On the Complexity of Verifying Consistency of XML Specifications

Citation for published version:

Digital Object Identifier (DOI):
10.1137/050646895

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
SIAM Journal on Computing

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

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

MARCELO ARENAS†, WENFEI FAN‡, AND LEONID LIBKIN§

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 [16] 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 previously studied constraints for XML data, and investigate the complexity of the consistency problem. Our main conclusion is that in the presence of foreign key constraints, compile-time verification of consistency is infeasible. We look at absolute constraints that hold in the entire document, and relative constraints that only hold in a part of the document. For absolute constraints, we prove decidability and establish complexity bounds for primary multi-attribute keys and unary foreign keys, and study unary constraints that involve regular expressions. For relative constraints, we prove that even for unary constraints, the consistency problem is undecidable. We also show that results continue to hold for extended DTDs, a more expressive typing mechanism for XML.

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 [10, 12, 18, 19, 23, 30, 31], as well as integrating several XML documents [2, 3, 4, 15].

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., [35, 38, 36, 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 documents 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, [16] showed that for 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 [38]. 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. We now give examples of new kinds of constraints considered here, and explain their consistency problem.

Constraints with regular expressions. 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 Figure 1.1, which conforms to the following DTD for schools:

```xml
<!ELEMENT r (students, courses, faculty, labs)>  
<!ELEMENT students (student*)>  
<!ELEMENT courses (cs340, cs108, cs434)>  
```

∗An extended abstract was presented at the 21st ACM Symposium on Principles of Database Systems (PODS 2002).
†Pontificia Universidad Católica de Chile, Santiago, Chile (marenas@ing.puc.cl)
‡School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK (wenfei@inf.ed.ac.uk)
§School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK (libkin@inf.ed.ac.uk)
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:

\[ r. (\text{student} \cup \text{prof}).\text{record}.@id \rightarrow r. (\text{student} \cup \text{prof}).\text{record}, \]
\[ r. (\text{cs434}.\text{takenBy}.@sid \subseteq \text{FK} r.\text{student}.\text{record}.@id, \]
\[ r. \text{dbLab}.\text{acc}.@num \subseteq \text{FK} r.\text{cs434}.\text{takenBy}.@sid. \]

Fig. 1.1. An XML document.

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 students and professors. 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. Recall that a foreign key also imposes a key constraint on the target elements, e.g., the last foreign key above also says that @sid is a key for students taking cs434.

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:

\[ r. \text{faculty}.\text{prof}.\text{record}.@id \subseteq \text{FK} r.\text{dbLab}.\text{acc}.@num. \]

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 [16]. We use results from [1, 16, 27] 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 in [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 1.2.
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 1.2 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:

\[
\text{country} \cdot \text{name} \rightarrow \text{country}, \\
\text{country} \cdot \text{name} \rightarrow \text{province}, \\
\text{country} \cdot \text{capital} \cdot \text{name} \rightarrow \text{province}, \\
\text{country} \cdot \text{capital} \cdot \text{inProvince} \subseteq \text{FK province} \cdot \text{name}.
\]

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 all province descendants of the country element and @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. The foreign keys assure that for each capital element in a country element (sub-document), its @inProvince attribute refers to a province in the same country (recall that capital elements immediately below country also denote province capitals). Note that these constraints are somewhat related to the notion of keys for weak entities in relational databases (cf. [33]). 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 second 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. We note that one can make the specification consistent by replacing \( \text{country} \cdot \text{capital} \cdot \text{inProvince} \rightarrow \text{province} \) with two keys:

\[
\text{country} \cdot \text{capital} \cdot \text{inProvince} \rightarrow \text{province}.
\]
country(capital.@inProvince → capital) and country(province.capital.@inProvince → province.capital),
which allow capital.@inProvince and province.capital.@inProvince to share the same value.

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.

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 [16] 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.

The main conclusion of this paper is that while many proposals such as XML Schema [38] and XML
Data [36] 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 [8, 9, 22].

A number of specifications for XML keys and foreign keys have been proposed, e.g., XML Schema [38],
XML-Data [36]. A recent proposal [5] introduced relative constraints. To the best of our knowledge, consistency of XML constraints in the presence of schema specifications was only investigated in [16]. However,
[16] 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., [1, 6, 17]. 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. Application of constraints in data transformations was studied in [23, 12]; usefulness of keys
and foreign keys in query optimization has also been recognized [13, 14].

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. Section 5 provides lower and upper bounds for the consistency problem for
extended DTDs, a slight extension of DTDs which captures unranked tree automata, and several different
classes of keys and foreign keys. Section 6 summarizes the main results of the paper.

2. Notations.

2.1. DTDs, XML Trees and Paths. Assume that we have the following disjoint sets: El of element
names, Att of attribute names, S of possible values of attributes and raw text, and Vert of node identifiers.
All attribute names start with the symbol @, and these are the only ones starting with this symbol. We let S
be a reserved symbol not in any of those sets.

We formalize the notion of DTDs as follows (cf. [35, 7, 25, 16]).

Definition 2.1. A DTD (Document Type Definition) is defined to be D = (E, A, P, R, r), where:
• E ⊆ El is a finite set of element types;
• A ⊆ Att is a finite set of attributes;
• P is a mapping from E to element type definitions: Given τ ∈ E, P(τ) = S or P(τ) is a regular
  expression α defined as follows:

  α ::= ϵ | τ′ | α|α | α ∈ A

where S denotes the string type, τ′ ∈ El, ϵ is the empty word, and “|”, “,” and “*” denote union,
concatenation, and the Kleene closure, respectively;
• R is a mapping from E to the powerset of A. If @l ∈ R(τ), we say that @l is defined for τ.
The parent-child edge relation on documents w.r.t. a DTD, along the same lines as XQuery [39], XML Schema [38] and DOM [34].

We normally denote element types by \( \tau \), and assume that \( R(r) = \emptyset \) and \( r \) does not appear in \( P(\tau) \) for any \( \tau \in E \). We also assume that each \( \tau \) in \( E \setminus \{ r \} \) is connected to \( r \), i.e., either \( \tau \) appears in \( P(r) \), or it appears in \( P(\tau') \) for some \( \tau' \) that is connected to \( r \). In this paper we also use the following shorthands for regular expressions: \( \alpha^+ \) for \( (\alpha, \alpha^*) \) and \( \alpha? \) for \( (\epsilon, \alpha) \). Finally, notice that mixed content is not allowed in XML trees; for every \( \tau \in E \), \( P(\tau) \) is either \( S \) or a regular expression over \( E \).

**Example 2.2.** Let us consider the DTD \( D \) given in Section 1 for storing information about countries and their administrative subdivisions. In our formalism, \( D \) can be represented as \( (E, A, P, R, r) \), where \( E = \{ db, \text{country}, \text{province}, \text{capital}, \text{city} \} \), \( A = \{ \text{@name}, \text{@inProvince} \} \), \( r = db \) and \( P, R \) are as follows:

\[
\begin{align*}
P(db) &= \text{country}^+ \\
P(\text{country}) &= (\text{province}^+, \text{capital}^+) \\
P(\text{province}) &= (\text{capital}, \text{city}^*) \\
P(\text{capital}) &= S \\
P(\text{city}) &= S
\end{align*}
\]

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 [39], XML Schema [38] and DOM [34].

**Definition 2.3.** 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 \((V, \text{lab}, \text{ele}, \text{att}, \text{root})\), where

- \( V \subseteq \text{Vert} \) is a finite set of nodes;
- \( \text{lab} : V \to E ; \text{if}\ lab(v) = \tau (v \in V) \text{, } \tau \text{ is said to be the element type of } v \);
- \( \text{ele} : V \to S \cup V^* \text{, where } V^* \text{ is the set of all the finite sequences of values from } V \text{, such that for every } v \in V \text{, if } P(\text{lab}(v)) = S \text{, then } \text{ele}(v) = [s] \text{, where } s \in S \text{, otherwise } \text{ele}(v) = [v_1, \ldots, v_n] \text{, and the string } \text{lab}(v_1) \cdots \text{lab}(v_n) \text{ is in the regular language defined by } P(\text{lab}(v)) \text{.}
- \text{att} \text{ is a partial function from } V \times A \text{ to } S \text{ such that for any } v \in V \text{ and } \emptyset \setminus A \text{, } \text{att}(v, \emptyset) \text{ is defined iff } \emptyset \setminus A \text{ is defined} \)
- \( \text{root} \text{ is the root of } T : \text{root} \in V \text{ and } \text{lab}(\text{root}) = r \).

The parent-child edge relation on \( V \), \( \{ \{v_1, v_2\} \mid v_2 \text{ occurs in ele}(v_1) \} \), is required to form 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 \( \tau \) in \( E \), then function \( \text{ele} \) defines the children of \( x \) and function \( \text{att} \) defines the attributes of \( x \). The children of \( x \) are ordered and their labels observe the regular expression \( P(\tau) \). In contrast, their attributes are unordered and are identified by their labels (names).

**Example 2.4.** Figure 2.1 (a) shows an XML document storing information about provinces in Canada and conforming to the DTD shown in Example 2.2. Figure 2.1 (b) shows an XML tree \( T = (V, \text{lab}, \text{ele}, \text{att}, v_0) \) representing this document. In this tree, \( V = \{ v_i \mid i \in [0, 6] \} \) and \( \text{lab} \) is defined as:

- \( \text{lab}(v_0) = \text{db} \)
- \( \text{lab}(v_2) = \text{province} \)
- \( \text{lab}(v_4) = \text{capital} \)
- \( \text{lab}(v_5) = \text{capital} \)

Furthermore, function \( \text{ele} \) is defined as:
Finally, function \( att \) is defined as:

\[
\begin{align*}
att(v_1, \text{name}) &= \text{Canada} \\
att(v_2, \text{name}) &= \text{Alberta} \\
att(v_3, \text{name}) &= \text{Manitoba} \\
att(v_4, \text{inProvince}) &= \text{Ontario} \\
att(v_5, \text{inProvince}) &= \text{Alberta} \\
att(v_6, \text{inProvince}) &= \text{Manitoba}
\end{align*}
\]

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 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 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 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 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 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 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 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 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 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 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 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 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
\(\beta\) is a regular expression over \(D\).

For any XML tree \(T\) that conforms to \(D\), the tree \(T\) satisfies \(\varphi\), denoted by \(T \models \varphi\), if
\[
\forall x, y \in \text{nodes}(\beta.\tau) \ (x[X] = y[X] \rightarrow x = y).
\]

- A foreign key over \(D\) is an expression \(\varphi\) of the form \(\beta_1.\tau_1[X] \subseteq_{FK} \beta_2.\tau_2[Y]\), where
  - \(\tau_i \in E\) for \(i = 1, 2\);
  - \(\beta_i\) is a regular expression over \(D\), for \(i = 1, 2\); and
  - \(X, Y\) are nonempty lists of attributes in \(R(\tau_1)\), \(R(\tau_2)\) of the same length.

Here \(T \models \varphi\) if \(T \models \beta_2.\tau_2[Y] \rightarrow \beta_2.\tau_2\), and
\[
\forall x \in \text{nodes}(\beta_1.\tau_1) \exists y \in \text{nodes}(\beta_2.\tau_2) \ (x[X] = y[Y]).
\]

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.

The above constraints are generally referred to as multi-attribute regular expression constraints as they may be defined with multiple attributes. A regular expression key (foreign key) is said to be unary if it is defined in terms of a single attribute; that is, \(|X| = 1\) \((|Y| = 1)\) in the above definition. In that case, we write \(\beta.\tau.\@l \rightarrow \beta.\tau\) for regular expression unary keys, and \(\beta_1.\tau_1.\@l_1 \subseteq_{FK} \beta_2.\tau_2.\@l_2\) for regular expression unary foreign keys.

From [16], we immediately obtain that the consistency problem for regular expression constraints is undecidable. Thus, in this paper we only study the consistency problem for unary constraints defined with regular expressions. We denote this class of constraints by \(\mathcal{AC}_{K,FK}^{reg}\), where subscripts \(K\) and \(FK\) stand for keys and foreign keys, respectively. For example, the constraints over the school DTD that we have seen in Section 1 are instances of \(\mathcal{AC}_{K,FK}^{reg}\).

**Constraints associated with element types.** A class of absolute keys and foreign keys, denoted by \(\mathcal{AC}_{K,FK}^{*,*}\) (we shall explain the notation shortly), has been studied in [16]. It is a special case of regular-expression constraints and is defined for element types as follows. An \(\mathcal{AC}_{K,FK}^{*,*}\)-constraint \(\varphi\) over a DTD \(D = (E, A, P, R, \tau)\) 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 \(\varphi\), denoted by \(T \models \varphi\), if
  \[
  \forall x, y \in \text{ext}(\tau) \ (x[X] = y[X] \rightarrow x = y).
  \]

- **Foreign key.** \(\tau_1[X] \subseteq_{FK} \tau_2[Y]\), where \(\tau_1, \tau_2 \in E\) and \(X, Y\) are nonempty lists of attributes in \(R(\tau_1)\), \(R(\tau_2)\) of the same length. It is satisfied by a tree \(T\) if \(T \models \tau_2[Y] \rightarrow \tau_2\), and in addition
  \[
  \forall x \in \text{ext}(\tau_1) \exists y \in \text{ext}(\tau_2) \ (x[X] = y[Y]).
  \]

That is, \(\tau[X] \rightarrow \tau\) says that the \(X\)-attribute values of a \(\tau\)-element uniquely identify the element in \(\text{ext}(\tau)\). Furthermore, \(\tau_1[X] \subseteq_{FK} \tau_2[Y]\) says that the list of \(X\)-attribute values of every \(\tau_1\)-node in \(T\) must match the list of \(Y\)-attribute values of some \(\tau_2\)-node in \(T\) and the \(Y\)-attribute values of a \(\tau_2\)-element uniquely identify the element in \(\text{ext}(\tau_2)\).

Note that an \(\mathcal{AC}_{K,FK}^{*,*}\)-constraint can be readily expressed as a regular-expression constraint, by using \(\tau.\@r.\tau\) for \(\tau\).

As for the case of regular expression constraints, an \(\mathcal{AC}_{K,FK}^{*,*}\)-constraint is generally referred to as a multi-attribute constraint as it 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. 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,
Meaning

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mathcal{A}_{FK}^{reg})</td>
<td>regular expression unary keys and foreign keys</td>
</tr>
<tr>
<td>(\mathcal{A}_{FK}^{PK})</td>
<td>multi-attribute primary keys, unary foreign keys</td>
</tr>
<tr>
<td>(\mathcal{A}_{FK})</td>
<td>unary keys and foreign keys</td>
</tr>
<tr>
<td>(\mathcal{A}_{FK}^{PK})</td>
<td>primary unary keys and unary foreign keys</td>
</tr>
<tr>
<td>(\mathcal{A}_{FK}^{reg})</td>
<td>regular expression unary keys and foreign keys</td>
</tr>
</tbody>
</table>

Fig. 2.2. Notation summary.

we mean that both keys and foreign keys are unary. We shall be dealing with the following subclasses of \(\mathcal{A}_{FK}^{*,*}\): \(\mathcal{A}_{FK}^{*,1}\) denotes the class of multi-attribute keys and unary foreign keys; \(\mathcal{A}_{FK}^{reg}\) is the class of primary multi-attribute keys and unary foreign keys; \(\mathcal{A}_{FK}\) is the class of unary keys and unary foreign keys; and \(\mathcal{A}_{FK}^{reg}\) is the class of primary unary keys and unary foreign keys. We note that since a key is part of a foreign key, the restriction of \(\mathcal{A}_{FK}^{*,*}\) to unary keys and multi-attributes foreign keys (\(\mathcal{A}_{FK}^{1,*}\)) does not make sense.

For easy reference, in Figure 2.2 we summarize our notation for absolute constraints.

2.3. The Consistency 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 \(\text{SAT}(\mathcal{C})\) as follows:

**Problem**: \(\text{SAT}(\mathcal{C})\)

**Input**: A DTD \(D\), a finite set \(\Sigma\) of \(\mathcal{C}\)-constraints.

**Question**: Is there an XML tree \(T\) such that \(T \models D\) and \(T \models \Sigma\)?

It is known [16] that \(\text{SAT}(\mathcal{A}_{FK}^{*,*})\) is undecidable, but \(\text{SAT}(\mathcal{A}_{FK}^{1,*})\) and \(\text{SAT}(\mathcal{A}_{FK}^{reg})\) are NP-complete.

Nothing was known however about \(\text{SAT}(\mathcal{A}_{FK}^{*,1})\), where only keys are allowed to be multi-attribute, or about \(\text{SAT}(\mathcal{A}_{FK}^{reg})\), where regular expressions are used to define unary keys and foreign keys. These problems will be studied in Section 3.

In what follows, we write \(T \models (D, \Sigma)\) instead of \(T \models D\) and \(T \models \Sigma\).

3. Absolute Integrity Constraints. In this section, we establish the decidability and lower bounds for \(\text{SAT}(\mathcal{A}_{FK}^{*,1})\) and \(\text{SAT}(\mathcal{A}_{FK}^{reg})\), the consistency problems for absolute primary multi-attribute keys and unary foreign keys, and for regular-expression unary keys and unary foreign keys.

3.1. Consistency of Multi-attribute Keys. We know that \(\text{SAT}(\mathcal{A}_{FK}^{1,*})\), the consistency problem for unary absolute keys and foreign keys, is NP-complete [16]. In contrast, \(\text{SAT}(\mathcal{A}_{FK}^{*,*})\) is undecidable [16]. This leaves a large gap: namely, \(\text{SAT}(\mathcal{A}_{FK}^{*,1})\), where only keys are allowed to be multi-attribute.

The reason for the undecidability of \(\text{SAT}(\mathcal{A}_{FK}^{*,*})\) is that the implication problem for functional and inclusion dependencies can be reduced to it [16]. However, this implication problem is known to be decidable – in fact, in cubic time – for single-attribute inclusion dependencies [11], thus giving us hope to get decidability for multi-attribute keys and unary foreign keys.

The problem we resolve here is \(\text{SAT}(\mathcal{A}_{FK}^{*,1})\): the consistency problem for primary multi-attribute keys and unary foreign keys. Recall that a set \(\Sigma\) of \(\mathcal{A}_{FK}^{*,1}\)-constraints is said to be primary if for each element type \(\tau\), there is at most one key in \(\Sigma\) defined for \(\tau\)-elements (including key dependencies defined by foreign key constraints). Even dealing with this version of \(\text{SAT}(\mathcal{A}_{FK}^{*,1})\) one encounters considerable difficulties: with a rather involved proof, we manage to show that this problem is equivalent to a certain decidable version of Diophantine equations problem whose exact complexity has been an open problem for a while [21]:

**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 [27]. Thus, PDE can be thought of as integer linear programming extended with inequalities of the form \( x \leq y - z \) among variables. It is therefore \( \mathsf{NP} \)-hard, and [21] proved an \( \mathsf{NEXPTIME} \) upper bound for PDE. The exact complexity of the problem remains unknown.

Recall that two problems \( P_1 \) and \( P_2 \) are \textit{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.** \( \mathsf{SAT}(\mathcal{AC}^*_{PK, FK}) \) and PDE are polynomially equivalent.

**Proof.** The proof consists of two PTIME reductions, one for each direction.

- **a)** A reduction from \( \mathsf{SAT}(\mathcal{AC}^*_{PK, FK}) \) to PDE. We first define a class of simplified DTDs called \textit{narrow} DTDs, and we explain how to reduce the consistency problem for \( \mathcal{AC}^*_{PK, FK} \)-constraints over arbitrary DTDs to that over narrow DTDs. Then we show how to encode the consistency problem for narrow DTDs and \( \mathcal{AC}^*_{PK, FK} \)-constraints by a prequadratic Diophantine system.

- **b)** A reduction from PDE to \( \mathsf{SAT}(\mathcal{AC}^*_{PK, FK}) \). We now show the following. We define a new DTD

\[
\tau \rightarrow \tau_1, \tau_2 \quad \tau \rightarrow \tau_1 | \tau_2 \quad \tau \rightarrow \tau_1^* \quad \tau \rightarrow \tau' \quad \tau \rightarrow S \quad \tau \rightarrow \epsilon
\]

where \( \tau, \tau_1, \tau_2 \) are element types in \( E \cup N \), \( \tau' \in E \), \( S \) is the string type and \( \epsilon \) denotes the empty word. More specifically, we conduct the following “narrowing” process on the production rule \( \tau \rightarrow \alpha \):

- If \( \alpha = (\alpha_1, \alpha_2) \), then introduce two new element types \( \tau_1, \tau_2 \) and replace \( \tau \rightarrow \alpha \) with a new rule \( \tau \rightarrow \tau_1, \tau_2 \). We proceed to process \( \tau \rightarrow \alpha_1 \) and \( \tau \rightarrow \alpha_2 \) in the same way.
- If \( \alpha = (\alpha_1) \), then introduce a new element type \( \tau_1 \) and replace \( \tau \rightarrow \alpha \) with a new rule \( \tau \rightarrow \tau_1 | \tau_2 \). We proceed to process \( \tau \rightarrow \alpha_1 \) and \( \tau \rightarrow \alpha_2 \) in the same way.
- If \( \alpha = \alpha_1 \), then introduce a new element type \( \tau_1 \) and replace \( \tau \rightarrow \alpha \) with \( \tau \rightarrow \tau_1^* \). We proceed to process \( \tau \rightarrow \alpha_1 \) in the same way.
- If \( \alpha \) is one of \( \tau' \in E, S \) or \( \epsilon \), then the rule for \( \tau \) remains unchanged.

We refer to the set of new element types introduced when processing \( \tau \rightarrow P(\tau) \) as \( N_\tau \) and the set of production rules generated/revised as \( P_\tau \). Observe that \( N_\tau \cap E = \emptyset \) for any \( \tau \in E \). We define a new DTD \( D_N = (E_N, A, P_N, R_N, r) \), referred to as the \textit{narrowed DTD} of \( D \) (or just a narrow DTD if \( D \) is clear from the context), where

- \( E_N = E \cup \bigcup_{\tau \in E} N_\tau \), i.e., all element types of \( E \) and new element types introduced in the narrowing process;
- \( P_N = \bigcup_{\tau \in E} P_\tau \), i.e., production rules generated/revised in the narrowing process;
- \( R_N(\tau) = R(\tau) \) for each \( \tau \in E \), and \( R_N(\tau) = \emptyset \) for each \( \tau \in E_N \setminus E \).

Note that the root element type \( \tau \) and the set \( A \) of attributes remain unchanged. Moreover, elements of any type in \( E_N \setminus E \) do not have any attribute. The only kind of \( P_N \) production rules whose right-hand side contains element type of \( E \) are of the form \( \tau \rightarrow \tau' \), where \( \tau' \in E \). It is easy to see that \( D_N \) is computable in polynomial time.

Obviously, any set \( \Sigma \) of \( \mathcal{AC}^*_{PK, FK} \)-constraints over \( D \) is also a set of \( \mathcal{AC}^*_{PK, FK} \)-constraints over the narrow DTD \( D_N \) of \( D \). The next lemma establishes the connection between \( D \) and \( D_N \), which allows us to consider only narrow DTDs from now on.

**Lemma 3.2.** Let \( D \) be a DTD, \( D_N \) the narrowed DTD of \( D \) and \( \Sigma \) a set of \( \mathcal{AC}^*_{PK, FK} \)-constraints over \( D \). Then there exists an XML tree \( T_1 \) such that \( T_1 \models (D, \Sigma) \) iff there exists an XML tree \( T_2 \) such that \( T_2 \models (D_N, \Sigma) \).

**Proof.** Given an element type \( \tau \) and a sequence of attributes \( \alpha l_1, \ldots, \alpha l_n \in R(\tau) \), define \( \text{values}(\tau[\alpha l_1, \ldots, \alpha l_n]) \) as \( \{x.\alpha l_1, \ldots, x.\alpha l_n \mid x \in \text{ext}(\tau)\} \).

To prove the lemma, it suffices to show the following:
Claim: Given any XML tree $T_1 \models D$ one can construct an XML tree $T_2$ by modifying $T_1$ such that $T_2 \models D_N$, and vice versa. Furthermore, for every element type $\tau$ in $D$ and $@l_1, \ldots, @l_n \in R(\tau)$, $|\text{ext}(\tau)|$ in $T_1$, and $\text{values}(\tau[\@l_1, \ldots, \@l_n])$ in $T_2$ equals $\text{values}(\tau[\@l_1, \ldots, \@l_n])$ in $T_1$.

For the claim holds, we can show the lemma as follows. Assume that there exists an XML tree $T_1$ such that $T_1 \models D$ and $T_1 \models \Sigma$. By the claim, there is $T_2$ such that $T_2 \models D_N$. Suppose, by contradiction, there is $\varphi \in \Sigma$ such that $T_2 \not\models \varphi$. (1) If $\varphi$ is a key $\tau[\@l_1, \ldots, @l_n] \rightarrow \tau$, then there exist two distinct nodes $x, y \in \text{ext}(\tau)$ in $T_2$ such that $x.@l_i = y.@l_i$ for every $i \in [1, n]$. In other words, $|\text{values}(\tau[\@l_1, \ldots, \@l_n])| < |\text{ext}(\tau)|$ in $T_2$. Since $T_1 \not\models \varphi$, it must be the case that $|\text{values}(\tau[\@l_1, \ldots, \@l_n])| = |\text{ext}(\tau)|$ in $T_1$ because the tuple $(x.@l_1, \ldots, x.@l_n)$ of each $x \in \text{ext}(\tau)$ uniquely identifies $x$ among $\text{ext}(\tau)$. This contradicts the claim that $|\text{ext}(\tau)|$ in $T_2$ equals $|\text{ext}(\tau)|$ in $T_1$ and $|\text{values}(\tau[\@l_1, \ldots, \@l_n])|$ in $T_2$ equals $|\text{values}(\tau[\@l_1, \ldots, \@l_n])|$ in $T_1$. (2) If $\varphi$ is a unary foreign key: $\tau_1[\@l_1] \subseteq \tau_2[\@l_2]$, then either $T_2 \not\models \tau_2[\@l_2]$, and there is $x \in \text{ext}(\tau_1)$ in $T_2$ such that for all $y \in \text{ext}(\tau_2)$ in $T_2$, $x.@l_1 \neq y.@l_2$. In the first case, we reach a contradiction as in (1). In the second case, we have $x.@l_1 \neq \text{values}(\tau_2[\@l_2])$. By the claim, $x.@l_1 \neq \text{values}(\tau_1[\@l_1])$. Since $T_1 \not\models \varphi$, $x.@l_1 \neq \text{values}(\tau_2[\@l_2])$ in $T_1$. Again by the claim, we have $x.@l_1 \neq \text{values}(\tau_2[\@l_2])$ in $T_2$, which contradicts the assumption. The proof for the other direction is similar.

We next verify the claim. Given an XML tree $T_1 = (V_1, lab_1, ele_1, att, root)$ such that $T_1 \models D$, we construct an XML tree $T_2$ by modifying $T_1$ such that $T_2 \models D_N$. Consider a $\tau$-element $v$ in $T_1$. Let $ele_2(v) = \{v_1, \ldots, v_n\}$ and $w = lab_1(v_1) \ldots lab_1(v_n)$. Recall $\mathcal{N}_\tau$ and $\mathcal{P}_\tau$, the set of nonterminals and the set of production rules generated when narrowing $\tau \rightarrow P(\tau)$. Let $\mathcal{Q}_\tau$ be the set of E symbols that appear in $P_\tau$ plus $\Sigma$. We can view $G = (Q_\tau, \mathcal{N}_\tau \cup \{\tau\}, \mathcal{P}_\tau, \tau)$ as an extended context-free grammar, where $Q_\tau$ is the set of terminals, $\mathcal{N}_\tau \cup \{\tau\}$ the set of nonterminals, $\mathcal{P}_\tau$ the set of production rules and $\tau$ the start symbol.

Let $T_1 \models D$, we have $w \in P(\tau)$. By a straightforward induction on the structure of $P(\tau)$ it can be verified that $w$ is in the language defined by $G$. Thus, there is a parse tree $T(w)$ w.r.t. the grammar $G$ for $w$, and $w$ is the frontier (the list of leaves from left to right) of $T(w)$. Without loss of generality, assume that the root of $T(w)$ is $\tau$, and the leaves are $v_1, \ldots, v_n$. Observe that the internal nodes of $T(w)$ are labeled with element types in $\mathcal{N}_\tau$ except that the root $\tau$ is labeled $\tau$. Intuitively, we construct $T_2$ by replacing each element $v$ in $T_1$ by such a parse tree. More specifically, let $T_2 = (V_2, lab_2, ele_2, att, root)$. Here $V_2$ consists of nodes in $V_1$ and the internal nodes introduced in the parse trees. For each $x \in V_2$, let $lab_2(x) = lab(x)$ if $x \in V_1$, and otherwise let $lab_2(x)$ be the node label of $x$ in the parse tree where $x$ belongs. Note that nodes in $V_2 \setminus V_1$ are elements of some type in $E_N \setminus E$. For every $x \in V_1$, let $ele_2(x)$ be the list of its children in the parse tree having $x$ as root. For every $x \in V_2 \setminus V_1$, let $ele_2(x)$ be the list of its children in the parse tree of an element in $V_1$ that contains $x$. Note that $att$ and $root$ remain unchanged. By the construction of $T_2$ it can be verified that $T_2 \models D_N$; and moreover, for every element type $\tau$ in $D$ and $@l_1, \ldots, @l_n \in R(\tau)$, $|\text{ext}(\tau)|$ in $T_2$ equals $|\text{ext}(\tau)|$ in $T_1$ and $\text{values}(\tau[\@l_1, \ldots, @l_n])$ in $T_2$ equals $\text{values}(\tau[\@l_1, \ldots, @l_n])$ in $T_1$ because, among other things, (1) none of the new nodes, i.e., nodes in $V_2 \setminus V_1$, is labeled with an $E$-type; (2) no new attributes are defined; and (3) attribute function $att$ is unchanged.

Conversely, assume that there is $T_2 = (V_2, lab_2, ele_2, att, root)$ such that $T_2 \models D_N$. We construct an XML tree $T_1$ by modifying $T_2$ such that $T_1 \models D$. For every node $v \in V_2$ with $lab_2(v) = \tau$ and $\tau \in E_N \setminus E$, we substitute $v$ in $ele_2(v')$ by the children of $v$, where $v'$ is the parent of $v$. In addition, we remove $v$ from $V_2$, $lab_2(v)$ from $lab_2(v)$, and $ele_2(v)$ from $ele_2$. Observe that by the definition of $D_N$, no attributes are defined for elements of any type in $E_N \setminus E$. We repeat the process until there is no node labeled with element type in $E_N \setminus E$. Now let $T_1 = (V_1, lab_1, ele_1, att, root)$, where $V_1$, $lab_1$ and $ele_1$ are $V_2$, $lab_2$ and $ele_2$ at the end of the process, respectively. Notice that $att$ and $root$ remain unchanged. By the definition of $T_1$ it can be verified that $T_1 \models D$; and in addition, for every element type $\tau$ in $D$ and $@l_1, \ldots, @l_n \in R(\tau)$, $|\text{ext}(\tau)|$ in $T_2$ equals $|\text{ext}(\tau)|$ in $T_1$ and $\text{values}(\tau[\@l_1, \ldots, @l_n])$ in $T_2$ equals $\text{values}(\tau[\@l_1, \ldots, @l_n])$ in $T_1$ because, among other things, none of the nodes removed is labeled with a type of $E$ and the attribute function $att$ is unchanged.

By Lemma 3.2, in the rest of this proof we consider only narrow DTDs. Next we show how to encode $\mathcal{A}^{\tau}_{PK, FK}$-constraints by a prequadratic Diophantine system. Let $D = (E, A, P, R, \tau)$ be a narrow DTD and $\Sigma$ be a set of $\mathcal{A}_{PK, FK}^{\tau}$-constraints, i.e., primary $\mathcal{A}_{PK, FK}^{\tau}$-constraints. We encode $\Sigma$ with a set $C_\Sigma$ of
integer constraints, referred to as the cardinality constraints determined by \( \Sigma \). For every \( \varphi \in \Sigma \),

- if \( \varphi \) is a key constraint \( \tau[@l_1, \ldots, @l_k] \rightarrow \tau \), then \( C_{\Sigma} \) contains \( |ext(\tau)| \leq |values(\tau[@l_1])| \cdot \ldots \cdot |values(\tau[@l_k])| \);
- if \( \varphi \) is a unary foreign key \( \tau_1[@l_1] \subseteq_{FK} \tau_2[@l_2] \), then \( C_{\Sigma} \) contains \( |values(\tau_1[@l_1])| \leq |values(\tau_2[@l_2])| \) and \( |ext(\tau_2)| \leq |values(\tau_2[@l_2])| \);
- furthermore, for any \( \tau \in E \), if \( R(\tau) = \emptyset \), then \( 0 \leq |ext(\tau)| \) is in \( C_{\Sigma} \). Otherwise, for every \( @l \in R(\tau) \),
  \[ |values(\tau.@l)| \leq |ext(\tau)| \] and \( 0 \leq |values(\tau.@l)| \) are in \( C_{\Sigma} \).

Observe that for a unary key \( \tau.@l \rightarrow \tau \) we have both \( |values(\tau.@l)| \leq |ext(\tau)| \) and \( |ext(\tau)| \leq |values(\tau.@l)| \) in \( C_{\Sigma} \). Thus \( C_{\Sigma} \) assures \( |ext(\tau)| = |values(\tau.@l)| \).

We write \( T \models C_{\Sigma} \) if \( T \) satisfies all the constraints of \( C_{\Sigma} \), and we write \( T \models (D, C_{\Sigma}) \) if \( T \) conforms to a narrow \( DDT \) and satisfies \( C_{\Sigma} \). Note that \( C_{\Sigma} \) is equivalent (in fact, can be converted in polynomial time) to a quantified Diophantine system as the \( x \leq y \cdot z \) can be written as constraints of the form \( x \leq y \cdot z \) by introducing a fresh variable, e.g., \( x \leq x_1 \cdot x_2 \cdot x_3 \cdot x_4 \) is equivalent to \( x \leq x_1 \cdot z_1 \cdot z_1 \leq x_2 \cdot z_2 \) and \( z_2 \leq x_3 \cdot x_4 \) (in the sense that the former is satisfiable iff the latter is). Thus, without loss of generality, assume that \( C_{\Sigma} \) consists of linear and quantified integer constraints only. It should be noted that \( C_{\Sigma} \) can be computed in time polynomial in the size of \( \Sigma \) and \( D \). The lemma below shows that \( C_{\Sigma} \) characterizes the consistency of \( \Sigma \) if keys in \( \Sigma \) are primary.

**Lemma 3.3.** Let \( D \) be a narrow \( DDT \) and \( \Sigma \) a set of \( AC^*_{\text{FK},FK} \)-constraints over \( D \). Then every XML tree conforming to \( D \) and satisfying \( \Sigma \) also satisfies \( C_{\Sigma} \). In addition, if there exists an XML tree \( T_2 \) such that \( T_2 \models (D, C_{\Sigma}) \), then there exists an XML tree \( T_1 \) such that \( T_1 \models (D, \Sigma) \).

**Proof.** It is easy to see that for every XML tree \( T_1 \) that satisfies \( \Sigma \), it must be the case that \( T_1 \models C_{\Sigma} \).

Conversely, we show that if there exists an XML tree \( T_2 = (V, \text{lab}, \text{ele}, \text{att}, \text{root}) \) such that \( T_2 \models (D, C_{\Sigma}) \), then we can construct an XML tree \( T_1 = (V, \text{lab}, \text{ele}, \text{att}, \text{root}) \) such that \( T_1 \models (D, \Sigma) \). We construct \( T_1 \) from \( T_2 \) by modifying the function \( \text{att} \) while leaving \( \text{lab} \), \( \text{ele} \) and \( \text{root} \) unchanged. More specifically, let \( S = \{ \tau.@l | \tau \in E, @l \in R(\tau) \} \). To define the new function, denoted by \( \text{att}_1 \), we first associate a set of string values with each \( \tau.@l \) in \( S \). Let \( N \) be the maximum cardinality of \( \text{values}(\tau.@l) \) in \( T_2 \), i.e., \( N \geq \text{values}(\tau.@l) \) in \( T_2 \) for all \( \tau.@l \in S \). Let \( V_S = \{ a_i | i \in [1, N] \} \) be a set of distinct string values. For each \( \tau.@l \in S \), let \( V_{\tau.@l} = \{ a_i | i \in [1, |\text{values}(\tau.@l)|] \} \), and for each \( x \in \text{ext}(\tau), \text{att}_1(x.@l) \) be a string value in \( V_{\tau.@l} \) such that in \( T_1 \), \( |\text{values}(\tau.@l)| = V_{\tau.@l} \). The value \( \text{att}_1(x.@l) \) can be selected in such a way that for each key \( \varphi = \tau[@l_1, \ldots, @l_k] \rightarrow \tau \) in \( \Sigma \), \( x[@l_1, \ldots, @l_k] \) is a distinct list of string values from \( V_{\tau.@l_1} \times \ldots \times V_{\tau.@l_k} \). This is possible because by the definition of \( T_1 \), \( (\text{ext}(\tau))_1 \text{equals } |\text{ext}(\tau)| \) in \( T_2 \); (2) \( |\text{values}(\tau.@l)| \) in \( T_1 \) equals \( |\text{values}(\tau.@l)| \) in \( T_2 \); (3) \( T_2 \models C_{\Sigma} \) and \( |\text{ext}(\tau)| \leq |\text{values}(\tau.@l)| \) is in \( C_{\Sigma} \); and (4) since \( \varphi \) is the only key defined for \( \tau \)-elements, when we populate attributes \( @l_1, \ldots, @l_k \) of \( x \), we only need to select the value of \( \text{att}_1(x.@l_i) \) from \( V_{\tau.@l_i} \), such that \( x[@l_1, \ldots, @l_k] \) is distinct, without worrying about whether the population may hamper “other keys” defined on \( x \) (note that in the absence of the primary key assumption, the populations of different keys may interact with each other and as a result, the simply population strategy given above may no longer work; this is why we assume primary keys). It should be noted that it may be the case that \( V_{\tau.@l_1} \subseteq V_{\tau.@l_2} \), even if \( \Sigma \) does not imply \( \tau_1.@l_1 \subseteq_{\text{FK}} \tau_2.@l_2 \). This does not lose generality as we do not intend to capture negation of foreign keys. We next show that \( T_1 \) is indeed what we want.

It is easy to verify that \( T_1 \models D \) given the construction of \( T_1 \) from \( T_2 \) and the assumption that \( T_2 \models D \). To show that \( T_1 \models \Sigma \), we consider \( \varphi \in \Sigma \) in the following cases. (1) If \( \varphi \) is a key \( \tau[@l_1, \ldots, @l_k] \rightarrow \tau \), it is immediate from the definition of \( T_1 \) that \( T_1 \models \varphi \) since for any \( x \in \text{ext}(\tau), x[@l_1, \ldots, @l_k] \) is a distinct list of string values from \( V_{\tau.@l_1} \times \ldots \times V_{\tau.@l_k} \). (2) If \( \varphi = \tau_1.@l_1 \subseteq_{\text{FK}} \tau_2.@l_2 \), then \( T_2 \models |\text{values}(\tau_1.@l_1)| \leq |\text{values}(\tau_2.@l_2)| \) in \( T_2 \models C_{\Sigma} \). By the definition of \( att_1 \), for \( i = 1, 2 \), \( V_{\tau.@l_i} = \{ a_i | i \in [1, |\text{values}(\tau.@l_i)|] \} \) and in \( T_1 \), \( |\text{values}(\tau_1.@l)| \leq |\text{values}(\tau_2.@l_2)| \) in \( T_1 \). Furthermore, given that \( |\text{ext}(\tau_2)| \leq |\text{values}(\tau_2.@l_2)| \) and \( |\text{values}(\tau_2.@l_2)| \leq |\text{ext}(\tau_2)| \) are both in \( C_{\Sigma} \), \( T_2 \models C_{\Sigma} \), \( |\text{ext}(\tau_2)| \) in \( T_2 \) is equal to \( |\text{ext}(\tau_2)| \) in \( T_2 \) and \( |\text{values}(\tau_2.@l_2)| \) in \( T_2 \) is equal to \( |\text{values}(\tau_2.@l_2)| \) in \( T_2 \), we conclude that \( |\text{ext}(\tau_2)| \) is equal to \( |\text{values}(\tau_2.@l_2)| \) in \( T_1 \) and, hence, \( T_1 \models \tau_2.@l_2 \rightarrow \tau_2 \) since each \( x \in \text{ext}(\tau_2) \) in \( T_1 \) has a distinct \( @l_2 \)-attribute value and thus the value of its \( @l_2 \)-attribute uniquely identifies \( x \) among nodes in \( \text{ext}(\tau_2) \). Therefore, \( T_1 \models \varphi \) and, thus, \( T_1 \models (D, \Sigma) \). This concludes the proof of the lemma.

The above lemma takes care of coding the constraints; the next step is to code DTDs. For that, we use the technique developed in [16]: for each narrow DTD \( D \), one can compute in polynomial time in the size
of $D$ a set $\Psi_D$ of linear inequalities on nonnegative integers, referred to as the set of cardinality constraints determined by $D$, which includes $|e(t(\tau))|$ as a variable for each element type $\tau$ in $D$, but it does not have $|vales(\tau, \theta@l)|$ as a variable for any attribute $\theta@l$ of $\tau$. More specifically, for each symbol $\tau \in E \cup \{S\}$, $|e(t(\tau))|$ is treated as a distinct variable, which keeps track of the number of all $\tau$ elements in an XML tree $T$ conforming to $D$. In addition, for each occurrence of $\tau$ in the definition $P(\tau')$ of some element type $\tau'$, we also create distinct variables as follows: if $P(\tau') = \tau_1$ for $\tau_1 \in E \cup \{S\}$, then we create a distinct variable $x_{\tau_1,\tau'}$; if $P(\tau') = (\tau_1, \tau_2)$ or $P(\tau') = (\tau_1 | \tau_2)$, then we create two distinct variables $x_{\tau_1,\tau'}$ and $x_{\tau_2,\tau'}$. Intuitively, for $i \in [1, 2]$, $x_{\tau_i,\tau'}$ keeps track of the number of $\tau_i$ subelements at position $i$ under all $\tau'$ elements in $T$. Let $X_{\tau}$ be the set of all variables of the form $x_{\tau, \tau'}$. Using these variables, for each $\tau \in E$, we define a set $\psi_\tau$ of linear integer constraints that characterizes $P(\tau)$ quantitatively, as follows:

- If $P(\tau) = \tau_1$ for $\tau_1 \in E \cup \{S\}$, then $\psi_\tau$ includes $|e(t(\tau))| = x_{\tau_1,\tau}$. Referring to an XML tree $T$ that conforms to $D$, this assures that each $\tau$ element has a unique $\tau_1$ subelement.
- If $P(\tau') = (\tau_1, \tau_2)$, then $\psi_\tau$ includes $|e(t(\tau))| = x_{\tau_1,\tau'}$ and $|e(t(\tau))| = x_{\tau_2,\tau'}$. These assure that each $\tau$ element in $T$ must have a unique $\tau_1$ subelement and a unique $\tau_2$ subelement.
- If $P(\tau') = (\tau_1 | \tau_2)$, then $\psi_\tau$ includes $|e(t(\tau))| = x_{\tau_1,\tau'} + x_{\tau_2,\tau'}$. These assure that each $\tau$ element in $T$ must have either a $\tau_1$ subelement or a $\tau_2$ subelement, and thus the sum of the numbers of these $\tau_1$ and $\tau_2$ subelements equals the number of $\tau$ elements in $T$.

The set $\Psi_D$ of cardinality constraints determined by DTD $D$ consists of the following:

- $|e(t(\tau))| = 1$; i.e., there is a unique root in any XML tree valid w.r.t. $D$;
- constraints of $\psi_\tau$ for each $\tau \in E$; these assure that $P(\tau)$ is satisfied;
- $|e(t(\tau))| = \sum_{x_{\tau,\tau'} \in X_{\tau}} x_{\tau,\tau'}$ for each $\tau \in (E \setminus \{\tau\}) \cup \{S\}$; this indicates that the set $e(t(\tau))$ includes all $\tau$ elements no matter where they occur in an XML tree;
- $x \geq 0$ for any variable $x$ used above; i.e., the number of elements (subelements) is nonnegative.

It has been shown [16] that $\Psi_D$ has a nonnegative integer solution if and only if there exists an XML tree $T$ conforming to $D$ such that the cardinality of $e(t(\tau))$ in $T$ equals the value of the variable $|e(t(\tau))|$ in the solution for each element type $\tau$ in $D$.

We now combine this coding with the coding for $\mathcal{AC}_{PK,FK}^{m,1}$-constraints. Given a narrow DTD $D$ and a set $\Sigma$ of $\mathcal{AC}_{PK,FK}^{m,1}$-constraints over $D$, we define the set of cardinality constraints determined by $D$ and $\Sigma$ to be

$$
\Psi(D, \Sigma) = \Psi_D \cup \Sigma \cup \{(|e(t(\tau))| > 0) \rightarrow (|vales(\tau, \theta@l)| > 0) \mid \tau \in E, \theta@l \in R(\tau)\),
$$

where $C_\Sigma$ is the set of cardinality constraints determined by $\Sigma$, $\Psi_D$ is the set of cardinality constraints determined by $D$, and constraints $(|e(t(\tau))| > 0) \rightarrow (|vales(\tau, \theta@l)| > 0)$ are to ensure that every $\tau$-element has an $\theta@l$-attribute (note that $|vales(\tau, \theta@l)| \leq |e(t(\tau))|$ is already in $C_\Sigma$). Constraints in $\Psi(D, \Sigma)$ are either linear integer constraints, or inequalities of the form $x \leq y \cdot z$, which come from $C_\Sigma$, or constraints of the form $x > 0 \rightarrow y > 0$. Note that if we leave out constraints of the form $x > 0 \rightarrow y > 0$, $\Psi(D, \Sigma)$ is a prequadratic Diophantine system. Also note that $\Psi(D, \Sigma)$ can be computed in polynomial time in the size of $D$ and $\Sigma$.

We say that $\Psi(D, \Sigma)$ is consistent if and only if $\Psi(D, \Sigma)$ admits a nonnegative integer solution. That is, there is a nonnegative integer assignment to the variables in $\Psi(D, \Sigma)$ such that all the constraints in $\Psi(D, \Sigma)$ are satisfied.

**Lemma 3.4.** Let $D$ be a narrow DTD and $\Sigma$ a set of $\mathcal{AC}_{PK,FK}^{m,1}$-constraints over $D$. Then $\Psi(D, \Sigma)$ is consistent if and only if there is an XML tree $T$ such that $T \models (D, \Sigma)$.

**Proof.** Suppose that there exists an XML tree $T$ such that $T \models (D, \Sigma)$. Then there is a nonnegative integer solution to $\Psi_D$ such that for each element type $\tau$ in $D$, the value of the variable $|e(t(\tau))|$ equals the number of $\tau$-elements in $T$ [16]. By Lemma 3.3 and $T \models \Sigma$, we have $T \models C_{\Sigma}$. We extend the solution of $\Psi_D$ to be one to $\Psi(D, \Sigma)$ by letting the variable $|vales(\tau, \theta@l)|$ equal the number of distinct $\theta@l$-attribute values of all $\tau$-elements in $T$, for each element type $\tau$ and attribute $\theta@l$ of $\tau$ in $D$. Since $T \models C_{\Sigma}$, this extended assignment satisfies all the constraints in $C_\Sigma$. In addition, if $|e(t(\tau))| > 0$ then $|vales(\tau, \theta@l)| > 0$ since every $\tau$-element in $T$ has an $\theta@l$-attribute. Hence the assignment is indeed a nonnegative solution to $\Psi(D, \Sigma)$ and, therefore, $\Psi(D, \Sigma)$ is consistent.
Conversely, suppose that \( \Psi(D,\Sigma) \) admits a nonnegative integer solution. Then there exists an XML tree \( T \) such that \( T \models D \) and moreover, for each element type \( \tau \) in \( D \), the cardinality of \( \text{ext}(\tau) \) in \( T \) equals the value of the variable \( |\text{ext}(\tau)| \) in the solution [16]. We construct a new tree \( T' \) from \( T \) by modifying the definition of the function \( \text{att} \) such that in \( T' \), for each element type \( \tau \) and attribute \( @l \) of \( \tau \), the number of distinct \( @l \)-attribute values of all \( \tau \)-elements equals the value of the variable \( |\text{values}(\tau.@l)| \) in the solution. This is possible since \( |\text{values}(\tau.@l)| \leq |\text{ext}(\tau)| \) and \( (|\text{ext}(\tau)| > 0) \rightarrow (|\text{values}(\tau.@l)| > 0) \) are in \( \Psi(D,\Sigma) \). The assignment is also a solution to \( C_{\Sigma} \). Thus \( T' \models D \) and \( T' \models C_{\Sigma} \). Hence by Lemma 3.3, there exists an XML tree \( T'' \) such that \( T'' \models (D,\Sigma) \). This concludes the proof of the lemma.

We now conclude the proof of reduction from \( \text{SAT}(\mathcal{AC}^*_{PK,FK}) \) to \( \text{PDE} \). By Lemma 3.2, given an arbitrary DTD \( D \) and a set \( \Sigma \) of \( \mathcal{AC}^*_{PK,FK} \)-constraints, one can compute a narrow DTD \( D_N \) such that \( (D,\Sigma) \) is consistent iff \( (D_N,\Sigma) \) is consistent by Lemma 3.4. \( (D_N,\Sigma) \) is consistent iff \( \Psi(D_N,\Sigma) \) has a nonnegative integer solution. Thus \( (D,\Sigma) \) has a nonnegative integer solution iff \( \Psi(D_N,\Sigma) \) has a nonnegative integer solution. Hence it takes polynomial time to compute \( \Phi(D_N,\Sigma) \) from \( D \) and \( \Sigma \). Therefore, there is a \( \text{PTIME} \) reduction from \( \text{SAT}(\mathcal{AC}^*_{PK,FK}) \) to \( \text{PDE} \).

\[ b) \text{ A reduction from } \text{PDE} \text{ to } \text{SAT}(\mathcal{AC}^*_{PK,FK}). \text{ We now move to the other direction. Given an instance of } \text{PDE}, \text{ i.e., a system } S \text{ consisting of a set } S_L \text{ of linear equations/inequalities on integers and a set } S_P \text{ of prequadratic constraints of the form } x \leq y \cdot z, \text{ we define a DTD } D \text{ and a set } \Sigma \text{ of } \mathcal{AC}^*_{PK,FK}-\text{constraints such that } S \text{ has a nonnegative integer solution if there is an XML tree } T \text{ satisfying } S \text{ and conforming to } D. \text{ We use } X = \{ x_i \mid i \in [1,n] \} \text{ to denote the set of all the variables in } S. \text{ Assume that } S_L = \{ e_j \mid j \in [1,m] \} \text{ and } e_j \text{ is of the form: } a^1_j x_1 + \ldots + a^n_j x_n + c_j \leq b^1_j x_1 + \ldots + b^n_j x_n + d_j, \text{ where } a^i_j (i \in [1,n]), b^i_j (i \in [1,n]), c_j \text{ and } d_j \text{ are nonnegative integers}. \text{ Also, assume that } S_P = \{ p_j \mid j \in [1,l] \}, \text{ where } p_j \text{ is a prequadratic equation of the form } x \leq y \cdot z. \]
3 We define production rules as follows. For the root of the DTD:

\[ P(r) = (X_1, U_1^{s_1, \ldots, s_i, \ldots, s_{i+1}}, \ldots, X_n, U_n^{s_{i+1}, \ldots, s_i})^*, \ldots, (X_n, U_n^{s_1, \ldots, s_i}, \ldots, U_n^{s_{i+1}, \ldots, s_i})^*, \]

where \( s_i \) is the set of indexes \( \{s_i \in [1, n] \} \) is the set of indexes \( \{s \mid p_s = x_i \leq x_j, x_k \in S_P \} \). Furthermore, for every \( i \in [1, n] \) and every \( j \in [1, m] \):

\[ \begin{align*}
    P(A_i^j) &= E_j, \\
    P(C_j) &= E_j, \\
    P(B_j^i) &= F_j, \\
    P(D_j^i) &= F_j, \\
    P(X_j) &= A_1^i, A_1^i, \ldots, A_1^i, A_m^j, A_m^j, \ldots, A_m^j, B_1^i, B_1^i, \ldots, B_1^i, B_m^j, B_m^j.
\end{align*} \]

Finally, for every \( i \in [1, n] \) and every \( s \in [1, l] \) such that \( p_s = x_i \leq x_j, x_k \in S_P \), \( P(U^s_i) = \epsilon \).

4 We define the attribute function \( R \) as follows: for every \( j \in [1, m] \), \( R(E_j) = R(F_j) = \{\@e\} \). In addition, for every \( i \in [1, n] \), \( R(X_i) = \{\@e\} \), and for every \( s \in [1, l] \) such that \( p_s = x_i \leq x_j, x_k \in S_P \), \( R(U^s_i) = \{\@e, \@d\} \). For all other element type \( \tau \), let \( R(\tau) \) be empty.

For example, Figure 3.1 (a) shows an XML tree conforming to the DTD constructed from the set of equations \( S_1 = \{2x_1 \leq x_2 + 4\} \) and \( S_P = \{x_1 \leq x_2 \cdot x_3\} \). We note that this tree codes solution \( x_1 = 1 \), \( x_2 = 2 \), \( x_3 = 1 \) for this system of equations.

Given DTD \( D \), we define a set \( \Sigma \) of \( \mathcal{AC}_{FK}^+ \)-constraints over \( D \). For each \( j \in [1, m] \), \( \Sigma \) includes keys \( E_j, \@e \Rightarrow E_j, F_j, \@e \Rightarrow F_j \) and foreign key \( E_j, \@e \subseteq_{FK} F_j, \@e \). Furthermore, for every \( i, j, k \in [1, n] \) and \( s \in [1, l] \) such that \( p_s = x_i \leq x_j, x_k \in S_P \), \( \Sigma \) includes the following constraints:

\[ U^s_i[\@e, \@d] \Rightarrow U^s_i, \quad U^s_i[\@e] \subseteq_{FK} X_j[\@e], \quad U^s_i[\@d] \subseteq_{FK} X_k[\@e]. \]

Clearly, the set \( \Sigma \) is primary, i.e., for any element type \( \tau \) there is at most one key defined. In fact, we use copies \( U^s_i \) of \( X \) just to ensure that \( \Sigma \) is primary.

We next show that the encoding is indeed a reduction from PDE to \( \text{SAT}(\mathcal{AC}_{FK}^+, \mathcal{FK}) \). Suppose that \( S \) has a nonnegative solution. Then we construct an XML tree \( T \) conforming to \( D \) as shown in Figure 3.1 (a). That is, for each \( i \in [1, n] \) we let \( \text{ext}(X_i) \) be the value of the variable \( x_i \) in the solution. We note that, by the definition of \( D \), this implies that for every \( s \in [1, l] \) such that \( p_s = x_i \leq x_j, x_k \in S_P \), \( \text{ext}(U^s_i) \) is also equal to the value of \( x_i \) in the solution. For every \( i \in [1, n] \) and every \( X_i \)-element \( x \) in \( T \), we let \( x, \@e \) be a distinct value such that in \( T \), \( \text{values}(X_i, \@e) \) is the value of \( x_i \). For every \( j \in [1, m] \) and every \( F_j \)-element \( x \) in \( T \), we let \( x, \@e \) be a distinct value such that in \( T \), \( \text{values}(F_j, \@e) \) is the value of \( x_i \). Likewise, we assign values to the \( \@e \)-attribute of the nodes in \( \text{ext}(F_j) \) in such a way that \( \text{values}(F_j, \@e) \) is the value of \( x_i \) in \( T \). Finally, for every \( i, j, k \in [1, n] \) and \( s \in [1, l] \) such that \( p_s = x_i \leq x_j, x_k \in S_P \), and for every node \( x \) in \( T \) of type \( U^s_i \), we
let $x[\@c, \@d]$ be a distinct list of string values from $\text{values}(X_j, \@e) \times \text{values}(X_k, \@e)$. This is possible since $x_i \leq x_j \cdot x_k \in S_P$ and by the definition of $T$, $|\text{ext}(U_i')| = |\text{ext}(X_i)| = x_i$, $|\text{values}(X_j, \@e)| = |\text{ext}(X_j)| = x_j$ and $|\text{values}(X_k, \@e)| = |\text{ext}(X_k)| = x_k$. Since $T$ codes a solution of $S$, it is straightforward to prove that $T \models C_{\Sigma}$, the set of cardinality constraints determined by $\Sigma$. Thus, by Lemma 3.3 we conclude that there exists an XML tree $T'$ such that $T' \models (D, \Sigma)$ and, hence, $(D, \Sigma)$ is consistent. Conversely, suppose that there exists an XML tree $T$ such that $T \models (D, \Sigma)$. We construct a solution of $S$ by letting variable $x_i$ equal $|\text{ext}(X_i)|$ in $T$. By the definitions of $D$ and $\Sigma$, it is easy to verify that this is indeed a nonnegative integer solution for $S$. In particular, each $p_i = x_i \leq x_j \cdot x_k$ in $S_P$ holds because $T \models (D, \Sigma)$ and, thus, $|\text{ext}(X_i)| = |\text{ext}(U_i^*)| \leq |\text{values}(U_i^*, \@e)| \cdot |\text{values}(U_i^*, \@d)| \leq |\text{values}(X_j, \@e)| \cdot |\text{values}(X_k, \@e)| \leq |\text{ext}(X_j)| \cdot |\text{ext}(X_k)|$.

We observe that the previous reduction is not polynomial since constants $a_k^i$, $b_k^i$ ($i \in [1, n]$, $j \in [1, m]$) and $c_j, d_j$ ($j \in [1, m]$) are coded in unary. To overcome this problem, next we show how to code in a DTD the binary representation of a number. We introduce this coding separately to simplify the presentation of this proof.

Assume that $a = \sum_{i=0}^{k} a_i \cdot 2^i$, where each $a_i$ ($i \in [0, k-1]$) is either 0 or 1 and $a_k = 1$, that is, the binary representation of $a$ is $a_k a_{k-1} \cdots a_1 a_0$. To code $a$ in a DTD we include element types $A, Y_0, \ldots, Y_k$ and we define $P$ on these elements as follows:

$$P(Y_i) = \begin{cases} \epsilon & \text{if } i = 0 \\ Y_{i-1} & \text{if } Y_i \neq \epsilon \end{cases}$$

and $P(A) = Y_1, \ldots, Y_k$, where $i_1 > \cdots > i_0 \geq 0$ and $\{i_1, \ldots, i_0\}$ is the set of indexes $\{i \in [0, k] | a_i = 1\}$.

We note that the size of this set of rules is polynomial in the size of $a$. Furthermore, if an XML tree $T$ conforms to this DTD, then $|\text{ext}(Y_0)| = a$ in $T$. For example, if $a = 5$, then $P(A) = Y_2, Y_0$, $P(Y_2) = Y_1, Y_1$, $P(Y_1) = Y_0, Y_0$ and $P(Y_0) = \epsilon$ and an XML tree conforming to these rules is of the form shown in Figure 3.1 (b).

Thus, by using this coding in our original reduction of PDE to $\text{SAT}(\mathcal{A}^{\tau_1}_{PK, FK})$ we can show that there is a PTIME reduction from PDE to $\text{SAT}(\mathcal{A}^{\tau_1}_{PK, FK})$. This completes the proof of Theorem 3.1. It is known that the linear integer programming problem is NP-hard and PDE is in NEXPTIME. Thus from Theorem 3.1 it follows immediately:

**Corollary 3.5.** $\text{SAT}(\mathcal{A}^{\tau_1}_{PK, FK})$ is NP-hard, and can be solved in NEXPTIME. \hfill $\Box$

Obviously we cannot obtain the exact complexity of $\text{SAT}(\mathcal{A}^{\tau_1}_{PK, FK})$ without resolving the corresponding question for PDE, which appears to be quite hard [21]. The result of Theorem 3.1 can be generalized to disjoint $\mathcal{A}^{\tau_1}_{FK}$-constraints: that is, a set $\Sigma$ of $\mathcal{A}^{\tau_1}_{FK}$-constraints in which for every element type $\tau$ and every two distinct keys $\tau[X] \to \tau$ and $\tau[Y] \to \tau$ in $\Sigma$ (including key dependencies defined by foreign key constraints), $X \cap Y = \emptyset$. The proof of Theorem 3.1 applies almost verbatim to show the following.

**Corollary 3.6.** The restriction of $\text{SAT}(\mathcal{A}^{\tau_1}_{FK})$ to disjoint constraints is polynomially equivalent to PDE and, thus, it is NP-hard and can be solved in NEXPTIME. \hfill $\Box$

### 3.2. Consistency of Regular Expression Constraints

Specifications of $\mathcal{A}^{\tau_1}_{FK}$-constraints are associated with element types. We next consider $\mathcal{A}^{\tau_{reg}}_{FK}$, the class of unary keys and foreign keys defined in terms of regular path expressions. For $\text{SAT}(\mathcal{A}^{\tau_{reg}}_{FK})$, 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. In fact, even the current proof of the upper bound is quite involved, and relies on combining the techniques from [16] for coding DTDs and constraints with integer linear inequalities, and from [1] for reasoning about constraints given by regular expressions by using the product automaton for all the expressions involved in the constraints.

**Theorem 3.7.**

a) $\text{SAT}(\mathcal{A}^{\tau_{reg}}_{FK})$ can be solved in 2-NEXPTIME.

b) $\text{SAT}(\mathcal{A}^{\tau_{reg}}_{FK})$ is PSPACE-hard, even for non-recursive no-star DTDs. \hfill $\Box$

**Proof.** We reduce $\text{SAT}(\mathcal{A}^{\tau_{reg}}_{FK})$ to the existence of solution of an (almost) instance of linear integer programming, which happens to be of double-exponential size; hence the 2-NEXPTIME bound. For the lower bound, we encode the quantified boolean formula problem (QBF) as an instance of $\text{SAT}(\mathcal{A}^{\tau_{reg}}_{FK})$. 

15
Proof of a) The proof is a bit long, so we first give a rough outline. The idea is similar to the proof of the NP membership for \( SAT(\mathcal{AC}_{FK}\beta) \) [16]: we code both the DTD and the constraints with linear inequalities over integers. However, compared to the proof of [16], 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 expressions in integrity constraints (those correspond to "vertical" paths through the trees). To eliminate this interaction, we first show how to reduce the problem to that over narrow DTDs, in which no wide horizontal regular expressions are allowed. The next problem is that regular expressions in constraints can interact with each other. Thus, to model them with linear inequalities, we extend the approach of [16] by taking into account all possible Boolean combinations of regular languages given by expressions used in constraints. 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 expressions used in constraints. For that, we adopt the technique of [1], and tag all the variables in the coding of DTDs with states of a certain automaton (the product automaton for all the automata corresponding to the regular expressions used in constraints).

Now it is time to fill in all the details. First, we need some additional notation. For every regular expression \( \beta \) and every attribute \( @l \), we write \( \text{values}(\beta, @l) \) to denote the set \( \{y, @l \mid y \in \text{nodes}(\beta) \text{ and } y @l \text{ is defined}\} \). Observe that for any \( \tau \in E \setminus \{\tau\} \), and \( @l \in R(\tau) \), \( \text{values}(\tau \ast \tau, @l) \) corresponds to our original definition of \( \text{values}(\tau, @l) \) if we say that a DTD \( D \) is one-attribut"}
node in $T_1$ such that $u \in \text{nodes}(\beta, \tau)$ and $\text{att}_1(u, \tau_l) = \text{att}_2(v, \tau_e)$. Thus, given that $\text{values}(\beta', \tau', \tau'_{\text{opt}}, \tau_e)$ in $T_2$ is equal to $\text{values}(\beta', \tau', \tau'_{\text{opt}})$ in $T_1$, we conclude that $\text{att}_1(u, \tau_l) \not\in \text{values}(\beta', \tau', \tau'_{\text{opt}})$ in $T_1$. Therefore, $T_1 \not\models \beta, \tau.l \subseteq_{FK} \beta', \tau', \tau'_{\text{opt}}$, which contradicts the assumption that $T_1 \models \Sigma$.

$(\Leftarrow)$ Let $T_2 = (V_2, \text{lab}_2, \text{ele}_2, \text{att}_2, \text{root})$ be an XML tree such that $T_2 \models (D_U, \Sigma_U)$. We define an XML tree $T_1$ from $T_2$ such that $T_1 \models (D, \Sigma)$. More specifically, $T_1 = (V_1, \text{lab}_1, \text{ele}_1, \text{att}_1, \text{root})$, where $V_1$, $\text{lab}_1$, $\text{ele}_1$ and $\text{att}_1$ are defined as follows. Let $v$ be a node in $T_2$ such that $\text{lab}_2(v) = \tau$, $\tau \in E$ and $R(\tau) = \{\tau_1, \ldots, \tau_k\}$. Then $V_1$ also contains node $v$ with $\text{lab}_1(v) = \tau$. Furthermore, if $P(\tau) = \mathbf{S}$, then $\text{ele}_2(v) = [v_{\text{opt}}, \ldots, v_{\text{opt}}]$, where $\text{lab}(v_{\text{opt}}) = \tau_{\text{opt}}$, ($j \in [1,k]$), and we define $\text{ele}_1(v)$ as $[\mathbf{S}]$, where $s$ is an arbitrary string in $s$, and we define $\text{att}_1(v, \tau_i)$ as $\text{att}_2(v_{\text{opt}}, \tau_e)$, for every $i \in [1,k]$. Otherwise, $P(\tau)$ is a regular expression over $E$ and $\text{ele}_2(v) = [v_1, \ldots, v_n, v_{\text{opt}}, \ldots, v_{\text{opt}}]$, where $\text{lab}(v_i) \in E$ ($i \in [1,n]$) and $\text{lab}(v_{\text{opt}}) = \tau_{\text{opt}}$, ($j \in [1,k]$), and we define $\text{ele}_1(v)$ as $[v_1, \ldots, v_n]$ and $\text{att}_1(v, \tau_i)$ as $\text{att}_2(v_{\text{opt}}, \tau_e)$, for every $i \in [1,k]$. Next we show that $T_1 \models (D, \Sigma)$.

By the definition of $D_U$ and given that $T_2 \models D_U$, it is easy to see that $T_1 \models D$. Assume that $T_1 \not\models \Sigma$. Then there exists $\varphi \in \Sigma$ such that $T_1 \not\models \varphi$. (1) If $\varphi$ is a key $\beta, \tau.l \rightarrow \beta, \tau$, then there exists distinct $u_1, u_2 \in \text{nodes}(\beta, \tau)$ in $T_1$ such that $\text{att}_1(u_1, \tau_l) = \text{att}_1(u_2, \tau_l)$. By the definition of $T_1$, $u_1$ and $u_2$ are also in $\text{nodes}(\beta, \tau)$ in $T_2$. Let $v_1, v_2$ be the children of $u_1, u_2$ in $T_2$ of type $\tau_{\text{opt}}$, respectively. Given that $u_1 \not= u_2$, we have that $v_1 \not= v_2$. Thus, by the definition of $T_1$, $v_1$ and $v_2$ are nodes in $T_2$ such that $v_1, v_2 \in \text{nodes}(\beta, \tau_{\text{opt}})$ and $\text{att}_2(v_1, \tau_e) = \text{att}_2(v_2, \tau_e) \neq \text{att}_2(u_1, \tau_l)$. Therefore, $T_2 \not\models \beta, \tau_{\text{opt}}, \tau_e \rightarrow \beta, \tau_{\text{opt}}$, which contradicts the assumption that $T_2 \models \Sigma_U$. (2) If $\varphi$ is a foreign key $\beta, \tau.l \subseteq_{FK} \beta', \tau', \tau'_{\text{opt}}$, then either $T_1 \not\models \beta, \tau.l \rightarrow \beta, \tau'$ or there exists $u \in \text{nodes}(\beta, \tau)$ such that $\text{att}_1(u, \tau_l) \not\in \text{values}(\beta', \tau', \tau'_{\text{opt}})$ in $T_1$. In the former case, we reach a contradiction as in (1). In the latter case, assume that $v$ is the child of $u$ in $T_2$ of type $\tau_{\text{opt}}$ (a node of $T_2$ by the definition of $\Sigma_U$). By the definition of $T_1$, we have that $v \in \text{nodes}(\beta, \tau_{\text{opt}})$ and $\text{att}_2(v, \tau_e) = \text{att}_1(u, \tau_l)$. Thus, given that $\text{values}(\beta', \tau', \tau'_{\text{opt}}, \tau_e)$ in $T_2$ is equal to $\text{values}(\beta', \tau', \tau'_{\text{opt}})$ in $T_1$, we conclude that $\text{att}_2(v, \tau_e) \not\in \text{values}(\beta', \tau', \tau'_{\text{opt}}, \tau_e)$ in $T_2$. Therefore, $T_2 \not\models \beta, \tau_{\text{opt}}, \tau_e \subseteq_{FK} \beta', \tau', \tau'_{\text{opt}}, \tau_e$, which contradicts the assumption that $T_2 \models \Sigma$. This proves the conclusion of the lemma.

By Lemma 3.9, from now on we consider only one-attribute DTDDs. Let $D = (E, \{\tau_l\}, P, R, r)$ be a one-attribute DTD and $D_N = (E_N, \{\tau_l\}, P_N, R_N, r)$ be the narrow DTD of $D$ (defined in the proof of Theorem 3.1). Observe that $D_N$ is also a one-attribute. Furthermore, observe that an XML tree $T$ valid w.r.t. $D$ may not conform to $D_N$ and vice versa. In addition, an $AC_{K,FK}^{\text{reg}}$-constraint $\varphi$ over $D$ may be satisfied by $T$ but it may not be satisfied by any XML tree conforming to $D_N$. To explore the connection between XML trees conforming to $D$ and those conforming to $D_N$, we replace $AC_{K,FK}^{\text{reg}}$-constraints over $D$ by new $AC_{K,FK}^{\text{con}}$-constraints over $D_N$. More precisely, given a set $\Sigma$ of $AC_{K,FK}^{\text{reg}}$-constraints over $D$, we define a set $\Sigma_N$ of $AC_{K,FK}^{\text{con}}$-constraints over $D_N$, referred to as the narrowed set of constraints of $\Sigma$, as follows. Let $f$ be a substitution for the element types in $E$ defined as $f(\tau) = \tau.(E_N \setminus \tau)'$ for every $\tau \in E$. Then for every key constraint $\beta, \tau.l \rightarrow \beta, \tau$ in $\Sigma$, $f(\beta, \tau.l \rightarrow f(\beta, \tau)$ is in $\Sigma_N$, and for every foreign key constraint $\beta, \tau.l \subseteq_{FK} \beta', \tau', \tau'_{\text{opt}} \beta_2, \tau_2, \tau_2$ in $\Sigma$ (recall that $\tau_l$ is the only attribute of $D$), $f(\beta_2), \tau_2, \tau_2 \subseteq_{FK} f(\beta_2), \tau_2, \tau_2$ is in $\Sigma_N$.

We are now ready to establish the connection between $D$ and $D_N$, which allows us to consider only narrow DTDs from now on.

**Lemma 3.9.** Let $D$ be a one-attribute DTD, $D_N$ the narrowed DTD of $D$, $\Sigma$ a set of $AC_{K,FK}^{\text{reg}}$-constraints over $D$ and $\Sigma_N$ the narrowed set of constraints of $\Sigma$. Then there exists an XML tree $T_1$ such that $T_1 \models (D, \Sigma)$ if and only if there exists an XML tree $T_2$ such that $T_2 \models (D_N, \Sigma_N)$.

**Proof.** It suffices to show the following:

**Claim:** Given any XML tree $T_1 \models D$ one can construct an XML tree $T_2$ by modifying $T_1$ such that $T_2 \models D_N$, and vice versa. Furthermore, for any regular expression $\beta, \tau.l \rightarrow \beta, \tau$ over $D$ and $\tau_l \subseteq_{FK} \beta, \tau.l \rightarrow \beta, \tau$ in $T_1$, we conclude that $\text{att}_1(u, \tau_l) \not\in \text{values}(\beta, \tau.l \rightarrow \beta, \tau)$ in $T_1$, where $f$ is the substitution defined above.

For the claim holds, we can show the lemma as follows. Assume that there exists an XML tree $T_1$ such that $T_1 \models (D, \Sigma)$. By the claim, there is $T_2$ such that $T_2 \models D_N$. Suppose, by contradiction, there is $\varphi \in \Sigma_N$ such that $T_2 \not\models \varphi$. (1) If $\varphi$ is a key $f(\beta, \tau.l \rightarrow \beta, \tau)$, then there exist two distinct nodes $x, y \in \text{nodes}(f(\beta, \tau))$ in $T_2$ such that $x.\tau_l = y.\tau_l$. In other words, $\text{values}(f(\beta, \tau.l \rightarrow \beta, \tau)) \not\in \text{values}(\beta, \tau.l \rightarrow \beta, \tau)$ in $T_2$. Since $T_1 \not\models \varphi$, it must be the case that $\text{values}(\beta, \tau.l \rightarrow \beta, \tau) \not\in \text{values}(\beta, \tau.l \rightarrow \beta, \tau)$. This concludes the proof of the lemma.
each $x \in \text{nodes}(\beta, \tau)$ uniquely identifies $x$ among $\text{nodes}(\beta, \tau)$. This contradicts the claim that $|\text{nodes}(f(\beta), \tau)|$ in $T_2$ equals $|\text{nodes}(\beta, \tau)|$ in $T_1$ and values$(f(\beta), \tau, \emptyset \emptyset)$ in $T_2$ equals values$(\beta, \tau, \emptyset \emptyset)$ in $T_1$. (2) If $\varphi$ is a foreign key: $f(\beta_1), \tau_1, \emptyset \emptyset \subseteq_{FK} f(\beta_2), \tau_2, \emptyset \emptyset$, then either $T_2 \neq f(\beta_2), \tau_2, \emptyset \emptyset \rightarrow f(\beta_2), \tau_2$ or there is $x \in \text{nodes}(f(\beta_1), \tau_1)$ such that for all $y \in \text{nodes}(f(\beta_2), \tau_2)$ in $T_2$, $x, \emptyset \emptyset \neq y, \emptyset \emptyset$. In the first case, we reach a contradiction as in (1). In the second case, we have $x, \emptyset \emptyset \notin \text{values}(f(\beta_2), \tau_2, \emptyset \emptyset)$ in $T_2$. By the claim, $x, \emptyset \emptyset \in \text{values}(f(\beta_1), \tau_1, \emptyset \emptyset)$ in $T_1$. Since $T_1 \models \varphi$, $x, \emptyset \emptyset \in \text{values}(\beta, \tau_1, \emptyset \emptyset)$ in $T_1$. Again by the claim, we have $x, \emptyset \emptyset \in \text{values}(f(\beta_2), \tau_2, \emptyset \emptyset)$ in $T_2$, which contradicts the assumption. The proof for the other direction is similar.

We next verify the claim. Given an XML tree $T_1 = (V_1, \text{lab}_1, e_{1e}, \text{att}, \text{root})$ such that $T_1 \models D$, we construct an XML tree $T_2$ by modifying $T_1$ such that $T_2 \models D$. Consider a $\tau$-element $v$ in $T_1$. Let $e_{1e}(v) = [v_1, \ldots, v_n]$ and $w = \text{lab}_1(v_1) \ldots \text{lab}_1(v_n)$. Recall $N_\tau$ and $P_\tau$, the set of nonterminals and the set of production rules generated when narrowing $\tau \rightarrow P(\tau)$ (see proof of Theorem 3.1). Let $Q_\tau$ be the set of $E$ symbols that appear in $P_\tau$. We can view $G = (Q_\tau, N_\tau \cup \{\tau\}, P_\tau, \tau)$ as an extended context-free grammar, where $Q_\tau$ is the set of terminals, $N_\tau \cup \{\tau\}$ the set of nonterminals, $P_\tau$ the set of production rules and $\tau$ the start symbol. Since $T_1 \models D$, we have $w \in P(\tau)$. By a straightforward induction on the structure of $P_\tau(\tau)$ it can be verified that $w$ is in the language defined by $G$. Thus there is a parse tree $T(w)$ w.r.t. the grammar $G$ for $w$, and $w$ is the frontier (the list of leaves from left to right) of $T(w)$. Without loss of generality, assume that the root of $T(w)$ is $v$, and the leaves are $v_1, \ldots, v_n$. Observe that the internal nodes of $T(w)$ are labeled with element types in $N_\tau$ except that the root $v$ is labeled $\tau$. Intuitively, we construct $T_2$ by replacing each element $v$ in $T_1$ by such a parse tree. More specifically, let $T_2 = (V_2, \text{lab}_2, e_{2e}, \text{att}, \text{root})$. Here $V_2$ consists of nodes in $V_1$ and the internal nodes introduced in the parse trees. For each $x$ in $V_2$, let $\text{lab}_2(x) = \text{lab}_1(x)$ if $x \in V_1$, and otherwise let $\text{lab}_2(x)$ be the node label of $x$ in the parse tree where $x$ belongs. Note that nodes in $V_2 \setminus V_1$ are elements of some type in $E_{N_\tau \setminus E}$. For every $x \in V_1$, let $e_{2e}(x)$ be the list of its children in the parse tree having $x$ as root. For every $x \in V_2 \setminus V_1$, let $e_{2e}(x)$ be the list of its children in the parse tree of an element in $V_2$ that contains $x$. Note that $\text{att}$ remains unchanged. By the construction of $T_2$ it can be verified that $T_2 \models D_N$; and moreover, for every regular expression $\beta.\tau$ over $D$ and $\emptyset \emptyset \in R(\tau)$, $|\text{nodes}(f(\beta), \tau)|$ in $T_2$ equals $|\text{nodes}(\beta, \tau)|$ in $T_1$ and values$(f(\beta), \tau, \emptyset \emptyset)$ in $T_2$ equals values$(\beta, \tau, \emptyset \emptyset)$ in $T_1$, because among other things, (1) if a string $r.\tau_1, \ldots, \tau_n.\tau$ over $E$ is in $\beta.\tau$, then for every sequence of strings $v_0, \ldots, v_n$ in $(E_\tau \setminus E)^*$, $w_0.\tau_1, \ldots, w_n.\tau$ is in $f(\beta).\tau$; (2) if a string $r.w_0.\tau_1, \ldots, \tau_n.\tau$ is in $f(\beta).\tau$, where $\tau_1, \ldots, \tau_n, \tau$ are element types in $E$ and $w_0, \ldots, w_n$ are strings in $(E_{N_\tau \setminus E})^*$, then $r.\tau_1, \ldots, \tau_n.\tau$ is in $\beta.\tau$; (3) none of the new nodes, i.e., nodes in $V_2 \setminus V_1$, is labeled with an $E$-type; (4) no new attributes are defined; and (5) the ancestor-descendant relation on $T_1$-elements is not changed in $T_2$.

Conversely, assume that there is $T_2 = (V_2, \text{lab}_2, e_{2e}, \text{att}, \text{root})$ such that $T_2 \models D_N$. We construct an XML tree $T_1$ by modifying $T_2$ such that $T_1 \models D$. For any node $v \in V_2$ with $\text{lab}(v) = \tau$ and $\tau \in E_{N_\tau \setminus E}$, we replace $v$ in $e_{2e}(v')$ by the children of $v$, where $v'$ is the parent of $v$. In addition, we remove $v$ from $V_2$, $\text{lab}_2(v)$ from $\text{lab}_2$, and $e_{2e}(v)$ from $e_{2e}$. Observe that by the definition of $D_N$, no attributes are defined for elements of any type in $E_{N_\tau \setminus E}$. We repeat the process until there is no node labeled with element type in $E_{N_\tau \setminus E}$. Now let $T_1 = (V_1, \text{lab}_1, e_{1e}, \text{att}, \text{root})$, where $V_1$, $\text{lab}_1$ and $e_{1e}$ are $V_2$, $\text{lab}_2$ and $e_{2e}$ at the end of the process, respectively. Observe that $\text{att}$ and $\text{root}$ remain unchanged. By the definition of $T_1$ it can be verified that $T_1 \models D$; and in addition, for any regular expression $\beta.\tau$ over $D$ and $\emptyset \emptyset \in R(\tau)$, $|\text{nodes}(f(\beta), \tau)|$ in $T_1$ equals $|\text{nodes}(f(\beta), \tau)|$ in $T_2$, and values$(f(\beta), \tau, \emptyset \emptyset)$ in $T_1$ equals values$(f(\beta), \tau, \emptyset \emptyset)$ in $T_2$, because of (1) and (2) above and, among other things, the fact that none of the nodes removed is labeled with a type of $E$ and the attribute function $\text{att}$ is unchanged.

We now move to encoding of DTDs, more specifically, narrow one-attribute DTDs. Let $D = (E, \{\emptyset \emptyset\}, P, R, r)$ be a narrow one-attribute DTD and $\Sigma$ a set of $\text{AC}_{\text{FK}}$-constraints over $D$. We encode $D$ with a system $\Theta \Sigma$ of integer constraints such that there exists an XML tree conforming to $D$ iff $\Psi \Sigma \Theta$ admits a nonnegative solution. The coding is developed w.r.t. $\Sigma$. More specifically, assume that $\beta_1, \tau_1, \emptyset \emptyset, \ldots, \beta_k, \tau_k, \emptyset \emptyset$ is an enumeration of all regular expressions and attributes that appear in $\Sigma$ and $\Theta$ be the set of functions $\theta : \{1, \ldots, k\} \rightarrow \{0, 1\}$ which are not identically $0$. For every $\theta \in \Theta$, define a regular

---

4As in the proof of Lemma 3.2, if $\tau$ is in $P(\tau)$, then we need to rename $\tau$ in $Q_+ \emptyset$ to ensure that $Q_+$ and $N_\tau \cup \{\tau\}$ are disjoint. It is straightforward to handle that case.
expression:

$$r_\theta = \left( \bigcap_{i: \theta(i)=1} \beta_i.\tau_i \right) \cap \left( \bigcap_{j: \theta(j)=0} \overline{\beta_j.\tau_j} \right),$$

where $\overline{\beta_j.\tau_j}$ is the complement $\beta_j.\tau_j$. We allow intersection and complement operators only in regular expressions $r_\theta$. We note that for every $i \in [1, k]$:

$$\beta_i.\tau_i = \bigcup_{\theta: \theta(i)=1} r_\theta.$$

Then to capture the interaction between $D$ and constraints of $\Sigma$, the system $\Psi_D^\Sigma$ has a variable $|\text{nodes}(\beta.\tau_i)|$, for every $i \in [1, k]$, and $|\text{nodes}(r_\theta)|$, for every $\theta \in \Theta$. In other words, $\Psi_D^\Sigma$ specifies the dependencies imposed by $D$ on the number of elements reachable by following $\beta_i.\tau_i$ ($i \in [1, k]$) and $r_\theta$ ($\theta \in \Theta$).

To capture $\beta_i.\tau_i$ ($i \in [1, k]$) and $r_\theta$ ($\theta \in \Theta$) in $\Psi_D^\Sigma$, consider, for each regular expression $\beta_i.\tau_i$ ($i \in [1, k]$), a deterministic automaton that recognizes that expression. Let $M$ be the deterministic automaton equivalent to the product of all these automata. We refer to $M$ as the DFA for $\Sigma$. Let $s_M$ be the start state of $M$ and $\delta$ be its transition function. Given an XML tree $T$ conforming to $D$, for each node $x$ in $T$ we define $\text{state}(x)$ as $s$, if there is a simple path $\rho$ over $D$ such that $T \models \rho(\text{root}, x)$ and $s = \delta(s_M, \rho)$. The connection between $M$ and $T$ w.r.t. $\beta_i.\tau_i$ ($i \in [1, k]$) is described by the following lemma:

**Lemma 3.10.** Let $D$ be a narrow one-attribute DTD, $\Sigma$ a set of $\mathcal{A}_{\mathcal{K},FK}$-constraints over $D$, $M$ the DFA for $\Sigma$ and $\beta_i.\tau_i$ a regular expression in $\Sigma$. Then for every XML tree $T$ conforming to $D$ and every $\tau_i$-element $x$ in $T$, $x \in \text{nodes}(\beta_i.\tau_i)$ in $T$ iff $\text{state}(x)$ contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$.

In other words, $\text{nodes}(\beta_i.\tau_i)$ in $T$ consists of all $\tau_i$-elements $x$ such that $\text{state}(x)$ (which is a tuple of states of automata corresponding to regular expressions in $\Sigma$) contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$. A similar idea was exploited in [1].

**Proof.** Since $T$ is a tree, there exists a unique simple path $\rho$ over $D$ such that $T \models \rho.\tau_i(\text{root}, x)$. Thus $x \in \text{nodes}(\beta_i.\tau_i)$ in $T$ iff $\rho.\tau_i \in \beta_i.\tau_i$. If $x \in \text{nodes}(\beta_i.\tau_i)$ in $T$, then $\rho.\tau_i \in \beta_i.\tau_i$ and, therefore, there must be a final state $f_{\beta_i.\tau_i}$ in the automaton for $\beta_i.\tau_i$ and a state $s$ in $M$ such that $s = \delta(s_M, \rho.\tau_i)$ and $s$ contains $f_{\beta_i.\tau_i}$. Thus $\text{state}(x) = s$ contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$. Conversely, if $\text{state}(x)$ contains a final state $f_{\beta_i.\tau_i}$ in the automaton for $\beta_i.\tau_i$, then $\rho.\tau_i \in \beta_i.\tau_i$ since $s = \delta(s_M, \rho.\tau_i)$. Therefore, $x \in \text{nodes}(\beta_i.\tau_i)$ in $T$.

We next define a system $\Psi_D^\Sigma$ of integer constraints. The variables used in the constraints of $\Psi_D^\Sigma$ are as follows. Let $\tau \in E$ be an element type and $s = \delta(s_M, \rho, \tau)$ for some simple path $\rho.\tau \in E^*$. For each such pair we create a distinct variable $x^s_{\tau}$. Intuitively, in an XML tree $T$ conforming to $D$, we use $x^s_{\tau}$ to keep track of the number of $\tau$-elements with state $s$. Furthermore, define $Y^s_\tau$ as the set of pairs $(\tau', s')$ such that $\tau' \in E$, $s' = \delta(s_M, \rho, \tau')$ for some simple path $\rho.\tau' \in E^*$, $\tau$ is mentioned in $P(\tau')$ and $s = \delta(s', \tau)$. For each such pair $(\tau', s')$, we create a variable $x^s_{\tau'}$ to keep track of the number of $\tau$-elements with state $s$ that are children of a node of type $\tau'$ with state $s'$. There are exponentially many variables (in the size of $D$ and $\Sigma$) in total since $M$ is a DFA. Using these, we define an integer constraint to specify $\tau \rightarrow P(\tau')$ at state $s$ as follows. Let us use $\Psi^s_\tau$ to denote the set of integer constraints defined for $\tau$ at $s$.

- If $P(\tau) = \tau_1$, then $\Psi^s_\tau$ includes $x^s_{\tau} = x^{s_1}_{\tau_1}$, where $s_1 = \delta(s, \tau_1)$.
- If $P(\tau) = (\tau_1, \tau_2)$, then $\Psi^s_\tau$ includes $x^s_{\tau} = x^{s_1}_{\tau_1} + x^{s_2}_{\tau_2}$, where $s_1 = \delta(s, \tau_1)$ for $i = 1, 2$.
  Referring to the XML tree $T$, these assure that each $\tau$-element in $T$ must have a $\tau_1$-subelement and a $\tau_2$-subelement.
- If $P(\tau) = (\tau_1 | \tau_2)$, then $\Psi^s_\tau$ includes $x^s_{\tau} = x^{s_1}_{\tau_1} + x^{s_2}_{\tau_2}$, where $s_1 = \delta(s, \tau_1)$ for $i = 1, 2$. This assures that each $\tau$-element in $T$ must have either a $\tau_1$-subelement or a $\tau_2$-subelement, and thus the sum of the number of these $\tau_1$-subelements and the number of $\tau_2$-subelements equals the number of $\tau$-elements.

\[4\]Recall that the regular language defined by a regular expression $\beta$ is denoted by $\beta$ as well.
• If \( P(\tau) = \tau_i^* \), then \( \Psi^r_\tau \) includes \((x^{s_1^*}; \tau) > 0 \rightarrow (x^*_\tau > 0)\), where \( s_1 = \delta(s, \tau_1) \).

In addition, \( \Psi^r_\tau \) includes \( x^*_\tau = \sum_{(\tau', s') \in Y'_\tau} x^{s'; \tau'} \).

Recall that \( \beta_1, \tau_1 \)\(, \ldots, \beta_k, \tau_k \)\(, \ldots, \beta_l \)\(, \ldots, \beta_l \) is an enumeration of all regular expressions and attributes that appear in \( \Sigma \), that \( \Theta \) is the set of functions \( \theta : \{1, \ldots, k\} \rightarrow \{0, 1\} \) which are not identically 0 and that for each such function \( \theta \), \( \tau_0 \) is a regular expression defined as in (3.1). For each \( i \in [1, k] \), we define \( F_{\beta_i, \tau_i} \) as the set of states \( s = (s_1, \ldots, s_k) \) of the DFA for \( \Sigma \) such that \( s_i \) is a final state of the DFA for \( \beta_i, \tau_i \). Notice that by Lemma 3.10, for every XML tree \( T \) conforming to \( D \) and every node \( x \) of \( T \), \( x \in \text{nodes}(\beta_i, \tau_i) \) in \( T \) if and only if \( \text{state}(x) \in F_{\beta_i, \tau_i} \). Furthermore, for each \( \theta \in \Theta \), we define \( F_\theta \) as the set of states \( s = (s_1, \ldots, s_k) \) of the DFA for \( \Sigma \) such that for every \( i \in [1, k] \), \( s_i \) is a final state of the DFA for \( \beta_i, \tau_i \) if and only if \( \theta(i) = 1 \). Notice that by Lemma 3.10, for every XML tree \( T \) conforming to \( D \) and every node \( x \) of \( T \), \( x \in \text{nodes}(\tau_0) \) in \( T \) if and only if \( \text{state}(x) \in F_\theta \). Finally, for each \( r_0 \neq \emptyset \), we have that for every \( i, j \in [1, k] \), if \( \theta(i) = \theta(j) = 1 \), then \( \tau_i = \tau_j \). In this case, we define \( \tau_0 \) as \( \tau_i \), for an arbitrary \( i \in [1, k] \) such that \( \theta(i) = 1 \).

By our restriction on regular expressions regarding element type \( r \), there is a unique variable \( x^*_r \) associated with \( r \), where \( s = \delta(s_M, \tau) \). We write \( x_r \) for \( x^*_r \). Then we define the set of cardinality constraints determined by DTD \( D \) w.r.t. a set \( \Sigma \) of \( \mathcal{A}^{\text{reg}}_{K, FK} \)-constraints over \( D \), denoted by \( \Psi^r_\Sigma \), as follows:

- For each \( \tau \in E \) and each state \( s \) given above, \( \Psi^s_\tau \) contains all the constraints in \( \Psi^r_\tau \).
- \( \Psi^r_\Sigma \) contains constraint \( x_r = 1 \); i.e., there is a unique root in each XML tree conforming to \( D \).
- For every \( i \in [1, k] \), \( \Psi^r_\Sigma \) contains constraint \( \sum_{s \in F_{\beta_i, \tau_i}} x^s_i \).
- For every \( \theta \in \Theta \) such that \( r_\theta \neq \emptyset \), \( \Psi^r_\Sigma \) contains constraint \( \sum_{s \in F_\theta} x^s_r = 0 \).

Note that \( \Psi^r_\Sigma \) can be computed in EXPTIME in the size of \( D \) and \( \Sigma \). We say that \( \Psi^r_\Sigma \) is consistent if it has a nonnegative solution. We next show that \( \Psi^r_\Sigma \) indeed characterizes narrow one-attribute DTD \( D \).

**Lemma 3.11.** Let \( D \) be a narrow one-attribute DTD, \( \Sigma \) a set of \( \mathcal{A}^{\text{reg}}_{K, FK} \)-constraints over \( D \) and \( \Psi^r_\Sigma \) the set of cardinality constraints determined by \( D \) w.r.t. \( \Sigma \). Then \( \Psi^r_\Sigma \) is consistent if there is an XML tree \( T \) such that \( T \models D \). In addition, for every \( i \in [1, k] \) and \( \theta \in \Theta \), \( \text{nodes}(\beta_i, \tau_i) \) and \( \text{nodes}(\tau_0) \) in \( T \) equal the value of variables \( \text{nodes}(\beta_i, \tau_i) \) and \( \text{nodes}(\tau_0) \) given by the solution to \( \Psi^r_\Sigma \).

**Proof.** First, assume that there is an XML tree \( T = (V, \text{lab, ele, att, root}) \) conforming to \( D \). We define a nonnegative solution of \( \Psi^r_\Sigma \) as follows. For each variable \( x^{s, s'}_{\tau, \tau'} \) in \( \Psi^r_\Sigma \), let its value be the number of \( \tau \)-elements \( x \) in \( T \) such that \( x \) is a child of a node \( y \) of type \( \tau' \) with \( \text{state}(x) = s \) and \( \text{state}(y) = s' \). Furthermore, let \( x_r \) be 1 and for every variable \( x^*_r \) in \( \Psi^r_\Sigma \), let \( x^*_r \) be the sum of the variables \( x^{s, s'}_{\tau, \tau'} \), where \( (\tau', s') \in Y'_\tau \). Finally, for every \( i \in [1, k] \) and every \( \theta \in \Theta \), let \( \text{nodes}(\beta_i, \tau_i) \) and \( \text{nodes}(\tau_0) \) be \( \sum_{s \in F_{\beta_i, \tau_i}} x^s_i \) and \( \sum_{s \in F_\theta} x^s_r \), respectively. This defines a nonnegative assignment since \( T \) is finite. It can be verified that the assignment is a solution of \( \Psi^r_\Sigma \). Indeed, it satisfies the constraint \( x_r = 1 \) and constraints of the form \( x^{s, s'}_{\tau, \tau'} \) and \( \sum_{s \in F_{\beta_i, \tau_i}} x^s_i \) and \( \sum_{s \in F_\theta} x^s_r \), by the definition of the assignment. Moreover, one can verify that it also satisfies the constraints of each \( \Psi^r_\tau \), by considering the four different cases corresponding to the four different types of regular expressions in \( D \). In particular, it satisfies constraints of the form \( (x^{s_1^*}; \tau) > 0 \rightarrow (x^*_\tau > 0) \) for each \( \tau \rightarrow \tau_1 \) in \( P \), since if \( x^{s_1^*}; \tau > 0 \), then there exists a \( \tau_1 \)-node in \( T \) having as its parent a \( \tau \)-node \( y \) with \( \text{state}(y) = s \). Thus, \( x^*_s \) \( > 0 \) by the definition of the assignment. Therefore, \( \Psi^r_\Sigma \) is consistent. Moreover, by Lemma 3.10, for every \( i \in [1, k] \) and \( \theta \in \Theta \), the values of variables \( \text{nodes}(\beta_i, \tau_i) \) and \( \text{nodes}(\tau_0) \) in the solution are indeed \( \text{nodes}(\beta_i, \tau_i) \) and \( \text{nodes}(\tau_0) \) in \( T \).

Conversely, assume that \( \Psi^r_\Sigma \) admits a nonnegative solution. We show that there exists an XML tree \( T = (V, \text{lab, ele, att, root}) \) such that \( T \models D \). To do so, for each element type \( \tau \) and state \( s \) for \( \tau \), we create \( x^*_s \) many distinct \( \tau \)-elements. Let \( \text{ext}(\tau) \) denote the set of all \( \tau \)-elements created above and

\[
V = \bigcup_{\tau \in E} \text{ext}(\tau).
\]
Then function $\text{lab}$ is defined as $\text{lab}(v) = \tau$ if $v \in \text{ext}(\tau)$, and function $\text{att}$ is defined as follows:

$$\text{att}(v, @l) = \begin{cases} 
\text{empty string} & \text{if } @l \in R(\text{lab}(v)) \\
\text{undefined} & \text{otherwise}
\end{cases}$$

It is easy to verify that these functions are well defined. Let $\text{root}$ be the node labeled $r$, which is unique since $x_r = 1$ is in $\Psi_D^\Sigma$. Finally, to define function $\text{ele}$, we do the following. For each $x_{\tau,\tau'}$, in $\Psi_D^\Sigma$, we choose $x_{\tau,\tau'}^s$ many distinct vertices labeled $\tau$ and mark them with $x_{\tau,\tau'}^s$. Note that every $\tau$-element in $V$ can be marked once and only once. Starting at $\text{root}$, for each $\tau$-element $x$ marked with $x_{\tau,\tau'}^s$, for some $(\tau', s') \in \mathcal{Y}_s$, consider $P(\tau)$ and constraints of $\Psi_D^\Sigma$. If $P(\tau) = (\tau_1, \tau_2)$, then we choose a $\tau_1$-element $y$ marked with $x_{\tau_1,\tau_2}^s$ and let $\text{ele}(x) = [y]$, where $x_{\tau_1,\tau_2}^s = x_{\tau_1,\tau_2}^s$. If $P(\tau) = (\tau_1, \tau_2)$, then choose a $\tau_1$-element $y$ marked with $x_{\tau_1,\tau_2}^s$ and let $\text{ele}(x) = [y]$, where $x_{\tau_1,\tau_2}^s = x_{\tau_1,\tau_2}^s$. Finally, to see that for every $x$ it is easy to state that $\text{state}(x) = s$. Since $T$ is a tree, there is a unique simple path $p \in E^*$ such that $T \models p(\text{root}, x)$. We show the claim by induction on the length $|p|$ of $p$. If $|p| = 1$, i.e., $\rho = r$, then $x$ is the root and obviously, $\text{state}(x) = \delta(s_M, r)$. Assume the claim for $\rho$ and show that the claim holds for $p.\tau$. Let $y$ be the $\tau'$-element in $T$ such that $T \models p(\text{root}, y)$ and $y$ is the parent of $x$. Suppose that $y$ is marked with $x_{\tau',\tau''}^s$ in the construction. By the induction hypothesis $\text{state}(y) = s'$. It is easy to state $\text{state}(x) = \delta(s', \tau)$. By the definition of $\Psi^s_D$, we have that $s$ is precisely the state $\delta(s', \tau)$. This proves the claim and thus the lemma.

We now move to encoding $\mathcal{AC}^{\mathcal{K},\mathcal{FK}}_{\Sigma}$-constraints in terms of integer constraints. Let $D$ be a DTD $(E, \{@l\}, P, R, r)$ and $\Sigma$ a set of $\mathcal{AC}^{\mathcal{K},\mathcal{FK}}_{\Sigma}$-constraints over $D$. By Lemmas 3.8 and 3.9, we assume, without loss of generality, that $D$ is a narrow one-attribute DTD. To encode $\Sigma$, let $\beta_1, \tau_1, @l_1, \ldots, \beta_k, \tau_k, @l_k$ be an enumeration of all regular expressions and attributes that appear in $\Sigma$, and for every function $\theta : \{1, \ldots, k\} \to \{0, 1\}$ which is not identically 0, let regular expression $r_\theta$ be defined as in (3.1). Then for every nonempty $\Omega \subseteq \Theta$, we introduce a new variables $z_\Omega$. In any XML tree conforming to $D$, the intended interpretations of $z_\Omega$ is the cardinality of

$$\left( \bigcap_{\theta : \theta \in \Omega} \text{values}(r_\theta, @l) \right) \setminus \left( \bigcup_{\theta : \theta \in \Theta \setminus \Omega} \text{values}(r_\theta, @l) \right).$$

Note that the number of new variables is double-exponential in the number of regular expression in $\Sigma$. Using these variables, we define the set of the cardinality constraints determined by $\Sigma$, denoted by $C_\Sigma$, which consists of the following:

\footnote{We assume that $\text{root}$ is marked with $x_r^s$, where $s = \delta(s_M, r)$ and $s_M$ is the initial state of the DFA for $\Sigma$.}
\[
\sum_{\Omega: \theta \in \Omega} z_\Omega = |\text{values}(r_\theta, \emptyset)| \quad \text{for every } \theta \in \Theta,
\]
\[
\sum_{\Omega: \Omega \cap (\theta | \theta(i) = 1) \neq \emptyset} z_\Omega = |\text{values}(\beta_i, \tau_i, \emptyset)| \quad \text{for every } i \in [1, k],
\]
\[
|\text{values}(\beta_i, \tau_i, \emptyset)| = |\text{nodes}(\beta_i, \tau_i)| \quad \text{for every } \beta_i, \tau_i \in \Sigma,
\]
\[
|\text{values}(\beta_i, \tau_i, \emptyset)| = |\text{nodes}(\beta_i, \tau_i)| \quad \text{for every } \beta_i, \tau_i \in \Sigma,
\]
\[
\sum_{\Omega: \Omega \cap (\theta | \theta(i) = 1) \neq \emptyset, \nonumber \quad \Omega \cap (\theta | \theta(j) = 1) = \emptyset} z_\Omega = 0 \quad \text{for every } \beta_i, \tau_i \in \Sigma,
\]
\[
|\text{values}(\beta_i, \tau_i, \emptyset)| \leq |\text{nodes}(\beta_i, \tau_i)| \quad \text{for every } i \in [1, k],
\]
\[
|\text{values}(r_\theta, \emptyset)| \leq |\text{nodes}(r_\theta)| \quad \text{for every } \theta \in \Theta.
\]

Note that the size of \( C_\Sigma \) is double-exponential in the size of \( \Sigma \).

We now combine the encodings for constraints and the DTDs, and present a system \( \Psi(D, \Sigma) \) of linear integer constraints for a DTD \( D \) and a set \( \Sigma \) of \( \mathcal{AC}_{K, FK}^\text{reg} \)-constraints. Assuming that \( D \) and \( \Sigma \) are as above, the set \( \Psi(D, \Sigma) \), called the set of cardinality constraints determined by \( D \) and \( \Sigma \), is defined to be:

\[
\Psi^\Sigma_D \cup C_\Sigma \cup \{(|\text{nodes}(\beta_i, \tau_i)| > 0) \rightarrow (|\text{values}(\beta_i, \tau_i, \emptyset)| > 0) \mid i \in [1, k]\} \cup \{(|\text{nodes}(r_\theta)| > 0) \rightarrow (|\text{values}(r_\theta, \emptyset)| > 0) \mid \theta \in \Theta\},
\]

where \( C_\Sigma \) is the set of cardinality constraints determined by \( \Sigma \), and \( \Psi^\Sigma_D \) is the set of cardinality constraints determined by \( D \) w.r.t. \( \Sigma \). The system \( \Psi(D, \Sigma) \) is said to be consistent if it has a nonnegative solution that satisfies all of its constraints. Observe that \( \Psi(D, \Sigma) \) can be partitioned into two sets: \( \Psi(D, \Sigma) = \Psi^\Sigma(D, \Sigma) \cup \Psi^\Omega(D, \Sigma) \), where \( \Psi^\Sigma(D, \Sigma) \) consists of linear integer constraints, and \( \Psi^\Omega(D, \Sigma) \) consists of constraints of the form \( (x > 0 \rightarrow y > 0) \). Also note that the size of \( \Psi(D, \Sigma) \) is double-exponential in the size of \( D \) and \( \Sigma \).

We next show that \( \Psi(D, \Sigma) \) indeed characterizes the consistency of \( D \) and \( \Sigma \).

**Lemma 3.12.** Let \( D \) be a narrow one-attribute DTD, \( \Sigma \) a finite set of \( \mathcal{AC}_{K, FK}^\text{reg} \)-constraints over \( D \) and \( \Psi(D, \Sigma) \) the set of cardinality constraints determined by \( D \) and \( \Sigma \). Then \( \Psi(D, \Sigma) \) is consistent if and only if there is an XML tree \( T \) such that \( T \models (D, \Sigma) \).

**Proof.** Suppose that there exists an XML tree \( T \) such that \( T \models (D, \Sigma) \). Then by Lemma 3.11, there exists a nonnegative solution for \( \Psi^\Sigma_D \) such that for every \( i \in [1, k] \) and \( \theta \in \Theta \), the values of variables \( |\text{nodes}(r_\theta)| \) and \( |\text{nodes}(\beta_i, \tau_i)| \) in this solution coincide with \( |\text{values}(r_\theta)| \) and \( |\text{values}(\beta_i, \tau_i)| \) in \( T \), respectively. From this solution, it is easy to generate a solution to \( \Psi(D, \Sigma) \) by assigning to variable \( |\text{values}(r_\theta, \emptyset)| \) the size of \( \text{values}(r_\theta, \emptyset) \) in \( T \), for every \( \theta \in \Theta \), assigning to variable \( |\text{values}(\beta_i, \tau_i, \emptyset)| \) the size of \( \text{values}(\beta_i, \tau_i, \emptyset) \) in \( T \), for every \( i \in [1, k] \), and then assigning to each variable \( z_\Omega \) the cardinality of set \( \text{values}(\beta_i, \tau_i, \emptyset) \) for some regular expression \( \beta \tau \) mentioned in \( \Sigma \), and leave \( \text{values}(\beta_i, \tau_i, \emptyset) \) unchanged otherwise. To do this, for each variable \( z_\Omega \) we create a set \( s_\Omega \) of distinct string values such that \( |s_\Omega| = z_\Omega \) and \( s_\Omega \cap s_\Omega' = \emptyset \) if \( \Omega \neq \Omega' \). Then for every \( \Omega \subseteq \Theta \), we let \( \text{values}(r_\theta, \emptyset) \) in \( T \) to contain \( s_\Omega \) if and only if \( \theta \in \Omega \). This is possible because (1) \( \sum_{\Omega: \theta \in \Omega} z_\Omega = |\text{values}(r_\theta, \emptyset)| \) is in \( C_\Sigma \), for every \( \theta \in \Theta \); (2) \( \sum_{\Omega: \Omega \cap (\theta | \theta(i) = 1) \neq \emptyset} z_\Omega = |\text{values}(\beta_i, \tau_i, \emptyset)| \) is in \( C_\Sigma \), for every \( i \in [1, k] \); (3) if \( r_\theta = \emptyset \), then
|nodes(rθ)| = 0 in ΨΣ, for every θ ∈ Θ; (4) |values(β, τ, Λl)| ≤ |nodes(β, τ)| is in CΣ, for every i ∈ [1, k]; (5) |values(rθ, Λl)| ≤ |nodes(rθ)| is in CΣ, for every θ ∈ Θ; and (6) nodes(β) in T equals nodes(β) in T′, for every regular expression β over D.

We next show that T has the desired properties. It is easy to verify T |= D given the construction of T from T′ and the assumption T′ |= D. By the definition of T, we have that for every i ∈ [1, k] and θ ∈ Θ, |nodes(βi, τi)|, |values(βi, τi, Λi)|, |nodes(rθ)| and |values(rθ, Λl)| in T equal the value of variables |nodes(β, τ)|, |values(β, τ, Λ)|, |nodes(rθ)| and |values(rθ, Λl)| given by the solution to Ψ(D, Σ). We use this property to show that T |= Σ. Let φ be a constraint in Σ. (1) If φ is a key β, τ, Λl → β, Λl, it is immediate from the definition of T that T |= φ since |values(β, τ, Λl)| = |nodes(β, τ)| is a constraint in CΣ and, hence, |values(β, τ, Λl)| = |nodes(β, τ)| in T. That is, each x ∈ nodes(β, τ) in T has a distinct @l-attribute value and thus the value of its @l-attribute uniquely identifies x among nodes in nodes(β, τ). (2) If φ is β, τ, @l ⊆ Fk β, τ, Λl, it is easy to see that in T:

\[ |values(β, τ, Λl) \setminus values(β, τ, Λl)| = \bigcup_{\Omega : \Omega \cap \{θ|θ(i)=1\} \neq \emptyset, \Omega \cap \{θ'|θ'(j)=1\} = \emptyset} s_\Omega, \]

Since \( s_\Omega \cap s_\Omega' = \emptyset \) if Ω ≠ Ω',

\[ |values(β, τ, Λl) \setminus values(β, τ, Λl)| = \sum_{\Omega : \Omega \cap \{θ|θ(i)=1\} \neq \emptyset, \Omega \cap \{θ'|θ'(j)=1\} = \emptyset} z_\Omega. \]

Thus, given that

\[ \sum_{\Omega : \Omega \cap \{θ|θ(i)=1\} \neq \emptyset, \Omega \cap \{θ'|θ'(j)=1\} = \emptyset} z_\Omega = 0 \]

is in CΣ (since β, τ, Λl ⊆ Fk β, τ, Λl ∈ Σ), we have |values(β, τ, Λl) \setminus values(β, τ, Λl)| = 0 in T, that is, values(β, τ, Λl) ⊆ values(β, τ, Λl) in T. Furthermore, T |= β, τ, Λl → β, τ since |values(β, τ, Λl)| = |nodes(β, τ)| is a constraint in CΣ. Thus T |= φ. This concludes the proof of the lemma.

We need another lemma for a mild generalization of linear integer constraints.

**Lemma 3.13.** Given a system \( Ax \leq b \) of linear integer constraints together with conditions of the form \( (x_i > 0) \rightarrow (x_j > 0), \) where A is an n × m matrix on integers, b is an n-vector on integers and 1 ≤ i, j ≤ m, the problem of determining whether the system admits a nonnegative integer solution is in NP.

**Proof.** Let \( c_1, \ldots, c_p \) enumerate the conditions of the form \((x > 0) \rightarrow (y > 0), \) \( c_k \) being \((x_k > 0) \rightarrow (x_k^2 > 0).\) Consider 2p instances \( I_j \) of integer programming obtained by adding, for each \( k \leq p, \) either \( x_k^1 = 0, \) or \( x_k^2 > 0 \) to \( Ax \leq b. \) Clearly, the original system of constraints has a solution iff some \( I_j \) has a solution. By [27], \( I_j \) has a solution iff it has a solution whose size is polynomial in A, b and p. Hence, to check if the original system of constraints has a solution, it suffices to guess a system \( I_j \) and then guess a polynomial size solution for it; thus, the problem is in NP.

We now conclude the proof of the first part of the theorem. By Lemma 3.8, given an arbitrary DTD D and a set Σ of \( A^n_{K,FK} \)-constraints over D, it is possible to compute a one-attribute DTD D′ and a set \( \Sigma' \) of \( A^n_{K,FK} \)-constraints over D′ such that \( (D, Σ) \) is consistent iff \( (D', Σ') \) is consistent. By Lemma 3.9, one can compute a narrow one-attribute DTD D′|N and a set \( \Sigma'_{N} \) of \( A^n_{K,FK} \)-constraints over D′|N such that \( (D', Σ') \) is consistent iff \( (D'_{N}, Σ'_{N}) \) is consistent. By Lemma 3.12, \( (D'_{N}, Σ'_{N}) \) is consistent iff \( Ψ(D'_{N}, Σ'_{N}) \) has a nonnegativity integer solution. Thus, \( (D, Σ) \) is consistent iff \( Ψ(D'_{N}, Σ'_{N}) \) has a nonnegativity integer solution. Note that \( (D', Σ') \) can be computed in polynomial time on \( |D| + |Σ|, (D'_{N}, Σ'_{N}) \) can be computed in polynomial time on \( |D'| + |Σ'|, \) and \( Ψ(D'_{N}, Σ'_{N}) \) can be computed in double-exponential time on \( |D'_{N}| + |Σ'_{N}|. \) Thus, by Lemma 3.13, one can check in 2-NEXPTIME whether there exists an XML tree T such that T |= (D, Σ).

**Proof of b)** We establish the PSPACE-hardness by reduction from the QBF-CNF problem. An instance of QBF-CNF is a quantified boolean formula in prenex conjunctive normal form. The problem is to determine whether this formula is valid. QBF-CNF is known to be PSPACE-complete [20, 28].

Let \( \theta \) be a formula of the form

\[ Q_1x_1 \cdots Q_mx_m\psi, \] (3.3)
where each \( Q_i \in \{ \forall, \exists \} \) (1 \( i \leq m \)) and \( \psi \) is a propositional formula in conjunctive normal form, say \( C_1 \land \cdots \land C_n \), that mentions variables \( x_1, \ldots, x_m \). We construct a DTD \( D_\theta \) and a set \( \Sigma_\theta \) of \( AC_{FK} \) constraint such that \( \theta \) is valid if and only if there is an XML tree conforming to \( D_\theta \) and satisfying \( \Sigma_\theta \).

We construct a DTD \( D_\theta = (E, A, P, R, r) \) as follows. \( E = \{ r, C \} \cup \bigcup_{i=1}^{m-1} \{ x_i, \bar{x}_i, N_{x_i}, P_{x_i} \} \), \( A = \{ \emptyset \} \) and \( P \) is defined by considering the quantifiers of \( \theta \). We use \( Q_1 \) to define \( P \) on the root:

\[
P(r) = \begin{cases}  \{ (N_{x_1}|P_{x_1}), C \} & Q_1 = \exists \\  \{ (N_{x_1}, P_{x_1}), C \} & Q_1 = \forall \end{cases}
\]

In general, for each 1 \( i \leq m - 1 \), we consider quantifier \( Q_{i+1} \) to define \( P(N_{x_i}) \) and \( P(P_{x_i}) \):

\[
P(N_{x_i}) = P(P_{x_i}) = \begin{cases}  \{ N_{x_{i+1}}|P_{x_{i+1}} \} & Q_{i+1} = \exists \\  \{ N_{x_{i+1}}, P_{x_{i+1}} \} & Q_{i+1} = \forall \end{cases}
\]

We represent formula \( \psi \) as a regular expression. Given a clause \( C_j = \bigvee_{i=1}^{n_j} y_i \lor \bigwedge_{j=1}^{n_j} \neg z_i \) (1 \( j \leq n \)), \( tr(C_j) \) is defined to be the regular expression \( y_1 \cdot \cdots \cdot y_{n_j} | \bar{z}_1 \cdot \cdots \cdot \bar{z}_{n_j} \). We define \( P \) on element types \( N_{x_m} \) and \( P_{x_m} \) as \( P(N_{x_m}) = P(P_{x_m}) = tr(C_1), \ldots, tr(C_n) \). For the remaining elements of \( E \), we define \( P \) as \( \epsilon \). We define function \( R \) as follows:

\[
R(r) = R(P_{x_m}) = R(N_{x_i}) = \emptyset \quad 1 \leq i \leq m \\
R(C) = R(x_i) = R(\bar{x}_i) = \{ \emptyset \} \quad 1 \leq i \leq m.
\]

Finally, \( \Sigma_\theta \) contains the following foreign keys:

\[
r_\cdot \cdot N_{x_i} \cdot x_i \cdot \emptyset l \subseteq\text{FK r.C.C.} \emptyset l, \quad r_\cdot \cdot P_{x_i} \cdot \cdot \bar{x}_i \cdot \emptyset l \subseteq\text{FK r.C.C.} \emptyset l, \quad i \in [1, m].
\]

For instance, for the formula \( \forall x_1 \exists x_2 \forall x_3 (x_1 \lor x_2 \lor \neg x_3) \), an XML tree conforming to \( D \) is shown in Figure 3.2. In this tree, a node of type \( N_{x_1} \) represents a negative value (0) for the variable \( x_1 \), and a node of type \( P_{x_1} \) represents a positive value (1) for this variable. Thus, given that the root has two children of types \( N_{x_1} \) and \( P_{x_1} \), the values 0 and 1 are assigned to \( x_1 \) (representing the quantifier \( \forall x_1 \)). Nodes of type \( N_{x_2} \) have one child of type either \( N_{x_2} \) or \( P_{x_2} \), and, therefore, either 0 or 1 is assigned to \( x_2 \) (representing the quantifier \( \exists x_2 \)). The same holds for nodes of type \( P_{x_2} \). The fourth level of the tree represents the quantifier \( \forall x_3 \). Note that in any XML tree \( T \) conforming to \( D \), there is no node in \( T \) reachable by following the path \( r.C.C. \).

In Figure 3.2, every path from the root \( r \) to a node of type either \( N_{x_3} \) or \( P_{x_3} \) represents a truth assignment for the variables \( x_1, x_2, x_3 \). For example, the path from the root to the node \( u \) represents the truth assignment \( \sigma_u \): \( \sigma_u(x_1) = 0 \), \( \sigma_u(x_2) = 1 \) and \( \sigma_u(x_3) = 0 \). To verify that all these assignments satisfy the formula \( x_1 \lor x_2 \lor \neg x_3 \), we use the set of constraint \( \Sigma_\theta \).

Next we prove that \( \theta \), defined in (3.3), is valid if and only if there is an XML tree \( T \) conforming to \( D_\theta \) and satisfying \( \Sigma_\theta \). We show only the “if” direction. The “only if” direction is similar.

Suppose that there is an XML tree \( T \) such that \( T \models (D_\theta, \Sigma_\theta) \). To prove that \( \theta \) is valid, it suffices to prove that each path from the root \( r \) to a node of type either \( N_{x_m} \) or \( P_{x_m} \) represents a truth assignment
are two nodes in an XML tree and let \( v \) be the node of type either \( N_{x_i} \) or \( P_{x_i} \) reachable from the root by following \( p \). We define the truth assignment \( \sigma_p \) as follows:

\[
\sigma_p(x_i) = \begin{cases} 
1 & \text{p contains a node of type } P_{x_i} \\
0 & \text{Otherwise.}
\end{cases}
\]

We have to prove that \( \sigma_p(C_i) = 1 \) for each \( i \in [1, n] \). Given that \( T \models D_{\theta} \), let \( v \) have as a child a node \( v' \) whose type is in \( tr(C_i) \). If the type of \( v' \) is \( x_i \), then given that \( T \models r.\vec{x}_j.x_j.\exists l \subseteq FK r.C.C.\exists l \) and that there exists no node in \( T \) reachable by following the path \( r.C.C \), \( p \) contains a node of type \( P_{x_j} \), and, therefore, \( \sigma_p(C_i) = 1 \) since \( \sigma_p(x_j) = 1 \). If the type of \( v' \) is \( \bar{x}_j \), then given that \( T \models r.\vec{x}_j.\exists l \subseteq FK r.C.C.\exists l \), \( p \) contains a node of type \( P_{x_j} \) and it does not contain a node of type \( P_{x_j} \), and, therefore, \( \sigma_p(C_i) = 1 \) since \( \sigma_p(\neg x_j) = 1 \). Thus, we conclude that \( \theta \) is valid. This concludes the proof of part b) of the theorem.

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], that only hold on certain sub-documents. Below we define relative keys and foreign keys. Recall that we use \( RC \) 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 \( x \).

We first define unary relative keys and foreign keys associated with element types. Let \( D = (E, A, P, R, \tau) \) be a DTD. A relative key is an expression \( \varphi \) of the form \( \tau(1, @l \rightarrow \tau_i) \), where \( @l \in R(\tau_1) \). It says that relative to each node \( x \) of element type \( \tau \), \( @l \) is a key for all the \( \tau_i \)-nodes that are descendants of \( x \). That is, if \( T \) conforms to \( D \), then \( T \models \varphi \) if

\[
\forall x \in ext(\tau) \forall y, z \in ext(\tau_1) \ ((x \prec y) \land (x \prec z) \land (y, @l = z, @l) \rightarrow y = z).
\]

A relative foreign key is an expression \( \varphi \) of the form \( \tau(\tau_1, @l_1 \subseteq FK \tau_2, @l_2) \), where \( @l_1 \in R(\tau_i) \), \( i = 1, 2 \). It indicates that for each \( x \) in \( ext(\tau) \), \( @l_1 \) is a foreign key of descendants of \( x \) of type \( \tau_1 \) that references a key \( @l_2 \) of \( \tau_2 \)-descendants of \( x \). That is, \( T \models \varphi \) if \( T \models \tau(\tau_1, @l_1 \rightarrow \tau_2) \) and \( T \) satisfies

\[
\forall x \in ext(\tau) \forall y_1 \in ext(\tau_1) \ ((x \prec y_1) \rightarrow \exists y_2 \in ext(\tau_2) \ ((x \prec y_2) \land y_1, @l_1 = y_2, @l_2)).
\]

Here \( \tau \) is called the context type of \( \varphi \). Note that absolute constraints are a special case of relative constraints when \( \tau = r \): i.e., \( r(\tau, @l \rightarrow \tau) \) is the usual absolute key. Thus, the consistency problem for multi-attribute relative constraints is undecidable [16], and hence we only consider unary relative constraints here.

Following the notations for \( AC \), we use \( RC_{FK} \) to denote the class of all unary relative keys and foreign keys defined for element types; \( RC_{FK,FK} \) means the primary key restriction. For example, the constraints given in Section 1 over the country/province/capital DTD can be expressed in \( RC_{FK,FK} \) as follows:

\[
\text{country, @name} \rightarrow \text{country}, \\
\text{country(province, @name} \rightarrow \text{province}), \\
\text{country(capital, @inProvince} \rightarrow \text{capital}), \\
\text{country(capital, @inProvince \subseteq FK province, @name}).
\]

A more general form of unary relative constraints is defined in terms of regular path expressions, along the same lines as \( AC_{FK} \). For example, the constraints given in Section 1 over the country/province/capital DTD are instances of this general form of relative constraints. Since \( RC_{FK,FK} \) constraints are a special case of the general regular-expression relative constraints (by substituting \( \vec{.} \tau \) for \( \tau \), the lower bound for \( SAT(AC_{FK,FK}) \) carries over to the consistency problem for unary relative constraints defined in terms of regular path expressions.

Recall that \( SAT(AC_{FK,FK}) \), the consistency problem for absolute unary constraints, is NP-complete. One would be tempted to think that \( SAT(RC_{FK,FK}) \), the consistency problems for relative unary constraints, is decidable as well. We next show, however, that there is an enormous difference between unary absolute constraints and unary relative constraints: while clearly \( SAT(RC_{FK,FK}) \) is r.e., it turns out that one cannot lower this bound.
Theorem 4.1. SAT($\mathcal{RC}_{K, FK}$) is undecidable.

Proof. We establish the undecidability of the consistency problem for unary relative keys and foreign keys by reduction from the Hilbert’s 10th problem [24]. To do this, we consider a variation of the Diophantine problem, referred as the positive Diophantine quadratic system problem. An instance of the problem is

\[ P_i(x_1, \ldots, x_k) = Q_1(x_1, \ldots, x_k) + c_1 \\
\]

\[ P_2(x_1, \ldots, x_k) = Q_2(x_1, \ldots, x_k) + c_2 \\
\]

\[ \vdots \\
\]

\[ P_n(x_1, \ldots, x_k) = Q_n(x_1, \ldots, x_k) + c_n \\
\]

where for $1 \leq i \leq n$, $P_i$ and $Q_i$ are polynomials in which all coefficients are positive integers; the degree of $P_i$ is at most 2 and the degree of each of its monomial is at least 1; each polynomial $Q_i$ satisfies the same condition, and each $c_i$ is a nonnegative integer constant. The problem is to determine, given any positive Diophantine quadratic system, whether it has a nonnegative integer solution.

The positive Diophantine quadratic system problem is undecidable. To prove this, it is straightforward to reduce to it another variation of the Diophantine problem, the positive Diophantine equation problem, which is known to be undecidable. An instance of this problem is $R(y) = S(y)$, where $R$ and $S$ are polynomials in which all coefficients are positive integers, and the problem is to determine whether it has a nonnegative integer solution.

In what follows, we show a reduction from the positive Diophantine quadratic system problem to SAT($\mathcal{RC}_{K, FK}$). More precisely, given a quadratic equation we show how to represent it by using a DTD and a set of constraints. It is straightforward to extend this representation to consider an arbitrary number of quadratic equations.

Consider the following equation:

\[ \sum_{i=1}^{m} a_i x_{a_i} + \sum_{i=m+1}^{n} a_i x_{\alpha_i} x_{\beta_i} = \sum_{i=1}^{p} b_i x_{\gamma_i} + \sum_{i=p+1}^{q} b_i x_{\gamma_i} x_{\delta_i} + a. \]  

(4.1)

In this equation, for every $i \in [1, n]$ and $j \in [m+1, n]$, $a_i$ is a positive integer and $x_{\alpha_i}, x_{\beta_j}$ represent variables, where $\alpha_i, \beta_j \in [1,k]$. Furthermore, for every $i \in [1, q]$ and $j \in [p+1, q]$, $b_i$ is a positive integer and $x_{\gamma_i}, x_{\delta_j}$ are variables, where $\gamma_i, \delta_j \in [1, k]$. Finally, $a$ is a nonnegative integer.

To code the previous equation, we need to define a DTD $D = (E, A, P, R, r)$ and a set of $\mathcal{RC}_{K, FK}$-constraints $\Sigma$. Here $D$ includes the following elements and attributes:

\[ E = \{ r, X, Y \} \cup \bigcup_{i=1}^{k} \{ \alpha_i \} \cup \bigcup_{i=1}^{n} \{ \alpha_i \} \cup \bigcup_{i=m+1}^{n} \{ \alpha_i, \beta_i, c_i, d_i, e_i \} \cup \bigcup_{i=1}^{q} \{ \gamma_i \} \cup \bigcup_{i=p+1}^{q} \{ \gamma_i, \delta_i, f_i, g_i, h_i \}, \]

$A = \{ @v \}$.

In this DTD, $r$ is the root. Intuitively, referring to an XML tree conforming to $D$, we use $|ext(n_i)|$ to code the value of the variable $x_i$, and we use $|ext(X)|$ and $|ext(Y)|$ to code the values of the left and the right hand sides of (4.1), respectively.

We define $P(r)$ as follows:

\[ P(r) = n_1, \ldots, n_k, \alpha_1^*, \ldots, \alpha_m^*, (\epsilon|\alpha_{m+1}|), \ldots, (\epsilon|\alpha_n|), \gamma_1^*, \ldots, \gamma_p^*, (\epsilon|\gamma_{p+1}|), \ldots, (\epsilon|\gamma_q|), Y, \ldots, Y \]

where $a_i$, $b_i$, and $p$ is the number of times $X, Y$, and $\epsilon$ are repeated, respectively.

We define the function $P$ on $\alpha_i$ and $\beta_i$ as follows:

\[
P(\alpha_i) = \underbrace{X, \ldots, X}_{a_i \text{ times}}^{} \quad 1 \leq i \leq m
\]

\[
P(\alpha_i) = \underbrace{(\beta_i, c_i, c_i, X, \ldots, X)^*}_{a_i \text{ times}}^{} \quad m+1 \leq i \leq n
\]

\[
P(\gamma_i) = \underbrace{Y, \ldots, Y}_{b_i \text{ times}}^{} \quad 1 \leq i \leq p
\]

\[
P(\gamma_i) = \underbrace{(\delta_i, f_i, f_i, Y, \ldots, Y)^*}_{b_i \text{ times}}^{} \quad p+1 \leq i \leq q
\]
To code (4.1) we need to capture the multiplication operator. To do this, we use $\alpha'_i$ and $\gamma'_i$:

$$P(\alpha'_i) = (\beta_i, d_i, e_i)^*, (\alpha'_i|c_i, e_i)^* \quad m + 1 \leq i \leq n$$

$$P(\gamma'_i) = (\beta_i, g_i, h_i)^*, (\gamma'_i|f_i, h_i)^* \quad p + 1 \leq i \leq q$$

For all the other element types $\tau$ in $D$, $P(\tau)$ is defined as $\epsilon$:

$$P(\beta_i) = \epsilon \quad m + 1 \leq i \leq n$$

$$P(c_i) = \epsilon \quad m + 1 \leq i \leq n$$

$$P(d_i) = \epsilon \quad m + 1 \leq i \leq n$$

$$P(g_i) = \epsilon \quad p + 1 \leq i \leq q$$

$$P(h_i) = \epsilon \quad p + 1 \leq i \leq q$$

Finally, we include the following attributes:

$$R(r) = \emptyset$$

$$R(\alpha_i) = \emptyset \quad 1 \leq i \leq k$$

$$R(\delta_i) = \emptyset \quad 1 \leq i \leq k$$

$$R(\gamma_i) = \emptyset \quad 1 \leq i \leq q$$

To ensure that XML documents that conform to $D$ indeed code equation (4.1) we need to define a set of $\mathcal{RC}_{K,FK}$-constraints $\Sigma$. This set contains the following absolute keys:

$$\tau(X, @v \rightarrow X)$$

$$\tau(\alpha_i, @v \rightarrow \alpha_i) \quad \text{for every} \ 1 \leq i \leq n$$

$$\tau(\beta_i, @v \rightarrow \beta_i) \quad \text{for every} \ m + 1 \leq i \leq n$$

$$\tau(\delta_i, @v \rightarrow \delta_i) \quad \text{for every} \ p + 1 \leq i \leq q$$

$$\tau(\gamma_i, @v \rightarrow \gamma_i) \quad \text{for every} \ p + 1 \leq i \leq q$$

$$\tau(\eta_i, @v \rightarrow \eta_i) \quad \text{for every} \ 1 \leq i \leq k$$

$\Sigma$ contains the following absolute foreign keys:

$$r(X, @v \subseteq_{FK} Y, @v)$$

$$r(\alpha_i, @v \subseteq_{FK} \alpha_i, @v)$$

$$r(\beta_i, @v \subseteq_{FK} \beta_i, @v)$$

$$r(\delta_i, @v \subseteq_{FK} \delta_i, @v)$$

$$r(\gamma_i, @v \subseteq_{FK} \gamma_i, @v)$$

$$r(\eta_i, @v \subseteq_{FK} \eta_i, @v)$$

Finally, $\Sigma$ contains the following relative foreign keys:

$$\alpha_i(\beta_i, @v \subseteq_{FK} \beta_i, @v)$$

$$\alpha'_i(\beta_i, @v \subseteq_{FK} \beta_i, @v)$$

$$\gamma_i(\beta_i, @v \subseteq_{FK} \beta_i, @v)$$

$$\gamma'_i(\beta_i, @v \subseteq_{FK} \beta_i, @v)$$

We show next that there is an XML tree $T$ such that $T \models (D, \Sigma)$ if and only if there exists a nonnegative integer solution for (4.1). To do this, we prove that every XML tree $T$ satisfying $D$ and $\Sigma$ codifies equation (4.1). More precisely, if the value of every variable $x_i$ is $v_i$ and $|ext(\eta_i)| = v_i$, for $i \in [1,k]$, then

$$|ext(X)| = \sum_{i=1}^{m} a_i v_{\alpha_i} + \sum_{i=m+1}^{n} a_i v_{\alpha_i} v_{\beta_i}$$

$$|ext(Y)| = \sum_{i=1}^{p} b_i v_{\gamma_i} + \sum_{i=p+1}^{q} b_i v_{\gamma_i} v_{\delta_i} + o.$$
Analogously, to show that (4.3) holds, we have to prove that the number of \( Y \)-nodes that are children of some node of type \( \gamma_i \) (\( i \in [1, q] \)) is equal to the \( i \)-th term of (4.3). We will only consider here the case of \( X \)-nodes, being the other case similar.

First, let \( i \in [1, m] \) and \( s \) be the value of \( \alpha_i \) in (4.2). Given that \( r(n_s, \exists v \subseteq_{FK} \alpha_i, \exists v) \), \( r(\alpha_i, \exists v \subseteq_{FK} n_s, \exists v) \) are in \( \Sigma \), by the definition of \( P(\alpha_i) \) the total number of \( X \)-nodes that are children of a node of type \( \alpha_i \) is equal to \( a_i v_s v_l \). Second, let \( i \in [m + 1, n] \) and \( s, t \) be the values of \( \alpha_i \) and \( \beta_i \) in (4.1), respectively. Next we prove that \( |\{ x \mid x \text{ is an } X \text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i \}| = a_i v_s v_l \).

Given that \( r(n_s, \exists v \subseteq_{FK} \alpha_i, \exists v), r(\alpha_i, \exists v \subseteq_{FK} n_s, \exists v) \) are in \( \Sigma \), \( |\text{ext}(\alpha_i)\text{ in } T \) is equal to \( |\text{ext}(n_s)| = v_s \). Thus, in \( T \) there are exactly \( v_s \) nodes of type \( \alpha_i \), each of them having exactly one child of type \( \alpha'_i \). Hence, there are exactly \( v_s \) nodes of type \( \alpha'_i \), being the last one of the form shown in Figure 4.1 (see node \( r_4 \)). By the definition of \( P(\alpha'_i) \), \( |\{ x \mid x \text{ is a child of } r_4 \text{ of type } \alpha_i \}| = |\{ x \mid x \text{ is a child of } r_4 \text{ of type } \alpha_i \}| \). Given that \( r(n_t, \exists v \subseteq_{FK} \alpha_i, \exists v), r(\alpha_i, \exists v \subseteq_{FK} n_t, \exists v) \) are in \( \Sigma \) and that every node of type \( \alpha_i \) in \( T \) is a child of \( r_4 \), \( |\{ x \mid x \text{ is a child of } r_4 \text{ of type } \alpha_i \}| = |\text{ext}(n_t)| = v_t \). Thus, since \( r_4 \) is a node of type \( \alpha'_i \) and \( \alpha'_i(\beta_i, \exists v \subseteq_{FK} c_i, \exists v) \), \( \alpha'_i(\beta_i, \exists v \subseteq_{FK} c_i, \exists v) \) are in \( \Sigma \), \( |\{ x \mid x \text{ is a child of } r_4 \text{ of type } \beta_i \}| = |\text{ext}(n_t)| = v_t \). In addition, by the definition of \( P(\alpha'_i) \), the number of children of \( r_4 \) of type \( d_i \) is \( 2 v_t \).

Given that \( r_3 \) is a node of type \( \alpha_i \) and \( \alpha_i(\beta_i, \exists v \subseteq_{FK} d_i, \exists v), \alpha_i(d_i, \exists v \subseteq_{FK} \beta_i, \exists v) \) are in \( \Sigma \), \( |\{ x \mid x \text{ is a child of } r_3 \text{ of type } \beta_i \}| = v_t \), since there are exactly \( 2 v_t \) descendants of \( r_3 \) of type \( d_i \) and \( v_t \) children of \( r_3 \) of type \( \beta_i \). Furthermore, by the definition of \( P(\alpha_i) \), the number of children of \( r_3 \) of type \( X \) is \( a_i v_t \) and the number of children of \( r_3 \) of type \( c_i \) is \( 2 v_t \). We can use the same argument to prove that the number of children of \( r_2 \) of types \( \beta_i \) and \( d_i \) are \( v_t \) and \( 2 v_t \), respectively. Thus, the number of children of \( r_1 \) of type \( X \) is \( a_i v_t \) and the number of descendants of \( r_1 \) of type \( X \) is \( 2 a_i v_t \). If we continue with this process we can prove, by induction, that the number of \( X \)-nodes in \( T \) that are children of some node of type \( \alpha_i \) is \( v_t a_i v_t \), since there are \( v_t \) nodes of type \( \alpha_i \) in \( T \). This conclude the proof, since \( |\{ x \mid x \text{ is an } X \text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i \}| = a_i v_t v_t \).

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

**Corollary 4.2.** \( \text{SAT}(\mathbb{R}_{FK}^{FK}) \), the restriction of \( \text{SAT}(\mathbb{R}_{FK}^{FK}) \) to primary keys, is undecidable. \( \blacksquare \)

5. Extended DTDs. In this section, we consider a slight extension of DTDs which captures unranked tree automata. An *extended DTD* \([32, 29]\) \( ED \) is a tuple \( (D', f, E) \), where \( D' = (E', A', P', R', r') \) is a DTD, \( E \) is a finite set of element types such that \( E \cap E' = \emptyset \) and \( f \) is a surjective mapping \( f: E' \rightarrow E \) such that for every \( \tau_1, \tau_2 \in E' \), we have that \( R'(\tau_1) = R'(\tau_2) \) if \( f(\tau_1) = f(\tau_2) \). We say that a tree \( T \) conforms to \( ED \) if there exists a tree \( T' \) that conforms to \( D' \) such that \( T = f(T') \), that is, \( T \) can be obtained by replacing each label \( \tau \) in \( T' \) by \( f(\tau) \). Extended DTDs support a subtyping mechanism (specialization), and have proven useful in data migration and integration, among other things (see, e.g., \([29]\) for examples of extended DTDs and their applications in data integration). It is also known that extended DTDs capture precisely MSO definable trees and the regular tree languages of finite unranked trees \([26, 29]\).

A constraint \( \varphi \) is said to be defined over extended DTD \( ED \) if every element type \( \tau \) mentioned in \( \varphi \) is in \( E \).

The consistency problem for extended DTDs is defined exactly as in the case of DTDs: Given a specification \( (ED, \Sigma) \), the problem is to verify whether there exists a tree \( T \) conforming to \( ED \) and satisfying \( \Sigma \).
Next we show that the consistency problem for extended DTDs can be efficiently reduced to the consistency problem for DTDs. Let $ED = (D', f, E)$ be an extended DTD, where $D' = (E', A', P', R', r')$, and $\Sigma$ be either a set of $AC_{K,FK}^{*,*}$-constraints over $ED$ or a set of $AC_{K,FK}^{reg}$-constraints over $ED$. We consider here only absolute constraints since, by Theorem 4.1, the consistency problem for extended DTDs and relative constraints is already undecidable for unary keys and foreign keys without regular expressions. We define DTD $D(ED) = (E_{ED}, A', P_{ED}, R_{ED}, r')$ as follows. Let $E_{ED} = E' \cup E$ and

$$P_{ED}(\tau) = \begin{cases} f(\tau), P'(\tau) & \tau \in E' \\ \epsilon & \tau \in E \end{cases}$$

$$R_{ED}(\tau) = \begin{cases} \emptyset & \tau \in E' \\ R'(\tau') & \tau \in E \text{ and } f(\tau') = \tau \end{cases}$$

Notice that $R_{ED}$ is well defined since $f$ is a surjective mapping such that $R'(\tau_1) = R'(\tau_2)$ whenever $f(\tau_1) = f(\tau_2)$. Moreover, we define a set of keys and foreign keys $\Gamma(ED, \Sigma)$ over DTD $D(ED)$ as follows. If $\Sigma$ is a set of $AC_{K,FK}^{*,*}$-constraints, then $\Gamma(ED, \Sigma) = \Sigma$. Otherwise, $\Sigma$ is a set of $AC_{K,FK}^{reg}$-constraints and for every $\varphi \in \Sigma$ we have the following constraint $\psi$ in $\Gamma(ED, \Sigma)$. For every element type $\tau \in E$, define the image of substitution $h$ on $\tau$ as $h(\tau) = (\tau_1 \cup \cdots \cup \tau_n)$, where $\{\tau_1, \ldots, \tau_n\}$ is the set of element types $\tau' \in E'$ such that $f(\tau') = \tau$. If $\varphi$ is a key $\beta.\tau.\@l \rightarrow \beta.\tau$, then $\psi = h(\beta.\tau.\@l) \rightarrow h(\beta.\tau.\@l)$. If $\varphi$ is a foreign key $\beta.\tau_1.\@l_1 \subseteq FK.\beta_2.\tau_2.\@l_2$, then $\psi = h(\beta.\tau_1.\@l_1 \subseteq FK.\beta_2.\tau_2.\@l_2)$. The following simple lemma shows that the consistency problems for $(ED, \Sigma)$ and $(D(ED), \Gamma(ED, \Sigma))$ are equivalent.

**Lemma 5.1.** Let $ED = (D', f, E)$ be an extended DTD and $\Sigma$ either a set of $AC_{K,FK}^{*,*}$-constraints over $ED$ or a set of $AC_{K,FK}^{reg}$-constraints over $ED$. Then $(ED, \Sigma)$ is consistent if and only if $(D(ED), \Gamma(ED, \Sigma))$ is consistent.

We note that if $C$ is one of $AC_{FK,FK}^{*}, AC_{K,FK}^{*}, AC_{FK,FK}^{1,*}$, and $AC_{K,FK}^{reg}$, and $\Sigma$ is a set of $C$-constraints over an extended DTD $ED$, then $\Gamma(ED, \Sigma)$ is a set of $C$-constraints over $D(ED)$. Thus, given that $D(ED)$ and $\Gamma(ED, \Sigma)$ can be constructed in polynomial time from $(ED, \Sigma)$, we obtain the following corollary from the previous lemma, Theorem 4.7 and Corollary 4.8 in [16], Corollary 3.5 and Theorem 3.7.

**Corollary 5.2.**

a) The consistency problem for extended DTDs and $AC_{K,FK}^{*}$-constraints ($AC_{FK,FK}^{*}$-constraints) is NP-complete.

b) The consistency problem for extended DTDs and $AC_{FK,FK}^{1,*}$-constraints is NP-hard, and can be solved in NEXPTIME.

c) The consistency problem for extended DTDs and $AC_{K,FK}^{reg}$-constraints is PSPACE-hard, and can be solved in 2-NEXPTIME.

**6. Conclusion.** We have studied the problem of statically checking XML specifications, which may include various schema definitions as well as integrity constraints. As observed earlier, such static validation

<table>
<thead>
<tr>
<th>Class</th>
<th>$AC_{K,FK}^{<em>,</em>}$ [16]</th>
<th>$AC_{F_{n,FK}}^{*,1}$</th>
<th>$AC_{K,FK}^{reg}$</th>
<th>$AC_{K,FK}$ [16]</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>multi-attribute keys and foreign keys</td>
<td>primary keys and attributes, unary foreign keys</td>
<td>unary regular path constraints (keys, foreign keys)</td>
<td>unary keys and foreign keys</td>
</tr>
<tr>
<td>Upper bound</td>
<td>undecidable</td>
<td>NEXPTIME</td>
<td>2-NEXPTIME</td>
<td>NP</td>
</tr>
<tr>
<td>Lower bound</td>
<td>undecidable</td>
<td>NP</td>
<td>PSPACE</td>
<td>NP</td>
</tr>
</tbody>
</table>

*Fig. 6.1. Complexity of the consistency problem for absolute constraints*

<table>
<thead>
<tr>
<th>Class</th>
<th>$RC_{K,FK}^{<em>,</em>}$ [16]</th>
<th>$RC_{K,FK}$</th>
<th>$RC_{F_{n,FK}}^{*,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>multi-attribute keys and foreign keys</td>
<td>unary keys and foreign keys</td>
<td>primary unary keys and foreign keys</td>
</tr>
<tr>
<td>Lower bound</td>
<td>undecidable</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

*Fig. 6.2. Complexity of the consistency problem for relative constraints*
is quite desirable as an alternative to dynamic checking, which would attempt to validate each document; indeed, in the case of repeated failures, one does not know whether the problems lies in the documents or in the specification. 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 6.1, 6.2 (we also included the main results from [16] in those figures for completeness). When one deals with absolute constraints, which hold in an entire document, the general consistency problem is undecidable. It is solvable in NEXPTIME 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.

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, SAT(\(A\&\&_FK\)) was proved to be equivalent to a problem related to Diophantine equations whose exact complexity remains unknown. In the case of SAT(\(A^C\&\&_K\)), 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 [37], and more complex schema specifications such as XML Schema [38] and the type system of XQuery [39]. Our lower bounds apply to those settings, but it is open whether upper bounds remain intact.

Acknowledgments. We are grateful to anonymous referees for their comments.

Most of this work was done while Arenas and Libkin were at the University of Toronto, supported by grants from NSERC, CITO, and PREA grants. M. Arenas was also supported by FONDECYT grant 1050701, W. Fan by NSF Career Award IIS-0093168, and grants EPSRC GR/S63205/01, EPSRC GR/T27433/01 and NSFC 60228006, and L. Libkin by the European Commission Marie Curie Excellence grant MEXC-CT-2005-024502 and EPSRC grant E005039.

REFERENCES


