On Verifying Consistency of XML Specifications

Marcelo Arenas
Department of Computer Science
University of Toronto
marenas@cs.toronto.edu

Wenfei Fan
Internet Management Research Dept
Bell Laboratories
wenfei@research.bell-labs.com

Leonid Libkin*
Department of Computer Science
University of Toronto
libkin@cs.toronto.edu

Abstract

XML specifications often consist of a type definition (typically, a DTD) and a set of integrity constraints. It has been shown previously that such specifications can be inconsistent, and thus it is often desirable to check consistency at compile-time. It is known that for general keys and foreign keys, and DTDs, the consistency problem is undecidable; however, it becomes NP-complete when all keys are one-attribute (unary), and tractable, if no foreign keys are used.

In this paper, we consider a variety of constraints for XML data, and study the complexity of the consistency problem. Our main conclusion is that in the presence of foreign keys, compile-time verification of consistency is usually infeasible. We look at two types of constraints: absolute (that hold in the entire document), and relative (that only hold in a part of the document). For absolute constraints, we extend earlier decidability results to the case of multi-attribute keys and unary foreign keys, and to the case of constraints involving regular expressions, providing lower and upper bounds in both cases. For relative constraints, we show that even for unary constraints, the consistency problem is undecidable. We also establish a number of restricted decidable cases.

1 Introduction

XML data, just like relational and object-oriented data, can be specified in a certain data definition language. While the exact details of XML data definition languages are still being worked out, it is clear that all of them would contain a form of document description, as well as integrity constraints. Constraints are naturally introduced when one considers transformations between XML and relational databases [16, 27, 26, 17, 20, 11], as well as integrating several XML documents [3, 4, 13].

Document descriptions usually come in the form of DTDs (Document Type Definition), and constraints are typically natural analogs of the most common relational integrity constraints: keys and foreign keys. Indeed, a large number of proposals (e.g., [29, 33, 30, 5]) support specifications for keys and foreign keys.

We investigate XML specifications with DTDs and keys and foreign keys. We study the *consistency*, or *satisfiability*, of such specifications: given a DTD and a set of constraints, whether there are XML documents conforming to the DTD and satisfying the constraints.

In other words, we want to validate XML specifications statically, at compile-time. Invalid XML specifications are likely to be more common than invalid specifications of other kinds of data, due to the rather complex interaction of DTDs and constraints. Furthermore, many specifications are not written at once, but rather in stages: as new requirements are discovered, they are added to the constraints, and thus it is quite possible that at some point they may be contradictory.

An alternative to the static validation would be a dynamic approach: simply attempt to validate a document with respect to a DTD and a set of constraints. This, however, would not tell us whether repeated failures are due to a bad specification, or problems with the documents.

The consistency analysis for XML specifications is not nearly as easy as for relational data (any set of keys and foreign keys can be declared on a set of relational attributes). Indeed, [14] showed that for

*Research affiliation: Bell Laboratories.

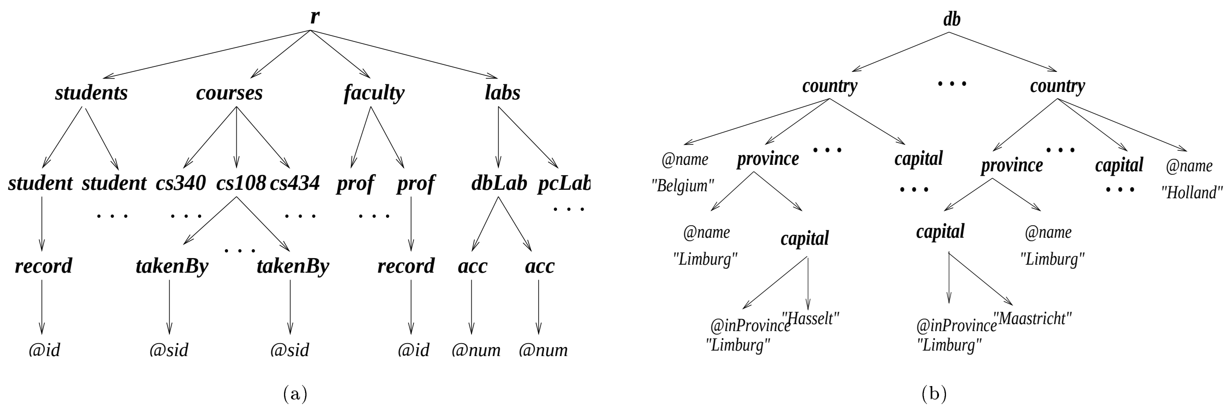

Figure 1: Examples of XML documents

DTDs and arbitrary keys and foreign keys, the consistency problem is undecidable. Furthermore, under the restriction that all keys and foreign keys are unary (single-attribute), the problem is NP-complete.

These results only revealed the tip of the iceberg, as many other flavors of XML constraints exist, and are likely to be added to future standards for XML such as XML Schema [33]. One of our goals is to study such constraints. In particular, we concentrate on constraints with regular expressions, and relative constraints that only hold in a part of the document. Furthermore, for classes of constraints with high lower bounds, we are interested in their tractable, or at least decidable restrictions. We now give examples of new kinds of constraints considered here, and explain the consistency problem for them.

<u>*Constraints with regular expressions.*</u> As XML data is hierarchically structured, one is often interested in constraints specified by regular expressions. For example, consider an XML document (represented as a node-labeled tree) in Fig. 1 (a), which conforms to the following DTD for schools:

```
<!ELEMENT  r  (students, courses, faculty, labs)>
<!ELEMENT  students   (student+)>
<!ELEMENT  courses    (cs340, cs108, cs434)>
<!ELEMENT  faculty    (prof+)>
<!ELEMENT  labs       (dbLab, pcLab)>
<!ELEMENT  student    (record)>
                /* similarly for prof
<!ELEMENT  cs434      (takenBy+)
                /* similarly for cs340, cs108
<!ELEMENT  dbLab      (acc+)
                /* similarly for pcLab
```

Here we omit the descriptions of elements whose type is string (PCDATA). Assume that each *record* element has an attribute *id*, each *takenBy* has an attribute *sid* (for student id), and each *acc* has an attribute *num*.

One may impose the following constraints over the DTD of that document:

$$\begin{aligned}
&r._^*.(student \cup prof).record.id \to \\
&\qquad\qquad r._^*.(student \cup prof).record,\\
&r._^*.cs434.takenBy.sid \subseteq r._^*.student.record.id,\\
&r._^*.dbLab.acc.num \subseteq r._^*.cs434.takenBy.sid,\\
&r._^*.cs434.takenBy.sid \to r._^*.cs434.takenBy.
\end{aligned}$$

Here $\_$ is a wildcard that matches any label (tag) and $\_^*$ is its Kleene closure that matches any path. The first constraint says that *id* is a key for all records of *student*s and *professor*s. Furthermore, *sid* is a key for students taking cs434. The other constraints specify foreign keys, which assert that cs434 can only be taken by students, and only students who are taking cs434 can have an account in the database lab.

Clearly, there is an XML tree satisfying both the DTD and the constraints. As was mentioned earlier, specifications are rarely written at once. Now suppose a new requirement is discovered: all faculty members must have a dbLab account. Consequently, one adds a new foreign key:

$$\begin{aligned}
r.faculty.prof.record.id &\subseteq r._^*.dbLab.acc.num,\\
r._^*.dbLab.acc.num &\to r._^*.dbLab.acc.
\end{aligned}$$

However, this addition makes the whole specification inconsistent. This is because previous constraints postulate that *dbLab* users are students taking cs434, and no professor can be a student since *id* is a key for both students and professors, while the new foreign key insists upon professors also being *dbLab* users and the DTD enforces at least one professor to be present in the document. Thus no XML document both conforms to the DTD and satisfies all the constraints.

The consistency problem for regular expression constraints is at least as hard as for constraints specified for element types with simple attributes: NP-hard in the unary case and undecidable in general [14]. We use results from [2, 14, 24] to show that in the unary case, the problem remains decidable, but the lower bound becomes PSPACE.

Relative integrity constraints. Many types of constraints are specified for an entire document. A different kind of constraints, called *relative*, was proposed recently [5] – those constraints only hold in a part of a document. As an example, consider an XML document that for each country lists its administrative subdivisions (e.g., into provinces or states), as well as capitals of provinces. A DTD is given below and an XML document conforming to it is depicted in Figure 1 (b).

```
<!ELEMENT  db       (country+)>
<!ELEMENT  country  (province+, capital+)>
<!ELEMENT  province (capital, city*)>
```

Each country has a nonempty sequence of provinces and a nonempty sequence of province capitals, and for each province we specify its capital and perhaps other cities. Each country and province has an attribute *name*, and each capital has an attribute *InProvince*.

Now suppose we want to define keys for countries and provinces. One can state that country *name* is a key for *country* elements. It is also tempting to say that *name* is a key for *province*, but this may not be the case. The example in Figure 1 (b) clearly shows that; which *Limburg* one is interested in probably depends on whether one's interests are in database theory, or in the history of the European Union. To overcome this problem, we define *name* to be a key for province *relative* to a country; indeed, it is extremely unlikely that two provinces of the same country would have the same name. Thus, our constraints are:

$$\begin{array}{l} \textit{country.name} \to \textit{country}, \\ \textit{country(province.name} \to \textit{province)}, \\ \textit{country(capital.inProvince} \to \textit{capital)}, \\ \textit{country(capital.inProvince} \subseteq \textit{province.name)}. \end{array}$$

The first constraint is like those we have encountered before: it is an *absolute* key, which applies to the entire document. The rest are *relative constraints* which are specified for sub-documents rooted at *country* elements. They assert that for each country, *name* is a key of *province* elements, *inProvince* is a key of all *capital* descendants of the country element and it is a foreign key referring to *name* of *province* elements in the same sub-document. In contrast to regular expression constraints given earlier, these constraints are defined for element types, e.g., the first constraint is a key for all *country* elements in the entire document, and the third constraint is a (relative) key for all *capital* elements in a sub-document rooted at a *country* node.

To illustrate the interaction between constraints and DTDs, observe that the above specification – which might look reasonable at first – is actually inconsistent!

To see this, let T be a tree that satisfies the specification. The constraints say that for any sub-document rooted at a country c, the number of its *capital* elements is at most the number of *province* elements among c's descendants. The DTD says that each *province* has a *capital* element as a child and that each *country* element has at least one *capital* child. Thus, the number of *capital* descendants of c is larger than the number of *province* descendants of c, which contradicts the previous bound. Hence, the specification is inconsistent.

Relative constraints appear to be quite useful for capturing information about XML documents that cannot possibly be specified by absolute constraints. It turns out, however, that the consistency problem is much harder for them: it is undecidable even for single-attribute keys and foreign keys.

Given this negative result, we look at restrictions that would give us decidability. They come in the form of extra conditions on the "geometry" of foreign keys that relate the two sides of the inclusion in the DTD tree representing a non-recursive DTD. We show that the problem is decidable if relative constraints are "hierarchical"; furthermore, if foreign keys do not talk about attributes that are "too far" from each other, the problem is PSPACE-complete.

Tractable and decidable restrictions. Since expensive lower bounds, and even undecidability, were established for most versions of the consistency problem, we would like to see some interesting tractable, or decidable, restrictions. In case of absolute constraints, the results of [14] consider either single attributes or multi-attribute sets for both keys and foreign keys, and thus say nothing about the intermediate case in which only keys are allowed to be multi-attribute. This class of constraints is rather common and arises when relational data is translated into XML. While often identifiers are used as single-attribute keys, other sets of attributes can form a key as well (e.g., via SQL `unique` declaration) and those typically contain more than one attribute. We show that the consistency problem for this class of constraints, when every key is primary (i.e., at most one key is defined for each element type), remains decidable.

A number of trivial restrictions for tractability of absolute constraints are known (e.g., a fixed DTD, no foreign keys). Restrictions on DTDs are unlikely to help: [14] showed that the consistency problem for unary absolute constraints is NP-hard for very simple DTDs (no Kleene star, no recursion). There are two further ways to restrict the problem: one can impose a bound on the number of constraints, or a bound on the depth of the DTDs. We show that neither one in isolation gives us tractability, but when the two restrictions are combined, the consistency problem is in NLOGSPACE.

The main conclusion of this paper is that while many proposals such as XML Schema [33] and XML Data [30] support the facilities provided by the DTDs as well as integrity constraints, and while it is possible to write inconsistent specifications, checking consistency at compile-time appears to be infeasible, even for fairly small specifications.

Related work. Consistency was studied for other data models, such as object-oriented and extended relational (e.g., with support for cardinality constraints), see [9, 10, 19].

A number of specifications for XML keys and foreign keys have been proposed, e.g., XML Schema [33], XML-Data [30]. A recent proposal [5] introduced relative constraints, which were further studied in [6]. To the best of our knowledge, consistency of XML constraints in the presence of schema specifications was only investigated in [14]. However, [14] did not consider relative constraints, constraints defined with regular expressions and the class of multi-attribute keys and unary foreign keys. Other constraints for semi-structured data, different from those considered here, were studied in, e.g. [2, 7, 15]. The latter also studies the consistency problem; the special form of constraints used there makes it possible to encode consistency as an instance of conjunctive query containment.

Organization. Section 2 defines DTDs, and absolute keys and foreign keys for XML. Section 3 studies the class of absolute multi-attribute keys and unary foreign keys, and the class of regular expression constraints which is an extension of absolute constraints with regular path expressions. Section 4 defines and investigates relative keys and foreign keys. We also provide several complexity results for implication of XML constraints. Section 5 summarizes the main results of the paper.

2 Notations

DTDs, XML trees, paths We formalize the notion of DTDs as follows (cf. [29, 8, 23, 14]).

Definition 2.1 *A* DTD *(Document Type Definition) is defined to be $D = (E, A, P, R, r)$, 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 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 $E \setminus \{r\}$ 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.

An XML document is typically modeled as a node-labeled tree. Below we describe valid XML documents w.r.t. a DTD, along the same lines as XQuery [34], XML Schema [33] and DOM [28].

Definition 2.2 *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 $(V, lab, ele, att, val, root)$, 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 $lab(v) = \tau$ and $\tau \in E$, an* attribute *if $lab(v) \in A$, and a* text node *if $lab(v) = \mathtt{S}$;*
- *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 $lab(v_1) \ldots lab(v_n)$ 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 $l \in A$, $att(v, l)$ is defined iff $lab(v) = \tau$, $\tau \in E$ and $l \in R(\tau)$;*
- *val is a partial function from V to string values such that for any node $v \in V$, $val(v)$ is defined iff $lab(v) = \mathtt{S}$ or $lab(v) \in A$;*
- *$root$ is the root of T: $root \in V$ and $lab(root) = r$.*

For any node $v \in V$, if $ele(v)$ is defined, then the nodes v' in $ele(v)$ are called the subelements *of v. For any $l \in A$, if $att(v, l) = v'$, 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.*

In an XML tree T, for each $v \in V$, there is a unique path of parent-child edges from the root to v, and each node has at most one incoming edge. The root is a unique node labeled with r. If a node x is labeled τ in E, then the functions *ele* and *att* define the children of x, which are partitioned into subelements and attributes. The subelements of x are ordered and their labels observe the regular expression $P(\tau)$. In contrast, its attributes are unordered and are identified by their labels (names). The function *val* assigns string values to attributes and to nodes labeled S.

Our model is simpler than the models of XQuery and XML Schema as DTDs support only one basic type (PCDATA or string) and do not have complex type constructs. Unlike the data model of XQuery, we do not consider nodes representing namespaces, processing instructions and references. These simplifications do not affect the lower bounds, however.

We also use the following notations. Referring to an XML tree T, if x is a τ element in T and l is an attribute in $R(\tau)$, then $x.l$ denotes the l attribute value of x, i.e., $x.l = val(att(x, l))$. If X is a list $[l_1, \ldots, l_n]$ of attributes in $R(\tau)$, then $x[X] = [x.l_1, \ldots, x.l_n]$. For any element type $\tau \in E$, $ext(\tau)$ denotes the set of all the τ elements in T. For any $l \in R(\tau)$, $ext(\tau.l)$ denotes $\{x.l \mid x \in ext(\tau)\}$, the set of all the l-attribute values of τ nodes.

Given a DTD $D = (E, A, P, R, r)$ and element types $\tau, \tau' \in E$, a string $\tau_1.\tau_2.\cdots.\tau_n$ over E is a *path in D from τ to τ'* if $\tau_1 = \tau$, $\tau_n = \tau'$ and for each $i \in [2, n]$, τ_i is a symbol in the alphabet of $P(\tau_{i-1})$. Moreover, $Paths(D) = \{p \mid$ there is $\tau \in E$ such that p is a path in D from r to $\tau\}$.

We say that a DTD is *non-recursive* if $Paths(D)$ 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 $P(\tau)$ (note that this is a stronger restriction than being $*$-free: a regular expression without the Kleene star yields a finite language, while the language of a $*$-free regular expression may still be infinite as it allows boolean operators including complement).

Keys and foreign keys We consider two forms of constraints for XML: *absolute constraints* that hold on the entire document, denoted by $\mathcal{AC}$; and *relative constraints* that hold on certain sub-documents, denoted by $\mathcal{RC}$. Below we define absolute keys and foreign keys; their variations using regular expressions will be defined in Section 3.2, and relative constraints will be formally defined in Section 4. The constraints given in Section 1 are instances of regular constraints and relative constraints, which are slightly different from what we present in this section.

A class of absolute keys and foreign keys, denoted by $\mathcal{AC}^{*,*}_{K,FK}$ (we shall explain the notation shortly), is defined for element types as follows. An $\mathcal{AC}^{*,*}_{K,FK}$ constraint φ over a DTD $D = (E, A, P, R, r)$ has one of the following forms:

- *Key.* $\tau[X] \to \tau$, where $\tau \in E$ and X is a nonempty set of attributes in $R(\tau)$. An XML tree T satisfies φ, denoted by $T \models \varphi$, if
$$\forall\, x, y \in ext(\tau)\ (\bigwedge_{l \in X} (x.l = y.l) \to x = y).$$
- *Foreign key.* It is a combination of two constraints: an inclusion constraint $\tau_1[X] \subseteq \tau_2[Y]$ and a key constraint $\tau_2[Y] \to \tau_2$, where $\tau_1, \tau_2 \in E$, X, Y are nonempty lists of attributes in $R(\tau_1)$, $R(\tau_2)$ of the same length. This constraint is satisfied by a tree T if $T \models \tau_2[Y] \to \tau_2$, and in addition
$$\forall\, x \in ext(\tau_1)\ \exists\, y \in ext(\tau_2)\ (x[X] = y[Y]).$$

That is, $\tau[X] \to \tau$ says that the X-attribute values of a τ element uniquely identify the element in $ext(\tau)$, and $\tau_1[X] \subseteq \tau_2[Y]$ says that the list of X-attribute values of every τ_1 node in T must match the list of Y-attribute values of some τ_2 node in T. We use two notions of equality to define keys: value equality is assumed when comparing attributes, and node identity is used when comparing elements. We shall use the same symbol '=' for both, as it will never lead to ambiguity.

Constraints of $\mathcal{AC}^{*,*}_{K,FK}$ are generally referred to as *multi-attribute* constraints as they may be defined with multiple attributes. An $\mathcal{AC}^{*,*}_{K,FK}$ constraint is said to be *unary* if it is defined in terms of a single attribute; that is, $|X| = |Y| = 1$ in the above definition. In that case, we write $\tau.l \to \tau$ for unary keys, and $\tau_1.l_1 \subseteq \tau_2.l_2$, $\tau_2.l_2 \to \tau_2$ for unary foreign keys. As in relational databases, we also consider *primary keys*: for each element type, at most one key can be defined.

We shall use the following notations for subclasses of $\mathcal{AC}^{*,*}_{K,FK}$: subscripts K and FK denote keys and foreign keys, respectively. When the primary key restriction is imposed, we use subscript PK instead of

K. The superscript '$*$' denotes multi-attribute, and '1' means unary. When both superscripts are left out, we mean that both keys and foreign keys are unary.

We shall be dealing with the following subclasses of $\mathcal{AC}^{*,*}_{K,FK}$: $\mathcal{AC}^{*,1}_{K,FK}$ denotes the class of multi-attribute keys and unary foreign keys; $\mathcal{AC}_{K,FK}$ is the class of unary keys and unary foreign keys; $\mathcal{AC}^{*,1}_{PK,FK}$ is the class of primary multi-attribute keys and unary foreign keys; and $\mathcal{AC}_{PK,FK}$ is the class of primary unary keys and unary foreign keys.

Consistency, or satisfiability problem We are interested in the consistency, or satisfiability problem for XML constraints considered together with DTDs: 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 we define the *XML specification consistency problem* $\mathsf{SAT}(\mathcal{C})$ as follows:

PROBLEM:	$\mathsf{SAT}(\mathcal{C})$
INPUT:	A DTD D, a set Σ of $\mathcal{C}$-constraints.
QUESTION:	Is there an XML tree T such that $T \models D$ and $T \models \Sigma$?

It is known [14] that $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable, but $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ and $\mathsf{SAT}(\mathcal{AC}_{PK,FK})$ are NP-complete.

Constraint implication Another classical problem is the *implication problem* for a class of constraints $\mathcal{C}$, denoted by $\mathsf{Impl}(\mathcal{C})$. Here, we consider it in the presence of DTDs. We write $(D, \Sigma) \vdash \phi$ if for every XML tree T, $T \models D$ and $T \models \Sigma$ imply $T \models \phi$. The implication problem $\mathsf{Impl}(\mathcal{C})$ is to determine, given any DTD D and any set $\Sigma \cup \{\phi\}$ of $\mathcal{C}$ constraints, whether or not $(D, \Sigma) \vdash \phi$. It was shown in [14] that $\mathsf{Impl}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable and $Impl(\mathcal{AC}_{K,FK})$ is coNP-complete.

3 Absolute integrity constraints

In this section, we establish the decidability and lower bounds for $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ and $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, the consistency problems for absolute primary multi-attribute keys and unary foreign keys, and for absolute regular unary keys and unary foreign keys. The class $\mathcal{AC}^{reg}_{K,FK}$ is an extension of $\mathcal{AC}_{K,FK}$ with regular path expressions, which will be defined shortly. We also study tractable restrictions of $\mathsf{SAT}(\mathcal{AC}_{K,FK})$.

3.1 Multi-attribute keys

We know that $\mathsf{SAT}(\mathcal{AC}_{K,FK})$, the consistency problem for unary absolute keys and foreign keys, is NP-complete. In contrast, $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable. This leaves a rather large gap: namely, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$, where only keys are allowed to be multi-attribute (note that since a key is part of a foreign key, the other restriction, to $\mathcal{AC}^{1,*}_{K,FK}$, does not make sense).

The reason for the undecidability of $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is that the implication problem for functional and inclusion dependencies can be reduced to it [14]. However, this implication problem is known to be decidable – in fact, in cubic time – for single-attribute inclusion dependencies [12], thus giving us hope to get decidability for multi-attribute keys and unary foreign keys.

While the decidability of the consistency problem for $\mathcal{AC}^{*,1}_{K,FK}$ is still an open problem, we resolve a closely-related problem, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. That is, the consistency problem for multi-attribute *primary* keys and unary foreign keys. Recall that a set Σ of $\mathcal{AC}^{*,1}_{K,FK}$ constraints is said to be *primary* if for each element type τ, there is at most one key in Σ defined for τ elements. We prove the decidability by showing that complexity-wise, the problem is equivalent to a certain extension of integer linear programming studied in [22]:

PROBLEM:	PDE (Prequadratic Diophantine Equations)
INPUT:	An integer $n \times m$ matrix A, a vector $\vec{b} \in \mathbb{Z}^n$, and a set $E \subseteq \{1, \ldots, m\}^3$.
QUESTION:	Is there a vector $\vec{x} \in \mathbb{N}^m$ such that $A\vec{x} \leq \vec{b}$ and $x_i \leq x_j \cdot x_k$ for all $(i, j, k) \in E$.

Note that for $E = \emptyset$, this is exactly the integer linear programming problem [24]. Thus, PDE can be thought of as integer linear programming extended with inequalities of the form $x \leq y \cdot z$ among variables. It is therefore NP-hard, and [22] proved an NEXPTIME upper bound for PDE. The exact complexity of the problem remains unknown.

Recall that two problems P_1 and P_2 are *polynomially equivalent* if there are PTIME reductions from P_1 to P_2 and from P_2 to P_1. We now show the following.

Theorem 3.1 *$\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ and PDE are polynomially equivalent.*

Proof sketch. The proof is by a careful extension of the coding used in [14] for unary keys and foreign keys; we show that conditions of the form $x \leq y \cdot z$ suffice to encode arbitrary keys. □

It is known that the linear integer programming problem is NP-hard and PDE is in NEXPTIME. Thus from Theorem 3.1 follows immediately:

Corollary 3.2 *$\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ is NP-hard, and can be solved in NEXPTIME.* □

Obviously we cannot obtain the exact complexity of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ without resolving the corresponding question for PDE, which appears to be quite hard [22].

The result of Theorem 3.1 can be generalized to *disjoint* $\mathcal{AC}^{*,1}_{K,FK}$ constraints: that is, a set Σ of $\mathcal{AC}^{*,1}_{K,FK}$ constraints in which for any two keys $\tau[X] \to \tau$ and $\tau[Y] \to \tau$ (on the same element type τ) in Σ, $X \cap Y = \emptyset$. The proof of Theorem 3.1 applies almost verbatim to show the following.

Corollary 3.3 *The restriction of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$ to disjoint constraints is polynomially equivalent to PDE.*

3.2 Regular expression constraints

Just as in XML-Data and XML Schema, specifications of $\mathcal{AC}^{*,*}_{K,FK}$ constraints are associated with element types. To capture the hierarchical nature of XML data, constraints can also be defined on a collection of elements identified by a regular path expression. It is common to find path expressions in query languages for XML (e.g., XQuery [34], XSL [32]).

We define a *regular (path) expression* over a DTD $D = (E, A, P, R, r)$ as follows:

$$\beta \ ::= \ \epsilon \ | \ \tau \ | \ \_ \ | \ \beta.\beta \ | \ \beta \cup \beta \ | \ \beta^*,$$

where ϵ denotes the empty word, τ is an element type in E, and '_' stands for wildcard that matches any symbol in E. We assume that β does not include the type r for the root element unless $\beta = r.\beta'$ where β' does not include r; thus, '_' is just a shorthand for $E \setminus \{r\}$. A regular expression defines a language over the alphabet E, which will be denoted by β as well.

Recall that a path in a DTD is a list of E symbols, that is, a string in E^*. Any pair of nodes x,y in an XML tree T with y a descendant of x uniquely determines the path, denoted by $\rho(x,y)$, 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 $nodes(\beta)$ stand for the set of nodes reachable from the root by following the regular expression β: $nodes(\beta) = \{y \mid T \models \beta(root,\ y)\}$. Note that for any element type $\tau \in E$, $nodes(r.\_^*.\tau) = ext(\tau)$.

We now define XML keys and foreign keys with regular expressions. Let $D = (E, A, P, R, r)$ be a DTD. Given a regular expression β over D, a *key* over D is an expression φ of the form $\beta.\tau.l \to \beta.\tau$, where $\tau \in E, l \in R(\tau)$. For any XML tree T that conforms to D, T satisfies φ ($T \models \varphi$) if for any $x,y \in nodes(\beta.\tau)$, $x.l = y.l$ implies $x = y$. Given two regular expressions β_1, β_2 over D, a *foreign key* over D is a combination of the inclusion constraint $\beta_1.\tau_1.l_1 \subseteq \beta_2.\tau_2.l_2$ and a key $\beta_2.\tau_2.l_2 \to \beta_2.\tau_2$, where $\tau_1, \tau_2 \in E$, $l_i \in R(\tau_i), i = 1,2$. Here $T \models \varphi$ if $T \models \beta_2.\tau_2.l_2 \to \beta_2.\tau_2$, and for every $x \in nodes(\beta_1.\tau_1)$ there exists $y \in nodes(\beta_2.\tau_2)$ such that $x.l_1 = y.l_2$.

We use $\mathcal{AC}^{reg}_{K,FK}$ to denote the set of all unary constraints defined with regular expressions. For example, the constraints over the school DTD that we have seen in Section 1 are instances of $\mathcal{AC}^{reg}_{K,FK}$. We do not consider multi-attribute constraints here, since they subsume $\mathcal{AC}^{*,*}_{K,FK}$ (by using $r.\_^*.\tau$ for τ), and thus consistency is undecidable for them.

For $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, we are able to establish both an upper and a lower bound. The lower bound already indicates that the problem is perhaps infeasible in practice, even for very simple DTDs. Finding the precise complexity of the problem remains open, and does not appear to be easy.

Theorem 3.4

a) $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ can be solved in NEXPTIME.

b) For non-recursive no-star DTDs, $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ is PSPACE-hard.

Proof sketch. a) Following [14], we code both the DTD and the constraints with linear inequalities over integers. However, compared to the proof of [14], the current proof is considerably harder due to the following. First, regular expressions in DTDs ("horizontal" regular expressions) interact in a certain way with regular path expressions in constraints (those correspond to "vertical" paths through the trees). To eliminate this interaction, we first reduce the problem to that over certain simple DTDs. The next problem is that regular path expressions in constraints can interact with each other. To model them with linear inequalities, we must introduce exponentially many variables that account for all possible Boolean combinations of those regular languages. The last problem is coding the DTDs in such a way that variables corresponding to each node have the information about the path leading to the node, and its relationship with the regular path expressions used in constraints. For that, we adopt the technique of [2], and tag all the variables in the coding of DTDs with states of the product automaton for all the automata corresponding to the regular expressions in constraints. Putting everything together, we reduce $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ to the existence of a solution of an (almost) instance of linear integer programming, which happens to be of exponential size; hence the NEXPTIME bound.

b) We encode the quantified boolean formula problem (QBF) as an instance of $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$. $\Box$

3.3 Restrictions for tractability

Since most flavors of the consistency problem for XML constraints are intractable, one is interested in finding suitable restrictions that admit polynomial-time algorithms. Some – rather severe – restrictions of this kind were given in [14]: for example, $\mathsf{SAT}(\mathcal{AC}_K)$ (no foreign keys) is solvable in PTIME, as is $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ for any fixed DTD. A more natural way of putting restrictions appears to be by specifying what kinds of regular expressions are allowed in the DTDs. However, the hardness result can be proved even for DTDs with neither recursion nor the Kleene star [14].

We show that the hardness result for $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is very robust, and withstands severe restrictions on constraints and DTDs: namely, a bound on the total number of constraints, and a bound on the depth of the DTD. However, imposing both of these bounds simultaneously makes $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ tractable.

For a non-recursive DTD D, the set $Paths(D)$ is finite. We define the *depth* of a non-recursive DTD D as $Depth(D) = \max_{p \in Paths(D)} length(p)$. By a depth-$d$ $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ we mean the restriction of $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ to pairs (D, Σ) with $Depth(D) \leq d$.

By a k-constraint $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ we mean the restriction of the consistency problem to pairs (D, Σ) where $|\Sigma| \leq k$ (considering each foreign key as one constraint). A k-constraint depth-d $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is a restriction to (D, Σ) with $|\Sigma| \leq k$ and $Depth(D) \leq d$.

Theorem 3.5 *For non-recursive no-star DTDs:*

a) both k-constraint $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ and depth-d $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ are NP-hard, for $k \geq 2$ and $d \geq 2$.

b) for any fixed $k, d > 0$, the k-constraint depth-d $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is solvable in NLOGSPACE. $\Box$

3.4 Lower bounds for implication

We now state a simple result that gives us lower bounds for the complexity of implication, if we know the complexity of the satisfiability problem. Recall that for a complexity class K, co$\mathsf{K}$ stands for $\{\bar{P} \mid P \in \mathsf{K}\}$.

Proposition 3.6 *For any class $\mathcal{C}$ of XML constraints that contains $\mathcal{AC}_{K,FK}$, if $\mathsf{SAT}(\mathcal{C})$ is K-hard for some complexity class K that contains DLOGSPACE, then $\mathsf{Impl}(\mathcal{C})$ is co$\mathsf{K}$-hard.* $\Box$

It was shown in [14] that $\mathsf{Impl}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable and $\mathsf{Impl}(\mathcal{AC}_{K,FK})$ is coNP-hard (in fact, coNP-complete). Now we derive:

Corollary 3.7 *$\mathsf{Impl}(\mathcal{AC}^{*,1}_{PK,FK})$ is coNP-hard, and $\mathsf{Impl}(\mathcal{AC}^{reg}_{K,FK})$ is PSPACE-hard.* $\Box$

4 Relative integrity constraints

Since XML documents are hierarchically structured, one may be interested in the entire document as well as in its sub-documents. The latter gives rise to *relative integrity constraints* [5, 6], that only hold on certain sub-documents. Below we define relative keys and foreign keys. Recall that we use $\mathcal{RC}$ to denote various classes of such constraints. We use the notation $x \prec y$ when x and y are two nodes in an XML tree and y is a descendant of x.

Let $D = (E, A, P, R, r)$ be a DTD. A *relative key* is an expression φ of the form $\tau(\tau_1.l \to \tau_1)$, where $l \in R(\tau_1)$. It says that relative to each node x of element type τ, l is a key for all the τ_1 nodes that are descendants of x. That is, if a tree T conforms to D, then $T \models \varphi$ if

$$\forall x \in ext(\tau)\ \forall y, z \in ext(\tau_1)$$
$$\big((x \prec y) \wedge (x \prec z) \wedge (y.l = z.l)\big) \to y = z.$$

A *relative foreign key* is an expression φ of the form $\tau(\tau_1.l_1 \subseteq \tau_2.l_2)$ and $\tau(\tau_2.l_2 \to \tau_2)$, where $l_i \in R(\tau_i), i = 1, 2$. This constraint indicates that for each x in $ext(\tau)$, l_1 is a foreign key of descendants of x of type τ_1 that references a key l_2 of τ_2-descendants of x. That is, $T \models \varphi$ iff $T \models \tau(\tau_2.l_2 \to \tau_2)$ and T satisfies

$$\forall x \in ext(\tau)\ \forall y_1 \in ext(\tau_1)\ \big((x \prec y_1) \to$$
$$\exists y_2 \in ext(\tau_2)\ \ ((x \prec y_2) \wedge y_1.l_1 = y_2.l_2)\big).$$

Here τ is called the *context type* of φ. Note that absolute constraints are a special case of the relative constraints when $\tau = r$: i.e., $r(\tau.l \to \tau)$ is the usual absolute key. Thus, the consistency problem for multi-attribute relative constraints is undecidable [14], and hence we only consider unary relative constraints here.

Following the notations for $\mathcal{AC}$, we use $\mathcal{RC}_{K,FK}$ to denote the class of all unary relative keys and foreign keys; $\mathcal{RC}_{PK,FK}$ means the primary key restriction.

For example, the constraints given in Section 1 over the country/province/capital DTD are instances of $\mathcal{RC}_{K,FK}$.

Recall that $\mathsf{SAT}(\mathcal{AC}_{K,FK})$, the consistency problems for absolute unary constraints, is NP-complete. One would be tempted to think that $\mathsf{SAT}(\mathcal{RC}_{K,FK})$, the consistency problems for relative unary constraints, is decidable as well. We show, however, in Section 4.1, that this is not the case. In light of this negative result, we identify several decidable subclasses of $\mathcal{RC}_{K,FK}$, which we call *hierarchical constraints*, in Section 4.2.

4.1 Undecidability of consistency

We now show that there is an enormous difference between unary absolute constraints, where $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is decidable in NP, and unary relative constraints. We consider the consistency problem for those, that is, $\mathsf{SAT}(\mathcal{RC}_{K,FK})$. Clearly, the problem is r.e.; it turns out that one cannot lower this bound.

Theorem 4.1 $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ *is undecidable.*

Proof sketch. By reduction from Hilbert's 10th problem [21]. □

In the proof of Theorem 4.1, all relative keys are primary. Thus, we obtain:

Corollary 4.2 $\mathsf{SAT}(\mathcal{RC}_{PK,FK})$*, the restriction of* $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ *to primary keys, is undecidable.* □

4.2 Decidable hierarchical constraints

Often, relative constraints for XML documents have a hierarchical structure. For example, to store information about books we can use the structure presented in Figure 2 (a), with four relative constraints:

$$library(book.isbn \to book), \tag{1}$$
$$book(author.name \to author), \tag{2}$$
$$book(chapter.number \to chapter), \tag{3}$$
$$chapter(section.title \to section). \tag{4}$$

(1) says that *isbn* is a key for books, (2) says that two distinct authors of the same book cannot have the same name and (3) says that two distinct chapters of the same book cannot have the same number. Constraint (4) asserts that two distinct sections of the same chapter cannot have the same title.

This specification has a hierarchical structure: there are three context types (*library*, *book*, and *chapter*), and if a constraint restricts one of them, it does not impose a restriction on the others. For instance, (1) imposes a restriction on the children of *library*, but it does not restrict the children of *book*. To verify if there is an XML document conforming to this schema, we can separately solve three consistency problems for absolute constraints: one for the subschema containing the element types *library*, *book* and *isbn*; another for *book, author, name, chapter* and *number*; and the last one for *chapter, section*, and *title*.

On the other hand, the example in figure 2 (b) does not have a hierarchical structure. In this case, *author_info* stores information about the authors of books, and, therefore, the following relative foreign key is included:

$$library(author\_info.name \to author\_info),$$
$$library(author.name \subseteq author\_info.name).$$

In this case, nodes of type *author* are restricted from context types *library* and *book*. Thus, we cannot separate the consistency problems for nodes of types *library* and *book*.

Below we formalize the notion of hierarchical relative constraints via the notion of *hierarchical* DTDs and sets of relative constraints. We prove that the consistency problem for this kind of DTDs and sets of constraints is decidable and show that under some additional restrictions, it is PSPACE-complete.

Let $D = (E, A, P, R, r)$ be a non-recursive DTD and Σ be a set of $\mathcal{RC}_{K,FK}$-constraints over D. We say that $\tau \in E$ is a restricted type if $\tau = r$ or τ is the context type of some Σ-constraint. A restricted node in an XML tree is a node whose type is a restricted type. The scope of a restricted node x is the subtree rooted at x consisting of: (1) all element nodes y that are reachable from x by following some path $\tau_1.\tau_2.\cdots.\tau_n$ ($n \geq 2$) such that for every $i \in [2, n-1]$, τ_i is not a restricted type, and (2) all the attributes of the nodes mentioned in (1). For instance, a node of type *book* in the example shown in figure 2 (a) is a restricted node and its scope includes a node of type *book* and some nodes of types *author*, *name*, *chapter* and *number*.

Given two restricted types τ_1 and τ_2, we say that τ_1, τ_2 is a conflicting pair in (D, Σ) if the scopes of the nodes of types τ_1 and τ_2 are related by a foreign key. Formally, $\tau_1, \tau_2 \in E$ is a *conflicting pair in* (D, Σ) iff $\tau_1 \neq \tau_2$ and (1) there is a path in D from τ_1 to τ_2 and τ_2 is the context type of some constraint in Σ; and (2) there is $\tau_3 \in E$ such that $\tau_2 \neq \tau_3$ and there exists a path in D from τ_2 to τ_3 and for some $\tau_4 \in E$, either $\tau_1(\tau_3.l_3 \subseteq \tau_4.l_4)$ or $\tau_1(\tau_4.l_4 \subseteq \tau_3.l_3)$ is in Σ. As an example, *library* and *book* in figure 2 (b) are a conflicting pair, whereas they are not in figure 2 (a).

If a specification (D, Σ) does not contain conflicting

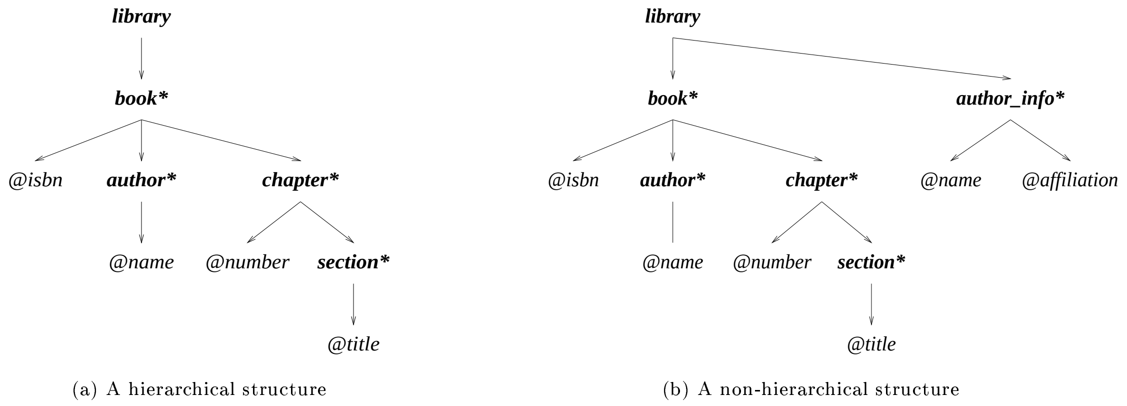

Figure 2: Two schemas for storing data in a library.

pairs, then we say that (D, Σ) is *hierarchical*. If this specification is consistent, then we can construct a tree conforming to D and satisfying Σ hierarchically, never looking at more than the scope of a single restricted node. We prove this property in theorem 4.3.

We define the language $\mathcal{HRC}_{K,FK}$ as $\{(D, \Sigma) \mid D$ is a non-recursive DTD, Σ is a set of $\mathcal{RC}_{K,FK}$-constraints and (D, Σ) is hierarchical$\}$. In this case, the input of $\mathsf{SAT}(\mathcal{HRC}_{K,FK})$ is $(D, \Sigma) \in \mathcal{HRC}_{K,FK}$, and the problem is to determine whether there is an XML tree conforming to D and satisfying Σ.

Theorem 4.3 $\mathsf{SAT}(\mathcal{HRC}_{K,FK})$ *is decidable.*

Proof sketch. To prove this theorem, first we prove a lemma stating the following. Suppose that $f : \mathbb{N} \to \mathbb{N}$ is a function such that for any consistent $(D, \Sigma) \in \mathcal{HRC}_{K,FK}$, there is a tree $T \models D$, $T \models \Sigma$ in which the size of the scope of each restricted node is at most the value of f on the size of the DTD naturally associated with that scope. Then $\mathsf{SAT}(\mathcal{HRC}_{K,FK})$ is in NSPACE(log(f)).

Second, by using the techniques of [14] we prove that $f(x)$ can be taken to be $2^{2^{x^k}}$, where $k \geq 1$ is a fixed constant. We conclude that $\mathsf{SAT}(\mathcal{HRC}_{K,FK})$ is in EXPSPACE. $\Box$

The algorithm in the proof gives an exponential space upper bound. We can lower it by imposing some further conditions on the "geometry" of constraints involved: namely, that for any inclusion constraint $\tau(\tau_1.l_1 \subseteq \tau_2.l_2)$, $\tau_1.l_1$ and $\tau_2.l_2$ are not too far from each other.

Formally, let D be a non-recursive DTD and Σ a set of $\mathcal{RC}_{K,FK}$-constraints over D such that (D, Σ) is hierarchical. Given $d > 1$, (D, Σ) is d-local if, whenever τ_1, τ_2 are restricted types, τ_2 is a descendant of τ_1 and no other node on the path from τ_1 to τ_2 is a context type of a Σ-constraint, then the length of that path is at most d.

Let d-$\mathcal{HRC}_{K,FK}$ be the language $\{(D, \Sigma) \mid (D, \Sigma) \in \mathcal{HRC}_{K,FK}$ and is d-local$\}$.

Theorem 4.4 *For any $d > 1$, $\mathsf{SAT}(d\text{-}\mathcal{HRC}_{K,FK})$ is PSPACE-complete.*

Proof sketch. The membership follows from the lemma used in the proof of Theorem 4.3. For hardness, we use reduction from QBF. $\Box$

4.3 Implication problem

Note that $\mathcal{RC}_{K,FK}$ and $\mathcal{HRC}_{K,FK}$ include $\mathcal{AC}_{K,FK}$. Thus from Proposition 3.6 we derive:

Corollary 4.5 $\mathsf{Impl}(\mathcal{RC}_{K,FK})$ *is undecidable, and* $\mathsf{Impl}(\mathcal{HRC}_{K,FK})$ *is PSPACE-hard.* $\Box$

5 Conclusion

We studied the problem of statically checking XML specifications, which may include various schema definitions as well as integrity constraints. As observed earlier, static validation is quite desirable as an alternative to dynamic checking. Our main conclusion is that, however desirable, the static checking is hard: even with very simple document definitions given by DTDs, and (foreign) keys as constraints, the complexity ranges from NP-hard to undecidable.

The main results are summarized in Figures 3, 4 (we also included the main results from [14] in those figures). When one deals with absolute constraints, which hold in an entire document, the general consistency problem is undecidable. It is solvable in NEXP-

Class	$\mathcal{AC}^{*,*}_{K,FK}$ [14]	$\mathcal{AC}^{*,1}_{PK,FK}$	$\mathcal{AC}^{reg}_{K,FK}$	$\mathcal{AC}_{K,FK}$ [14]
description	multi-attribute keys and foreign keys	multi-attribute primary keys, unary foreign keys	unary regular path constraints (keys, foreign keys)	unary keys, foreign keys
Upper bound	undecidable	NEXPTIME	NEXPTIME	NP
Lower bound	undecidable	NP	PSPACE	NP

Figure 3: Complexity of the consistency problem for absolute constraints

Class	$\mathcal{RC}^{*,*}_{K,FK}$ [14]	$\mathcal{RC}_{K,FK}$	$\mathcal{HRC}_{K,FK}$	d-$\mathcal{HRC}_{K,FK}$, $d > 1$
description	multi-attribute keys, foreign keys	unary keys foreign keys	unary hierarchical constraints	unary hierarchical constraints, d-local
Upper bound	undecidable	undecidable	EXPSPACE	PSPACE
Lower bound	undecidable	undecidable	PSPACE	PSPACE

Figure 4: Complexity of the consistency problem for relative constraints

TIME if foreign keys are single-attribute, and is NP-complete if so are all the keys as well. However, if regular expressions are allowed in single-attribute constraints, the lower bounds becomes at least PSPACE.

For relative constraints, which are only required to hold in a part of a document, the situation is quite bleak, as even the very simple case of single-attribute constraints is undecidable. By imposing certain restrictions on the "geometry" of those constraints, we can show that the problem is decidable, although PSPACE-hard; further restrictions make it PSPACE-complete. We also saw that these results are quite robust, as hardness is often achieved on relatively simple constraints and DTDs.

Although most of the results of the paper are negative, the techniques developed in the paper help study consistency of individual XML specification with type and constraints. These techniques include, e.g., the connection between regular expression constraints and integer linear programming and automata.

One open problem is to close the complexity gaps. However, these are by no means trivial: for example, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ was proved to be equivalent to a problem related to Diophantine equations whose exact complexity remains unknown. In the cases of $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ and $\mathsf{SAT}(\mathcal{HRC}_{K,FK})$, we think that it is more likely that our lower bounds correspond to the exact complexity of those problems. However, the algorithms are quite involved, and we do not yet see a way to simplify them to prove the matching upper bounds.

Another topic for future work is to study the interaction between more complex XML constraints, e.g., those defined in terms of XPath [31], and more complex schema specifications such as XML Schema [33] and the type system of XQuery [34]. Our lower bounds apply to those settings, but it is open whether upper bounds remain intact.

Acknowledgments We thank Michael Benedikt for his comments. M. Arenas and L. Libkin are supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and from Bell University Laboratories. W. Fan is currently on leave from Temple University, and is supported in part by NSF grant IIS 00-93168.

References

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

[2] S. Abiteboul and V. Vianu. Regular path queries with constraints. *JCSS*, 58(4):428–452, 1999.

[3] C. Baru et al. XML-based information mediation with MIX. In *SIGMOD'99*, pages 597–599.

[4] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *ICDT'99*, pages 296–313.

[5] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. In *WWW'10*, 2001.

[6] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. In *DBPL'01*.

[7] P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured databases. *JCSS*, 61(2):146–193, 2000.

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *JLC* 9 (1999), 295–318.

[9] D. Calvanese, M. Lenzerini. Making object-oriented schemas more expressive. In *PODS'94*, pages 243–254.

[10] D. Calvanese and M. Lenzerini. On the interaction between ISA and cardinality constraints. In *ICDE'94*, pages 204–213.

[11] M. Carey et al. XPERANTO: Publishing object-relational data as XML. In *WebDB 2000*.

[12] S. S. Cosmadakis, P. C. Kanellakis, and M. Y. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *J. ACM*, 37(1):15–46, Jan. 1990.

[13] A. Eyal and T. Milo. Integrating and customizing heterogeneous e-commerce applications. *VLDB Journal*, 10(1):16–38, 2001.

[14] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. In *PODS'01*, pages 114–125.

[15] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Verifying integrity constraints on web sites. In *IJCAI'99*, pages 614–619.

[16] M. Fernandez, A. Morishima, D. Suciu, and W. Tan. Publishing relational data in XML: the SilkRoute approach. *IEEE Data Eng. Bull.*, 24(2):12–19, 2001.

[17] D. Florescu and D. Kossmann. Storing and querying XML data using an RDMBS. *IEEE Data Eng. Bull.*, 22(3):27–34, 1999.

[18] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[19] P. C. Kanellakis. On the computational complexity of cardinality constraints in relational databases. *Information Processing Letters*, 11(2):98–101, 1980.

[20] D. Lee and W. W. Chu. Constraints-preserving transformation from XML document type definition to relational schema. In *ER'2000*.

[21] Y. Matiyasevich. *Hilbert's 10th Problem*. MIT Press, 1993.

[22] D. McAllester, R. Givan, C. Witty, and D. Kozen. Tarskian set constraints. In *LICS'96*, pages 138–147.

[23] F. Neven. Extensions of attribute grammars for structured document queries. In *DBPL'99*, pages 99–116.

[24] C. H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

[25] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[26] J. Shanmugasundaram et al. Efficiently publishing relational data as XML documents. In *VLDB'2000*.

[27] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *VLDB'1999*.

[28] W3C. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, Oct. 1998.

[29] W3C. Extensible Markup Language (XML) 1.0. W3C Recommendation, Feb. 1998.

[30] W3C. XML-Data. W3C Note, Jan. 1998.

[31] W3C. XML Path Language (XPath). Nov. 1999.

[32] W3C. XSL Transformations (XSLT). Nov. 1999.

[33] W3C. XML Schema. W3C Working Draft, May 2001.

[34] W3C. XQuery 1.0: An XML Query Language. W3C Working Draft, June 2001.