Consistency of XML Specifications

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

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

Abstract. Specifications of XML documents typically consist of typing information (for example, a DTD), and integrity constraints (for example, keys and foreign keys). We show that combining the two may lead to seemingly reasonable specifications that are nevertheless inconsistent: there is no XML document that both conforms to the DTD and satisfies the constraints. We then survey results on the complexity of consistency checking, and show that, depending on the classes of DTDs and constraints involved, it ranges from linear time to undecidable. Furthermore, we show that for some of the most common classes of specifications checking consistency is intractable.

1 Introduction

Although a number of dependency formalisms were developed for relational databases, functional and inclusion dependencies are the ones used most often. In fact, two subclasses of functional and inclusion dependencies, namely, keys and foreign keys, are most commonly found in practice. Both are fundamental to conceptual database design, and are supported by the SQL standard [34]. They provide a mechanism by which one can uniquely identify a tuple in a relation and refer to a tuple from another relation. They have proved useful in update anomaly prevention, query optimization and index design [1, 41].

XML (eXtensible Markup Language [11]) has become the prime standard for data exchange on the Web. XML data typically originates in databases. If XML is to represent data currently residing in databases, it should support keys and foreign keys, which are an essential part of the semantics of the data. A number of key and foreign key specifications have been proposed for XML, e.g., the XML standard (Document Type Definition, DTD) [11], XML Data [31] and XML Schema [40]. Keys and foreign keys for XML are important in, among other things, query optimization [37], data integration [7, 8, 22, 27], and in data transformations between XML and database formats [9, 18, 25, 26, 32, 38, 39].

XML data usually comes with a DTD[1] that specifies how a document is organized. Thus, a specification of an XML document may consist of both a DTD

[1] Throughout the chapter, by a DTD we mean its type specification; we ignore its ID/IDREF constraints since their limitations have been well recognized [12, 24].

L. Bertossi et al. (Eds.): Inconsistency Tolerance, LNCS 3300, pp. 15–41, 2004.

and a set of integrity constraints, such as keys and foreign keys. A legitimate question then is whether such a specification is *consistent*, or meaningful: that is, whether there exists an XML document that both satisfies the constraints and conforms to the DTD.

In the relational database setting, such a question would have a trivial answer: one can write arbitrary (`primary`) `key` and `foreign key` specifications in SQL, without worrying about consistency. However, DTDs (and other schema specifications for XML) are more complex than relational schema: in fact, XML documents are typically modeled as node-labeled trees, e.g., in XSLT [19], XQuery [10], XML Schema [40], XPath [20] and DOM [3]. Consequently, DTDs may interact with keys and foreign keys in a rather nontrivial way, as shown in the following examples.

Example 1. As a simple example, consider the DTD given below:

```
<!ELEMENT db  (foo)>
<!ELEMENT foo (foo)>
```

Observe that there exists no finite XML tree conforming to this DTD, and hence this specification – that consists only of a DTD and no constraints – is inconsistent. □

Example 2. To illustrate the interaction between XML DTDs and key/foreign key constraints, consider a DTD D, which specifies a (nonempty) collection of teachers:

```
<!ELEMENT teachers (teacher+)>
<!ELEMENT teacher  (teach, research)>
<!ELEMENT teach    (subject, subject)>
```

It says that a teacher teaches two subjects. Here we omit the descriptions of elements whose type is string (i.e., PCDATA in XML).

Assume that each teacher has an attribute `name` and each subject has an attribute `taught_by`. Attributes are single-valued. That is, if an attribute l is defined for an element type τ in a DTD, then in a document conforming to the DTD, each element of type τ must have a unique l attribute with a string value. Consider a set of unary key and foreign key constraints, Σ:

$$\begin{aligned} \mathit{teacher.name} &\rightarrow \mathit{teacher}, \\ \mathit{subject.taught\_by} &\rightarrow \mathit{subject}, \\ \mathit{subject.taught\_by} &\subseteq_{FK} \mathit{teacher.name}. \end{aligned}$$

That is, `name` is a key of `teacher` elements, `taught_by` is a key of `subject` elements and it is also a foreign key referencing `name` of `teacher` elements. More specifically, referring to an XML tree T, the first constraint asserts that two distinct `teacher` nodes in T cannot have the same `name` attribute value: the (string) value of `name` attribute uniquely identifies a `teacher` node. It should

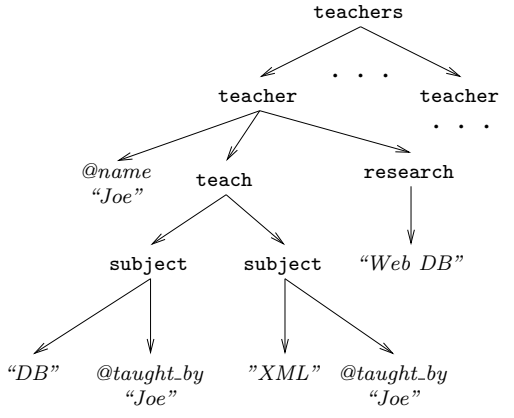


Fig. 1. An XML tree conforming to D

be mentioned that two notions of equality are used in the definition of keys: we assume string *value* equality when comparing `name` attribute values, and *node* identity when it comes to comparing `teacher` elements. The second key states that the `taught_by` attribute value uniquely identifies a `subject` node in T. The third constraint asserts that for any `subject` node x, there is a `teacher` node y in T such that the `taught_by` attribute value of x equals the `name` attribute value of y. Since `name` is a key of `teacher`, the `taught_by` attribute of any `subject` node refers to a unique `teacher` node.

Obviously, there exists an XML tree conforming to D, as shown in Figure 1. However, there is no XML tree that both conforms to D and satisfies Σ. To see this, let us first define some notation. Given an XML tree T and an element type τ, we use $ext(\tau)$ to denote the set of all the nodes labeled τ in T. Similarly, given an attribute l of τ, we use $ext(\tau.l)$ to denote the set of l attribute values of all τ elements. Then immediately from Σ follows a set of dependencies:

$$\begin{aligned} |ext(teacher.name)| &= |ext(teacher)|, \\ |ext(subject.taught\_by)| &= |ext(subject)|, \\ |ext(subject.taught\_by)| &\leq |ext(teacher.name)|, \end{aligned}$$

where $|\cdot|$ is the cardinality of a set. Therefore, we have

$$|ext(subject)| \leq |ext(teacher)|. \tag{1}$$

On the other hand, the DTD D requires that each teacher must teach two subjects. Since no sharing of nodes is allowed in XML trees and the collection of `teacher` elements is nonempty, from D follows:

$$1 \ < \ 2 \ \cdot \ |ext(teacher)| = |ext(subject)|. \tag{2}$$

Thus $|ext(teacher)| < |ext(subject)|$. Obviously, (1) and (2) contradict each other and as an immediate result, there exists no XML document that both satisfies Σ and conforms to D. In particular, the XML tree in Figure 1 violates the key $subject.taught\_by \rightarrow subject$. □

This example demonstrates that a DTD may impose dependencies on the cardinalities of certain sets of objects in XML trees. These *cardinality constraints* interact with keys and foreign keys. More specifically, keys and foreign keys also enforce cardinality constraints that interact with those imposed by DTD. This makes the consistency analysis of keys and foreign keys for XML far more intriguing than its relational counterpart.

The constraints in this example are fairly simple: there is an immediate analogy between such XML constraints and relational keys and foreign keys. There have been a number of proposals for supporting more powerful keys and foreign keys for XML (e.g., [11, 12, 40, 31]). Not surprisingly, the interaction between DTDs and those complicated XML constraints is more involved.

In light of this we are interested in the following family of the *consistency* (or *satisfiability*) problems, where $\mathcal{C}$ ranges over classes of integrity constraints:

PROBLEM	:	$\mathsf{SAT}(\mathcal{C})$.
INPUT	:	A DTD D, a set Σ of $\mathcal{C}$-constraints.
QUESTION	:	Is there an XML document that conforms to D and satisfies Σ?

In other words, we want to validate XML specifications statically, at compile-time. The main reason is twofold: first, complex interactions between DTDs and constraints are likely to result in inconsistent specifications, and second, an alternative dynamic approach to validation (simply check a document to see if it conforms to the DTD and satisfies the constraints) would not tell us whether repeated failures are due to a bad specification, or problems with the documents.

This chapter presents the complexity of the consistency analysis of XML specifications. We consider DTDs and a variety of XML keys and foreign keys commonly encountered in real-life XML specifications.

The next section gives a brief introduction to XML DTDs and XML documents. It is followed by the definitions of two basic forms of XML constraints, namely, *absolute* constraints that hold in the entire document, and *relative* constraints that only hold in a part of the document. Section 4 is devoted to the consistency analyses of XML specifications with absolute constraints, and Section 5 considers relative constraints. Extensions of the basic XML constraints by means of path expressions (regular expressions and XPath [20]), such as constraints proposed by XML Schema [40], are treated in Section 6. Finally, Section 7 identifies open problems for further study, and provides references to the original papers.

2 DTDs and XML Trees

In this section, we present a formalism of XML DTDs [11] and review the XML tree model.

Document Type Definition. We formalize the notion of DTDs as follows (cf. [11, 15, 35, 23]).

Definition 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*:*

$$\alpha \quad ::= \quad \mathtt{S} \mid \tau' \mid \epsilon \mid \alpha|\alpha \mid \alpha, \alpha \mid \alpha*,$$

  where S *denotes the* string *type,* $\tau' \in E$*,* ϵ *is the empty word, and "*$|$*", ","* *and "*$*$*" denote union, concatenation, and the Kleene closure, respectively. In this chapter we also use the following shorthands:* $\alpha+$ *for* $(\alpha, \alpha*)$ *and* $\alpha?$ *for* $(\epsilon|\alpha)$*. We refer to the set of* E *types appearing in* $P(\tau)$ *as* the alphabet of $P(\tau)$*.*
- *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.

Example 3. Let us consider the DTD D given in Example 2. In our formalism, D can be represented as (E, A, P, R, r), where $E = \{teachers, teacher, teach, research, subject\}$, $A = \{name, taught\_by\}$, $r = teachers$ and P, R are as follows:

$P(teachers) = teacher+$	$R(teachers) = \emptyset$
$P(teacher) = teach,\ research$	$R(teacher) = \{name\}$
$P(teach) = subject,\ subject$	$R(teach) = \emptyset$
$P(subject) = \mathtt{S}$	$R(subject) = \{taught\_by\}$
$P(research) = \mathtt{S}$	$R(research) = \emptyset$

□

XML Trees. 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 [10], XML Schema [40] and DOM [3].

Definition 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.* □

Intuitively, V is the set of nodes of the tree T. The mapping *lab* labels every node of V with a symbol (tag) from $E \cup A \cup \{\mathtt{S}\}$. Text nodes and attributes are leaves. For an element x of type τ, the functions *ele* and *att* define the children of x, which are partitioned into *subelements* and *attributes* according to $P(\tau)$ and $R(\tau)$ in the DTD D. The subelements of x are ordered and their labels satisfy 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 text nodes. We consider single-valued attributes. That is, if $l \in R(\tau)$ then every element of type τ has a unique l attribute with a string value. Since T has a tree structure, sharing of nodes is not allowed in T.

For example, Figure 1 depicts an XML tree valid w.r.t. the DTD given in Example 2.

Our model is simpler than the models of XQuery [10] and XML Schema [40] as DTDs support only one basic type (PCDATA or string) and do not have complex type constructs. Furthermore, we do not have nodes representing namespaces, processing instructions and references. These simplifications allow us to concentrate on the essence of the DTD/constraint interaction. It should further be noticed that they do not affect the lower bounds results in the chapter. It is also worth mentioning that we consider ordered XML trees in this paper, but removal of ordering does not affect the semantics of XML constraints and the complexity of their consistency and implication analyses.

Notation. In this chapter, we also use the following notation. 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 $x[X] = [x.l_1, \ldots, x.l_n]$. We write $|S|$ for the cardinality of a set S.

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, we define $Paths(D) = \{p \mid \text{there is } \tau \in E \text{ such that } p \text{ is a path in } D \text{ from } r \text{ to } \tau\}$.

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, which is a well-accepted concept with a standard definition [42]: 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).

3 Integrity Constraints for XML

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 both classes of constraints. A variation of $\mathcal{AC}$ using regular expressions will be defined in Section 6.1.

3.1 Absolute Keys and Foreign Keys

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*: $\tau[X] \rightarrow \tau$, where $\tau \in E$ and X is a nonempty set of attributes in $R(\tau)$. An XML tree T satisfies this constraint, denoted by $T \models \tau[X] \rightarrow \tau$, if

$$\forall\, x, y \in ext(\tau)\ (x[X] = y[X] \rightarrow x = y).$$

- *Foreign key*: $\tau_1[X] \subseteq_{FK} \tau_2[Y]$, where $\tau_1, \tau_2 \in E$, X and Y are nonempty lists of attributes in $R(\tau_1)$ and $R(\tau_2)$, respectively, and $|X| = |Y|$. This constraint is satisfied by a tree T, denoted by $T \models \tau_1[X] \subseteq_{FK} \tau_2[Y]$, if $T \models \tau_2[Y] \rightarrow \tau_2$, and in addition

$$\forall\, x \in ext(\tau_1)\ \exists\, y \in ext(\tau_2)\ (x[X] = y[Y]).$$

That is, $\tau[X] \rightarrow \tau$ says that the X-attribute values of a τ element uniquely identify the element in $ext(\tau)$, and $\tau_1[X] \subseteq_{FK} \tau_2[Y]$ says that the Y-attribute values of a τ_2 element uniquely identify the element in $ext(\tau_2)$ and 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. It is worth remarking that keys and foreign keys are defined in terms of XML attributes, which are of the string type and can not be null values.

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 $\tau.l \rightarrow \tau$ for unary keys, and $\tau_1.l_1 \subseteq_{FK} \tau_2.l_2$ for unary foreign keys. For example, the set of constraints considered in Example 2 are unary. As in relational databases, we also consider *primary keys*: for each element type, at most one key can be defined.

Example 4. To illustrate keys and foreign keys of $\mathcal{AC}^{*,*}_{K,FK}$, let us consider a DTD $D_1 = (E_1, A_1, P_1, R_1, r_1)$, where $E_1 = \{school, course, student, subject, enroll, name\}$, $A_1 = \{student\_id, course\_no, dept\}$, $r_1 = school$ and P_1, R_1 are as follows:

$P_1(school) = course*,\ student*$	$R_1(school) = \emptyset$
$P_1(course) = subject,\ enroll*$	$R_1(course) = \{dept,\ course\_no\}$
$P_1(student) = name$	$R_1(student) = \{student\_id\}$
$P_1(subject) = \mathtt{S}$	$R_1(subject) = \emptyset$
$P_1(enroll) = \epsilon$	$R_1(enroll) = \{student\_id\}$
$P_1(name) = \mathtt{S}$	$R_1(name) = \emptyset$

Typical $\mathcal{AC}^{*,*}_{K,FK}$ constraints over D_1 include:

$$\begin{aligned} student.student\_id &\rightarrow student, \\ course[dept, course\_no] &\rightarrow course, \\ enroll.student\_id &\subseteq_{FK} student.student\_id, \end{aligned}$$

The first two constraints are keys in $\mathcal{AC}^{*,*}_{K,FK}$ and the last constraint is a foreign key. The first and the last constraint are unary. □

We shall use the following notation 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. The first of these superscripts refers to keys, and the second to foreign keys.

In this chapter we shall be dealing with the following subclasses of $\mathcal{AC}^{*,*}_{K,FK}$:

- $\mathcal{AC}^{*,1}_{K,FK}$ is the class of multi-attribute keys and unary foreign keys;
- $\mathcal{AC}^{*,1}_{PK,FK}$ is the class of primary multi-attribute keys and unary foreign keys;
- $\mathcal{AC}^{1,1}_{K,FK}$ is the class of unary keys and unary foreign keys;
- $\mathcal{AC}^{1,1}_{PK,FK}$ is the class of primary unary keys and unary foreign keys;
- $\mathcal{AC}^{*}_{K}$ is the class of multi-attribute keys.

Since every foreign key implicitly contains a key, the class $\mathcal{AC}^{1,*}_{K,FK}$ of unary keys and multi-attributes foreign keys is equal to $\mathcal{AC}^{*,*}_{K,FK}$. Thus, we do not consider $\mathcal{AC}^{1,*}_{K,FK}$ in this chapter.

3.2 Relative Keys and Foreign Keys

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* [12, 13], 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.

A class of relative keys and foreign keys, denoted by $\mathcal{RC}^{*,*}_{K,FK}$, is defined as follows. An $\mathcal{RC}^{*,*}_{K,FK}$ constraint φ over a DTD $D = (E, A, P, R, r)$ has one of the following forms:

- *Relative key*: $\tau(\tau_1[X] \rightarrow \tau_1)$, where $\tau, \tau_1 \in E$ and X is a nonempty set of attributes in $R(\tau_1)$. It says that relative to each node x of element type τ, the set of attributes X 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

$$\forall\, x \in ext(\tau)\ \forall\, y, z \in ext(\tau_1)\ \big((x \prec y) \wedge (x \prec z) \wedge y[X] = z[X] \rightarrow y = z\big).$$

- *Relative foreign key*: $\tau(\tau_1[X] \subseteq_{FK} \tau_2[Y])$, where $\tau, \tau_1, \tau_2 \in E$, X and Y are nonempty lists of attributes in $R(\tau_1)$ and $R(\tau_2)$, respectively, and $|X| = |Y|$. It indicates that for each x in $ext(\tau)$, X is a foreign key of descendants of x of type τ_1 that references a key Y of τ_2-descendants of x. That is, T satisfies φ, denoted by $T \models \tau(\tau_1[X] \subseteq_{FK} \tau_2[Y])$, if $T \models \tau(\tau_2[Y] \rightarrow \tau_2)$ and T satisfies

$$\forall\ x \in ext(\tau)\ \forall\ y \in ext(\tau_1)\ \big((x \prec y) \rightarrow \exists\ z \in ext(\tau_2)\ \ ((x \prec z) \wedge y[X] = z[Y])\big).$$

Here τ is called the *context type* of φ. Note that absolute constraints are a special case of relative constraints when $\tau = r$: i.e., $r(\tau[X] \rightarrow \tau)$ is the usual absolute key. As in the case of absolute constraints, a relative 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 $\tau(\tau_1.l \rightarrow \tau)$ for relative unary keys, and $\tau(\tau_1.l_1 \subseteq_{FK} \tau_2.l_2)$ for relative unary foreign keys.

Example 5. Let us 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 2.

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

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

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

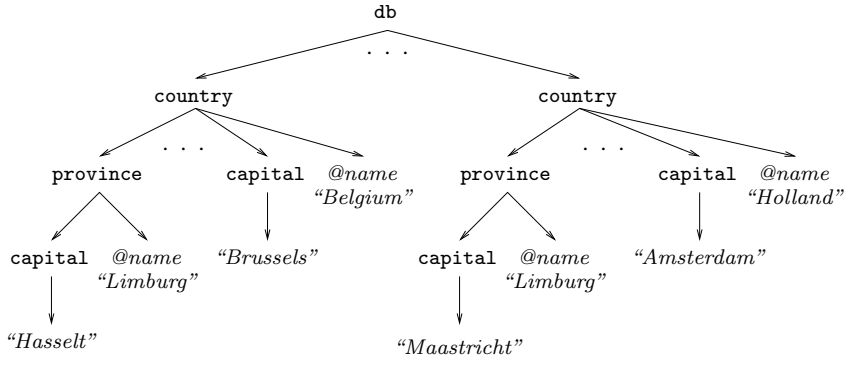


Fig. 2. An XML document storing information about countries and their administrative subdivisions

say that *name* is a key for *province*, but this may not be the case. The example in Figure 2 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:

$$country.name \rightarrow country,$$
$$country(province.name \rightarrow province).$$

The first constraint is like those we have encountered before: it is an *absolute* key, which applies to the entire document. The second one is a *relative constraint* which is specified for sub-documents rooted at *country* elements. It asserts that for each country, *name* is a key of *province* elements. Note that relative constraints are somewhat related to the notion of keys for weak entities in relational databases (cf. [41]). □

Following the notation for $\mathcal{AC}$, we denote subclasses of $\mathcal{RC}$ as follows:

- $\mathcal{RC}^{*,1}_{K,FK}$: the class of relative multi-attribute keys and unary foreign keys;
- $\mathcal{RC}^{*,1}_{PK,FK}$: the class of relative primary multi-attribute keys and unary foreign keys;
- $\mathcal{RC}^{1,1}_{K,FK}$: the class of relative unary keys and unary foreign keys;
- $\mathcal{RC}^{1,1}_{PK,FK}$: the class of relative primary unary keys and unary foreign keys;
- $\mathcal{RC}^{*}_{K}$: the class of relative multi-attribute keys.

As in the case of absolute constraints, every relative foreign key implicitly contains a relative key and, hence, the class $\mathcal{RC}^{1,*}_{K,FK}$ of unary keys and multi-attributes foreign keys is equal to $\mathcal{RC}^{*,*}_{K,FK}$. Thus, there is no need to consider $\mathcal{RC}^{1,*}_{K,FK}$.

4 Consistency of Absolute Keys and Foreign Keys

In this section we study the complexity of the consistency problem for absolute keys and foreign keys. We show that, in general, this problem is undecidable, and we identify several special cases of the problem that are decidable.

4.1 Undecidability of Consistency

The following result shows that in general it is not possible to verify statically whether an XML specification is consistent.

Theorem 1. $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ *is undecidable.* □

This theorem was proved in [23] by showing that the implication problem associated with keys and foreign keys in relational databases is undecidable, and then reducing (the complement of) the implication problem to the consistency problem for $\mathcal{AC}^{*,*}_{K,FK}$ constraints.

Given this negative result, it is desirable to find some restrictions on $\mathcal{AC}^{*,*}_{K,FK}$ that lead to decidable cases. We identify several of these classes in the next subsections.

4.2 Multi-attribute Keys

The reason for the undecidability of $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is that the implication problem for functional and inclusion dependencies in relational databases can be reduced to it [23]. However, this implication problem is known to be decidable – in fact, in cubic time – for single-attribute inclusion dependencies [21], 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, a closely-related problem, the consistency problem for multi-attribute *primary* keys and unary foreign keys, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$, has shown to be decidable [4]. 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. The decidability of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ is shown by proving that, complexity-wise, the problem is equivalent to a certain extension of integer linear programming studied in [33]:

PROBLEM	:	PDE (Prequadratic Diophantine Equations).
INPUT	:	An integer $n \times m$ matrix A, a vector $\boldsymbol{b} \in \mathbb{Z}^n$, and a set $E \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\} \times \{1, \ldots, m\}$.
QUESTION	:	Is there a vector $\boldsymbol{x} \in \mathbb{N}^m$ such that $A\boldsymbol{x} \leq \boldsymbol{b}$ 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 [36]. 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 [33] 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 vice versa. It is shown in [4] that $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ and PDE are polynomially equivalent. The following theorem is an immediate consequence of this result.

Theorem 2. $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ *is NP-hard, and can be solved in NEXPTIME.* □

Obviously the exact complexity of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ cannot be obtained without resolving the corresponding question for PDE, which appears to be quite hard [33].

The result of Theorem 2 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 2 applies almost verbatim to show the following.

Corollary 1. *The restriction of* $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$ *to disjoint constraints is NP-hard, and can be solved in NEXPTIME.* □

4.3 Unary Keys and Foreign Keys

One important subclass of $\mathcal{AC}^{*,*}_{K,FK}$ is $\mathcal{AC}^{1,1}_{K,FK}$, the class of unary keys and unary foreign keys. A cursory examination of existing XML specifications reveals that most keys and foreign keys are single-attribute constraints, i.e., unary. In particular, in XML DTDs, one can only specify unary constraints with ID and IDREF attributes.

The exact complexity of $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ was established in [23] by showing that this problem is polynomially equivalent to linear integer programming [36]:

PROBLEM	:	Linear Integer Programming.
INPUT	:	An integer $n \times m$ matrix A and vector $\boldsymbol{b} \in \mathbb{Z}^n$.
QUESTION	:	Is there a vector $\boldsymbol{x} \in \mathbb{N}^m$ such that $A\boldsymbol{x} \leq \boldsymbol{b}$?

Given that linear integer programming is known to be NP-complete, the following theorem is an immediate consequence of the polynomial equivalence of the two problems.

Theorem 3. $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ *is NP-complete.* □

Since all the flavors of the consistency problem presented so far are intractable, we next want to find suitable restrictions that admit polynomial-time algorithms. For instance, one might think that the primary key restriction would simplify the consistency analysis of $\mathcal{AC}^{1,1}_{K,FK}$ constraints. Unfortunately, as shown in [23], this is not the case.

Theorem 4. $\mathsf{SAT}(\mathcal{AC}^{1,1}_{PK,FK})$ *remains NP-complete.* □

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 [23]. In the rest of this section, we show that the hardness result for $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ 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 $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ tractable.

Recall that for a non-recursive DTD D, the set $Paths(D)$ is finite. We define the *depth* of a non-recursive DTD D as $\max_{p \in Paths(D)} length(p)$, denoted by $Depth(D)$. By a depth-d $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ we mean the restriction of $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ to pairs (D, Σ) with $Depth(D) \leq d$. By a k-constraint $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ we mean the restriction of the consistency problem to pairs (D, Σ) where $|\Sigma| \leq k$. A k-constraint depth-d $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ is a restriction to (D, Σ) with $|\Sigma| \leq k$ and $Depth(D) \leq d$. The following theorem was proved in [4].

Theorem 5. *For non-recursive no-star DTDs:*

a) both k-constraint $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ and depth-d $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ are NP-hard, for $k \geq 2$ and $d \geq 2$.
b) for any fixed $k, d > 0$, the k-constraint depth-d $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ is solvable in NLOGSPACE. □

4.4 Linear Time Decidable Cases

While the general consistency problem is undecidable, it is possible to identify some decidable cases of low complexity. The first one is checking whether a DTD has a valid XML tree. This is a special case of the consistency problem, namely, when the given set of $\mathcal{AC}^{*,*}_{K,FK}$ constraints is empty. A more interesting special case involves keys only.

It was shown in [23] that the problem of verifying whether a given DTD has a valid XML tree can be reduced to the emptiness problem for a context free grammar. Given that this reduction can be computed in linear time and the emptiness problem for a context free grammar can be solved in linear time (cf. [30]), the problem of checking whether a DTD has a valid XML tree can be solved in linear time. It was also shown in [23] that given any DTD D and any set Σ of keys in $\mathcal{AC}^{*}_{K}$ over D, Σ can be satisfied by an XML tree valid w.r.t. D if and only if D has a valid XML tree. Thus, the following theorem is a consequence of our previous discussion.

Theorem 6. *The following problems are decidable in linear time:*

a) Given any DTD D, whether there exists an XML tree valid w.r.t. D.
b) $\mathsf{SAT}(\mathcal{AC}^{*}_{K})$. □

4.5 The Implication Problem

Another classical problem, which is closely related to the consistency 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 $(D, \Sigma) \vdash \phi$.

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

Proposition 1. *For any class $\mathcal{C}$ of XML constraints that contains $\mathcal{AC}^{1,1}_{PK,FK}$, 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.* □

Along the same lines as Section 4.3 one can define k-constraint $\mathsf{Impl}(\mathcal{AC}^{1,1}_{K,FK})$ and depth-d $\mathsf{Impl}(\mathcal{AC}^{1,1}_{K,FK})$. Proposition 1 in fact remains intact under the depth-d and the k-constraint restrictions for $d \geq 2$ and $k \geq 2$. It has also been shown [23] that $\mathsf{Impl}(\mathcal{AC}^{*}_{K})$ is decidable in linear time. From these and the lower-bounds established for the consistency problem, we derive:

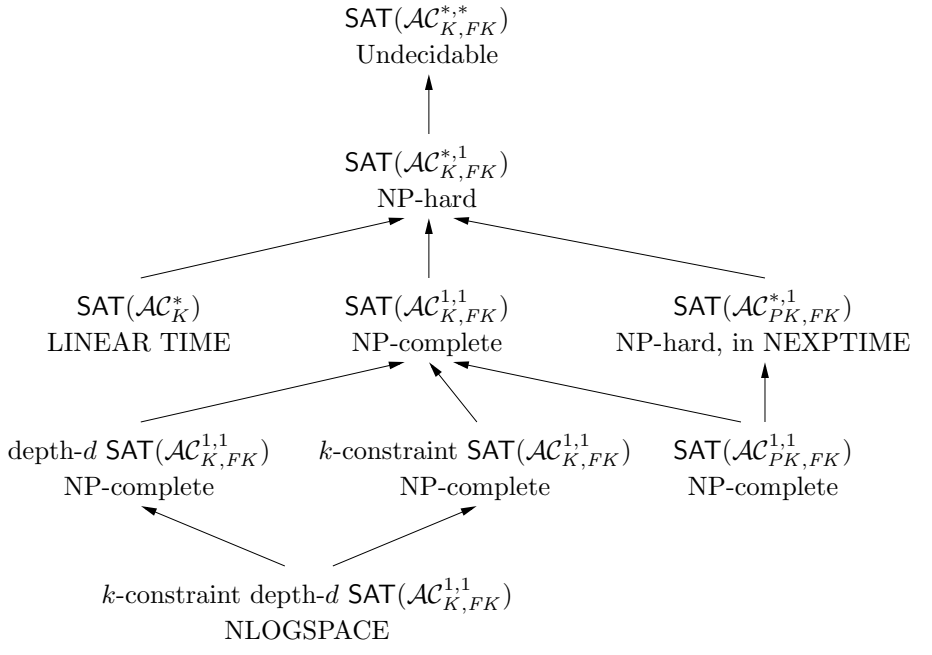


Fig. 3. A summary of the known complexity bounds for the consistency problem for absolute keys and foreign keys

Corollary 2. *For the implication problem for XML constraints,*

- $\mathsf{Impl}(\mathcal{AC}^{*,*}_{K,FK})$ *is undecidable;*
- *both* k*-constraint* $\mathsf{Impl}(\mathcal{AC}^{1,1}_{K,FK})$ *and depth-*d $\mathsf{Impl}(\mathcal{AC}^{1,1}_{K,FK})$ *are coNP-hard for* $d \geq 2$ *and* $k \geq 2$*, and so is* $\mathsf{Impl}(\mathcal{AC}^{*,1}_{PK,FK})$*;*
- $\mathsf{Impl}(\mathcal{AC}^{*,1}_{PK,FK})$ *is coNP-hard, and so are* $\mathsf{Impl}(\mathcal{AC}^{*,1}_{K,FK})$ *(and its restriction to disjoint constraints) and* $\mathsf{Impl}(\mathcal{AC}^{1,1}_{PK,FK})$*;*
- $\mathsf{Impl}(\mathcal{AC}^{*}_{K})$ *is in linear time.* □

4.6 Summary

Figure 3 shows a summary of the lower and upper bounds for the consistency problem for absolute keys and foreign keys. Note that in many cases we have matching lower and upper bounds. Also notice that for k-constraint $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$, depth-$d$ $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ and k-constraint depth-d $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ we are only considering non-recursive no-star DTDs.

5 Consistency of Relative Keys and Foreign Keys

In this section we study the consistency problem for relative keys and foreign keys. Relative constraints appear to be quite useful for capturing information about (hierarchical) XML documents that cannot possibly be specified by absolute constraints. However, it turns out that the complexity of their consistency analysis is, in general, higher than the complexity of the consistency problem for absolute constraints. In particular, we show that even for relative unary constraints the consistency problem is undecidable. In light of this negative result, we also identify some special cases of this problem that are decidable.

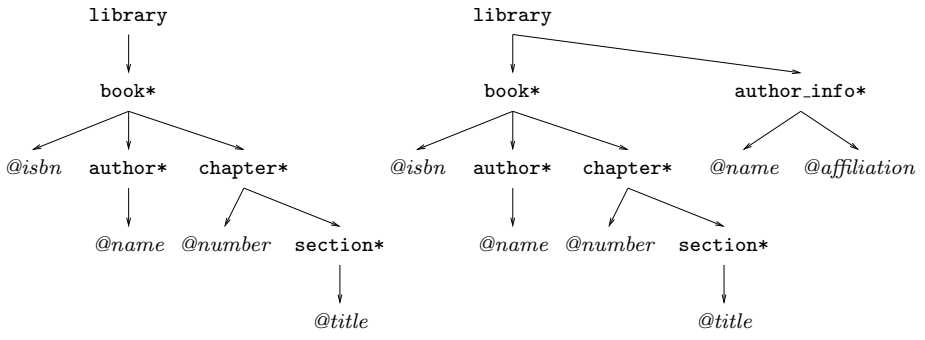
5.1 Undecidability of Consistency Analysis

Given that $\mathcal{RC}^{*,*}_{K,FK}$ contains $\mathcal{AC}^{*,*}_{K,FK}$ as a proper subclass, from Theorem 1 we obtain the following corollary.

Corollary 3. $\mathsf{SAT}(\mathcal{RC}^{*,*}_{K,FK})$ *is undecidable.* □

Since $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$, the consistency problem associated with absolute multi-attribute keys and unary foreign keys, is decidable, one would be tempted to think that $\mathsf{SAT}(\mathcal{RC}^{*,1}_{PK,FK})$, the consistency problem for relative multi-attribute keys and unary foreign keys, is also decidable. Even more, given that $\mathsf{SAT}(\mathcal{AC}^{1,1}_{K,FK})$ is NP-complete, one would be tempted to believe that $\mathsf{SAT}(\mathcal{RC}^{1,1}_{K,FK})$, the consistency problem for relative unary keys and foreign keys, must be decidable. However, it was shown in [4] that $\mathsf{SAT}(\mathcal{RC}^{1,1}_{K,FK})$ is not decidable, even if the primary key restriction is imposed.

Theorem 7. $\mathsf{SAT}(\mathcal{RC}^{1,1}_{PK,FK})$ *is undecidable.* □



(a) A hierarchical structure (b) A non-hierarchical structure

Fig. 4. Two schemas for storing data in a library

This undecidability was established by reduction from the Hilbert's 10th problem [29], a well known undecidable problem.

Corollary 4. $\mathsf{SAT}(\mathcal{RC}^{*,1}_{K,FK})$, $\mathsf{SAT}(\mathcal{RC}^{*,1}_{PK,FK})$ *and* $\mathsf{SAT}(\mathcal{RC}^{1,1}_{K,FK})$ *are undecidable.* □

5.2 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 given in Figure 4 (a), with four relative constraints:

$$library(book.isbn \rightarrow book), \tag{3}$$
$$book(author.name \rightarrow author), \tag{4}$$
$$book(chapter.number \rightarrow chapter), \tag{5}$$
$$chapter(section.title \rightarrow section). \tag{6}$$

(3) says that *isbn* is a key for books, (4) says that two distinct authors of the same book cannot have the same name and (5) says that two distinct chapters of the same book cannot have the same number. Constraint (6) 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, then it does not impose a restriction on the others. For instance, (3) 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 4 (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:

$$library(author.name \subseteq_{FK} author\_info.name).$$

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*.

The notion of hierarchical relative constraints was introduced in [4]. Below we introduce this notion via the notion of *hierarchical* DTDs and sets of relative constraints. Then, we show that the consistency problem for these kinds 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 $\mathcal{RC}^{1,1}_{K,FK}$-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 4 (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 are 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, $\tau_1, \tau_2 \in E$ are 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 $\tau_1(\tau_3.l_3 \subseteq_{FK} \tau_4.l_4)$ or $\tau_1(\tau_4.l_4 \subseteq_{FK} \tau_3.l_3)$ is in Σ. As an example, *library* and *book* in Figure 4 (b) are a conflicting pair, whereas they are not in Figure 4 (a).

If a specification (D, Σ) does not contain conflicting pairs, then (D, Σ) is said to be *hierarchical* [4]. We define the language $\mathcal{HRC}^{1,1}_{K,FK}$ as $\{(D, \Sigma) \mid D$ is a non-recursive DTD, Σ is a set of $\mathcal{RC}^{1,1}_{K,FK}$-constraints and (D, Σ) is hierarchical$\}$. In this case, the input of $\mathsf{SAT}(\mathcal{HRC}^{1,1}_{K,FK})$ is $(D, \Sigma) \in \mathcal{HRC}^{1,1}_{K,FK}$, and the problem is to determine whether there is an XML tree conforming to D and satisfying Σ.

It was shown in [4] that if a $\mathcal{HRC}^{1,1}_{K,FK}$-specification is consistent, then a tree conforming to D and satisfying Σ can be constructed hierarchically, never looking at more than the scope of a single restricted node. More precisely, it was shown in [4] that:

Theorem 8. $\mathsf{SAT}(\mathcal{HRC}^{1,1}_{K,FK})$ *is PSPACE-hard. The problem can be solved in EXPSPACE.* □

The exponential space upper bound can be lowered by imposing some further conditions on the "geometry" of constraints involved: namely, that for any inclu-

sion constraint $\tau(\tau_1.l_1 \subseteq_{FK} \tau_2.l_2)$, $\tau_1.l_1$ 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 $\mathcal{RC}^{1,1}_{K,FK}$-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 a 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 $d\text{-}\mathcal{HRC}^{1,1}_{K,FK}$ be the language $\{(D, \Sigma) \mid (D, \Sigma) \in \mathcal{HRC}^{1,1}_{K,FK}$ and is d-local$\}$. It was shown in [4] that:

Theorem 9. *For any $d > 1$, $\mathsf{SAT}(d\text{-}\mathcal{HRC}^{1,1}_{K,FK})$ is PSPACE-complete.* □

5.3 A Linear Time Decidable Case

As in the case of absolute keys, it can be shown that given any DTD D and any set Σ of keys in $\mathcal{RC}^*_K$ over D, Σ can be satisfied by an XML tree valid w.r.t. D if and only if D has a valid XML tree. Thus, the following theorem is analogous to Theorem 6.

Theorem 10. $\mathsf{SAT}(\mathcal{RC}^*_K)$ *can be solved in linear time.* □

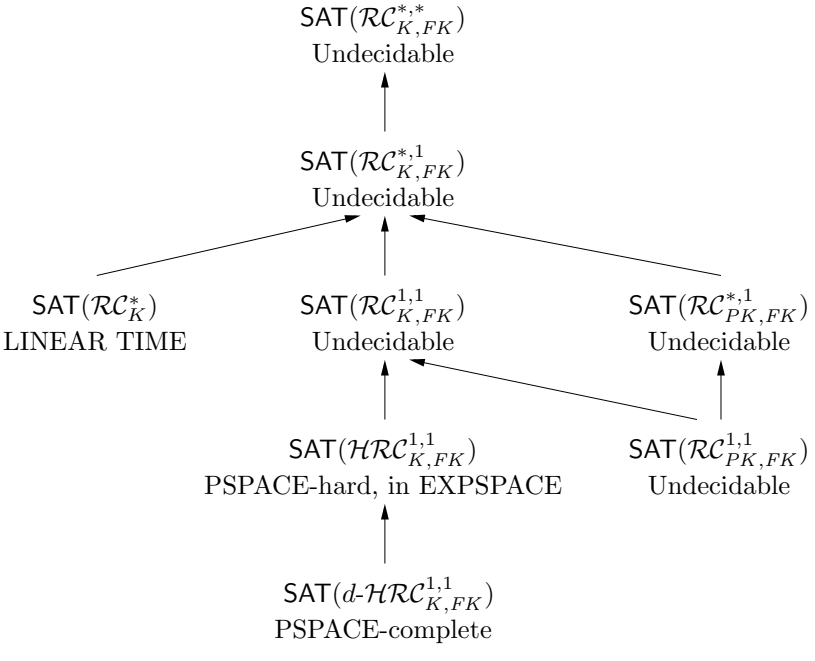


Fig. 5. A summary of the complexity bounds for the consistency problem for relative keys and foreign keys

For implication of relative constraints, note that $\mathcal{RC}^{1,1}_{PK,FK}$ and $\mathcal{HRC}^{1,1}_{K,FK}$ contain $\mathcal{AC}^{1,1}_{PK,FK}$. Thus from Proposition 1 and the lower-bounds for consistency analyses presented above. we derive:

Corollary 5. *For implication of relative constraints,*

- $\mathsf{Impl}(\mathcal{RC}^{1,1}_{PK,FK})$ *is undecidable, and so are* $\mathsf{Impl}(\mathcal{RC}^{*,1}_{K,FK})$, $\mathsf{Impl}(\mathcal{RC}^{*,1}_{PK,FK})$, $\mathsf{Impl}(\mathcal{RC}^{1,1}_{K,FK})$ *and* $\mathsf{Impl}(\mathcal{AC}^{*,1}_{PK,FK})$*;*
- $\mathsf{Impl}(\mathcal{HRC}^{1,1}_{K,FK})$ *is PSPACE-hard.* □

5.4 Summary

Figure 5 shows a summary of the complexity for the consistency problem for relative keys and foreign keys.

6 Consistency of Path-Expression Constraints

All the XML constraints that we have seen so far are defined for element types and in terms of attributes. As XML data is hierarchically structured, it is common to find path expressions in query languages for XML (e.g., XQuery [10], XSLT [19]). For the same reason, one is often interested in constraints specified with path expressions, either regular expressions [12, 13] or XPath [20] expressions [40]. In this section, we consider two classes of XML constraints defined with path expressions, namely, an extension of absolute constraints with regular expressions, and the class of constraints proposed by XML Schema [40] that is an extension of absolute constraints with XPath expressions.

6.1 Consistency of Regular Expression Constraints

To capture the hierarchical nature of XML data, we extend $\mathcal{AC}^{*,*}_{K,FK}$ to define absolute constraints on a collection of elements identified by a regular path expression.

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

$$\beta \quad ::= \quad \epsilon \;\mid\; \tau \;\mid\; \_ \;\mid\; \beta.\beta \;\mid\; \beta \cup \beta \;\mid\; \beta^*,$$

where ϵ denotes the empty word, τ is an element type in E, '_' stands for wildcard that matches any symbol in E and '.', '$\cup$' and '$*$' denote concatenation, union and Kleene closure, respectively. We assume that β is of the form $r.\beta'$ 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 the class $\mathcal{AC}^{reg}_{K,FK}$ of XML keys and foreign keys with regular expressions. Here we only consider unary constraints. An XML $\mathcal{AC}^{reg}_{K,FK}$ constraint φ over a DTD $D = (E, A, P, R, r)$ has one of the following forms:

- *Key*: $\beta.\tau.l \rightarrow \beta.\tau$, where $\tau \in E$, $l \in R(\tau)$ and β is a regular expression over D. An XML tree T satisfies this constraint, denoted by $T \models \beta.\tau.l \rightarrow \beta.\tau$, if

$$\forall\, x, y \in nodes(\beta.\tau)\ (x.l = y.l \rightarrow x = y).$$

- *Foreign key*: $\beta_1.\tau_1.l_1 \subseteq_{FK} \beta_2.\tau_2.l_2$, where $\tau_1, \tau_2 \in E$, $l_1 \in R(\tau_1)$, $l_2 \in R(\tau_2)$ and β_1, β_2 are regular expressions over D. An XML tree T satisfies this constraint, denoted by $T \models \beta_1.\tau_1.l_1 \subseteq_{FK} \beta_2.\tau_2.l_2$, if $T \models \beta_2.\tau_2.l_2 \rightarrow \beta_2.\tau_2$ and

$$\forall\, x \in nodes(\beta_1.\tau_1)\ \exists\, y \in nodes(\beta_2.\tau_2)\ (x.l_1 = y.l_2).$$

In other words, an $\mathcal{AC}^{reg}_{K,FK}$ constraint $\beta.\tau.l \rightarrow \beta.\tau$ defines a key for the set $nodes(\beta.\tau)$ of elements, i.e., all the elements reachable via the regular path expression $\beta.\tau$; similarly, an $\mathcal{AC}^{reg}_{K,FK}$ constraint of the form $\beta_1.\tau_1.l_1 \subseteq_{FK} \beta_2.\tau_2.l_2$ defines a foreign key for the set $nodes(\beta_1.\tau_1)$ of elements that references elements in the set $nodes(\beta_2.\tau_2)$.

Example 6. Consider the XML document depicted in Figure 6, 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* (account) has an attribute *num*. One may impose the following constraints over the DTD of that document:

$$r.\_^*.(student \cup prof).record.id \ \rightarrow\ r.\_^*.(student \cup prof).record,$$
$$r.\_^*.cs434.takenBy.sid \subseteq_{FK} r.\_^*.student.record.id,$$
$$r.\_^*.dbLab.acc.num \subseteq_{FK} r.\_^*.cs434.takenBy.sid.$$

The first constraint says that *id* is a key for all records of *student*s and *professor*s. 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. □

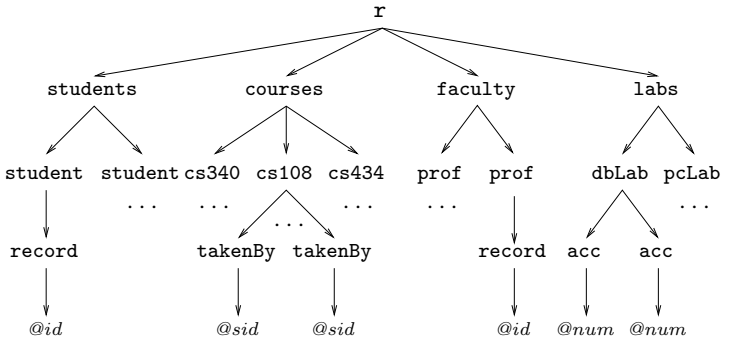


Fig. 6. An XML document

Both an upper and a lower bound for $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ were established in [4]. 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. In fact, even the current proof of the upper bound is quite involved, and relies on combining the techniques from [23] for coding DTDs and constraints as integer linear inequalities, and from [2] for reasoning about constraints given by regular expressions by using the product automaton for all the expressions involved in the constraints.

Theorem 11. $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ *is PSPACE-hard, and can be solved in NEXPTIME.* □

The PSPACE-hardness of $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ can be proved even for non-recursive DTDs without the Kleene star [4].

Observe that $\mathcal{AC}^{reg}_{K,FK}$ is a proper extension of the class $\mathcal{AC}^{1,1}_{K,FK}$ of unary constraints: substituting $r._^*.\tau$ for τ in $\mathcal{AC}^{1,1}_{K,FK}$ constraints yields equivalent $\mathcal{AC}^{reg}_{K,FK}$ constraints. Similarly, an extension of multi-attribute $\mathcal{AC}^{*,*}_{K,FK}$ constraints can be defined in terms of regular expressions, denoted by $\mathcal{AC}^{reg(*,*)}_{K,FK}$. The undecidability of the consistency problem for $\mathcal{AC}^{reg(*,*)}_{K,FK}$ is immediate from Theorem 1.

For the implication analysis of regular-expression constraints, from Proposition 1 it follows immediately:

Corollary 6. $\mathsf{Impl}(\mathcal{AC}^{reg}_{K,FK})$ *is PSPACE-hard, and* $\mathsf{Impl}(\mathcal{AC}^{reg(*,*)}_{K,FK})$ *is undecidable.*

Observe that there are practical $\mathcal{AC}^{reg}_{K,FK}$ constraints that are not expressible in $\mathcal{AC}^{1,1}_{K,FK}$, e.g., the foreign keys given in Example 6 are not definable in $\mathcal{AC}^{1,1}_{K,FK}$. In other words, $\mathcal{AC}^{reg}_{K,FK}$ is strictly more expressive than $\mathcal{AC}^{1,1}_{K,FK}$.

6.2 Consistency of XML Schema Specifications

All the results shown so far are for DTDs and keys and foreign keys. These days, the prime standard for specifying XML data is *XML Schema* [40]. 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 section 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.

Example 7. Recall that given any DTD D and any set Σ of keys in $\mathcal{AC}^*_K$ ($\mathcal{RC}^*_K$) over D, Σ can be satisfied by an XML tree valid w.r.t. D if and only if D has a valid XML tree. Thus, any XML specification (D, Σ) where D is non-recursive and Σ is a set of keys in $\mathcal{AC}^*_K$ ($\mathcal{RC}^*_K$) is consistent. We show here that a specification in XML Schema may not be consistent even for non-recursive DTDs in the absence of foreign keys.

Consider the following specification $S = (D, \Sigma)$ for biomedical data, where D is the following DTD:

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

and Σ contains only one key:

$$seq.clone._^*.DNA \rightarrow seq.clone.$$

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 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. We note that the syntax of XML Schema constraints (to be formally introduced later) is different from the syntax for XML constraints presented so far in that it allows a regular expression ($_^*.DNA$ in our example) to be the identifier of an element type.

This specification is inconsistent. XML Schema requires that for any XML document satisfying a key, the identifier (that is, $_^*.DNA$ in our example) must *exist* and be *unique*. However, as depicted in Fig. 7, in any XML document that conforms to the DTD D, a `clone` element must have two `DNA` descendants. Thus, it violates the uniqueness requirement of the key in Σ. □

The goal of this section is to show that the interaction of types with integrity constraints under the XML Schema semantics is more complicated than under the usual semantics for XML constraints. To focus on the nature of the interaction and to simplify the discussion, we first consider XML Schema specifications in which the type is a DTD and the constraints are absolute keys. We show that

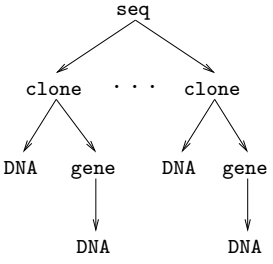


Fig. 7. An XML document conforming to the DTD D shown in Example 7

keys of XML Schema already suffice to demonstrate the complications caused by the interaction between types and constraints.

Before showing the main result of the section, we need to define the syntax and semantics of absolute keys for XML Schema specifications. Given a DTD $D = (E, A, P, R, r)$, a *key over* D is a constraint of the form

$$P[Q_1, \ldots, Q_n] \to P, \tag{7}$$

where $n \geq 1$ and $P, Q_1, \ldots, Q_n$ are regular expressions over the alphabet $E \cup A$. If $n = 1$, then the key is called unary and is denoted by $P.Q_1 \to P$. Expression P is called the *selector* of the key and is a regular expression conforming to the following BNF grammar [40] (abusing the XPath syntax):

$$\begin{array}{lll} \mathit{selector} & ::= & \mathit{path} \mid \mathit{path} \cup \mathit{selector} \\ \mathit{path} & ::= & r.\_^*.\mathit{sequence} \\ \mathit{sequence} & ::= & \tau \mid \_ \mid \mathit{sequence}.\mathit{sequence} \end{array}$$

Here $\tau \in E$ and $\_^*$ represents any possible finite sequence of node labels. The expressions $Q_1, \ldots, Q_n$ are called the *fields* of the key and are regular expressions conforming to the following BNF grammar [40]:

$$\begin{array}{lll} \mathit{field} & ::= & \mathit{path} \mid \mathit{path} \cup \mathit{field} \\ \mathit{path} & ::= & \_^*.\mathit{sequence}.\mathit{last} \mid \mathit{sequence}.\mathit{last} \\ \mathit{sequence} & ::= & \epsilon \mid \tau \mid \_ \mid \mathit{sequence}.\mathit{sequence} \\ \mathit{last} & ::= & \tau \mid \_ \mid @l \mid @\_ \end{array}$$

Here $@\_$ is a wildcard that matches any attribute and $@l \in A$. This grammar differs from the one above in allowing the final step to match an attribute node.

Definition 3. *Given an XML tree $T = (V, lab, ele, att, val, root)$, T satisfies the constraint $P[Q_1, \ldots, Q_n] \to P$, denoted by $T \models P[Q_1, \ldots, Q_n] \to P$, if*

1) *For each $x \in nodes(P)$ 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 $x_1, x_2 \in nodes(P)$, if y_i^1, y_i^2 are the only nodes such that $T \models Q_i(x_1, y_i^1)$ and $T \models Q_i(x_2, y_i^2)$ $(i = 1, \ldots, n)$, and $val(y_i^1) = val(y_i^2)$ for every $i \in [1, n]$, then $x_1 = x_2$.* □

That is, $P[Q_1, \ldots, Q_n] \rightarrow P$ defines a key for the set $nodes(P)$ of elements, i.e., the nodes reachable from the root by following path P, by asserting that the values of $Q_1, \ldots, Q_n$ uniquely identify the elements in $nodes(P)$. It further asserts that starting from each element in $nodes(P)$ there is a unique label path conforming to the regular expression Q_i ($i \in [1, n]$).

Observe that condition 1 in the previous definition requires the uniqueness and existence of the fields involved. For example, the XML tree depicted in Fig. 7 does not satisfy the key $seq.clone._^*.DNA \rightarrow seq.clone$ 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 [12, 13, 23, 4].

Since $\mathsf{SAT}(\mathcal{AC}^*_K)$ and $\mathsf{SAT}(\mathcal{RC}^*_K)$, the consistency problems for absolute and relative keys, respectively, are decidable in linear time, one would be tempted to think that the consistency problem for keys under the XML Schema semantics can be solved efficiently. Somewhat surprisingly, it was shown in [5] that this is not the case; the uniqueness and existence condition makes the problem intractable, even for unary keys and very simple DTDs:

Theorem 12. *The consistency problem is NP-hard for unary keys of the form (7), even for non-recursive no-star DTDs.* □

This result shows that the interaction of types and constraints under the XML Schema semantics is so intricate that the consistency check of XML Schema specifications is infeasible.

7 Selected Topics and Bibliographic Remarks

This chapter has shown that the consistency analysis of XML specifications with DTDs and constraints (keys, foreign keys) introduces new challenges and is in sharp contrast with its trivial counterpart for relational databases. Indeed, in the presence of foreign keys, compile-time verification of consistency for XML specifications is usually infeasible: the complexity ranges from NP-hard to undecidable. Worse still, the semantics of XML-Schema constraints makes the consistency analysis of specifications even more intricate.

These negative results suggest that one develops efficient approximate algorithms for static checking of XML specifications. One open question is to find performance guarantees for the approximate algorithms to prevent excessive overkill of consistent specifications. The techniques of [4, 5, 23] for establishing the complexity results of this chapter may help develop such performance guarantees; they may also help study consistency of individual XML specifications with types and constraints.

Another 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 $\mathsf{SAT}(\mathcal{HRC}^{1,1}_{K,FK})$, 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.

Bibliographic Notes. The complexity results of this chapter are taken from [4, 5, 23]: the results for the consistency analysis of absolute constraints were mostly established by [23]; relative constraints were studied in [4]; and a full treatment of XML-Schema specifications was given in [5].

Keys, foreign keys and the more general inclusion and functional dependencies have been well studied for relational databases (cf. [1]). The interaction between cardinality constraints and database schemas has been studied for object-oriented [16, 17] and extended relational data models [28]. These interactions are quite different from what we explore in this chapter because XML DTDs are defined in terms of extended context free grammars and they yield cardinality constraints more complex than those studied for traditional databases.

A number of specifications for XML keys and foreign keys have been proposed, e.g., XML Schema [40], XML-Data [31]. The notion of relative constraints was introduced by [12], which was further studied in [13]. It is worth remarking that although through the use of ID attributes in a DTD [11], one can uniquely identify an element within an XML document, it is not clear that ID attributes are intended to be used as keys rather than internal "pointers". For example, ID attributes are not scoped. In contrast to keys, they are unique within the entire document rather than among a designated set of elements. As a result, one cannot, for example, allow a student (element) and a person (element) to use the same SSN as an ID. Moreover using ID attributes as keys means that we are limiting ourselves to unary keys. Finally, one can specify at most one ID attribute for an element type, while in practice one may want more than one key.

Other constraints for semi-structured data were studied in, e.g., [2, 14]. In particular, [14] also studied the interaction between path constraints and traditional database schemas, which are quite different from XML constraints and DTDs considered here. Functional dependencies, an extension of XML keys, were recently proposed to define a normal form for XML documents [6].

Acknowledgments. M. Arenas and L. Libkin are supported in part by grants from NSERC, BUL, and PREA. W. Fan is supported in part by NSF Career Award IIS-0093168, NSFC 60228006 and EPSRC GR/S63205/01.

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. *J. Computer and System Sciences (JCSS)*, 58(4):428–452, 1999.

3. V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson and L. Wood. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, Oct. 1998. `http://www.w3.org/TR/REC-DOM-Level-1/`.
4. M. Arenas, W. Fan and L. Libkin. On verifying consistency of XML specifications. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 259–270, 2002.
5. M. Arenas, W. Fan and L. Libkin. What's Hard about XML Schema Constraints? In *Proc. Int'l Conf. on Database and Expert Systems Applications (DEXA)*, pages 269–278, 2002.
6. M. Arenas and L. Libkin. A Normal Form for XML Documents. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 85–96, 2002.
7. C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-based information mediation with MIX. In *Proc. of ACM SIGMOD Conf. on Management of Data (SIGMOD)*, pages 597–599, 1999.
8. C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *Proc. Int'l Conf. on Database Theory (ICDT)*, pages 296–313, 1999.
9. M. Benedikt, C. Chan, W. Fan, J. Freire, and R. Rastogi. Capturing both Types and Constraints in Data Integration. In *Proc. of ACM SIGMOD Conf. on Management of Data (SIGMOD)*, pages 277–288, 2003.
10. S. Boag, D. Chamberlin, M. Fernández, D. Florescu, J. Robie and J. Siméon. XQuery 1.0: An XML Query Language. W3C Working Draft, Nov. 2003. `http://www.w3.org/TR/xquery`.
11. T. Bray, J. Paoli and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, Feb. 1998. `http://www.w3.org/TR/REC-xml/`.
12. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.
13. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. *Information Systems*, 28(8):1037–1063, 2003.
14. P. Buneman, W. Fan, and S. Weinstein. Interaction between path and type constraints. *ACM Trans. on Computational Logic (TOCL)*, 4(4):530–577, 2003.
15. 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.
16. D. Calvanese and M. Lenzerini. Making object-oriented schemas more expressive. In *Proc. ACM Symp. on Principles of Database Systems (PODS)*, pages 243–254, 1994.
17. D. Calvanese and M. Lenzerini. On the interaction between ISA and cardinality constraints. In *Proc. IEEE Int'l Conf. on Data Engineering (ICDE)*, pages 204–213, 1994.
18. M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. Subramanian. XPERANTO: Publishing object-relational data as XML. In *Proc. Int'l Workshop on the Web and Databases (WebDB)*, 2000.
19. J. Clark. XSL Transformations (XSLT). W3C Recommendation, Nov. 1999. `http://www.w3.org/TR/xslt`.
20. J. Clark and S. DeRose. XML Path Language (XPath). W3C Recommendation, Nov. 1999. `http://www.w3.org/TR/xpath`.
21. 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.

22. A. Eyal and T. Milo. Integrating and customizing heterogeneous e-commerce applications. *VLDB Journal*, 10(1):16–38, 2001.
23. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3):368–406, 2002.
24. W. Fan and J. Siméon. Integrity constraints for XML. In *PODS'00*, pages 23–34.
25. 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.
26. D. Florescu and D. Kossmann. Storing and querying XML data using an RDMBS. *IEEE Data Eng. Bull.*, 22(3):27–34, 1999.
27. D. Florescu, L. Raschid and P. Valduriez. A methodology for query reformulation in CIS using semantic knowledge. *Int'l J. Cooperative Information Systems (IJCIS)*, 5(4):431–468, 1996.
28. P. C. Kanellakis. On the computational complexity of cardinality constraints in relational databases. *Information Processing Letters*, 11(2):98–101, Oct. 1980.
29. Y. Matiyasevich. *Hilbert's 10th Problem*. MIT Press, 1993.
30. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation* (2nd Edition). Addison Wesley, 2000.
31. A. Layman, E. Jung, E. Maler, H. Thompson, J. Paoli, J. Tigue, N. Mikula and S. De Rose. XML-Data. W3C Note, Jan. 1998. `http://www.w3.org/TR/1998/NOTE-XML-data`.
32. D. Lee and W. W. Chu. Constraint-preserving transformation from XML document type to relational schema. In *Proc. Int'l Conf. on Conceptual Modeling (ER)*, pages 323–338, 2000.
33. D. McAllester, R. Givan, C. Witty and D. Kozen. Tarskian set constraints. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 138–147, 1996.
34. J. Melton and A. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufman, 1993.
35. F. Neven. Extensions of attribute grammars for structured document queries. In *Proc. Int'l Workshop on Database Programming Languages (DBPL)*, pages 99–116, 1999.
36. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.
37. L. Popa. *Object/Relational Query Optimization with Chase and Backchase*. PhD thesis, University of Pennsylvania, 2000.
38. J. Shanmugasundaram et al. E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. In *Proc. of Int'l Conf. on Very Large Databases (VLDB)*, pages 65–76, 2000.
39. 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 *Proc. of Int'l Conf. on Very Large Databases (VLDB)*, pages 302–314, 1999.
40. H. Thompson, D. Beech, M. Malone and N. Mendelsohn. XML Schema. W3C Recommendation, May 2001 `http://www.w3.org/XML/Schema`.
41. J. D. Ullman. *Database and Knowledge Base Systems*. Computer Science Press, 1988.
42. S. Yu. Regular Languages. In G. Rosenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 41–110. Springer, 1996.