What's Hard about XML Schema Constraints?

Marcelo Arenas[1], Wenfei Fan[2], and Leonid Libkin[1]

[1] Department of Computer Science, University of Toronto.
{marenas,libkin}@cs.toronto.edu
[2] Bell Laboratories
wenfei@research.bell-labs.com

Abstract. Data description for XML usually comes in the form of a type specification (e.g., a DTD) together with integrity constraints. XML Schema allows one to mix DTD features with semantic information, such as keys and foreign keys. It was shown recently [2,7] that the interaction of DTDs with constraints may be rather nontrivial. In particular, testing if a general specification is consistent is undecidable, but for the most common case of single-attribute constraints it is NP-complete, and linear time if no foreign keys are present.
However, XML Schema design did not adopt the form of constraints prevalent in the database literature, and slightly changed the semantics of keys, foreign keys, and unique constraints. In this paper we demonstrate the very costly effect of this slight change on the feasibility of consistency checking. In particular, all the known hardness results extend to the XML Schema case, but tractability results do not. We show that even without foreign keys, and with very simple DTD features, checking consistency of XML-Schema specifications is intractable.

1 Introduction

Any data-central system must provide a data definition language as well as a data manipulation language. For commercial relational DBMSs, these languages are well-understood. As a lot of data is becoming available in XML [11], and much of database research focus is shifting from the traditional relational model to semistructured data and XML [1,6,5,9,10], it is important to understand new issues that arise in the context of describing and querying XML.

One such issue is the semantics of XML data specifications. Traditionally, XML data was described by DTDs [11][1]. But just as in the relational context, where simple SQL's `create table` must be supplemented with various constraints to provide semantic information, constraints must be added to XML specifications as well. Most of the proposals deal with constraints similar to those found in relational databases: keys and foreign keys [3,4,12]. However, unlike traditional relational constraints, XML keys and foreign keys interact in a nontrivial way with DTDs, allowing one to write seemingly perfect specifications that nevertheless are *inconsistent*: no document can satisfy them.

[1] Throughout the paper, by a DTD we mean its type specification; we ignore its ID/IDREF constraints since their limitations have been well recognized [3].

In [2,7], we studied this problem, and demonstrated the following. First, if arbitrary keys and foreign keys are added to DTDs, the consistency problem is undecidable. Second, with the restriction to one-attribute constraints (*unary* constraints, by far the most common in practice), the problem is intractable: depending on the exact flavor of constraints, it is anywhere from NP-complete (simple element-type absolute constraints [7]) to PSPACE-hard (regular-expression-based constraints [2]) to undecidable (relative constraints [2]). However, without foreign keys, the problem is tractable: it is solvable in *linear* time.

Those results were shown for DTDs and (foreign) keys. These days, the prime standard for specifying XML data is *XML Schema* [14]. It is a rather rich language that supports specifications of both types and integrity constraints. Its types subsume DTDs [11], and its constraints – even keys and foreign keys – have a slightly different semantics from what has been primarily studied in the database literature. In this paper we investigate specifications that consist of a DTD and a set of constraints with the semantics proposed by XML Schema. We show that this little change of semantics complicates things considerably, as far as consistency checking is concerned.

We say that an XML document satisfies a specification if and only if it conforms to the DTD and satisfies the constraints, and that a specification is *consistent* if there is a document that conforms to it. A specification may be inconsistent due to the interaction between the type and the constraint parts.

As an example, consider a specification in XML Schema $S_1 = (D_1, \Sigma_1)$, where D_1 is a simple DTD describing insurance policies for a transportation vehicle and Σ_1 is a set of keys and foreign keys:

D_1:
```
<!ELEMENT  vehicle  ((registr | plate), policy, policy)>
<!ATTLIST  registr  num  CDATA  #REQUIRED>
<!ATTLIST  plate    num  CDATA  #REQUIRED>
<!ATTLIST  policy   ref  CDATA  #REQUIRED>
```

Σ_1: $(vehicle/registr \cup vehicle/plate,\ \{@num\})$,
$(vehicle/policy,\ \{@ref\})$,
$(vehicle/policy,\ \{@ref\}) \subseteq_{FK} (vehicle/registr \cup vehicle/plate,\ \{@num\})$

Here we omit the definition of elements whose type is string. The DTD says that each vehicle must present either a registration number or a plate number, and must purchase two insurance policies. The first constraint in Σ_1 is a key asserting that each vehicle can be uniquely identified by either its registration number or its plate number[2]. The second constraint, another key, says that the policies should use different references. The third constraint in Σ_1 is a foreign key. It says that a policy reference must be either the registration number or the plate number. This schema is inconsistent: on one hand, as indicated in Figs. 1 (a) and (b), for any XML document conforming to the DTD D_1, the `vehicle` element must have either a `registr` or a `plate` subelement, but it cannot have both; on the other hand, the constraints enforce the presence of both a `registr` subelement and a `plate` subelement, since otherwise two `policy` references cannot be distinct. As a result, there is no XML document that both

[2] We define the syntax and semantics of keys and foreign keys in Sec. 2.2.

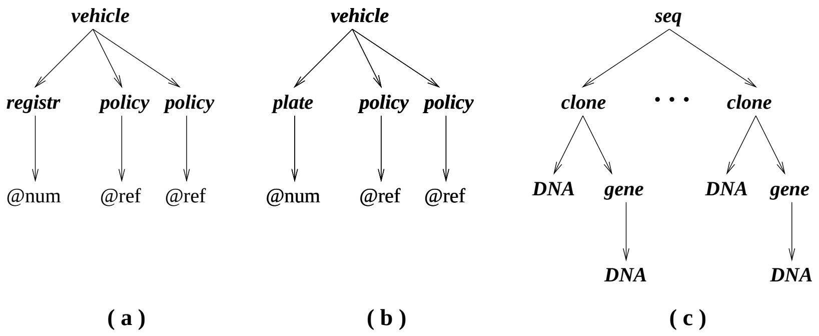

Fig. 1. XML documents (represented as trees) conforming to DTDs D_1 and D_2

conforms to D_1 and satisfies Σ_1. This example demonstrates that the DTD and constraints in an XML-Schema specification may interact with each other, and the interaction leads to the inconsistency of the specification.

Worse still, a specification in XML Schema may not be consistent even in the absence of foreign keys. As another example, consider the following specification $S_2 = (D_2, \Sigma_2)$ for biomedical data:

D_2:
```
<!ELEMENT  seq   (clone+)>
<!ELEMENT  clone (DNA, gene)>
<!ELEMENT  gene  (DNA)>
```

Σ_2: $(seq/clone,\ \{//DNA\})$

The DTD describes a nonempty sequence of `clone` elements: each `clone` has a `DNA` subelement and a `gene` subelement, and `gene` in turn has a `DNA` subelement, while `DNA` carries text data (PCDATA). The key in Σ_2 attempts to enforce the following semantic information: there exist no two `clone` elements that have the same `DNA` no matter where the `DNA` appears as their descendant. Again this specification is inconsistent. To see this, recall that XML Schema requires that for any XML document satisfying a key, the "fields" (that is, `//DNA` in our example) must *exist* and be *unique*. However, as depicted in Fig. 1 (c), in any XML document that conforms to the DTD D_2, a `clone` element must have two `DNA` descendants. Thus, it violates the uniqueness requirement of the key in Σ_2.

Is it possible to test consistency of an XML-Schema specification at compile time? That is, given a specification (D, Σ), whether or not there exists an XML document that both conforms to the DTD D and satisfies the constraints Σ. We refer to this problem as the *consistency problem* for XML Schema. The question is important as one wants to know whether or not a specification makes sense before attempting to create or validate an XML document w.r.t. it.

The central technical problem investigated in this paper is the consistency problem for XML Schema. Our *main conclusion* is that the semantics of keys and foreign keys in XML-Schema makes the consistency analysis rather intricate and intractable. Indeed, all the hardness and undecidability results of [2,7] carry over to specifications of XML Schema. However, using a new technique, we show that the most important tractable cases under the standard key semantics, become intractable under the semantics of XML Schema. We also identify several restrictions, commonly used in practice, that still allow relatively-efficient consistency checking.

Organization. Sec. 2 introduces a formalism for XML-Schema specifications. We show in Sec. 3 that the consistency problem is highly intricate in general. In Sec. 4 we identify restricted cases that allow relatively efficient consistency checking. We summarize our results in Sec. 5. Proofs can be found in [15].

2 XML Schema

XML Schema [14] defines both a type system and a class of integrity constraints. Its type system subsumes DTDs. It supports a variety of atomic types (e.g., string, integer, float, double, byte), complex type constructs (e.g., sequence, choice) and inheritance mechanisms (e.g., extension, restriction). Its integrity constraints include keys, foreign keys and unique constraints. In XML Schema, a specification for XML data consists of a type and a set of integrity constraints.

The goal of this paper is to understand how types interact with integrity constraints under the XML-Schema semantics. To focus on the nature of the interaction and to simplify the discussion, we consider XML-Schema specifications in which the type is a DTD and the constraints are simple keys and foreign keys[3]. We show that even in this simple setting, the interaction is already highly intricate such that the consistency check of XML-Schema specifications is infeasible. Note that in practice, an XML-Schema specification typically consists of a mild extension of a DTD as its type, as well as simple keys and foreign keys.

In this section, we first provide a formalism of DTDs, and then define keys and foreign keys under the XML-Schema semantics.

2.1 DTDs and XML Trees

Following [5,7], we formalize the definition of DTDs as follows. A *DTD* (Document Type Definition) is a tuple $D = (E, A, P, R, r)$, where:

- E is a finite set of *element types*;
- A is a finite set of *attributes*, disjoint from E.
- For each $\tau \in E$, $P(\tau)$ is a regular expression α, called the *element type definition* of τ: $\alpha \ ::= \ \mathtt{S} \mid \tau' \mid \epsilon \mid \alpha|\alpha \mid \alpha,\alpha \mid \alpha^*$, where $\mathtt{S}$ denotes the *string* type, $\tau' \in E$, ϵ is the empty word, and "|", "," and "∗" denote union, concatenation, and the Kleene closure;
- For each $\tau \in E$, $R(\tau)$ is a set of attributes in A;
- $r \in E$ and is called *the element type of the root*.

We normally denote element types by τ and attributes by $@l$, and assume that r does not appear in $P(\tau)$ for any $\tau \in E$. We also assume that each τ in $E \setminus \{r\}$

[3] We do not consider *relative* keys and foreign keys [2,3] here as the simple constraints suffice to demonstrate the complications caused by their interaction with types.

is *connected to* r, i.e., either τ appears in $P(r)$, or it appears in $P(\tau')$ for some τ' that is connected to r.

For example, recall the two DTDs D_1, D_2 given in the previous section. These DTDs can be naturally expressed in the formalism given above.

Given a DTD $D = (E, A, P, R, r)$, a *path* in D is a string $w_1 \cdots w_m$ over the alphabet $E \cup A \cup \{\mathtt{S}\}$ such that w_{i+1} is a symbol in the alphabet of $P(w_i)$ for each $i \in [1, m-2]$, and $w_m \in R(w_{m-1})$ or w_m is a symbol in the alphabet of $P(w_{m-1})$. Let $\mathit{Paths}(D) = \{p \mid p \text{ is a path in } D\}$. We say that a DTD is *non-recursive* if $\mathit{Paths}(D)$ is finite, and recursive otherwise. We also say that D is a *no-star* DTD if the Kleene star does not occur in any regular expression $P(\tau)$.

An XML document is typically modeled as a node-labeled tree. Below we describe valid XML documents of a DTD along the same lines as XML Schema [14].

Let $D = (E, A, P, R, r)$ be a DTD. An *XML tree* T *conforming to* D, written $T \models D$, is defined to be $(V, \mathit{lab}, \mathit{ele}, \mathit{att}, \mathit{val}, \mathit{root})$, where

- V is a finite set of *nodes*;
- *lab* is a function that maps each node in V to a label in $E \cup A \cup \{\mathtt{S}\}$; a node $v \in V$ is called an *element of type* τ if $\mathit{lab}(v) = \tau$ and $\tau \in E$, an *attribute* if $\mathit{lab}(v) \in A$, and a *text node* if $\mathit{lab}(v) = \mathtt{S}$;
- *ele* is a function that for any $\tau \in E$, maps each element v of type τ to a (possibly empty) list $[v_1, ..., v_n]$ of elements and text nodes in V such that $\mathit{lab}(v_1) \ldots \mathit{lab}(v_n)$ is in the regular language defined by $P(\tau)$;
- *att* is a partial function from $V \times A$ to V such that for any $v \in V$ and $@l \in A$, $\mathit{att}(v, @l)$ is defined iff $\mathit{lab}(v) = \tau$, $\tau \in E$ and $@l \in R(\tau)$;
- *val* is a partial function from V to string values such that for any node $v \in V$, $\mathit{val}(v)$ is defined iff $\mathit{lab}(v) = \mathtt{S}$ or $\mathit{lab}(v) \in A$;
- *root* is the root of T, $\mathit{root} \in V$ and $\mathit{lab}(\mathit{root}) = r$.

For any node $v \in V$, if $\mathit{ele}(v)$ is defined, then the nodes v' in $\mathit{ele}(v)$ are called the *subelements* of v. For any $@l \in A$, if $\mathit{att}(v, @l) = v'$, then v' is called *an attribute* of v. In either case we say that there is a *parent-child edge* from v to v'. The subelements and attributes of v are called its *children*. The graph defined by the parent-child relation is required to be a rooted tree.

For example, Figs. 1 (a) and (b) depict two XML trees that conform to the DTD D_1, and Fig. 1 (c) shows an XML tree that conforms to D_2.

In an XML tree T, the root is a unique node labeled with r. The subelements of an element of τ are ordered and their labels observe the regular expression $P(\tau)$. In contrast, its attributes are unordered and are identified by their labels. The function *val* assigns string values to attributes and to nodes labeled $\mathtt{S}$.

2.2 Keys and Foreign Keys

Given a DTD $D = (E, A, P, R, r)$, a *key over* D is a constraint of the form

$$(P, \{Q_1, \ldots, Q_n\}), \tag{1}$$

where $n \geq 1$ and P, Q_1, ..., Q_n are regular expressions over the alphabet $E \cup A \cup \{\mathtt{S}\}$. Expression P is called the *selector* of the key and is a regular expression conforming to the following BNF grammar [14].

$$\begin{array}{lll} \textit{selector} & ::= \textit{path} \mid \textit{path} \cup \textit{selector} \\ \textit{path} & ::= r//\textit{sequence} \mid \textit{sequence} \\ \textit{sequence} & ::= \tau \mid \_ \mid \textit{sequence}/\textit{sequence} \end{array}$$

Here _ is a wildcard that matches any element type, $\tau \in E$ and // represents the Kleene closure of _, that is, any possible finite sequence of node labels. The expressions Q_1, ..., Q_n are called the *fields* of the key and are defined by [14]:

$$\begin{array}{lll} \textit{field} & ::= \textit{path} \mid \textit{path} \cup \textit{field} \\ \textit{path} & ::= //\textit{sequence}/\textit{last} \mid /\textit{sequence}/\textit{last} \\ \textit{sequence} & ::= \epsilon \mid \tau \mid \_ \mid \textit{sequence}/\textit{sequence} \\ \textit{last} & ::= \mathtt{S} \mid @l \mid @\_ \end{array}$$

Here @_ is a wildcard that matches any attribute and $@l \in A$. This grammar differs from the one above in restricting the final step to match a text node or an attribute. A key containing exactly one field is called *unary*.

It should be mentioned that XML Schema expresses selectors and fields with *restricted* fragments of XPath [13], which are precisely the regular expressions defined above. In XPath, '_' represents *child* and '//' denotes *descendant*[4].

A *foreign key* over a DTD D is an expression of the form

$$(P, \{Q_1, \ldots, Q_n\}) \subseteq_{FK} (U, \{S_1, \ldots, S_n\}), \tag{2}$$

where P and U are the selectors of the foreign key, $n \geq 1$ and Q_1, ..., Q_n, S_1, ..., S_n are its fields. A foreign key containing one field in its left hand side and one field in its right hand side is called *unary*.

To define the notion of satisfaction of keys and foreign keys, we need to introduce some additional notation. Any pair of nodes x, y in an XML tree T with y a descendant of x uniquely determines the path, $\rho(x, y)$, from x to y. We say that y is *reachable* from x by following a regular expression β over D, denoted by $T \models \beta(x, y)$, iff $\rho(x, y) \in \beta$. For any fixed T, let $\textit{nodes}_\beta(x)$ stand for the set of nodes reachable from a node x by following the regular expression β: $\textit{nodes}_\beta(x) = \{y \mid T \models \beta(x, y)\}$. If there is only one node y such that $T \models \beta(x, y)$, then we define $x.\beta = y$.

Definition 1. *Given an XML tree $T = (V, \textit{lab}, \textit{ele}, \textit{att}, \textit{val}, \textit{root})$, T satisfies a key $(P, \{Q_1, \ldots, Q_n\})$, denoted by $T \models (P, \{Q_1, \ldots, Q_n\})$, if*

1. *For each $x \in \textit{nodes}_P(\textit{root})$ and $i \in [1, n]$, there is exactly one node y_i such that $T \models Q_i(x, y_i)$. Furthermore, $\textit{lab}(y_i) \in A$ or $\textit{lab}(y_i) = \mathtt{S}$.*
2. *For each $x_1, x_2 \in \textit{nodes}_P(\textit{root})$, if $\textit{val}(x_1.Q_i) = \textit{val}(x_2.Q_i)$ for all $i \in [1, n]$, then $x_1 = x_2$.*

That is, the values of Q_1, ..., Q_n uniquely identify the nodes reachable from the root by following path P. It further asserts that starting from each one of these nodes there is a single path conforming to the regular expression Q_i ($i \in [1, n]$).

[4] XPath [13] uses '*' to denote wildcard. Here we use '_' instead to avoid overloading the symbol '*' with the Kleene star found in DTDs.

Definition 2. *An XML tree $T = (V, lab, ele, att, val, root)$ satisfies a foreign key $(P, \{Q_1, \ldots, Q_n\}) \subseteq_{FK} (U, \{S_1, \ldots, S_n\})$, denoted by $T \models (P, \{Q_1, \ldots, Q_n\}) \subseteq_{FK} (U, \{S_1, \ldots, S_n\})$, if $T \models (U, \{S_1, \ldots, S_n\})$ and*

1. *For each $x \in nodes_P(root)$ and $i \in [1, n]$, there is exactly one node y_i such that $T \models Q_i(x, y_i)$. Furthermore, $lab(y_i) \in A$ or $lab(y_i) = \mathtt{S}$.*
2. *For each $x \in nodes_P(root)$ there exists a node $x' \in nodes_U(root)$ such that $val(x.Q_i) = val(x'.S_i)$ for each $i \in [1, n]$.*

The foreign key asserts that $(U, \{S_1, \ldots, S_n\})$ is a key and that for every node x reachable from the root by following path P, there is a node x' reachable from the root by following path U such that the $Q_1, \ldots, Q_n$-values of x are equal to the $S_1, \ldots, S_n$-values of x'.

Observe that condition 1 of Defs. 1 and 2 requires the *uniqueness* and *existence* of the fields involved. For example, the XML tree depicted in Fig. 1 (c) does not satisfy the key $(seq/clone, \{//DNA\})$ because the uniqueness condition imposed by the key is violated. Uniqueness conditions are required by the XML Schema semantics, but they are not present in various earlier proposals for XML keys coming from the database community [3,4,7,2].

Given an XML tree T and a set of keys and foreign keys Σ, we say that T satisfies Σ, denoted by $T \models \Sigma$, if $T \models \varphi$ for each $\varphi \in \Sigma$.

3 Consistency Problem: The General Case

We are interested in the consistency, or satisfiability, problem for XML-Schema specifications; that is, whether a given set of constraints and a DTD are satisfiable by an XML tree. Formally, for a class $\mathcal{C}$ of integrity constraints and a class $\mathcal{D}$ of DTDs, the input of the *consistency problem* $\mathsf{SAT}(\mathcal{D},\mathcal{C})$ is a DTD $D \in \mathcal{D}$ and a set of constraints $\Sigma \subseteq \mathcal{C}$ and the problem is to determine whether there is an XML tree T such that $T \models D$ and $T \models \Sigma$.

The same problem was considered in [7]. The constraint language introduced there is properly contained in the language defined in the previous section. Given a DTD D, element types τ, τ' and attributes $@l_1, \ldots, @l_n, @l'_1, \ldots, @l'_n$, keys and foreign keys in [7] are of the form

$$(r//\tau, \{@l_1, \ldots, @l_n\}), \tag{3}$$
$$(r//\tau, \{@l_1, \ldots, @l_n\}) \subseteq_{FK} (r//\tau', \{@l'_1, \ldots, @l'_n\}), \tag{4}$$

respectively. Then, from [7] we immediately derive:

Corollary 1. *The consistency problem for XML-Schema specifications, i.e., arbitrary DTDs and keys, foreign keys of the form (1) and (2), is undecidable.*

Observe that given an XML tree T conforming to a DTD D, for every node x reachable from the root by following a path $r//\tau$, there exists exactly one node reachable from x by following a path $@l_i$, which correspond to the attribute $@l_i$ of x. In this case, to check the consistency of an XML-Schema specification one does not need to consider the first condition of Defs. 1 and 2. Also from results of [7], for keys of such a form alone, and for arbitrary DTDs, there exists a *linear time* algorithm for the consistency problem.

However, none of the previous results give us any hint as to what happens when the first condition of Defs. 1 is imposed on arbitrary XML-Schema keys. Somewhat surprisingly, this extra condition makes the problem intractable, even for unary keys and very simple DTDs. By using a reduction from SAT-CNF [8], we can show the following:

Theorem 1. *The consistency problem is NP-hard for unary keys of form (1) and for non-recursive and no-star DTDs.* □

From these one can see that the consistency analysis is impossible for general XML-Schema specifications, and it is still not practical even if only unary keys are considered. In light of these we consider restricted cases of specifications in the next section, by imposing restrictions on the fields of keys and foreign keys.

4 Consistency Problem: A Restricted Case

In this section we study a class of XML-Schema constraints that are commonly found in practice, and investigate their consistency analysis. More specifically, we consider keys and foreign keys of the form

$$(P, \{@l_1, \ldots, @l_n\}), \tag{5}$$

$$(P, \{@l_1, \ldots, @l_n\}) \subseteq_{FK} (U, \{@l'_1, \ldots, @l'_n\}), \tag{6}$$

where P and U are regular expressions defined by the BNF grammar for *selector* expressions given in the previous section. Furthermore, if these constrains are defined over a DTD $D = (E, A, P, R, r)$, then they must satisfy the following *existence condition*: for each $\tau \in \mathit{last}(P)$, $\{@l_1, \ldots, @l_n\} \subseteq R(\tau)$, and for each $\tau' \in \mathit{last}(U)$, $\{@l'_1, \ldots, @l'_n\} \subseteq R(\tau')$, where $\mathit{last}(P)$ is the set of element types that are the last symbol of some string in the regular language defined by P. Note that these conditions can be checked in polynomial time.

Observe that the keys and foreign keys satisfying these conditions trivially satisfy requirement 1 of Defs. 1 and 2. For this kind of constraints, one can show the following by reduction to the emptiness problem of finite state automata.

Proposition 1. *For keys of the form (5) satisfying the existence condition and for arbitrary DTDs, the consistency problem is decidable in linear time.*

In practice, *unary* constraints are most commonly used, with the form:

$$(P, \{@l\}), \tag{7}$$

$$(P, \{@l\}) \subseteq_{FK} (U, \{@l'\}). \tag{8}$$

The next result tells us that when constraints are restricted to be unary and defined with attributes, the consistency problem is decidable even in the presence of foreign keys. This follows from results of [2]. However, the complexity is very high.

Proposition 2. *For constraints of the form (7), (8) satisfying the existence condition and for arbitrary DTDs, the consistency problem is PSPACE-hard and decidable.*

Obviously it is completely impractical to solve a PSPACE-hard problem. Thus one may want to consider further restrictions to get lower complexity. One approach is to further restrict constraints. Observe that constraints of the form (3) and (4) are a restriction of (7) and (8): P and U are required to be of the form $(r//\tau)$ for some element type τ. This helps, but not much: from [7] we get:

Proposition 3. *The consistency problem for unary constraints of form (3) and (4) is NP-complete for arbitrary DTDs, and is in PTIME for a fixed DTD.*

Note that Proposition 3 does not require the existence condition as it can be checked in linear time for constraints of form (3) and (4). The motivation for considering a fixed DTD is because in practice, one often defines the DTD of a specification at one time, but writes constraints in stages: constraints are added incrementally when new requirements are discovered.

Alternatively, one may want to further restrict the DTDs involved. However, this again does not help much: even under some rather severe restriction on DTDs, the consistency problem remains intractable. More precisely, we show that even if DTDs contain a fixed number of elements and attributes, the consistency problem for unary keys and foreign keys is NP-hard.

Let $k > 0$ be a fixed constant and let $\mathcal{D}_k$ be the class of DTDs $D = (E, A, P, R, r)$ such that $|E \cup A| \leq k$.

Theorem 2. *If $\mathcal{C}$ is the class of unary keys and foreign of the form (7), (8) satisfying the existence condition, then for each $k \geq 11$, $\mathsf{SAT}(\mathcal{D}_k, \mathcal{C})$ is NP-hard.*

This again is a new result that does not follow from previously published results on the consistency checking for XML.

5 Conclusion

We have shown that the semantics of XML-Schema constraints makes the consistency analysis of specifications rather intricate. The main results of the paper are summarized in Fig. 2, which indicate that static consistency checking for XML-Schema specifications is very hard: in general it is beyond reach (undecidable); for extremely restricted DTDs and constraints, it is still rather expensive

	DTD [7]	XML Schema
Keys and foreign keys	undecidable	undecidable
Unary keys and foreign keys	NP-complete	PSPACE-hard
Keys only	linear time	NP-hard
No constraints	linear time	linear time

Fig. 2. Complexity of the consistency problem

(NP-hard and PSPACE-hard). In particular, with only unary keys, the consistency problem is NP-hard under the XML-Schema semantics, in contrast to its linear-time decidability under the standard key semantics [2,7].

These negative results tell us that under the current semantics of XML-Schema constraints, there is no hope to efficiently check whether or not an XML-Schema specification makes sense. One may find that a seemingly perfect specification turns out to be inconsistent, after repeated failures to validate documents. The designers of XML Schema might want to take these results into account when revising the W3C recommendation.

Acknowledgments

Marcelo Arenas and Leonid Libkin are supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and from Bell University Laboratories. Wenfei Fan is currently on leave from Temple University, and is supported in part by NSF Career Award IIS-0093168.

References

1. S. Abiteboul, P. Buneman and D. Suciu *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufman, 2000.
2. M. Arenas, W. Fan and L. Libkin. On verifying consistency of XML specifications. In *PODS'02*, pages 259–270.
3. P. Buneman, S. Davidson, W. Fan, C. Hara and W. Tan. Keys for XML. In *WWW'10*, 2001, pages 201–210.
4. P. Buneman, S. Davidson, W. Fan, C. Hara and W. Tan. Reasoning about Keys for XML. In *DBPL*, 2001.
5. D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *J. Logic and Computation* 9(3):295–318, 1999.
6. S. Ceri, P. Fraternali, S. Paraboschi. XML: Current developments and future challenges for the database community. In *EDBT 2000*, pages 3–17.
7. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. In *PODS'01*, pages 114–125.
8. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.
9. D. Lee and W. W. Chu. Constraints-preserving transformation from XML document type definition to relational schema. In *ER'2000*, pages 323–338.
10. V. Vianu. A Web odyssey: From Codd to XML. In *PODS'01*, pages 1–15.
11. W3C. Extensible Markup Language (XML) 1.0. W3C Recommendation, Feb. 1998.
12. W3C. XML-Data, W3C Working Draft, Jan. 1998.
13. W3C. XML Path Language (XPath). W3C Working Draft, Nov. 1999.
14. W3C. XML Schema. W3C Recommendation, May 2001.
15. Full version: `http://www.cs.toronto.edu/~marenas/publications/xsc.pdf`.