SIAM J. COMPUT.
Vol. 38, No. 3, pp. 841–880

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 document type definition (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 [W. Fan and L. Libkin, *J. ACM*, 49 (2002), pp. 368–406] that for general keys, 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 hold only in a part of the document. For absolute constraints, we prove decidability and establish complexity bounds for primary multiattribute 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.

Key words. XML, keys and foreign keys, document type definition, consistency, complexity

AMS subject classifications. 03D05, 68P15

DOI. 10.1137/050646895

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 definitions), and constraints are typically natural analogues 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 specifications statically, at compile time. Invalid XML specifications are likely to be more common than invalid

*Received by the editors December 7, 2005; accepted for publication (in revised form) January 17, 2008; published electronically June 6, 2008. An extended abstract was presented at the 21st ACM Symposium on Principles of Database Systems (PODS 2002).

http://www.siam.org/journals/sicomp/38-3/64689.html

†Escuela de Ingenieria, Pontificia Universidad Católica de Chile, Santiago 22, Chile (marenas@ing.puc.cl). This author's research was partially supported by grants from NSERC, CITO, and PREA, as well as by FONDECYT grant 1050701.

‡School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK (wenfei@inf.ed.ac.uk, libkin@inf.ed.ac.uk). The second author's research was supported by NSF Career Award IIS-0093168 and grants EPSRC GR/S63205/01, EPSRC GR/T27433/01, and NSFC 60228006. The third author's research was partially supported by grants from NSERC, CITO, and PREA and by the European Commission Marie Curie Excellence grant MEXC-CT-2005-024502 and EPSRC grant E005039.

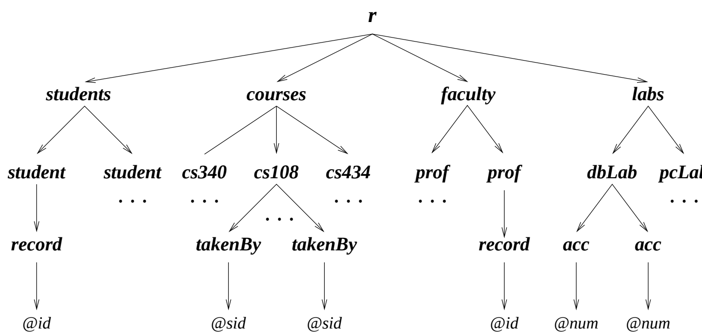


FIG. 1.1. *An XML document.*

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 revealed only 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 hold only 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 are 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:

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

Here we omit the descriptions of elements whose type is string (PCDATA). Assume that each *record* element has an attribute @*id*, each *takenBy* has an attribute @*sid* (for student id), and each *acc* has an attribute @*num*. One may impose the following constraints over the DTD of that document:

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

Here "_" is a wild card 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 be taken only 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:

$$r.faculty.prof.record.@id \subseteq_{FK} r.\_^*.dbLab.acc.@num.$$

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 the requirement that at least one professor be present in the document. Thus no XML document both conforms to the DTD and satisfies all of 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 hold only 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:

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

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

Now suppose that 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

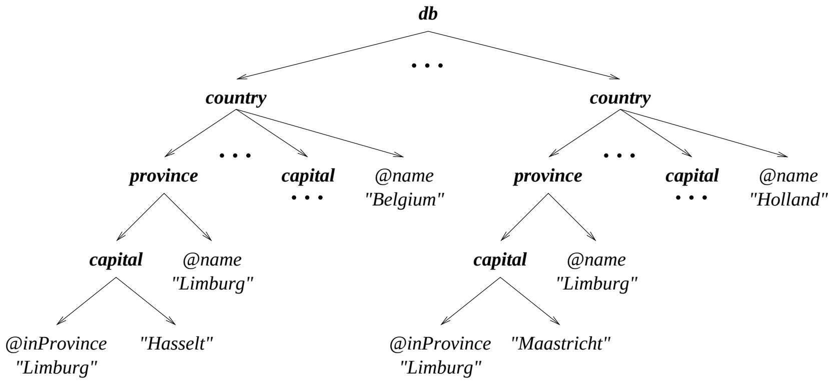


Fig. 1.2. *An XML document storing information about countries and their administrative subdivisions.*

Union. To overcome this problem, we define *@name* to be a key for *province relative* to a country; indeed, it is extremely unlikely that two provinces of the same country would have the same name. Thus, our constraints are

$$\begin{array}{rcl} \mathit{country}.@\mathit{name} & \to & \mathit{country}, \\ \mathit{country}(\mathit{province}.@\mathit{name} & \to & \mathit{province}), \\ \mathit{country}(\_^*.\mathit{capital}.@\mathit{inProvince} & \to & \_^*.\mathit{capital}), \\ \mathit{country}(\_^*.\mathit{capital}.@\mathit{inProvince} & \subseteq_{FK} & \_^*.\mathit{province}.@\mathit{name}). \end{array}$$

The first constraint is like those we have encountered before: it is an *absolute* key, which applies to the entire document. The rest are *relative constraints* which are specified for subdocuments 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 a foreign key referring to *@name* of *province* elements in the same subdocument. The foreign keys assure that for each *capital* element in a *country* element (subdocument) 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 subdocument 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 subdocument 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 $\mathit{country}(\_^*.\mathit{capital}.@\mathit{inProvince} \to \_^*.\mathit{capital})$ with two keys: $\mathit{country}(\mathit{capital}.@\mathit{inProvince} \to \mathit{capital})$ and $\mathit{country}(\mathit{province}.\mathit{capital}.@\mathit{inProvince} \to$

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 the case of absolute constraints, the results of [16] consider either single attributes or multiattribute sets for both keys and foreign keys and thus say nothing about the intermediate case in which only keys are allowed to be multiattribute. 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] and 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 investigated only in [16]. However, [16] did not consider relative constraints, constraints defined with regular expressions, and the class of multiattribute keys and unary foreign keys. Other constraints for semistructured 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, absolute keys, and foreign keys for XML. Section 3 studies the class of absolute multiattribute 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 $\mathtt{S}$ 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 *is defined to be* $D = (E, A, P, R, r)$*, where:*

- $E \subseteq El$ *is a finite set of* element types*;*
- $A \subseteq Att$ *is a finite set of* attributes*;*
- P *is a mapping from* E *to* element type definitions*: Given* $\tau \in E$*,* $P(\tau) = \mathtt{S}$ *or* $P(\tau)$ *is a regular expression* α *defined as follows:*

$$\alpha ::= \epsilon \mid \tau' \mid \alpha|\alpha \mid \alpha, \alpha \mid \alpha^*,$$

where $\mathtt{S}$ *denotes the* string *type,* $\tau' \in E$*,* ϵ *is the empty word, and "|", ",", and "*" denote union, concatenation, and the Kleene closure, respectively;*
- R *is a mapping from* E *to the powerset of* A*. If* $@l \in R(\tau)$*, we say that* $@l$ *is* defined for τ*;*
- $r \in E$ *and is called* the element type of the root.

We normally denote element types by τ and assume that $R(r) = \emptyset$ and r does not appear in $P(\tau)$ for any $\tau \in E$. We also assume that each τ in $E \setminus \{r\}$ is *connected to* r; i.e., either τ appears in $P(r)$, or it appears in $P(\tau')$ for some τ' that is connected to r. In this paper we also use the following shorthand for regular expressions: α^+ for (α, α^*) 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 $\mathtt{S}$ 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 $E = \{db, country, province, capital, city\}$, $A = \{@name, @inProvince\}$, $r = db$, and P, R are as follows:

$$\begin{array}{lcllcl}
P(db) & = & country^+, & R(db) & = & \emptyset, \\
P(country) & = & (province^+, capital^+), & R(country) & = & \{@name\}, \\
P(province) & = & (capital, city^*), & R(province) & = & \{@name\}, \\
P(capital) & = & \mathtt{S}, & R(capital) & = & \{@inProvince\}, \\
P(city) & = & \mathtt{S}, & R(city) & = & \emptyset.
\end{array}$$

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* $(V, lab, ele, att, root)$*, where*

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

The parent-child edge relation on V*,* $\{(v_1, v_2) \mid v_2$ *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

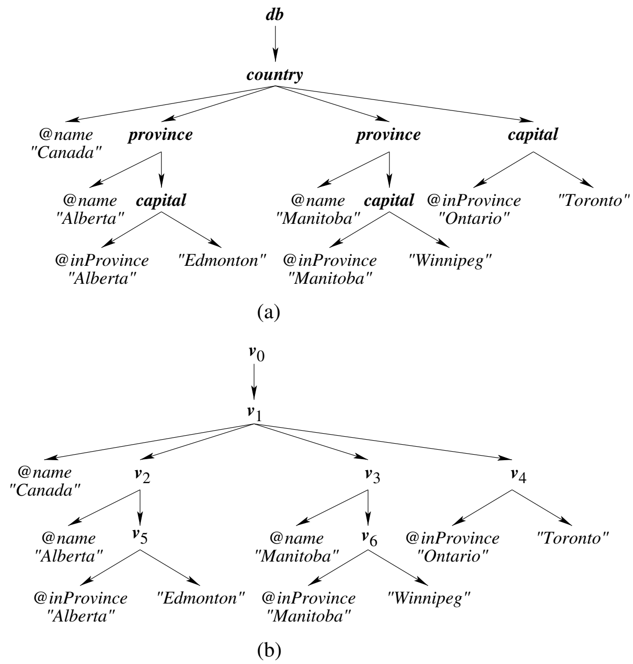


FIG. 2.1. *An XML document represented as a tree.*

unique node labeled with r. If a node x is labeled τ in E, then function ele defines the children of x and function att defines the attributes of x. The children of x are ordered, and their labels observe the regular expression $P(\tau)$. In contrast, its attributes are unordered and are identified by their labels (names).

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 $T = (V,\ lab,\ ele,\ att,\ v_0)$ representing this document. In this tree, $V = \{v_i \mid i \in [0, 6]\}$ and lab is defined as:

$$\begin{array}{llll} lab(v_0) = db, & lab(v_2) = province, & lab(v_4) = capital, & lab(v_6) = capital. \\ lab(v_1) = country, & lab(v_3) = province, & lab(v_5) = capital, & \end{array}$$

Furthermore, function ele is defined as:

$$\begin{array}{llll} ele(v_0) = [v_1], & ele(v_2) = [v_5], & ele(v_4) = [\text{Toronto}], & ele(v_6) = [\text{Winnipeg}]. \\ ele(v_1) = [v_2, v_3, v_4], & ele(v_3) = [v_6], & ele(v_5) = [\text{Edmonton}], & \end{array}$$

Finally, function att is defined as:

$$\begin{array}{ll} att(v_1, @name) = \text{Canada}, & att(v_4, @inProvince) = \text{Ontario}, \\ att(v_2, @name) = \text{Alberta}, & att(v_5, @inProvince) = \text{Alberta}, \\ att(v_3, @name) = \text{Manitoba}, & att(v_6, @inProvince) = \text{Manitoba}. \end{array}$$

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

We also use the following notations. Referring to an XML tree T, if x is a τ-element in T and $@l$ is an attribute in $R(\tau)$, then $x.@l$ denotes the $@l$-attribute value of x, i.e., $x.@l = att(x, @l)$. If X is a list $[@l_1, \ldots, @l_n]$ of attributes in $R(\tau)$, then $x[X] = [x.@l_1, \ldots, x.@l_n]$. For any element type $\tau \in E$, $ext(\tau)$ denotes the set of all of the τ-elements in T. For any $@l \in R(\tau)$, $values(\tau.@l)$ denotes $\{x.@l \mid x \in ext(\tau)\}$, the set of all of the $@l$-attribute values of τ-nodes. We write $|S|$ for the cardinality of a set S. Given a DTD D and a set Σ of constraints, we also use $|D|$ and $|\Sigma|$ to denote their sizes, respectively.

Given a DTD $D = (E, A, P, R, r)$ and element types $\tau, \tau' \in E$, a string $\tau_1.\tau_2.\ldots.\tau_n$ over E is a *path in* D *from* τ *to* τ' if $\tau_1 = \tau$, $\tau_n = \tau'$, and for each $i \in [2, n]$, τ_i is a symbol in the alphabet of $P(\tau_{i-1})$. Moreover, $paths(D) = \{p \mid$ there is $\tau \in E$ such that p is a path in D from r to $\tau\}$. We say that a DTD is *nonrecursive* if $paths(D)$ is finite and recursive otherwise. We also say that D is a *no-star* DTD if the Kleene star does not occur in any regular expression $P(\tau)$ (note that this is a stronger restriction than being $*$-free: a regular expression without the Kleene star yields a finite language, while the language of a $*$-free regular expression may still be infinite as it allows boolean operators including complement).

2.2. Keys and foreign keys. We consider two forms of constraints for XML: *absolute constraints* that hold on the entire document, denoted by $\mathcal{AC}$, and *relative constraints* that hold on certain subdocuments, denoted by $\mathcal{RC}$. Below we define absolute keys and foreign keys, and we shall define relative constraints in section 4. The constraints given in section 1 are instances of absolute constraints and relative constraints.

Regular expression constraints. To capture the hierarchical nature of XML data, absolute constraints, in their general form, are defined on a collection of elements identified by a regular path expression. It is common to find path expressions in specification and query languages for XML (e.g., XML Schema [38], XQuery [39], and XSL [40]). We define a *regular (path) expression* over a set of element types E as follows:

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

where ϵ denotes the empty word, τ is an element type in E, and ".", "$\cup$", and "$*$" denote concatenation, union, and Kleene closure, respectively. A regular expression defines a language over the alphabet E, which will be denoted by β as well. Given a DTD $D = (E, A, P, R, r)$ and a regular expression β over E, we say that β is a *regular (path) expression over* D if β is of the form $r.\beta'$, where β' does not include r. In this section, we use "$\_$" as shorthand for $E \setminus \{r\}$.

Recall that a path in a DTD is a list of E symbols, that is, a string in E^*. Given an XML tree $T = (V, lab, ele, att, root)$, a pair of nodes x, y in T, with y a descendant of x, and a path $w = \tau_1.\ldots.\tau_n$ over E, we say that w *is a path from* x *to* y if there exists a sequence of nodes $v_1, \ldots, v_n$ in T such that (1) $v_1 = x$ and $v_n = y$, (2) v_{i+1} is a child of v_i in T, for every $i \in [1, n-1]$, and (3) $lab(v_i) = \tau_i$, for every $i \in [1, n]$. Any pair of nodes x, y in an XML tree T with y a descendant of x uniquely determines the path, denoted by $\rho(x, y)$, from x to y. We say that y is *reachable* from x by following a regular expression β over D, denoted by $T \models \beta(x,\ y)$, iff $\rho(x, y) \in \beta$. For any fixed T, let $nodes(\beta)$ stand for the set of nodes reachable from the root by following the regular expression β: $nodes(\beta) = \{y \mid T \models \beta(root,\ y)\}$. Note that, for any element type $\tau \in E \setminus \{r\}$, $nodes(r.\_^*.\tau) = ext(\tau)$.

We now define unary XML keys and foreign keys with regular path expressions. Let DTD $D = (E, A, P, R, r)$.

- A *key* over D is an expression φ of the form $\beta.\tau[X] \to \beta.\tau$, where

– $\tau \in E$;
– X is a nonempty set of attributes in $R(\tau)$; and
– β is a regular expression over D.

For any XML tree T that conforms to D, the tree T satisfies φ, denoted by $T \models \varphi$, if

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

- A *foreign key* over D is an expression φ of the form $\beta_1.\tau_1[X] \subseteq_{FK} \beta_2.\tau_2[Y]$, where
  – $\tau_i \in E$ for $i = 1, 2$;
  – β_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

$$\forall\, x \in nodes(\beta_1.\tau_1)\ \exists\, y \in nodes(\beta_2.\tau_2)\ \big(x[X] = y[Y]\big).$$

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 *multiattribute* 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, $|X| = 1$ ($|X| = |Y| = 1$) 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 study only the consistency problem for unary constraints defined with regular expressions. We denote this class of constraints by $\mathcal{AC}^{reg}_{K,FK}$, 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 $\mathcal{AC}^{reg}_{K,FK}$.

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 φ over a DTD $D = (E, A, P, R, r)$ has one of the following forms:

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

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

- *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

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

That is, $\tau[X] \rightarrow \tau$ says that the X-attribute values of a τ-element uniquely identify the element in $ext(\tau)$. Furthermore, $\tau_1[X] \subseteq_{FK} \tau_2[Y]$ says that the list of X-attribute values of every τ_1-node in T must match the list of Y-attribute values of some τ_2-node in T and the Y-attribute values of a τ_2-element uniquely identify the element in $ext(\tau_2)$.

Table 2.1
Notation summary.

Notation	Meaning
$\mathcal{AC}^{*,*}_{K,FK}$	Multiattribute keys and foreign keys
$\mathcal{AC}^{*,1}_{PK,FK}$	Multiattribute primary keys, unary foreign keys
$\mathcal{AC}_{K,FK}$	Unary keys and foreign keys
$\mathcal{AC}_{PK,FK}$	Primary unary keys and unary foreign keys
$\mathcal{AC}^{reg}_{K,FK}$	Regular expression unary keys and foreign keys

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

As for the case of regular expression constraints, an $\mathcal{AC}^{*,*}_{K,FK}$-constraint is generally referred to as a *multiattribute* 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 multiattribute, and "1" means unary. When both superscripts are left out, we mean that both keys and foreign keys are unary. We shall be dealing with the following subclasses of $\mathcal{AC}^{*,*}_{K,FK}$: $\mathcal{AC}^{*,1}_{K,FK}$ denotes the class of multiattribute keys and unary foreign keys; $\mathcal{AC}^{*,1}_{PK,FK}$ is the class of primary multiattribute keys and unary foreign keys; $\mathcal{AC}_{K,FK}$ is the class of unary keys and unary foreign keys; and $\mathcal{AC}_{PK,FK}$ 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 $\mathcal{AC}^{*,*}_{K,FK}$ to unary keys and multiattribute foreign keys ($\mathcal{AC}^{1,*}_{K,FK}$) does not make sense.

For easy reference, in Table 2.1 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* $\mathsf{SAT}(\mathcal{C})$ as follows:

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

It is known [16] that $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable, but $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ and $\mathsf{SAT}(\mathcal{AC}_{PK,FK})$ are NP-complete. Nothing was known, however, about $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$, where only keys are allowed to be multiattribute, or about $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, 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 $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ and $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ and the consistency problems for absolute primary multiattribute keys and unary foreign keys and for regular expression unary keys and unary foreign keys.

3.1. Consistency of multiattribute keys.

We know that $\mathsf{SAT}(\mathcal{AC}_{K,FK})$, the consistency problem for unary absolute keys and foreign keys, is NP-complete [16]. In contrast, $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable [16]. This leaves a large gap: namely, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$, where only keys are allowed to be multiattribute.

The reason for the undecidability of $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is that the implication problem for functional and inclusion dependencies can be reduced to it [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 multiattribute keys and unary foreign keys.

The problem we resolve here is $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$: the consistency problem for *primary* multiattribute keys and unary foreign keys. Recall that a set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints is said to be *primary* if for each element type τ there is at most one key in Σ defined for τ-elements (including key dependencies defined by foreign key constraints). Even dealing with this version of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$ 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 $x \leq y \cdot z$ among variables. It is therefore NP-hard, and [21] proved an NEXPTIME upper bound for PDE. The exact complexity of the problem remains unknown.

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

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

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

(a) *A reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE.* We first define a class of simplified DTDs called *narrow DTDs*, and we explain how to reduce the consistency problem for $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over arbitrary DTDs to that over narrow DTDs. Then we show how to encode the consistency problem for narrow DTDs and $\mathcal{AC}^{*,1}_{PK,FK}$-constraints by a prequadratic Diophantine system.

We start by explaining the process of narrowing the DTDs. Intuitively, we replace long "horizontal" regular expressions in $P(\tau)$ by shorter ones. Formally, consider a DTD $D = (E, A, P, R, r)$. D is basically an extended regular grammar (cf. [7, 25]); for each $\tau \in E$, $P(\tau)$ is a regular expression α, and, thus, $\tau \to \alpha$ can be viewed as the production rule for τ. We rewrite the regular expression by introducing a set N of new element types (nonterminals) such that the production rules of the new DTD have one of the following forms:

$$\tau \to \tau_1,\ \tau_2, \qquad \tau \to \tau_1 \mid \tau_2, \qquad \tau \to \tau_1^*, \qquad \tau \to \tau', \qquad \tau \to \ \mathtt{S}, \qquad \tau \to \epsilon,$$

where τ, τ_1, τ_2 are element types in $E \cup N$, $\tau' \in E$, $\mathtt{S}$ is the string type, and ϵ denotes the empty word. More specifically, we conduct the following "narrowing" process on the production rule $\tau \to \alpha$:

- If $\alpha = (\alpha_1, \alpha_2)$, then we introduce two new element types τ_1 and τ_2 and replace $\tau \to \alpha$ with a new rule $\tau \to \tau_1, \tau_2$. We proceed to process $\tau_1 \to \alpha_1$ and $\tau_2 \to \alpha_2$ in the same way.
- If $\alpha = (\alpha_1|\alpha_2)$, then we introduce two new element types τ_1 and τ_2 and replace $\tau \to \alpha$ with a new rule $\tau \to \tau_1 \mid \tau_2$. We proceed to process $\tau_1 \to \alpha_1$ and $\tau_2 \to \alpha_2$ in the same way.
- If $\alpha = \alpha_1^*$, then we introduce a new element type τ_1 and replace $\tau \to \alpha$ with $\tau \to \tau_1^*$. We proceed to process $\tau_1 \to \alpha_1$ in the same way.
- If α is one of $\tau' \in E$, $\mathtt{S}$, or ϵ, then the rule for τ remains unchanged.

We refer to the set of new element types introduced when processing $\tau \to P(\tau)$ as N_τ and the set of production rules generated/revised as P_τ. 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 *narrowed DTD of D* (or just a narrow DTD if D is clear from the context), where

- $E_N = E \ \cup \ \cup_{\tau \in E} N_\tau$, i.e., all element types of E and new element types introduced in the narrowing process;
- $P_N = \cup_{\tau \in E} P_\tau$, 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 r 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 E are of the form $\tau \to \tau'$, where $\tau' \in E$. It is easy to see that D_N is computable in polynomial time.

Obviously, any set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D is also a set of $\mathcal{AC}^{*,1}_{PK,FK}$-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 Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-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 τ and a sequence of attributes $@l_1, \ldots, @l_n \in R(\tau)$, define $values(\tau[@l_1, \ldots, @l_n])$ as $\{(x.@l_1, \ldots, x.@l_n) \mid x \in ext(\tau)\}$.

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 τ in D and $@l_1, \ldots, @l_n \in R(\tau)$, $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1, and $values(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $values(\tau[@l_1, \ldots, @l_n])$ in T_1.

If 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, that there is $\varphi \in \Sigma$ such that $T_2 \not\models \varphi$. (1) If φ is a key $\tau[@l_1, \ldots, @l_n] \to \tau$, then there exist two distinct nodes $x, y \in ext(\tau)$ in T_2 such that $x.@l_i = y.@l_i$ for every $i \in [1, n]$. In other words, $|values(\tau[@l_1, \ldots, @l_n])| < |ext(\tau)|$ in T_2. Since $T_1 \models \varphi$, it must be the case that $|values(\tau[@l_1, \ldots, @l_n])| = |ext(\tau)|$ in T_1 because the tuple $(x.@l_1, \ldots, x.@l_n)$ of each $x \in ext(\tau)$ uniquely identifies x

among $ext(\tau)$. This contradicts the claim that $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1 and $values(\tau[@l_1, \dots, @l_n])$ in T_2 equals $values(\tau[@l_1, \dots, @l_n])$ in T_1. (2) If φ is a unary foreign key $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then either $T_2 \not\models \tau_2.@l_2 \to \tau_2$ or there is $x \in ext(\tau_1)$ in T_2 such that, for all $y \in ext(\tau_2)$ 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 $x.@l_1 \notin values(\tau_2.@l_2)$ in T_2. By the claim, $x.@l_1 \in values(\tau_1.@l_1)$ in T_1. Since $T_1 \models \varphi$, $x.@l_1 \in values(\tau_2.@l_2)$ in T_1. Again by the claim, we have $x.@l_1 \in values(\tau_2.@l_2)$ 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 τ-element v in T_1. Let $ele_1(v) = [v_1, \dots, v_n]$ and $w = lab_1(v_1) \dots lab_1(v_n)$. Recall N_τ and P_τ, the set of nonterminals and the set of production rules generated when narrowing $\tau \to P(\tau)$. Let Q_τ be the set of E symbols that appears in P_τ plus $\mathtt{S}$. We can view $G = (Q_\tau, N_\tau \cup \{\tau\}, P_\tau, \tau)$ as an extended context free grammar, where Q_τ is the set of terminals, $N_\tau \cup \{\tau\}$ the set of nonterminals, P_τ the set of production rules, and τ the start symbol.[1] Since $T_1 \models D$, we have $w \in P(\tau)$. By a straightforward induction on the structure of $P_N(\tau)$, 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 $v_1, \dots, v_n$. Observe that the internal nodes of $T(w)$ are labeled with element types in N_τ except that the root v is labeled τ. 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 $lab_2(x) = lab_1(x)$ 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 $T_2 \models D_N$; moreover, for every element type τ in D and $@l_1, \dots, @l_n \in R(\tau)$, $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1 and $values(\tau[@l_1, \dots, @l_n])$ in T_2 equals $values(\tau[@l_1, \dots, @l_n])$ 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) the 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 $lab(v) = \tau$ 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, 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 τ in D and $@l_1, \dots, @l_n \in R(\tau)$, $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1 and $values(\tau[@l_1, \dots, @l_n])$ in T_2 equals $values(\tau[@l_1, \dots, @l_n])$ 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.

[1]If τ is in $P(\tau)$, i.e., if τ is recursively defined, we need to rename τ in Q_τ to ensure that Q_τ and $N_\tau \cup \{\tau\}$ are disjoint. It is straightforward to handle that case.

By Lemma 3.2, in the rest of this proof we consider only narrow DTDs. Next we show how to encode $\mathcal{AC}^{*,1}_{PK,FK}$-constraints by a prequadratic Diophantine system. Let $D = (E, A, P, R, r)$ be a narrow DTD and Σ be a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints, i.e., primary $\mathcal{AC}^{*,1}_{K,FK}$-constraints. We encode Σ with a set C_Σ of integer constraints, referred to as *the cardinality constraints determined by* Σ. For every $\varphi \in \Sigma$,

- if φ is a key constraint $\tau[@l_1, \ldots, @l_k] \to \tau$, then C_Σ contains $|ext(\tau)| \leq |values(\tau.@l_1)| \cdot \ldots \cdot |values(\tau.@l_k)|$;
- if φ is a unary foreign key $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then C_Σ 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_Σ. Otherwise, for every $@l \in R(\tau)$, $|values(\tau.@l)| \leq |ext(\tau)|$ and $0 \leq |values(\tau.@l)|$ are in C_Σ.

Observe that for a unary key $\tau.@l \to \tau$ we have both $|values(\tau.@l)| \leq |ext(\tau)|$ and $|ext(\tau)| \leq |values(\tau.@l)|$ in C_Σ. Thus C_Σ assures $|ext(\tau)| = |values(\tau.@l)|$.

We write $T \models C_\Sigma$ if T satisfies all of the constraints of C_Σ, and we write $T \models (D, C_\Sigma)$ if T conforms to a narrow DTD D and satisfies C_Σ. Note that C_Σ is equivalent (in fact, can be converted in polynomial time) to a prequadratic Diophantine system since $x \leq x_1 \cdot \ldots \cdot x_k$ can be written as constraints of the form $x \leq y \cdot z$ by introducing $k-2$ fresh variables; e.g., $x \leq x_1 \cdot x_2 \cdot x_3 \cdot x_4$ is equivalent to $x \leq x_1 \cdot z_1$, $z_1 \leq x_2 \cdot z_2$, 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_Σ consists of linear and prequadratic integer constraints only. It should be noted that C_Σ can be computed in time polynomial in the size of Σ and D. The lemma below shows that C_Σ characterizes the consistency of Σ if keys in Σ are primary.

LEMMA 3.3. *Let D be a narrow DTD and Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. Then every XML tree conforming to D and satisfying Σ also satisfies C_Σ. 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 Σ, it must be the case that $T_1 \models C_\Sigma$.

Conversely, we show that if there exists an XML tree $T_2 = (V, lab, ele, att_2, root)$ such that $T_2 \models (D, C_\Sigma)$, then we can construct an XML tree $T_1 = (V, lab, ele, att_1, root)$ such that $T_1 \models (D, \Sigma)$. We construct T_1 from T_2 by modifying the function att_2 while leaving V, lab, ele, and $root$ unchanged. More specifically, let $S = \{\tau.@l \mid \tau \in E, @l \in R(\tau)\}$. To define the new function, denoted by att_1, we first associate a set of string values with each $\tau.@l$ in S. Let N be the maximum cardinality of $values(\tau.@l)$ in T_2, i.e., $N \geq |values(\tau.@l)|$ in T_2 for all $\tau.@l \in S$. Let $V_S = \{a_i \mid i \in [1, N]\}$ be a set of distinct string values. For each $\tau.@l \in S$, let $V_{\tau.@l} = \{a_i \mid i \in [1, |values(\tau.@l)|]\}$, and for each $x \in ext(\tau)$, let $att_1(x, @l)$ be a string value in $V_{\tau.@l}$ such that, in T_1, $values(\tau.@l) = V_{\tau.@l}$. The value $att_1(x, @l)$ can be selected in such a way that, for each key $\varphi = \tau[@l_1, \ldots, @l_k] \to \tau$ in Σ, $x[@l_1, \ldots, @l_k]$ is a distinct list of string values from $V_{\tau.@l_1} \times \cdots \times V_{\tau.@l_k}$. This is possible because by the definition of T_1, (1) $ext(\tau)$ in T_1 equals $ext(\tau)$ in T_2; (2) $|values(\tau.@l)|$ in T_1 equals $|values(\tau.@l)|$ in T_2; (3) $T_2 \models C_\Sigma$ and $|ext(\tau)| \leq |values(\tau.@l_1)| \cdot \ldots \cdot |values(\tau.@l_k)|$ is in C_Σ; and (4) since φ is the only key defined for τ-elements, when we populate attributes $@l_1, \ldots, l_k$ of x, we need only to select the value of $att_1(x, @l_i)$ 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 simple 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 $V_{\tau_1.@l_1} \subseteq V_{\tau_2.@l_2}$ even if Σ does not imply $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$. 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 φ is a key $\tau[@l_1, \ldots, @l_k] \to \tau$, it is immediate from the definition of T_1 that $T_1 \models \varphi$ since for any $x \in ext(\tau)$, $x[@l_1, \ldots, @l_k]$ is a distinct list of string values from $V_{\tau.@l_1} \times \cdots \times V_{\tau.@l_k}$. (2) If φ is $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then $T_2 \models |values(\tau_1.@l_1)| \leq |values(\tau_2.@l_2)|$ by $T_2 \models C_\Sigma$. By the definition of att_1, for $i = 1, 2$, $V_{\tau_i.@l_i} = \{a_i \mid i \in [1, |values(\tau_i.@l_i)|]\}$ and in T_1, $values(\tau_i.@l_i) = V_{\tau_i.@l_i}$. Thus $values(\tau_1.@l_1) \subseteq values(\tau_2.@l_2)$ in T_1. Furthermore, given that $|ext(\tau_2)| \leq |values(\tau_2.@l_2)|$ and $|values(\tau_2.@l_2)| \leq |ext(\tau_2)|$ are both in C_Σ, $T_2 \models C_\Sigma$, $|ext(\tau_2)|$ in T_2 is equal to $|ext(\tau_2)|$ in T_1, and $|values(\tau_2.@l_2)|$ in T_2 is equal to $|values(\tau_2.@l_2)|$ in T_1, we conclude that $|ext(\tau_2)|$ is equal to $|values(\tau_2.@l_2)|$ in T_1 and, hence, $T_1 \models \tau_2.@l_2 \to \tau_2$ since each $x \in ext(\tau_2)$ 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 $ext(\tau_2)$. Therefore, $T_1 \models \varphi$ and, thus, $T_1 \models (D, \Sigma)$. This concludes the proof of the lemma. $\square$

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 Ψ_D of linear inequalities on nonnegative integers, referred to as the set of cardinality constraints determined by D, which includes $|ext(\tau)|$ as a variable for each element type τ in D, but it does not have $|values(\tau.@l)|$ as a variable for any attribute $@l$ of τ. More specifically, for each symbol $\tau \in E \cup \{\mathtt{S}\}$, $|ext(\tau)|$ is treated as a distinct variable, which keeps track of the number of all τ elements in an XML tree T conforming to D. In addition, for each occurrence of τ in the definition $P(\tau')$ of some element type τ', we also create distinct variables as follows: if $P(\tau') = \tau_1$ for $\tau_1 \in E \cup \{\mathtt{S}\}$, then we create a distinct variable $x^1_{\tau_1,\tau'}$; if $P(\tau') = (\tau_1, \tau_2)$ or $P(\tau') = (\tau_1|\tau_2)$, then we create two distinct variables $x^1_{\tau_1,\tau'}$ and $x^2_{\tau_2,\tau'}$. Intuitively, for $i \in [1, 2]$, $x^i_{\tau_i,\tau'}$ keeps track of the number of τ_i subelements at position i under all τ' elements in T. Let X_τ be the set of all variables of the form $x^i_{\tau,\tau'}$. By using these variables, for each $\tau \in E$, we define a set ψ_τ of linear integer constraints that characterizes $P(\tau)$ quantitatively, as follows:

- If $P(\tau) = \tau_1$ for $\tau_1 \in E \cup \{\mathtt{S}\}$, then ψ_τ includes $|ext(\tau)| = x^1_{\tau_1,\tau}$. Referring to an XML tree T that conforms to D, this assures that each τ element has a unique τ_1 subelement.
- If $P(\tau') = (\tau_1, \tau_2)$, then ψ_τ includes $|ext(\tau)| = x^1_{\tau_1,\tau}$ and $|ext(\tau)| = x^2_{\tau_2,\tau}$. These assure that each τ element in T must have a unique τ_1 subelement and a unique τ_2 subelement.
- If $P(\tau') = (\tau_1|\tau_2)$, then ψ_τ includes $|ext(\tau)| = x^1_{\tau_1,\tau} + x^2_{\tau_2,\tau}$. These assure that each τ element in T must have either a τ_1 subelement or a τ_2 subelement, and thus the sum of the numbers of these τ_1 and τ_2 subelements equals the number of τ elements in T.

The set Ψ_D of cardinality constraints determined by DTD D consists of the following:

- $|ext(r)| = 1$; i.e., there is a unique root in any XML tree valid w.r.t. D;
- constraints of ψ_τ for each $\tau \in E$; these assure that $P(\tau)$ is satisfied;
- $|ext(\tau)| = \sum_{x^i_{\tau,\tau'} \in X_\tau} x^i_{\tau,\tau'}$ for each $\tau \in (E \setminus \{r\}) \cup \{\mathtt{S}\}$; this indicates that the set $ext(\tau)$ includes all τ 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 Ψ_D has a nonnegative integer solution iff there exists an XML tree T conforming to D such that the cardinality of $ext(\tau)$ in T equals the value of the variable $|ext(\tau)|$ in the solution for each element type τ in D.

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

$$\Psi(D,\,\Sigma) \;=\; \Psi_D \cup C_\Sigma \cup \{(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0) \mid \tau \in E,\ @l \in R(\tau)\},$$

where C_Σ is the set of cardinality constraints determined by Σ, Ψ_D is the set of cardinality constraints determined by D, and constraints $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ are to ensure that every τ-element has an $@l$-attribute (note that $|values(\tau.@l)| \leq |ext(\tau)|$ is already in C_Σ). Constraints in $\Psi(D,\,\Sigma)$ are either linear integer constraints, or inequalities of the form $x \leq y \cdot z$, which come from C_Σ, 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 Σ.

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

LEMMA 3.4. *Let D be a narrow DTD and Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. Then $\Psi(D,\,\Sigma)$ is consistent iff 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 Ψ_D such that for each element type τ in D, the value of the variable $|ext(\tau)|$ equals the number of τ-elements in T [16]. By Lemma 3.3 and $T \models \Sigma$, we have $T \models C_\Sigma$. We extend the solution of Ψ_D to be one to $\Psi(D,\,\Sigma)$ by letting the variable $|values(\tau.@l)|$ equal the number of distinct $@l$-attribute values of all τ-elements in T, for each element type τ and attribute $@l$ of τ in D. Since $T \models C_\Sigma$, this extended assignment satisfies all of the constraints in C_Σ. In addition, if $|ext(\tau)| > 0$, then $|values(\tau.@l)| > 0$ since every τ-element in T has an $@l$-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 τ in D, the cardinality of $ext(\tau)$ in T equals the value of the variable $|ext(\tau)|$ in the solution [16]. We construct a new tree T' from T by modifying the definition of the function att such that in T', for each element type τ and attribute $@l$ of τ, the number of distinct $@l$-attribute values of all τ-elements equals the value of the variable $|values(\tau.@l)|$ in the solution. This is possible since $|values(\tau.@l)| \leq |ext(\tau)|$ and $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ are in $\Psi(D,\,\Sigma)$. The assignment is also a solution to C_Σ. 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 $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE. By Lemma 3.2, given an arbitrary DTD D and a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints, one can compute a narrow DTD D_N such that (D,Σ) is consistent iff (D_N,Σ) is consistent. By Lemma 3.4, (D_N,Σ) is consistent iff $\Psi(D_N,\Sigma)$ has a nonnegative integer solution. Such a solution requires $|values(\tau.@l)| > 0$ if $|ext(\tau)| > 0$. To ensure this, let $\Phi(D_N,\Sigma)$ be a system that includes all linear integer constraints and prequadratic constraints in $\Psi(D_N, \Sigma)$ and, moreover, $|ext(\tau)| \leq |values(\tau.@l)| \cdot |ext(\tau)|$ for each $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ in $\Psi(D_N, \Sigma)$. Now $\Phi(D_N, \Sigma)$ is a prequadratic Diophantine system. In addition, $\Psi(D_N, \Sigma)$ has a nonnegative integer solution iff $\Phi(D_N, \Sigma)$ has a nonnegative integer solution. To see this, observe that, for any nonnegative integer assignment to $|ext(\tau)|$ and $|values(\tau.@l)|$, $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ iff $|ext(\tau)| \leq |values(\tau.@l)| \cdot |ext(\tau)|$. Thus, (D, Σ) is consistent iff the prequadratic Diophantine system $\Phi(D_N, \Sigma)$ has a nonnegative integer solution. Note that D_N can be computed in polynomial time in the size of D, $\Psi(D_N, \Sigma)$ can be computed in polynomial time in the size of D_N and Σ, and $\Phi(D_N, \Sigma)$ can be computed in polynomial time in the size of $\Psi(D_N, \Sigma)$. Hence, it takes polynomial time to compute $\Phi(D_N, \Sigma)$ from D and Σ. Therefore, there is a PTIME reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE.

(b) *A reduction from PDE to* $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. We now move to the other direction. Given an instance of PDE, i.e., a system S consisting of a set S_L of linear equations/inequalities on integers and a set S_P of prequadratic constraints of the form $x \leq y \cdot z$, we define a DTD D and a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints such that S has a nonnegative solution iff there is an XML tree T satisfying Σ and conforming to D. We use $X = \{x_i \mid i \in [1, n]\}$ to denote the set of all of the variables in S. Assume that $S_L = \{e_j \mid j \in [1, m]\}$ and e_j is of the form: $a_1^j\ x_1 + \cdots + a_n^j\ x_n + c_j \leq b_1^j\ x_1 + \cdots + b_n^j\ x_n + d_j$, where a_i^j $(i \in [1, n])$, b_i^j $(i \in [1, n])$, c_j, and d_j are nonnegative integers.[2] Also, assume that $S_P = \{p_j \mid j \in [1, l]\}$, where p_j is a prequadratic equation of the form $x \leq y \cdot z$. Then we define DTD $D = (E,\ A,\ P,\ R,\ r)$ as follows:

(1) For each variable x_i, we define an element type X_i. In addition, for each $p_s \in S_P$ of the form $x_i \leq x_j \cdot x_k$, we define an element type U_i^s. For each linear constraint e_j, we define distinct element types E_j, A_1^j, …, A_n^j, C_j, F_j, B_1^j, …, B_n^j, D_j. We use r to denote the root element type. That is,

$$\begin{aligned} E \ &= \ \{r\} \cup \{X_i \mid i \in [1, n]\} \cup \\ &\{E_j, A_1^j, \ldots, A_n^j, C_j, F_j, B_1^j, \ldots, B_n^j, D_j \mid j \in [1, m]\} \cup \{U_i^s \mid p_s = x_i \leq x_j \cdot x_k \in S_P\}. \end{aligned}$$

Intuitively, referring to an XML tree conforming to D, we use $|ext(X_i)|$ to code the value of the variable x_i in S. For every equation e_j, we use $|ext(A_1^j)|$, …, $|ext(A_n^j)|$, $|ext(C_j)|$ to code the values of constants a_1^j, …, a_n^j, c_j; $|ext(E_j)|$ to code the value of the expression $a_1^j x_1 + \cdots + a_n^j x_n + c_j$; $|ext(B_1^j)|$, …, $|ext(B_n^j)|$, $|ext(D_j)|$ to code the values of constants b_1^j, …, b_n^j, d_j; and $|ext(F_j)|$ to code the value of the expression $b_1^j x_1 + \cdots + b_n^j x_n + d_j$. Furthermore, for each prequadratic equation $p_s = x_i \ \leq \ x_j \cdot x_k$ in S_P, we create a distinct copy U_i^s of X_i. The reason to use U_i^s instead of X_i is to ensure that the set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints defined below is primary.

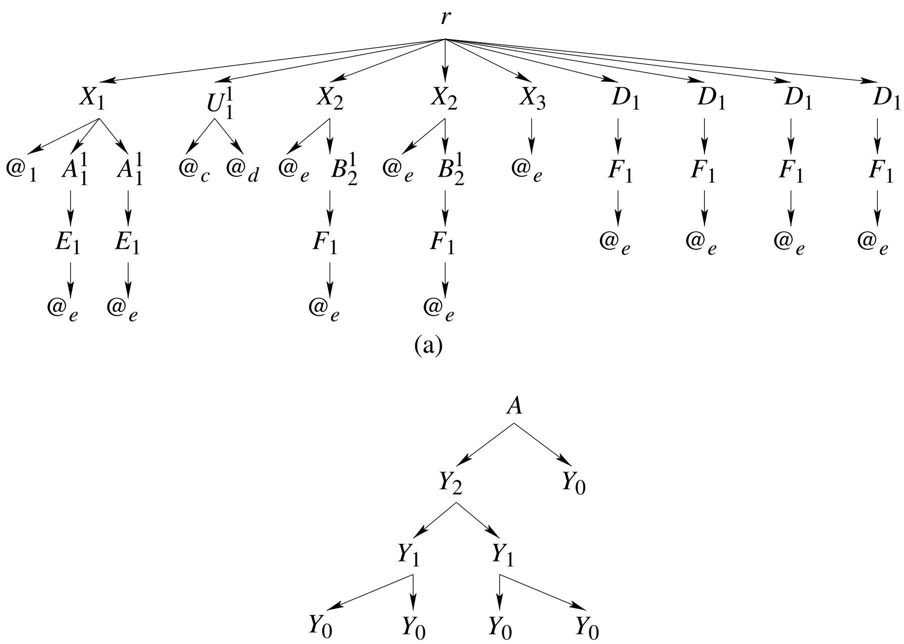
(2) $A = \{@c,\ @d,\ @e\}$. Intuitively, we shall define $@e$ as a key and use $@c$ and $@d$ to code prequadratic constraints of the form $x \leq y \cdot z$.

(3) We define production rules as follows. For the root of the DTD:

$$\begin{aligned} P(r) \ = \ &(X_1, U_1^{s_{1,1}}, \ldots, U_1^{s_{1,j_1}})^*, \ldots, (X_n, U_n^{s_{n,1}}, \ldots, U_n^{s_{n,j_n}})^*, \\ &\underbrace{C_1, \ldots, C_1}_{c_1 \text{ times}}, \ldots, \underbrace{C_m, \ldots, C_m}_{c_m \text{ times}}, \underbrace{D_1, \ldots, D_1}_{d_1 \text{ times}}, \ldots, \underbrace{D_m, \ldots, D_m}_{d_m \text{ times}}, \end{aligned}$$

where $\{s_{i,1},\ \ldots,\ s_{i,j_i}\}$ $(i \in [1, n])$ is the set of indexes $\{s \mid p_s = x_i \leq x_j \cdot x_k \in S_P\}$.

[2] For example, we represent the equation $-3x + 5y \leq -7$ as $0x + 5y + 7 \leq 3x + 0y + 0$.



(a)

(b)

FIG. 3.1. *Trees used in the proof of Theorem* 3.1.

Furthermore, for every $i \in [1, n]$ and every $j \in [1, m]$:

$$\begin{array}{rcl} P(A_i^j) & = & E_j, \\ P(C_j) & = & E_j, \\ P(B_i^j) & = & F_j, \\ P(D_j) & = & F_j, \\ P(X_i) & = & \underbrace{A_i^1, \ldots, A_i^1}_{a_i^1 \text{ times}}, \ldots, \underbrace{A_i^m, \ldots, A_i^m}_{a_i^m \text{ times}}, \underbrace{B_i^1, \ldots, B_i^1}_{b_i^1 \text{ times}}, \ldots, \underbrace{B_i^m, \ldots, B_i^m}_{b_i^m \text{ times}}. \end{array}$$

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

(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 $p_s = x_i \leq x_j \cdot x_k \in S_P$, $R(U_i^s) = \{@c, @d\}$. For all other element types τ, 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 $S_L = \{2x_1 \leq x_2 + 4\}$ and $S_P = \{x_1 \leq x_2 \cdot x_3\}$. We note that this tree codes the solution $x_1 = 1$, $x_2 = 2$, $x_3 = 1$ for this system of equations.

Given DTD D, we define a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. For each $j \in [1, m]$, Σ includes keys $E_j.@e \to E_j$ and $F_j.@e \to F_j$ 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 $p_s = x_i \leq x_j \cdot x_k \in S_P$, Σ includes the following constraints:

$$U_i^s[@c, @d] \to U_i^s, \qquad U_i^s.@c \subseteq_{FK} X_j.@e, \qquad U_i^s.@d \subseteq_{FK} X_k.@e.$$

Clearly, the set Σ is primary; i.e., for any element type τ there is at most one key defined. In fact, we use copies U_i^s of X_i just to ensure that Σ is primary.

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

We observe that the previous reduction is not polynomial since constants a_i^j, b_i^j ($i \in [1, n]$, $j \in [1, m]$) 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 $a_k a_{k-1} \ldots a_1 a_0$. 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:

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

and $P(A) = Y_{i_1}, \ldots, Y_{i_l}$, where $i_1 > \cdots > i_l \geq 0$ and $\{i_1, \ldots, i_l\}$ is the set of indexes $\{j \in [0, k] \mid a_j = 1\}$. 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 $|ext(Y_0)| = a$ 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 $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$, we can show that there is a PTIME reduction from PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. This completes the proof of Theorem 3.1. $\square$

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

COROLLARY 3.5. $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ *is NP-hard and can be solved in NEXPTIME.*

Obviously we cannot obtain the exact complexity of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ without resolving the corresponding question for PDE, which appears to be quite hard [21]. The result of Theorem 3.1 can be generalized to *disjoint* $\mathcal{AC}^{*,1}_{K,FK}$-constraints: that is, a set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints in which, for every element type τ and every two distinct keys $\tau[X] \to \tau$ and $\tau[Y] \to \tau$ in Σ (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* $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$ *to disjoint constraints is polynomially equivalent to PDE, and, thus, it is NP-hard and can be solved in NEXPTIME.*

3.2. Consistency of regular expression constraints. Specifications of $\mathcal{AC}^{*,*}_{K,FK}$-constraints are associated with element types. We next consider $\mathcal{AC}^{reg}_{K,FK}$, the class of unary keys and foreign keys defined in terms of regular path expressions. For $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, we are able to establish both an upper and a lower bound. The lower bound already indicates that the problem is perhaps infeasible in practice, even for very simple DTDs. Finding the precise complexity of the problem remains open and does not appear to be easy. 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 of the expressions involved in the constraints.

THEOREM 3.7.

(a) $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ *can be solved in* 2*-NEXPTIME.*

(b) $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ *is PSPACE-hard, even for nonrecursive no-star DTDs.*

Proof. We reduce $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ 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 $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$.

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 $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ [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 of the variables in the coding of DTDs with states of a certain automaton (the product automaton for all of the automata corresponding to the regular expressions used in constraints).

Now it is time to fill in all of the details. First, we need some additional notation. For every regular expression β and every attribute $@l$, we write $values(\beta.@l)$ to denote the set $\{y.@l \mid y \in nodes(\beta) \text{ and } y.@l \text{ is defined}\}$. Observe that, for any $\tau \in E \setminus \{r\}$ and $@l \in R(\tau)$, $values(r.\_^*.\tau.@l)$ corresponds to our original definition of $values(\tau.@l)$.

We say that a DTD D is *one-attribute* if D contains only one attribute and no element type τ such that $P(\tau) = \mathtt{S}$. We start by showing that $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$

can be reduced to the consistency problem for regular expression constraints over one-attribute DTDs. Let $D = (E, A, P, R, r)$ be a DTD and Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D. First, define DTD $D_U = (E_U, A_U, P_U, R_U, r)$ as follows. For every $\tau \in E$ and $@l \in R(\tau)$, assume that $\tau_{@l}$ is a fresh element type symbol. Then define E_U as $E \cup \{\tau_{@l} \mid \tau \in E \text{ and } @l \in R(\tau)\}$ and $A_U = \{@e\}$, where $@e$ is a fresh attribute symbol. Furthermore, define functions P_U and R_U as follows:

- For every $\tau \in E$ such that $P(\tau) = \mathtt{S}$, if $R(\tau) = \{@l_1, \ldots, @l_n\}$, where $n \geq 0$, then $P_U(\tau) = \tau_{@l_1}, \ldots, \tau_{@l_n}$ and $R_U(\tau) = \emptyset$.
- For every $\tau \in E$ such that $P(\tau)$ is a regular expression over E, if $R(\tau) = \{@l_1, \ldots, @l_n\}$, where $n \geq 0$, then $P_U(\tau) = P(\tau), \tau_{@l_1}, \ldots, \tau_{@l_n}$ and $R_U(\tau) = \emptyset$.
- For every $\tau \in E$ and $@l \in R(\tau)$, $P_U(\tau_{@l}) = \epsilon$ and $R_U(\tau_{@l}) = \{@e\}$.

We note that if $P(\tau) = \mathtt{S}$ and $R(\tau) = \emptyset$, then $P_U(\tau) = \epsilon$.

Second, define the set Σ_U of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_U as follows. For every key constraint $\beta.\tau.@l \to \beta.\tau$ in Σ, we include $\beta.\tau.\tau_{@l}.@e \to \beta.\tau.\tau_{@l}$ in Σ_U, and, for every foreign key constraint $\beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$ in Σ, we add $\beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$ to Σ_U.

Lemma 3.8. *Let D be a DTD, Σ be a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, and D_U, Σ_U be as defined above. 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 $T_2 \models (D_U, \Sigma_U)$.*

Proof. ($\Rightarrow$) Let $T_1 = (V_1, lab_1, ele_1, att_1, root)$ be an XML tree such that $T_1 \models (D, \Sigma)$. We define an XML tree T_2 from T_1 such that $T_2 \models (D_U, \Sigma_U)$. More specifically, $T_2 = (V_2, lab_2, ele_2, att_2, root)$, where V_2, lab_2, ele_2, and att_2 are defined as follows. Let v be a node in T_1 such that $lab_1(v) = \tau \in E$ and $R(\tau) = \{@l_1, \ldots, @l_k\}$. Then V_2 contains node v and fresh nodes $v_{@l_1}, \ldots, v_{@l_k}$ such that $lab_2(v) = \tau$ and $lab_2(v_{@l_i}) = \tau_{@l_i}$, for every $i \in [1, k]$. Furthermore, if $ele_1(v) = [s]$, where $s \in \mathtt{S}$, then $ele_2(v) = [v_{@l_1}, \ldots, v_{@l_k}]$. Otherwise, $ele_1(v) = [v_1, \ldots, v_n]$, where $n \geq 0$ and each v_i is an element node, and $ele_2(v) = [v_1, \ldots, v_n, v_{@l_1}, \ldots, v_{@l_k}]$. Finally, $att_2(v, @e)$ is not defined and $att_2(v_{@l_i}, @e) = att_1(v, @l_i)$ for every $i \in [1, k]$. Next we show that $T_2 \models (D_U, \Sigma_U)$.

By the definition of D_U and given that $T_1 \models D$, it is easy to see that $T_2 \models D_U$. Assume that $T_2 \not\models \Sigma_U$. Then there exist $\varphi \in \Sigma_U$ such that $T_2 \not\models \varphi$. (1) If φ is a key $\beta.\tau.\tau_{@l}.@e \to \beta.\tau.\tau_{@l}$, then there exist distinct $v_1, v_2 \in nodes(\beta.\tau.\tau_{@l})$ in T_2 such that $att_2(v_1, @e) = att_2(v_2, @e)$. Let u_1 and u_2 be the parents of v_1 and v_2 in T_2, respectively. By the definition of D_U and given that $v_1 \neq v_2$, we have $u_1 \neq u_2$. Thus, by the definition of T_2, u_1 and u_2 are nodes in T_1 such that $u_1, u_2 \in nodes(\beta.\tau)$ and $att_1(u_1, @l) = att_1(u_2, @l) = att_2(v_1, @e)$. Therefore, $T_1 \not\models \beta.\tau.@l \to \beta.\tau$, which contradicts the assumption that $T_1 \models \Sigma$. (2) If φ is a foreign key $\beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$, then either $T_2 \not\models \beta'.\tau'.\tau'_{@l'}.@e \to \beta'.\tau'.\tau'_{@l'}$ or there exists $v \in nodes(\beta.\tau.\tau_{@l})$ such that $att_2(v, @e) \notin values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2. In the former case, we reach a contradiction as in (1). In the latter case, assume that u is the parent of v in T_2. By the definition of T_2, we have that u is a node in T_1 such that $u \in nodes(\beta.\tau)$ and $att_1(u, @l) = att_2(v, @e)$. Thus, given that $values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2 is equal to $values(\beta'.\tau'.@l')$ in T_1, we conclude that $att_1(u, @l) \notin values(\beta'.\tau'.@l')$ in T_1. Therefore, $T_1 \not\models \beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$, which contradicts the assumption that $T_1 \models \Sigma$.

($\Leftarrow$) Let $T_2 = (V_2, lab_2, ele_2, att_2, root)$ 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, $T_1 = (V_1, lab_1, ele_1, att_1, root)$, where V_1, lab_1, ele_1, and att_1 are defined as follows. Let v be a node in T_2 such that $lab_2(v) = \tau$, $\tau \in E$, and $R(\tau) = \{@l_1, \ldots, @l_k\}$.

Then V_1 also contains node v with $lab_1(v) = \tau$. Furthermore, if $P(\tau) = \mathtt{S}$, then $ele_2(v) = [v_{@l_1}, \ldots, v_{@l_k}]$, where $lab(v_{@l_j}) = \tau_{@l_j}$ ($j \in [1, k]$), and we define $ele_1(v)$ as $[s]$, where s is an arbitrary string in S, and we define $att_1(v, @l_i)$ as $att_2(v_{@l_i}, @e)$ for every $i \in [1, k]$. Otherwise, $P(\tau)$ is a regular expression over E and $ele_2(v) = [v_1, \ldots, v_n, v_{@l_1}, \ldots, v_{@l_k}]$, where $lab(v_i) \in E$ ($i \in [1, n]$) and $lab(v_{@l_j}) = \tau_{@l_j}$ ($j \in [1, k]$), and we define $ele_1(v)$ as $[v_1, \ldots, v_n]$ and $att_1(v, @l_i)$ as $att_2(v_{@l_i}, @e)$ 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 φ is a key $\beta.\tau.@l \to \beta.\tau$, then there exist distinct $u_1, u_2 \in nodes(\beta.\tau)$ in T_1 such that $att_1(u_1, @l) = att_1(u_2, @l)$. By the definition of T_1, u_1 and u_2 are also in $nodes(\beta.\tau)$ in T_2. Let v_1 and v_2 be the children of u_1 and u_2 in T_2 of type $\tau_{@l}$, respectively. Given that $u_1 \neq u_2$, we have $v_1 \neq v_2$. Thus, by the definition of T_1, v_1 and v_2 are nodes in T_2 such that $v_1, v_2 \in nodes(\beta.\tau.\tau_{@l})$ and $att_2(v_1, @e) = att_2(v_2, @e) = att_1(u_1, @l)$. Therefore, $T_2 \not\models \beta.\tau.\tau_{@l}.@e \to \beta.\tau.\tau_{@l}$, which contradicts the assumption that $T_2 \models \Sigma_U$. (2) If φ is a foreign key $\beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$, then either $T_1 \not\models \beta'.\tau'.@l' \to \beta'.\tau'$ or there exists $u \in nodes(\beta.\tau)$ such that $att_1(u, @l) \notin values(\beta'.\tau'.@l')$ 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 $\tau_{@l}$ (u is a node of T_2 by the definition of T_1). By the definition of T_1, we have $v \in nodes(\beta.\tau.\tau_{@l})$ and $att_2(v, @e) = att_1(u, @l)$. Thus, given that $values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2 is equal to $values(\beta'.\tau'.@l')$ in T_1, we conclude that $att_2(v, @e) \notin values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2. Therefore, $T_2 \not\models \beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$, which contradicts the assumption that $T_2 \models \Sigma_U$. This concludes the proof of the lemma. ∎

By Lemma 3.8, from now on we consider only one-attribute DTDs. Let $D = (E, \{@l\}, P, R, r)$ be a one-attribute DTD and $D_N = (E_N, \{@l\}, P_N, R_N, r)$ be the narrow DTD of D (defined in the proof of Theorem 3.1). Observe that D_N is also 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 $\mathcal{AC}^{reg}_{K,FK}$-constraint φ 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 $\mathcal{AC}^{reg}_{K,FK}$-constraints over D by new $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_N. More precisely, given a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, we define a set Σ_N of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_N, referred to as the *narrowed set of constraints of* Σ, as follows. Let f be a substitution for the element types in E defined as $f(\tau) = \tau.(E_N \setminus E)^*$ for every $\tau \in E$. Then for every key constraint $\beta.\tau.@l \to \beta.\tau$ in Σ, $f(\beta).\tau.@l \to f(\beta).\tau$ is in Σ_N, and, for every foreign key constraint $\beta_1.\tau_1.@l \subseteq_{FK} \beta_2.\tau_2.@l$ in Σ (recall that $@l$ is the only attribute of D), $f(\beta_1).\tau_1.@l \subseteq_{FK} f(\beta_2).\tau_2.@l$ is in Σ_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, Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, and Σ_N the narrowed set of constraints of Σ. 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 $T_2 \models (D_N, \Sigma_N)$.*

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 $\beta.\tau$ over D and $@l \in R(\tau)$, $|nodes(f(\beta).\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1, and $values(f(\beta).\tau.@l)$ in T_2 equals $values(\beta.\tau.@l)$ in T_1, where f is the substitution defined above.

If 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, that there is $\varphi \in \Sigma_N$ such that $T_2 \not\models \varphi$. (1) If φ is a key $f(\beta).\tau.@l \to f(\beta).\tau$, then there exist two distinct nodes $x, y \in \mathit{nodes}(f(\beta).\tau)$ in T_2 such that $x.@l = y.@l$. In other words, $|\mathit{values}(f(\beta).\tau.@l)| < |\mathit{nodes}(f(\beta).\tau)|$ in T_2. Since $T_1 \models \varphi$, it must be the case that $|\mathit{values}(\beta.\tau.@l)| = |\mathit{nodes}(\beta.\tau)|$ in T_1 because the value $x.@l$ of each $x \in \mathit{nodes}(\beta.\tau)$ uniquely identifies x among $\mathit{nodes}(\beta.\tau)$. This contradicts the claim that $|\mathit{nodes}(f(\beta).\tau)|$ in T_2 equals $|\mathit{nodes}(\beta.\tau)|$ in T_1 and $\mathit{values}(f(\beta).\tau.@l)$ in T_2 equals $\mathit{values}(\beta.\tau.@l)$ in T_1. (2) If φ is a foreign key $f(\beta_1).\tau_1.@l \subseteq_{FK} f(\beta_2).\tau_2.@l$, then either $T_2 \not\models f(\beta_2).\tau_2.@l \to f(\beta_2).\tau_2$ or there is $x \in \mathit{nodes}(f(\beta_1).\tau_1)$ such that, for all $y \in \mathit{nodes}(f(\beta_2).\tau_2)$ in T_2, $x.@l \neq y.@l$. In the first case, we reach a contradiction as in (1). In the second case, we have $x.@l \notin \mathit{values}(f(\beta_2).\tau_2.@l)$ in T_2. By the claim, $x.@l \in \mathit{values}(\beta_1.\tau_1.@l)$ in T_1. Since $T_1 \models \varphi$, $x.@l \in \mathit{values}(\beta_2.\tau_2.@l)$ in T_1. Again by the claim, we have $x.@l \in \mathit{values}(f(\beta_2).\tau_2.@l)$ in T_2, which contradicts the assumption. The proof for the other direction is similar.

We next verify the claim. Given an XML tree $T_1 = (V_1, \mathit{lab}_1, \mathit{ele}_1, \mathit{att}, \mathit{root})$ 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 τ-element v in T_1. Let $\mathit{ele}_1(v) = [v_1, \ldots, v_n]$ and $w = \mathit{lab}_1(v_1) \ldots \mathit{lab}_1(v_n)$. Recall N_τ and P_τ, the set of nonterminals and the set of production rules generated when narrowing $\tau \to P(\tau)$ (see proof of Theorem 3.1), respectively. Let Q_τ be the set of E symbols that appears in P_τ. We can view $G = (Q_\tau, N_\tau \cup \{\tau\}, P_\tau, \tau)$ as an extended context free grammar, where Q_τ is the set of terminals, $N_\tau \cup \{\tau\}$ the set of nonterminals, P_τ the set of production rules, and τ the start symbol.[3] Since $T_1 \models D$, we have $w \in P(\tau)$. By a straightforward induction on the structure of $P_N(\tau)$ 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_τ except that the root v is labeled τ. Intuitively, we construct T_2 by replacing each element v in T_1 by such a parse tree. More specifically, let $T_2 = (V_2, \mathit{lab}_2, \mathit{ele}_2, \mathit{att}, \mathit{root})$. 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 $\mathit{lab}_2(x) = \mathit{lab}_1(x)$ if $x \in V_1$, and otherwise let $\mathit{lab}_2(x)$ 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 $\mathit{ele}_2(x)$ be the list of its children in the parse tree having x as the root. For every $x \in V_2 \setminus V_1$, let $\mathit{ele}_2(x)$ be the list of its children in the parse tree of an element in V_1 that contains x. Note that att 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 $@l \in R(\tau)$, $|\mathit{nodes}(f(\beta).\tau)|$ in T_2 equals $|\mathit{nodes}(\beta.\tau)|$ in T_1 and $\mathit{values}(f(\beta).\tau.@l)$ in T_2 equals $\mathit{values}(\beta.\tau.@l)$ 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 $w_0, \ldots, w_n$ in $(E_N \setminus E)^*$, $r.w_0.\tau_1.w_1.\ldots.\tau_n.w_n.\tau$ is in $f(\beta).\tau$; (2) if a string $r.w_0.\tau_1.w_1.\ldots.\tau_n.w_n.\tau$ 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 $(E_N \setminus E)^*$, 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.

[3]As in the proof of Lemma 3.2, if τ is in $P(\tau)$, then we need to rename τ in Q_τ to ensure that Q_τ and $N_\tau \cup \{\tau\}$ are disjoint. It is straightforward to handle that case.

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 any node $v \in V_2$ with $lab(v) = \tau$ and $\tau \in E_N \setminus E$, we replace 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, 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 an 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. Observe that att and $root$ remain unchanged. By the definition of T_1 it can be verified that $T_1 \models D$; in addition, for any regular expression $\beta.\tau$ over D and $@l \in R(\tau)$, $|nodes(\beta.\tau)|$ in T_1 equals $|nodes(f(\beta).\tau)|$ in T_2, and $values(\beta.\tau.@l)$ in T_1 equals $values(f(\beta).\tau.@l)$ 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 att is unchanged.

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

$$r_\theta = \left(\bigcap_{i\,:\,\theta(i)=1} \beta_i.\tau_i \right) \cap \left(\bigcap_{j\,:\,\theta(j)=0} \overline{\beta_j.\tau_j} \right), \tag{3.1}$$

where $\overline{\beta_j.\tau_j}$ is the complement $\beta_j.\tau_j$. We allow intersection and complement operators only in regular expressions r_θ. We note that, for every $i \in [1, k]$,[4]

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

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

To capture $\beta_i.\tau_i$ ($i \in [1, k]$) and r_θ ($\theta \in \Theta$) in Ψ^{Σ}_D, 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 of these automata. We refer to M as the DFA for Σ. Let s_M be the start state of M and δ be its transition function. Given an XML tree T conforming to D, for each node x in T we define $state(x)$ as s, if there is a simple path ρ over D such that $T \models \rho(root, x)$ 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, Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, M the DFA for Σ, and $\beta_i.\tau_i$ a regular expression in Σ. Then for every XML tree T conforming to D and every τ_i-element x in T, $x \in nodes(\beta_i.\tau_i)$ in T iff $state(x)$ contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$.*

[4] Recall that the regular language defined by a regular expression β is denoted by β as well.

In other words, $nodes(\beta_i.\tau_i)$ in T consists of all τ_i-elements x such that $state(x)$ (which is a tuple of states of automata corresponding to regular expressions in Σ) 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 ρ over D such that $T \models \rho.\tau_i(root,\ x)$. Thus $x \in nodes(\beta.\tau_i)$ in T iff $\rho.\tau_i \in \beta.\tau_i$. If $x \in nodes(\beta.\tau_i)$ in T, then $\rho.\tau_i \in \beta.\tau_i$, and, therefore, there must be a final state $f_{\beta.\tau_i}$ in the automaton for $\beta.\tau_i$ and a state s in M such that $s = \delta(s_M, \rho.\tau_i)$ and s contains $f_{\beta_i.\tau_i}$. Thus $state(x) = s$ contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$. Conversely, if $state(x)$ contains a final state $f_{\beta_i.\tau_i}$ in the automaton for $\beta_i.\tau_i$, then $\rho.\tau_i \in \beta.\tau_i$ since $s = \delta(s_M, \rho.\tau_i)$. Therefore, $x \in nodes(\beta_i.\tau_i)$ in T.

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

- If $P(\tau) = \tau_1$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$, where $s_1 = \delta(s, \tau_1)$.
- If $P(\tau) = (\tau_1, \tau_2)$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ and $x_\tau^s = x_{\tau_2,\tau}^{s_2,s}$, where $s_i = \delta(s, \tau_i)$ for $i = 1, 2$. Referring to the XML tree T, these assure that each τ-element in T must have a τ_1-subelement and a τ_2-subelement.
- If $P(\tau) = (\tau_1|\tau_2)$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s} + x_{\tau_2,\tau}^{s_2,s}$, where $s_i = \delta(s, \tau_i)$ for $i = 1, 2$. This assures that each τ-element in T must have either a τ_1-subelement or a τ_2-subelement, and thus the sum of the number of these τ_1-subelements and the number of τ_2-subelements equals the number of τ-elements.
- If $P(\tau) = \tau_1^*$, then Ψ_τ^s includes $(x_{\tau_1,\tau}^{s_1,s} > 0) \to (x_\tau^s > 0)$, where $s_1 = \delta(s, \tau_1)$.

In addition, Ψ_τ^s includes $x_\tau^s = \sum_{(\tau',s') \in Y_\tau^s} x_{\tau,\tau'}^{s,s'}$.

Recall that $\beta_1.\tau_1.@l, \ldots, \beta_k.\tau_k.@l$ is an enumeration of all regular expressions and attributes that appear in Σ, that Θ is the set of functions $\theta : \{1, \ldots, k\} \to \{0, 1\}$ which are not identically 0, and that, for each such function θ, r_θ 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 Σ 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, $x \in nodes(\beta_i.\tau_i)$ in T iff $state(x) \in F_{\beta_i.\tau_i}$. Furthermore, for each $\theta \in \Theta$, we define F_θ as the set of states $s = (s_1, \ldots, s_k)$ of the DFA for Σ such that for every $i \in [1, k]$, s_i is a final state of the DFA for $\beta_i.\tau_i$ iff $\theta(i) = 1$. Notice that by Lemma 3.10, for every XML tree T conforming to D and every node x of T, $x \in nodes(r_\theta)$ in T iff $state(x) \in F_\theta$. Finally, for each $r_\theta \neq \emptyset$, 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 τ_θ as τ_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^s associated with r, where $s = \delta(s_M, r)$. We write x_r for x_r^s. Then we define *the set of cardinality constraints determined by DTD D* w.r.t. a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, denoted by Ψ^{Σ}_{D}, as follows:

- For each $\tau \in E$ and each state s given above, Ψ^{Σ}_{D} contains all of the constraints in Ψ^{s}_{τ}.
- Ψ^{Σ}_{D} contains constraint $x_r = 1$; i.e., there is a unique root in each XML tree conforming to D.
- For every $i \in [1, k]$, Ψ^{Σ}_{D} contains constraint $|nodes(\beta_i.\tau_i)| = \sum_{s\,:\,s \in F_{\beta_i.\tau_i}} x^{s}_{\tau_i}$.
- For every $\theta \in \Theta$ such that $r_\theta \neq \emptyset$, Ψ^{Σ}_{D} contains constraint $|nodes(r_\theta)| = \sum_{s\,:\,s \in F_\theta} x^{s}_{\tau_\theta}$.
- For every $\theta \in \Theta$ such that $r_\theta = \emptyset$, Ψ^{Σ}_{D} contains constraint $|nodes(r_\theta)| = 0$.

Note that Ψ^{Σ}_{D} can be computed in EXPTIME in the size of D and Σ. We say that Ψ^{Σ}_{D} is *consistent* iff it has a nonnegative solution. We next show that Ψ^{Σ}_{D} indeed characterizes the narrow one-attribute DTD D. ∎

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

Proof. First, assume that there is an XML tree $T = (V, lab, ele, att, root)$ conforming to D. We define a nonnegative solution of Ψ^{Σ}_{D} as follows. For each variable $x^{s,s'}_{\tau,\tau'}$ in Ψ^{Σ}_{D}, let its value be the number of τ-elements x in T such that x is a child of a node y of type τ' with $state(x) = s$ and $state(y) = s'$. Furthermore, let x_r be 1, and, for every variable x^{s}_{τ} in Ψ^{Σ}_{D}, let x^{s}_{τ} be the sum of the variables $x^{s,s'}_{\tau,\tau'}$ where $(\tau', s') \in Y^{s}_{\tau}$. Finally, for every $i \in [1, k]$ and every $\theta \in \Theta$, let $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ be $\sum_{s\,:\,s \in F_{\beta_i.\tau_i}} x^{s}_{\tau_i}$ and $\sum_{s\,:\,s \in F_\theta} x^{s}_{\tau_\theta}$, respectively. This defines a nonnegative assignment since T is finite. It can be verified that the assignment is a solution of Ψ^{Σ}_{D}. Indeed, it satisfies the constraint $x_r = 1$ and constraints of the form $x^{s}_{\tau} = \sum_{(\tau',s') \in Y^{s}_{\tau}} x^{s,s'}_{\tau,\tau'}$, $|nodes(\beta_i.\tau_i)| = \sum_{s\,:\,s \in F_{\beta_i.\tau_i}} x^{s}_{\tau_i}$, and $|nodes(r_\theta)| = \sum_{s\,:\,s \in F_\theta} x^{s}_{\tau_\theta}$ by the definition of the assignment. Moreover, one can verify that it also satisfies the constraints of each Ψ^{s}_{τ}, by considering four different cases corresponding to the four different types of regular expressions in D. In particular, it satisfies constraints of the form $(x^{s_1,s}_{\tau_1,\tau} > 0) \to (x^{s}_{\tau} > 0)$ for each $\tau \to \tau_1^*$ in P, since if $x^{s_1,s}_{\tau_1,\tau} > 0$, then there exists a τ_1-node in T having as its parent a τ-node y with $state(y) = s$. Thus, $x^{s}_{\tau} > 0$ by the definition of the assignment. Therefore, Ψ^{Σ}_{D} is consistent. Moreover, by Lemma 3.10, for every $i \in [1, k]$ and $\theta \in \Theta$, the values of variables $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in the solution are indeed $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in T.

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

$$V = \bigcup_{\tau \in E} ext(\tau).$$

Then function lab is defined as $lab(v) = \tau$ if $v \in ext(\tau)$, and function att is defined as follows:

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

It is easy to verify that these functions are well defined. Let *root* be the node labeled r, which is unique since $x_r = 1$ is in Ψ_D^Σ. Finally, to define function *ele*, we do the following. For each $x_{\tau,\tau'}^{s,s'}$ in Ψ_D^Σ, we choose $x_{\tau,\tau'}^{s,s'}$ many distinct vertices labeled τ and mark them with $x_{\tau,\tau'}^{s,s'}$. Note that every τ-element in V can be marked once and only once. Starting at *root*, for each τ-element x marked with $x_{\tau,\tau'}^{s,s'}$ for some $(\tau', s') \in Y_\tau^s$, consider $P(\tau)$ and constraints of Ψ_D^Σ.[5] If $P(\tau)$ is $\tau_1 \in E$, then we choose a distinct τ_1-element y marked with $x_{\tau_1,\tau}^{s_1,s}$ and let $ele(x) = [y]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ is in Ψ_D^Σ. If $P(\tau) = (\tau_1, \tau_2)$, then we choose a τ_1-element y_1 marked with $x_{\tau_1,\tau}^{s_1,s}$ and a τ_2-element y_2 marked with $x_{\tau_2,\tau}^{s_2,s}$ and let $ele(x) = [y_1, y_2]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ and $x_\tau^s = x_{\tau_2,\tau}^{s_2,s}$ are in Ψ_D^Σ. If $P(\tau) = (\tau_1|\tau_2)$, then we choose an element y marked with either $x_{\tau_1,\tau}^{s_1,s}$ or $x_{\tau_2,\tau}^{s_2,s}$ and let $ele(x) = [y]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s} + x_{\tau_2,\tau}^{s_2,s}$ is in Ψ_D^Σ. If $P(\tau) = \tau_1^*$, then we choose a list $[y_1, \ldots, y_n]$ ($n \geq 0$) of τ_1-elements marked with $x_{\tau_1,\tau}^{s_1,s}$ and let $ele(x) = [y_1, \ldots, y_n]$, where $(x_{\tau_1,\tau}^{s_1,s} > 0) \to (x_\tau^s > 0)$ is in Ψ_D^Σ. By the constraints in Ψ_D^Σ, each element of V can be chosen once and only once. One can verify that T defined in this way is indeed an XML tree and $T \models D$. Hence, there exists an XML tree conforming to D.

Finally, to see that, for every $i \in [1, k]$ and $\theta \in \Theta$, $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in T equal the values of variables $|nodes(\beta.\tau)|$ and $|nodes(r_\theta)|$ in the solution, respectively, it suffices to show, by Lemma 3.10, that for each node x in T, if x is marked with $x_{\tau,\tau'}^{s,s'}$ in the construction, then $state(x) = s$. Since T is a tree, there is a unique simple path $\rho \in E^*$ such that $T \models \rho(root, x)$. We show the claim by induction on the length $|\rho|$ of ρ. If $|\rho| = 1$, i.e., $\rho = r$, then x is the root and, obviously, $state(x) = \delta(s_M, r)$. Assume the claim for ρ, and we show that the claim holds for $\rho.\tau$. Let y be the τ'-element in T such that $T \models \rho(root, y)$ and y is the parent of x. Suppose that y is marked with $x_{\tau',\tau''}^{s',s''}$ in the construction. By the induction hypothesis, $state(y) = s'$. It is easy to see that $state(x) = \delta(s', \tau)$. By the definition of $\Psi_{\tau'}^{s'}$, we have that s is precisely the state $\delta(s', \tau)$. Thus $state(x) = s$. This proves the claim and thus the lemma. □

We now move to encoding $\mathcal{AC}_{K,FK}^{reg}$-constraints in terms of integer constraints. Let D be a DTD $(E, \{@l\}, P, R, r)$ and Σ a set of $\mathcal{AC}_{K,FK}^{reg}$-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 Σ, let $\beta_1.\tau_1.@l, \ldots, \beta_k.\tau_k.@l$ be an enumeration of all regular expressions and attributes that appear in Σ, and, for every function $\theta : \{1, \ldots, k\} \to \{0, 1\}$ which is not identically 0, let regular expression r_θ be defined as in (3.1). Then for every nonempty $\Omega \subseteq \Theta$, we introduce a new variable z_Ω. In any XML tree conforming to D, the intended interpretation of z_Ω is the cardinality of

$$\left(\bigcap_{\theta\,:\,\theta \in \Omega} values(r_\theta.@l) \right) \setminus \left(\bigcup_{\theta\,:\,\theta \in \Theta \setminus \Omega} values(r_\theta.@l) \right). \tag{3.2}$$

Note that the number of new variables is double-exponential in the number of regular expression in Σ. By using these variables, we define the set of *the cardinality constraints determined by* Σ, denoted by C_Σ, which consists of the following:

[5] We assume that *root* is marked with x_r^s, where $s = \delta(s_M, r)$ and s_M is the initial state of the DFA for Σ.

$$
\begin{array}{rcll}
\displaystyle\sum_{\Omega\,:\,\theta\in\Omega} z_\Omega & = & |values(r_\theta.@l)| & \text{for every } \theta\in\Theta,\\[2ex]
\displaystyle\sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset} z_\Omega & = & |values(\beta_i.\tau_i.@l)| & \text{for every } i\in[1,k],\\[2ex]
|values(\beta_i.\tau_i.@l)| & = & |nodes(\beta_i.\tau_i)| & \text{for every } \beta_i.\tau_i.@l\to\beta_i.\tau_i \text{ in } \Sigma,\\[2ex]
|values(\beta_j.\tau_j.@l)| & = & |nodes(\beta_j.\tau_j)| & \text{for every } \beta_i.\tau_i.@l\subseteq_{FK}\beta_j.\tau_j.@l \text{ in } \Sigma,\\[2ex]
\displaystyle\sum_{\substack{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\\ \Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset}} z_\Omega & = & 0 & \text{for every } \beta_i.\tau_i.@l\subseteq_{FK}\beta_j.\tau_j.@l \text{ in } \Sigma,\\[2ex]
|values(\beta_i.\tau_i.@l)| & \leq & |nodes(\beta_i.\tau_i)| & \text{for every } i\in[1,k],\\[2ex]
|values(r_\theta.@l)| & \leq & |nodes(r_\theta)| & \text{for every } \theta\in\Theta.
\end{array}
$$

Note that the size of C_Σ is double-exponential in the size of Σ.

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 Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints. Assuming that D and Σ are as above, the set $\Psi(D, \Sigma)$, called the *set of cardinality constraints determined by* D *and* Σ, is defined to be:

$$
\begin{aligned}
\Psi^\Sigma_D \;\cup\; C_\Sigma \;\cup\; \{(|nodes(\beta_i.\tau_i)| > 0) \to (|values(\beta_i.\tau_i.@l)| > 0) \mid i\in[1,k]\} \cup \\
\{(|nodes(r_\theta)| > 0) \to (|values(r_\theta.@l)| > 0) \mid \theta\in\Theta\},
\end{aligned}
$$

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

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

LEMMA 3.12. *Let D be a narrow one-attribute DTD, Σ a finite set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, and $\Psi(D, \Sigma)$ the set of cardinality constraints determined by D and Σ. Then $\Psi(D, \Sigma)$ is consistent iff 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 Ψ^Σ_D such that, for every $i \in [1, k]$ and $\theta \in \Theta$, the values of variables $|nodes(r_\theta)|$ and $|nodes(\beta_i.\tau_i)|$ in this solution coincide with $|nodes(r_\theta)|$ and $|nodes(\beta_i.\tau_i)|$ in T, respectively. From this solution, it is easy to generate a solution to $\Psi(D, \Sigma)$ by assigning to variable $|values(r_\theta.@l)|$ the size of $values(r_\theta.@l)$ in T, for every $\theta \in \Theta$, assigning to variable $|values(\beta_i.\tau_i.@l)|$ the size of $values(\beta_i.\tau_i.@l)$ in T, for every $i \in [1, k]$, and then assigning to each variable z_Ω the cardinality of set (3.2) above. It is straightforward to verify that this assignment is a solution to $\Psi(D, \Sigma)$.

Conversely, suppose that $\Psi(D, \Sigma)$ has an integer solution. We show that there is an XML tree T such that $T \models (D, \Sigma)$. By Lemma 3.11, given an integer solution to $\Psi(D, \Sigma)$, we can construct an XML tree $T' = (V, lab, ele, att', root)$ such that

$T' \models D$. Moreover, for every $i \in [1,k]$, there are exactly $n_{\beta_i.\tau_i}$ elements in T' reachable by following $\beta_i.\tau_i$, where $n_{\beta_i.\tau_i}$ is the value of the variable $|nodes(\beta_i.\tau_i)|$ in $\Psi(D, \Sigma)$, and, for every $\theta \in \Theta$, there are exactly n_{r_θ} elements in T' reachable by following r_θ, where n_{r_θ} is the value of the variable $|nodes(r_\theta)|$ in $\Psi(D, \Sigma)$. We modify the definition of the function att', while leaving V, lab, ele, and $root$ unchanged, to generate a tree $T = (V, lab, ele, att, root)$ such that $T \models (D, \Sigma)$. More specifically, we modify $att'(v, @l)$ if v is in $nodes(\beta.\tau)$ for some regular expression $\beta.\tau$ mentioned in Σ and leave $att'(v, @l)$ unchanged otherwise. To do this, for each variable z_Ω we create a set s_Ω 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 $values(r_\theta.@l)$ in T contain s_Ω iff $\theta \in \Omega$. This is possible because (1) $\sum_{\Omega : \theta \in \Omega} z_\Omega = |values(r_\theta.@l)|$ is in C_Σ for every $\theta \in \Theta$; (2) $\sum_{\Omega : \Omega \cap \{\theta | \theta(i)=1\} \neq \emptyset} z_\Omega = |values(\beta_i.\tau_i.@l)|$ is in C_Σ for every $i \in [1,k]$; (3) if $r_\theta = \emptyset$, then $|nodes(r_\theta)| = 0$ is in Ψ_D^Σ for every $\theta \in \Theta$; (4) $|values(\beta_i.\tau_i.@l)| \leq |nodes(\beta_i.\tau_i)|$ is in C_Σ for every $i \in [1,k]$; (5) $|values(r_\theta.@l)| \leq |nodes(r_\theta)|$ is in C_Σ for every $\theta \in \Theta$; and (6) $nodes(\beta)$ in T equals $nodes(\beta)$ in T' for every regular expression β over D.

We next show that T has the desired properties. It is easy to verify $T \models D$ given the construction of T from T' and the assumption $T' \models D$. By the definition of T, we have that, for every $i \in [1,k]$ and $\theta \in \Theta$, $|nodes(\beta_i.\tau_i)|$, $|values(\beta_i.\tau_i.@l)|$, $|nodes(r_\theta)|$, and $|values(r_\theta.@l)|$ in T equal the value of variables $|nodes(\beta_i.\tau_i)|$, $|values(\beta_i.\tau_i.@l)|$, $|nodes(r_\theta)|$, and $|values(r_\theta.@l)|$, respectively, given by the solution to $\Psi(D,\Sigma)$. We use this property to show that $T \models \Sigma$. Let φ be a constraint in Σ. (1) If φ is a key $\beta_i.\tau_i.@l \to \beta_i.@l$, it is immediate from the definition of T that $T \models \varphi$ since $|values(\beta_i.\tau_i.@l)| = |nodes(\beta_i.\tau_i)|$ is a constraint in C_Σ and, hence, $|values(\beta_i.\tau_i.@l)| = |nodes(\beta_i.\tau_i)|$ in T. That is, each $x \in nodes(\beta_i.\tau_i)$ in T has a distinct $@l$-attribute value, and thus the value of its $@l$-attribute uniquely identifies x among nodes in $nodes(\beta_i.\tau_i)$. (2) If φ is $\beta_i.\tau_i.@l \subseteq_{FK} \beta_j.\tau_j.@l$, it is easy to see that in T:

$$values(\beta_i.\tau_i.@l) \setminus values(\beta_j.\tau_j.@l) = \bigcup_{\Omega : \Omega \cap \{\theta | \theta(i)=1\} \neq \emptyset,\, \Omega \cap \{\theta' | \theta'(j)=1\} = \emptyset} s_\Omega.$$

Since $s_\Omega \cap s_{\Omega'} = \emptyset$ if $\Omega \neq \Omega'$,

$$|values(\beta_i.\tau_i.@l) \setminus values(\beta_j.\tau_j.@l)| = \sum_{\Omega : \Omega \cap \{\theta | \theta(i)=1\} \neq \emptyset,\, \Omega \cap \{\theta' | \theta'(j)=1\} = \emptyset} z_\Omega.$$

Thus, given that

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

is in C_Σ (since $\beta_i.\tau_i.@l \subseteq_{FK} \beta_j.\tau_j.@l \in \Sigma$), we have $|values(\beta_i.\tau_i.@l) \setminus values(\beta_j.\tau_j.@l)| = 0$ in T, that is, $values(\beta_i.\tau_i.@l) \subseteq values(\beta_j.\tau_j.@l)$ in T. Furthermore, $T \models \beta_j.\tau_j.@l \to \beta_j.\tau_j$ since $|values(\beta_j.\tau_j.@l)| = |nodes(\beta_j.\tau_j)|$ is a constraint in C_Σ. Thus $T \models \varphi$. 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 $A\vec{x} \leq \vec{b}$ of linear integer constraints together with conditions of the form $(x_i > 0) \to (x_j > 0)$, where A is an $n \times m$ matrix on integers, $\vec{b}$ is an n-vector on integers, and $1 \leq i, j \leq m$, 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 $(x > 0) \to (y > 0)$, c_k being $(x_k^1 > 0) \to (x_k^2 > 0)$. Consider 2^p instances $\mathcal{I}_j$ of integer linear programming obtained by adding, for each $k \leq p$, either $x_k^1 = 0$ or $x_k^2 > 0$ to $A\vec{x} \leq \vec{b}$. Clearly, the original system of constraints has a solution iff some $\mathcal{I}_j$ has a solution. By [27], $\mathcal{I}_j$ has a solution iff it has a solution whose size is polynomial in A, $\vec{b}$, and p. Hence, to check if the original system of constraints has a solution, it suffices to guess a system $\mathcal{I}_j$ 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 $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, it is possible to compute a one-attribute DTD D' and a set Σ' of $\mathcal{AC}^{reg}_{K,FK}$-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 Σ'_N of $\mathcal{AC}^{reg}_{K,FK}$-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 $\Psi(D'_N, \Sigma'_N)$ has a nonnegative integer solution. Thus, (D, Σ) is consistent iff $\Psi(D'_N, \Sigma'_N)$ has a nonnegative integer solution. Note that (D', Σ') can be computed in polynomial time on $|D| + |\Sigma|$, (D'_N, Σ'_N) can be computed in polynomial time on $|D'|+|\Sigma'|$, and $\Psi(D'_N, \Sigma'_N)$ can be computed in double-exponential time on $|D'_N|+|\Sigma'_N|$. Thus, by Lemma 3.13, one can check in 2-NEXPTIME whether there exists an XML tree T such that $T \models (D, \Sigma)$. ☐

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 θ be a formula of the form

$$Q_1 x_1 \ldots Q_m x_m \psi, \tag{3.3}$$

where each $Q_i \in \{\forall, \exists\}$ $(1 \leq i \leq m)$ and ψ is a propositional formula in conjunctive normal form, say, $C_1 \wedge \cdots \wedge C_n$, that mentions variables $x_1, \ldots, x_m$. We construct a DTD D_θ and a set Σ_θ of $\mathcal{AC}^{reg}_{K,FK}$-constraint such that θ is valid iff there is an XML tree conforming to D_θ and satisfying Σ_θ.

We construct a DTD $D_\theta = (E, A, P, R, r)$ as follows. $E = \{r, C\} \cup \bigcup_{i=1}^m \{x_i, \bar{x}_i, N_{x_i}, P_{x_i}\}$, $A = \{@l\}$, and P is defined by considering the quantifiers of θ. 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 $1 \leq i \leq m-1$, 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 ψ as a regular expression. Given a clause $C_j = \bigvee_{i=1}^p y_i \vee \bigvee_{i=1}^q \neg z_i$ $(j \in [1, n])$, $tr(C_j)$ is defined to be the regular expression $y_1| \ldots |y_p|\bar{z}_1| \ldots |\bar{z}_q$. 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 ϵ. We define function R as follows:

$$\begin{array}{lllllll} R(r) & = & R(P_{x_i}) & = & R(N_{x_i}) & = & \emptyset, \quad 1 \leq i \leq m, \\ R(C) & = & R(x_i) & = & R(\bar{x}_i) & = & \{@l\}, \quad 1 \leq i \leq m. \end{array}$$

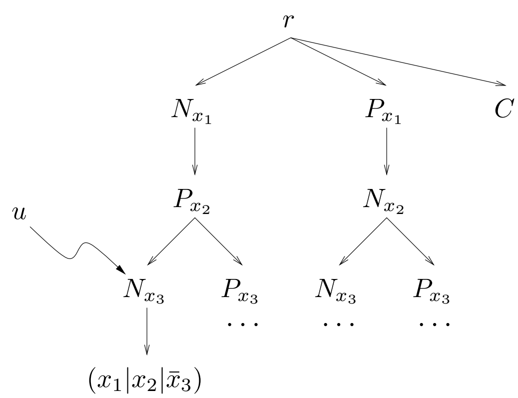


FIG. 3.2. *An XML tree conforming to the DTD constructed from* $\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee \neg x_3)$.

Finally, Σ_θ contains the following foreign keys:

$$r.\_^*.N_{x_i}.\_^*.x_i.@l \subseteq_{FK} r.C.C.@l, \qquad r.\_^*.P_{x_i}.\_^*.\bar{x}_i.@l \subseteq_{FK} r.C.C.@l, \qquad i \in [1, m].$$

For instance, for the formula $\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee \neg x_3)$, an XML tree conforming to D is shown in Figure 3.2. In this tree, a node of type N_{x_i} represents a negative value (0) for the variable x_i, and a node of type P_{x_i} 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_1} 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 σ_u: $\sigma_u(x_1) = 0$, $\sigma_u(x_2) = 1$, and $\sigma_u(x_3) = 0$. To verify that all of these assignments satisfy the formula $x_1 \vee x_2 \vee \neg x_3$ we use the set of constraints Σ_θ.

Next we prove that θ, defined in (3.3), is valid iff there is an XML tree T conforming to D_θ and satisfying Σ_θ. 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 θ 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 satisfying ψ. Let p be one of these paths, and let v be the node of type either N_{x_m} or P_{x_m} reachable from the root by following p. We define the truth assignment σ_p as follows:

$$\sigma_p(x_i) = \begin{cases} 1 & \text{if } p \text{ contains a node of type } P_{x_i}, \\ 0 & \text{otherwise.} \end{cases}$$

We have to prove that $\sigma_p(C_i) = 1$ for each $i \in [1, n]$. Given that $T \models D_\theta$, v has as a child a node v' whose type is in $tr(C_i)$. If the type of v' is x_j, then, given that $T \models r.\_^*.N_{x_j}.\_^*.x_j.@l \subseteq_{FK} r.C.C.@l$ 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 $T \models r.\_^*.P_{x_j}.\_^*.\bar{x}_j.@l \subseteq_{FK} r.C.C.@l$, p contains a node of type N_{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 θ 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 subdocuments. The latter gives rise to *relative integrity constraints* [5] that hold only on certain subdocuments. Below we define relative keys and foreign keys. Recall that we use $\mathcal{RC}$ to denote various classes of such constraints. We use the notation $x \prec y$ when x and y are two nodes in an XML tree and y is a descendant of x.

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

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

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

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

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

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

$$\begin{aligned}
country.@name &\to country,\\
country(province.@name &\to province),\\
country(capital.@inProvince &\to capital),\\
country(capital.@inProvince &\subseteq_{FK} province.@name).
\end{aligned}$$

A more general form of unary relative constraints is defined in terms of regular path expressions, along the same lines as $\mathcal{AC}^{reg}_{K,FK}$. For example, the constraints given in section 1 over the country/province/capital DTD are instances of this general form of relative constraints. Since $\mathcal{RC}_{K,FK}$ constraints are a special case of the general regular-expression relative constraints (by substituting $\_^*.\tau$ for τ), the lower bound for $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ carries over to the consistency problem for unary relative constraints defined in terms of regular path expressions.

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

THEOREM 4.1. $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ *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 to as the *positive Diophantine quadratic system problem*. An instance of the problem is

$$\begin{aligned} P_1(x_1,\dots,x_k) &= Q_1(x_1,\dots,x_k) + c_1,\\ P_2(x_1,\dots,x_k) &= Q_2(x_1,\dots,x_k) + c_2,\\ &\dots\\ P_n(x_1,\dots,x_k) &= Q_n(x_1,\dots,x_k) + c_n, \end{aligned}$$

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 monomials 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 $R(\bar{y}) = S(\bar{y})$, 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 $\mathsf{SAT}(\mathcal{RC}_{K,FK})$. 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:

$$\sum_{i=1}^{m} a_i x_{\alpha_i} + \sum_{i=m+1}^{n} a_i x_{\alpha_i} x_{\beta_i} = \sum_{i=1}^{p} b_i x_{\gamma_i} + \sum_{i=p+1}^{q} b_i x_{\gamma_i} x_{\delta_i} + o. \tag{4.1}$$

In this equation, for every $i \in [1,n]$ and $j \in [m+1,n]$, a_i is a positive integer and x_{α_i} and x_{β_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_{γ_i} and x_{δ_j} are variables, where $\gamma_i, \delta_j \in [1,k]$. Finally, o 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 Σ. Here D includes the following elements and attributes:

$$\begin{aligned} E = \{r, X, Y\} &\cup \bigcup_{i=1}^{k}\{n_i\} \cup \bigcup_{i=1}^{n}\{\alpha_i\} \cup \bigcup_{i=m+1}^{n}\{\alpha_i', \beta_i, c_i, d_i, e_i\}\\ &\cup \bigcup_{i=1}^{q}\{\gamma_i\} \cup \bigcup_{i=p+1}^{q}\{\gamma_i', \delta_i, f_i, g_i, h_i\},\\ A = \{@v\}&. \end{aligned}$$

In this DTD, r is the root. Intuitively, by 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:

$$P(r) = n_1^*, \ldots, n_k^*, \alpha_1^*, \ldots, \alpha_m^*, (\epsilon|\alpha_{m+1}), \ldots, (\epsilon|\alpha_n),$$
$$\gamma_1^*, \ldots, \gamma_p^*, (\epsilon|\gamma_{p+1}), \ldots, (\epsilon|\gamma_q), \underbrace{Y, \ldots, Y}_{o \text{ times}}.$$

We define the function P on α_i and β_i as follows:

$$\begin{array}{lcll}
P(\alpha_i) & = & \underbrace{X, \ldots, X}_{a_i \text{ times}}, & 1 \leq i \leq m, \\
P(\alpha_i) & = & (\beta_i, c_i, c_i, \underbrace{X, \ldots, X}_{a_i \text{ times}})^*, \alpha_i', & m+1 \leq i \leq n, \\
P(\gamma_i) & = & \underbrace{Y, \ldots, Y}_{b_i \text{ times}}, & 1 \leq i \leq p, \\
P(\gamma_i) & = & (\delta_i, f_i, f_i, \underbrace{Y, \ldots, Y}_{b_i \text{ times}})^*, \gamma_i', & p+1 \leq i \leq q.
\end{array}$$

To code (4.1) we need to capture the multiplication operator. To do this, we use α_i' and γ_i':

$$\begin{array}{lcll}
P(\alpha_i') & = & (\beta_i, d_i, d_i)^*, (\alpha_i|(c_i, e_i)^*), & m+1 \leq i \leq n, \\
P(\gamma_i') & = & (\delta_i, g_i, g_i)^*, (\gamma_i|(f_i, h_i)^*), & p+1 \leq i \leq q.
\end{array}$$

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

$$\begin{array}{llllllll}
P(\beta_i) = \epsilon, & m+1 \leq i \leq n, & P(\delta_i) = \epsilon, & p+1 \leq i \leq q, & P(X) = \epsilon, & \\
P(c_i) = \epsilon, & m+1 \leq i \leq n, & P(f_i) = \epsilon, & p+1 \leq i \leq q, & P(Y) = \epsilon, & \\
P(d_i) = \epsilon, & m+1 \leq i \leq n, & P(g_i) = \epsilon, & p+1 \leq i \leq q, & P(n_i) = \epsilon, & 1 \leq i \leq k, \\
P(e_i) = \epsilon, & m+1 \leq i \leq n, & P(h_i) = \epsilon, & p+1 \leq i \leq q. & &
\end{array}$$

Finally, we include the following attributes:

$$\begin{array}{ll}
R(r) = \emptyset, & \\
R(n_i) = \{@v\}, & 1 \leq i \leq k, \\
R(X) = R(Y) = \{@v\}, & \\
R(\alpha_i) = \{@v\}, & 1 \leq i \leq n, \\
R(\gamma_i) = \{@v\}, & 1 \leq i \leq q, \\
R(\beta_i) = R(c_i) = R(d_i) = R(e_i) = \{@v\}, & m+1 \leq i \leq n, \\
R(\delta_i) = R(f_i) = R(g_i) = R(h_i) = \{@v\}, & p+1 \leq i \leq q, \\
R(\alpha_i') = \emptyset, & m+1 \leq i \leq n, \\
R(\gamma_i') = \emptyset, & p+1 \leq i \leq q.
\end{array}$$

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

$$\begin{array}{ll}
r(X.@v \to X), & \\
r(\alpha_i.@v \to \alpha_i) & \text{for every } 1 \leq i \leq n, \\
r(\beta_i.@v \to \beta_i) & \text{for every } m+1 \leq i \leq n, \\
r(c_i.@v \to c_i) & \text{for every } m+1 \leq i \leq n, \\
r(d_i.@v \to d_i) & \text{for every } m+1 \leq i \leq n, \\
r(e_i.@v \to e_i) & \text{for every } m+1 \leq i \leq n, \\
r(n_i.@v \to n_i) & \text{for every } 1 \leq i \leq k, \\
r(Y.@v \to Y), & \\
r(\gamma_i.@v \to \gamma_i) & \text{for every } 1 \leq i \leq q,
\end{array}$$

$$
\begin{array}{ll}
r(\delta_i.@v \rightarrow \delta_i) & \text{for every } p+1 \leq i \leq q, \\
r(f_i.@v \rightarrow f_i) & \text{for every } p+1 \leq i \leq q, \\
r(g_i.@v \rightarrow g_i) & \text{for every } p+1 \leq i \leq q, \\
r(h_i.@v \rightarrow h_i) & \text{for every } p+1 \leq i \leq q.
\end{array}
$$

Σ contains the following absolute foreign keys:

$$
\begin{array}{lll}
r(X.@v \subseteq_{FK} Y.@v), & r(Y.@v \subseteq_{FK} X.@v), & \\
r(n_s.@v \subseteq_{FK} \alpha_i.@v), & r(\alpha_i.@v \subseteq_{FK} n_s.@v), & 1 \leq i \leq n \text{ and the value of } \alpha_i \text{ in} \\
& & \text{(4.1) is equal to } s, \\
r(n_s.@v \subseteq_{FK} e_i.@v), & r(e_i.@v \subseteq_{FK} n_s.@v), & m+1 \leq i \leq n \text{ and the value of } \beta_i \\
& & \text{in (4.1) is equal to } s, \\
r(n_s.@v \subseteq_{FK} \gamma_i.@v), & r(\gamma_i.@v \subseteq_{FK} n_s.@v), & 1 \leq i \leq q \text{ and the value of } \gamma_i \text{ in} \\
& & \text{(4.1) is equal to } s, \\
r(n_s.@v \subseteq_{FK} h_i.@v), & r(h_i.@v \subseteq_{FK} n_s.@v), & p+1 \leq i \leq q \text{ and the value of } \delta_i \\
& & \text{in (4.1) is equal to } s.
\end{array}
$$

Finally, Σ contains the following relative foreign keys:

$$
\begin{array}{lll}
\alpha_i(\beta_i.@v \subseteq_{FK} d_i.@v), & \alpha_i(d_i.@v \subseteq_{FK} \beta_i.@v), & m+1 \leq i \leq n, \\
\alpha_i'(\beta_i.@v \subseteq_{FK} c_i.@v), & \alpha_i'(c_i.@v \subseteq_{FK} \beta_i.@v), & m+1 \leq i \leq n, \\
\gamma_i(\delta_i.@v \subseteq_{FK} g_i.@v), & \gamma_i(g_i.@v \subseteq_{FK} \delta_i.@v), & p+1 \leq i \leq q, \\
\gamma_i'(\delta_i.@v \subseteq_{FK} f_i.@v), & \gamma_i'(f_i.@v \subseteq_{FK} \delta_i.@v), & p+1 \leq i \leq q.
\end{array}
$$

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

$$
|ext(X)| = \sum_{i=1}^{m} a_i v_{\alpha_i} + \sum_{i=m+1}^{n} a_i v_{\alpha_i} v_{\beta_i}, \tag{4.2}
$$

$$
|ext(Y)| = \sum_{i=1}^{p} b_i v_{\gamma_i} + \sum_{i=p+1}^{q} b_i v_{\gamma_i} v_{\delta_i} + o. \tag{4.3}
$$

Let T be an XML tree conforming to D. Then every node of type X in T appears as a child of some node of type α_i ($i \in [1, n]$). Thus, to prove (4.2) it suffices to show that the number of X-nodes that are children of some node of type α_i ($i \in [1, n]$) is equal to the ith term of (4.2), that is,

$$
\begin{array}{l}
|\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| \\
\quad = a_i v_{\alpha_i}, \quad 1 \leq i \leq m, \\
|\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| \\
\quad = a_i v_{\alpha_i} v_{\beta_i}, \quad m+1 \leq i \leq n.
\end{array}
$$

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 γ_i ($i \in [1, q]$) is equal to the ith term of (4.3). We will consider here only the case of X-nodes, the other case being similar.

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

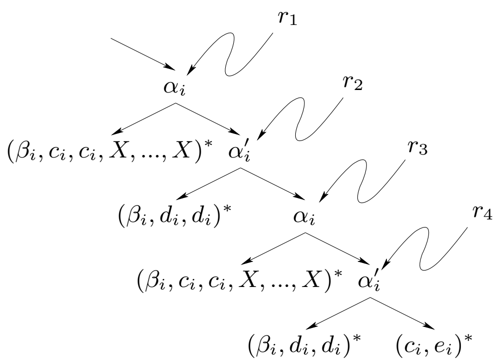


FIG. 4.1. *Part of the XML tree used in the proof of Theorem* 4.1.

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

Given that r_3 is a node of type α_i and $\alpha_i(\beta_i.@v \subseteq_{FK} d_i.@v)$ and $\alpha_i(d_i.@v \subseteq_{FK} \beta_i.@v)$ are in Σ, $|\{x \mid x \text{ is a child of } r_3 \text{ of type } \beta_i\}| = v_t$, since there are $2v_t$ descendants of r_3 of type d_i and v_t children of r_4 of type β_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 $2v_t$. We can use the same argument to prove that the number of children of r_2 of types β_i and d_i are v_t and $2v_t$, 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 $2a_i v_t$. 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 α_i is $v_s a_i v_t$, since there are v_s nodes of type α_i in T. This concludes the proof, since $|\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| = a_i v_s v_t$. ∎

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

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

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 $f : E' \to E$ such that, for every $\tau_1, \tau_2 \in E'$, we have $R'(\tau_1) = R'(\tau_2)$ 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 τ 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 (monadic second-order logic) definable trees and the regular tree languages of finite unranked trees [26, 29].

A constraint φ is said to be defined over extended DTD ED if every element type τ mentioned in φ is in E.

The consistency problem for extended DTDs is defined exactly as in the case of DTDs: given a specification (ED, Σ), the problem is to verify whether there exists a tree T conforming to ED and satisfying Σ. 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 Σ be either a set of $\mathcal{AC}^{*,*}_{K,FK}$-constraints over ED or a set of $\mathcal{AC}^{reg}_{K,FK}$-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

$$P_{ED}(\tau) = \begin{cases} f(\tau), P'(\tau), & \tau \in E', \\ \epsilon, & \tau \in E, \end{cases} \qquad R_{ED}(\tau) = \begin{cases} \emptyset, & \tau \in E', \\ R'(\tau'), & \tau \in E \text{ and } f(\tau') = \tau. \end{cases}$$

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 Σ is a set of $\mathcal{AC}^{*,*}_{K,FK}$-constraints, then $\Gamma(ED, \Sigma) = \Sigma$. Otherwise, Σ is a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints, and for every $\varphi \in \Sigma$ we have the following constraint ψ in $\Gamma(ED, \Sigma)$. For every element type $\tau \in E$, define the image of substitution h on τ 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 φ is a key $\beta.\tau.@l \to \beta.\tau$, then $\psi = h(\beta.\tau).\tau.@l \to h(\beta.\tau).\tau$. If φ is a foreign key $\beta_1.\tau_1.@l_1 \subseteq_{FK} \beta_2.\tau_2.@l_2$, then $\psi = h(\beta_1.\tau_1).\tau_1.@l_1 \subseteq_{FK} h(\beta_2.\tau_2).\tau_2.@l_2$. The following simple lemma shows that the consistency problems for (ED, Σ) and $(D(ED), \Gamma(ED, \Sigma))$ are equivalent.

LEMMA 5.1. *Let $ED = (D', f, E)$ be an extended DTD and Σ be either a set of $\mathcal{AC}^{*,*}_{K,FK}$-constraints over ED or a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over ED. Then (ED, Σ) is consistent iff $(D(ED), \Gamma(ED, \Sigma))$ is consistent.*

We note that if $\mathcal{C}$ is one of $\mathcal{AC}_{PK,FK}$, $\mathcal{AC}_{K,FK}$, $\mathcal{AC}^{*,1}_{PK,FK}$, and $\mathcal{AC}^{reg}_{K,FK}$ and Σ is a set of $\mathcal{C}$-constraints over an extended DTD ED, then $\Gamma(ED, \Sigma)$ is a set of $\mathcal{C}$-constraints over $D(ED)$. Thus, given that $D(ED)$ and $\Gamma(ED, \Sigma)$ can be constructed in polynomial time from (ED, Σ), 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 $\mathcal{AC}_{K,FK}$-constraints ($\mathcal{AC}_{PK,FK}$-constraints) is NP-complete.*

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

(c) *The consistency problem for extended DTDs and $\mathcal{AC}^{reg}_{K,FK}$-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 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

Table 6.1
Complexity of the consistency problem for absolute constraints.

Class	$\mathcal{AC}^{*,*}_{K,FK}$ [16]	$\mathcal{AC}^{*,1}_{PK,FK}$	$\mathcal{AC}^{reg}_{K,FK}$	$\mathcal{AC}_{K,FK}$ [16]
Description	Multiattribute keys and foreign keys	Primary multiattribute keys, unary foreign keys	Unary regular path constraints (keys, foreign keys)	Unary keys and foreign keys
Upper bound	Undecidable	NEXPTIME	2-NEXPTIME	NP
Lower bound	Undecidable	NP	PSPACE	NP

Table 6.2
Complexity of the consistency problem for relative constraints.

Class	$\mathcal{RC}^{*,*}_{K,FK}$ [16]	$\mathcal{RC}_{K,FK}$	$\mathcal{RC}_{PK,FK}$
Description	Multiattribute keys and foreign keys	Unary keys and foreign keys	Primary unary keys and foreign keys
Lower bound	Undecidable	Undecidable	Undecidable

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 Tables 6.1 and 6.2 (we also included the main results from [16] in those tables 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 of the keys as well. However, if regular expressions are allowed in single-attribute constraints, the lower bounds become at least PSPACE. For relative constraints, which are required to hold only 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, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ was proved to be equivalent to a problem related to Diophantine equations whose exact complexity remains unknown. In the case of $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, we think that it is more likely that our lower bounds correspond to the exact complexity of those problems. However, the algorithms are quite involved, and we do not yet see a way to simplify them to prove the matching upper bounds.

Another topic for future work is to study the interaction between more complex XML constraints, e.g., those defined in terms of XPath [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.

REFERENCES

[1] S. Abiteboul and V. Vianu, *Regular path queries with constraints*, J. Comput. System Sci., 58 (1999), pp. 428–452.

[2] C. K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu, *XML-based information mediation with MIX*, in Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, Philadelphia, PA, 1999, pp. 597–599.

[3] C. Beeri and T. Milo, *Schemas for integration and translation of structured and semi-structured data*, in Proceedings of the 7th International Conference on Database Theory, Lecture Notes in Comput. Sci. 1540, Springer, New York, 1999, pp. 296–313.

[4] M. Benedikt, C. Y. Chan, W. Fan, J. Freire, and R. Rastogi, *Capturing both types and constraints in data integration*, in Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, CA, 2003, pp. 277–288.

[5] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan, *Keys for XML*, Comput. Networks, 39 (2002), pp. 473–487.

[6] P. Buneman, W. Fan, and S. Weinstein, *Path constraints in semistructured databases*, J. Comput. System Sci., 61 (2000), pp. 146–193.

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

[8] D. Calvanese and M. Lenzerini, *Making object-oriented schemas more expressive*, in Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Minneapolis, MN, 1994, pp. 243–254.

[9] D. Calvanese and M. Lenzerini, *On the interaction between ISA and cardinality constraints*, in Proceedings of the 10th International Conference on Data Engineering, IEEE, Piscataway, NJ, 1994, pp. 204–213.

[10] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian, *XPERANTO: Publishing object-relational data as XML*, in International Workshop on the Web and Databases (Informal Proceedings), Lecture Notes in Comput. Sci. 1997, Springer, New York, 2000, pp. 105–110.

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

[12] S. B. Davidson, W. Fan, and C. S. Hara, *Propagating XML constraints to relations*, J. Comput. System Sci., 73 (2007), pp. 316–361.

[13] A. Deutsch, L. Popa, and V. Tannen, *Physical data independence, constraints, and optimization with universal plans*, in Proceedings of the 25th International Conference on Very Large Data Bases, Morgan Kaufmann, San Francisco, 1999, pp. 459–470.

[14] A. Deutsch and V. Tannen, *Reformulation of XML queries and constraints*, in Proceedings of the 9th International Conference on Database Theory, Lecture Notes in Comput. Sci. 2572, Springer, New York, 2003, pp. 225–241.

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

[16] W. Fan and L. Libkin, *On XML integrity constraints in the presence of DTDs*, J. ACM, 49 (2002), pp. 368–406.

[17] M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu, *Verifying integrity constraints on web sites*, in Proceedings of the 16th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Francisco, 1999, pp. 614–619.

[18] M. F. Fernandez, A. Morishima, D. Suciu, and W. C. Tan, *Publishing relational data in XML: The SilkRoute approach*, IEEE Data Engineering Bulletin, 24 (2001), pp. 12–19.

[19] D. Florescu and D. Kossmann, *Storing and querying XML data using an RDMBS*, IEEE Data Engineering Bulletin, 22 (1999), pp. 27–34.

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

[21] R. Givan, D. A. McAllester, C. Witty, and D. Kozen, *Tarskian set constraints*, Inform. and Comput., 174 (2002), pp. 105–131.

[22] P. C. Kanellakis, *On the computational complexity of cardinality constraints in relational databases*, Inform. Process. Lett., 11 (1980), pp. 98–101.

[23] D. Lee and W. W. Chu, *Constraints-preserving transformation from XML document type definition to relational schema*, in Proceedings of the 19th International Conference on Conceptual Modeling, Lecture Notes in Comput. Sci. 1920, Springer, New York, 2000, pp. 323–338.

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

[25] F. Neven, *Extensions of attribute grammars for structured document queries*, in Proceedings of the 7th International Workshop on Database Programming Languages, Lecture Notes in Comput. Sci. 1949, Springer, New York, 1999, pp. 99–116.

[26] F. Neven and T. Schwentick, *Query automata over finite trees*, Theoret. Comput. Sci., 275 (2002), pp. 633–674.

[27] C. Papadimitriou, *On the complexity of integer programming*, J. ACM, 28 (1981), pp. 765–768.
[28] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.
[29] Y. Papakonstantinou and V. Vianu, *DTD inference for views of XML data*, in Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Dallas, TX, 2000, pp. 35–46.
[30] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald, *Efficiently publishing relational data as XML documents*, The VLDB Journal, 10 (2001), pp. 133–154.
[31] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. DeWitt, and J. Naughton, *Relational databases for querying XML documents: Limitations and opportunities*, in Proceedings of the 25th International Conference on Very Large Data Bases, Morgan Kaufmann, San Francisco, 1999, pp. 302–314.
[32] J. W. Thatcher, *Characterizing derivation trees of context-free grammars through a generalization of finite automata theory*, J. Comput. System Sci., 1 (1967), pp. 317–322.
[33] J. Ullman, *Principles of Database and Knowledge-Base Systems, Volume* I, Computer Science Press, Rockville, MD, 1988.
[34] W3C, *Document Object Model (DOM) Level* 1 *Specification*, W3C Recommendation, October 1998, http://www.w3.org/TR/REC-DOM-Level-1/.
[35] W3C, *Extensible Markup Language (XML)* 1.0 (third edition), W3C Recommendation, February 2004, http://www.w3.org/TR/2004/REC-xml-20040204/.
[36] W3C, *XML-Data*, W3C Note, January 1998, http://www.w3.org/TR/1998/NOTE-XML-data-0105/.
[37] W3C, *XML Path Language (XPath) Version* 1.0, W3C Recommendation, November 1999, http://www.w3.org/TR/xpath.
[38] W3C, *XML Schema*, W3C Recommendation, October 2004, http://www.w3.org/XML/Schema.
[39] W3C, *XQuery* 1.0 : *An XML Query Language*, W3C Working Draft 29, October 2004, http://www.w3.org/TR/2004/WD-xquery-20041029/.
[40] W3C, *XSL Transformations (XSLT) Version* 1.0, W3C recommendation, November 1999, http:// www.w3.org/TR/xslt.