, W. Fan, C. Hara, W.-C. Tan. Keys for XML. *Computer Networks* 39(5):473-487 (2002).
10. A. Calì, D. Lembo, R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. *PODS'03*, pages 260-271.
11. D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Regular XPath: constraints, query containment and view-based answering for XML documents. In *LID'08*, 2008.
12. C. David, L. Libkin, T. Tan. Efficient reasoning about data trees via integer linear programming. *ACM TODS*, 37(3): 19 (2012).
13. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *Journal of the ACM* 49 (2002), 368–406.
14. D. Figueira. Forward-XPath and extended register automata on data-trees. In *ICDT'10*, pages 231-241.
15. D. Figueira. Bottom-up automata on data trees and vertical XPath. In *STACS'11*, pages 93-104.
16. A. Gheerbrant, L. Libkin, T. Tan. On the complexity of query answering over incomplete XML documents. In *ICDT'12*, pages 169–181.
17. G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees. *Journal of the ACM* 53(2):238-272, 2006.
18. T. Imieliński and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.
19. G. Jan Bex, F. Neven, J. Van den Bussche. DTD versus XML Schema: A Practical Study. *WEBDB04*, pages 79–84 (2004).
20. A. Laender, M. Moro, C. Nascimento, P. Martins. An X-Ray on Web-Available XML Schemas. *SIGMOD Record* 38(1) (2009), 37-42.
21. L. Libkin, C. Sirangelo. Reasoning about XML with temporal logics and automata. *J. Applied Logic*, 8:2, 210–232 (2010).
22. W. Martens, F. Neven, T. Schwentick. Simple off the shelf abstractions for XML schema. *SIGMOD Record* 36(3): 15-22 (2007).
23. L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL 2006*, pages 41-57.
24. L. Segoufin. Static analysis of XML processing with data values. *SIGMOD Record* 36(1): 31-38 (2007).

25. T. Tan. An automata model for trees with ordered data values. In *LICS'12*.
26. J.W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *JCSS* 1 (1967), 317-322.