On Verifying Consistency of XML Specifications

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

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

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

Abstract

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

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

1 Introduction

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

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

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

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

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

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

*Research affiliation: Bell Laboratories.

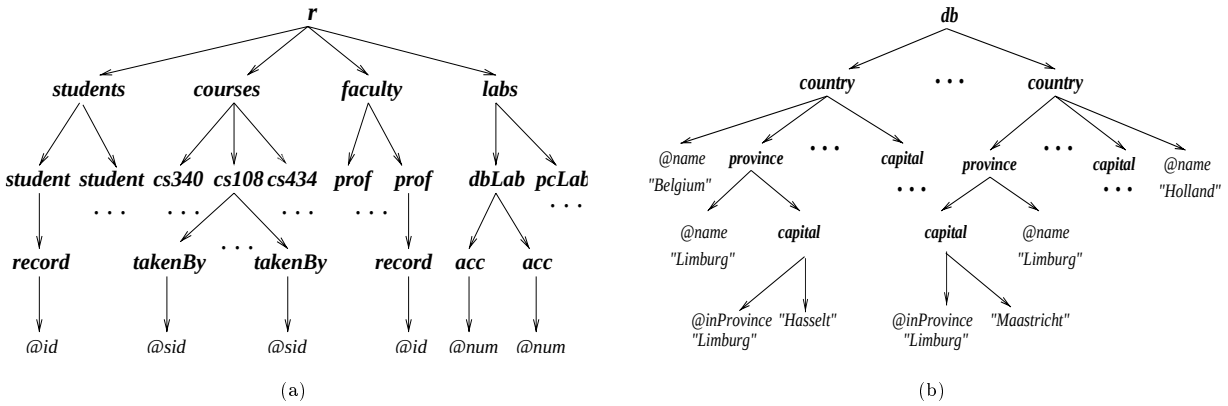


Figure 1: Examples of XML documents

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

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

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

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

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

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

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

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

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

$$r.faculty.prof.record.id \subseteq r.\_^*.dbLab.acc.num,$$
$$r.\_^*.dbLab.acc.num \rightarrow r.\_^*.dbLab.acc.$$

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

in the document. Thus no XML document both conforms to the DTD and satisfies all the constraints.

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

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

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

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

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

$country.name \rightarrow country,$
$country(province.name \rightarrow province),$
$country(capital.inProvince \rightarrow capital),$
$country(capital.inProvince \subseteq province.name).$

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

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

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

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

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

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

A number of trivial restrictions for tractability of absolute constraints are known (e.g., a fixed DTD, no

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

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

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

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

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

2 Notations

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

Definition 2.1 *A* DTD *(Document Type Definition) is defined to be* $D = (E, A, P, R, r)$*, where:*

- E *is a finite set of* element types*;*
- A *is a finite set of* attributes*, disjoint from* E*;*
- *for each* $\tau \in E$*,* $P(\tau)$ *is a regular expression* α*, called the* element type definition *of* τ*:*

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

  where S *denotes the* string *type,* $\tau' \in E$*,* ϵ *is the empty word, and "*$|$*", "*$,$*" and "*$*$*" denote union, concatenation, and the Kleene closure;*
- *for each* $\tau \in E$*,* $R(\tau)$ *is a set of attributes in* A*;*
- $r \in E$ *and is called* the element type of the root.

We normally denote element types by τ and attributes by l, and assume that r does not appear in $P(\tau)$ for any $\tau \in E$. We also assume that each τ in $E \backslash \{r\}$ is *connected to* r, i.e., either τ appears in $P(r)$, or it appears in $P(\tau')$ for some τ' that is connected to r.

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

Definition 2.2 *Let* $D = (E, A, P, R, r)$ *be a DTD. An* XML tree T conforming to D*, written* $T \models D$*, is defined to be* $(V, lab, ele, att, val, root)$*, where*

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

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

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

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

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

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

We say that a DTD is *non-recursive* if $Paths(D)$ is finite, and recursive otherwise. We also say that D is a *no-star* DTD if the Kleene star does not occur in any regular expression $P(\tau)$ (note that this is a stronger restriction than being $*$-free: a regular expression without the Kleene star yields a finite language, while the language of a $*$-free regular expression may still be infinite as it allows boolean operators including complement).

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

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

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

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

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

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

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

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

Consistency, or satisfiability problem We are interested in the consistency, or satisfiability problem for XML constraints considered together with DTDs: that is, whether a given set of constraints and a DTD are satisfiable by an XML tree. Formally, for a class $\mathcal{C}$ of integrity constraints we define the *XML specification consistency problem* $\mathsf{SAT}(\mathcal{C})$ as follows:

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

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

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

3 Absolute integrity constraints

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

3.1 Multi-attribute keys

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

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Regular expression constraints

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

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

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

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

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

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

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

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

Theorem 3.4

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

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

Proof sketch. a) Following [14], we code both the DTD and the constraints with linear inequalities over integers. However, compared to the proof of [14], the current proof is considerably harder due to the following. First, regular expressions in DTDs ("horizontal" regular expressions) interact in a certain way with regular path expressions in constraints (those correspond to "vertical" paths through the trees). To eliminate this interaction, we first reduce the problem to that over certain simple DTDs. The next problem is that regular path expressions in constraints can interact with each other. To model them with linear inequalities, we must introduce exponentially many variables that account for all possible Boolean combinations of those regular languages. The last problem is coding the DTDs in such a way that variables corresponding to each node have the information about the path leading to the node, and its relationship with the regular path expressions used in constraints. For that, we adopt the technique of [2], and tag all the variables in the coding of DTDs with states of the product automaton for all the automata corresponding to the regular expressions in constraints. Putting

everything together, we reduce $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ to the existence of a solution of an (almost) instance of linear integer programming, which happens to be of exponential size; hence the NEXPTIME bound.

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

3.3 Restrictions for tractability

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

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

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

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

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

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

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

3.4 Lower bounds for implication

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

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

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

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

4 Relative integrity constraints

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

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

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

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

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

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

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

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

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

4.1 Undecidability of consistency

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

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

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

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

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

4.2 Decidable hierarchical constraints

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

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

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

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

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

$$\mathit{library}(\mathit{author\_info.name} \rightarrow \mathit{author\_info}),$$
$$\mathit{library}(\mathit{author.name} \subseteq \mathit{author\_info.name}).$$

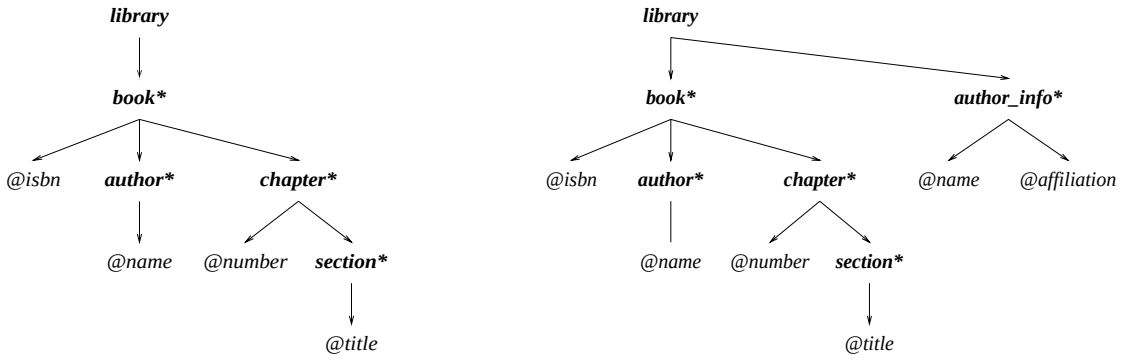
In this case, nodes of type *author* are restricted from context types *library* and *book*. Thus, we cannot separate the consistency problems for nodes of types *library* and *book*.

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

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

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

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



(a) A hierarchical structure

(b) A non-hierarchical structure

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

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

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

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

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

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

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

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

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

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

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

4.3 Implication problem

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

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

5 Conclusion

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

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

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

Figure 3: Complexity of the consistency problem for absolute constraints

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

Figure 4: Complexity of the consistency problem for relative constraints

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

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

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

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

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

APPENDIX

Proofs

Proof of Theorem 3.1

We start by showing a reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE. As the first step, we encode $\mathcal{AC}^{*,1}_{PK,FK}$ constraints by a prequadratic Diophantine system.

Let D be a DTD (E, A, P, R, r) 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 any $\varphi \in \Sigma$,

- if φ is a key constraint $\tau[l_1, \ldots, l_k] \rightarrow \tau$, then C_Σ contains $|ext(\tau)| \leq |ext(\tau.l_1)| \cdot \ldots \cdot |ext(\tau.l_k)|$;
- if φ is a unary foreign key $\tau_1.l_1 \subseteq \tau_2.l_2$ and $\tau_2.l_2 \rightarrow \tau_2$, then C_Σ contains $|ext(\tau_1.l_1)| \leq |ext(\tau_2.l_2)|$ and $|ext(\tau_2)| \leq |ext(\tau_2.l_2)|$.

In addition, for any $\tau \in E$ and $l \in R(\tau)$, $|ext(\tau.l)| \leq |ext(\tau)|$ and $0 \leq |ext(\tau.l)|$ are in C_Σ. Observe that for a unary key $\tau.l \rightarrow \tau$ we have both $|ext(\tau.l)| \leq |ext(\tau)|$ and $|ext(\tau)| \leq |ext(\tau.l)|$ in C_Σ. Thus C_Σ assures $|ext(\tau)| = |ext(\tau.l)|$.

We write $T \models C_\Sigma$ if T satisfies all the constraints of C_Σ. Note that C_Σ is equivalent (in fact, can be converted in linear 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 linear 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 1 *Let D be a DTD, Σ be a set of $\mathcal{AC}^{*,1}_{PK,FK}$ constraints over D, and C_Σ be the set of cardinality constraints determined by Σ. Then there exists an XML tree T_1 such that $T_1 \models D$ and $T_1 \models \Sigma$ iff there exists an XML tree T_2 such that $T_2 \models D$ and $T_2 \models C_\Sigma$. In addition, any XML tree valid w.r.t. D and satisfying Σ also satisfies C_Σ.* □

Proof. It is easy to see that for any 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 $T_2 = (V, lab, ele, att, val, root)$ such that $T_2 \models D$ and $T_2 \models C_\Sigma$, then we can construct an XML tree T_1 such that $T_1 \models D$ and $T_1 \models \Sigma$.

We construct T_1 from T_2 by modifying the function val while leaving V, lab, ele, att and $root$ unchanged. More specifically, we modify $val(v)$ if $lab(v) \in A$, i.e., if v is an attribute, and leave $val(v)$ unchanged otherwise. Let $S = \{\tau.l \mid \tau \in E, l \in R(\tau)\}$. To define the new function, denoted by val', we first associate a set of string values with each $\tau.l$ in S. Let N be the maximum cardinality of $ext(\tau.l)$ in T_2, i.e., $N \geq |ext(\tau.l)|$ 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 $V_{\tau.l} = \{a_i \mid i \in [1, |ext(\tau.l)|]\}$, and for each $x \in ext(\tau)$, let $val'(att(x, l))$ be a string value in $V_{\tau.l}$ such that in T_1, $ext(\tau.l) = V_{\tau.l}$. In addition, for each key $\tau[l_1, \ldots, l_k] \rightarrow \tau$ in Σ, let $x[l_1, \ldots, l_k]$ be a distinct list of string values from $V_{\tau.l_1} \times \ldots \times V_{\tau.l_k}$. This is possible because by the definition of T_1, (1) $ext(\tau)$ in T_1 equals $ext(\tau)$ in T_2; (2) $|ext(\tau.l)|$ in T_1 equals $|ext(\tau.l)|$ in T_2; (3) $T_2 \models C_\Sigma$ and $|ext(\tau)| \leq |ext(\tau.l_1)| \cdot \ldots \cdot |ext(\tau.l_k)|$ is in C_Σ; and (4) since φ is the only key defined for τ elements, the population of the attributes $l_1, \ldots, l_k$ of x is independent of the population of any other attributes of x. 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 $\tau_1.l_1 \subseteq \tau_2.l_2$. 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 $\tau[l_1, \ldots, l_k] \rightarrow \tau$, 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 $V_{\tau.l_1} \times \ldots \times V_{\tau.l_k}$. (2) If φ is $\tau_1.l_1 \subseteq \tau_2.l_2$, then $T_2 \models |ext(\tau_1.l_1)| \leq |ext(\tau_2.l_2)|$ by $T_2 \models C_\Sigma$. Recall that by the definition of val', for $i \in [1, 2]$, $V_{\tau_i.l_i} = \{a_i \mid i \in [1, |ext(\tau_i.l_i)|]\}$ and in T_1, $ext(\tau_i.l_i) = V_{\tau_i.l_i}$. Thus $ext(\tau_1.l_1) \subseteq ext(\tau_2.l_2)$ in T_1. That is, $T_1 \models \varphi$. Therefore, $T_1 \models D$ and $T_1 \models \Sigma$. □

The above lemma takes care of coding the constraints; the next step is to code DTDs. For that, we use the technique developed in [14]: for each DTD D, one can compute in linear 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 $|ext(\tau.l)|$ as a variable for any attribute l of τ. Moreover, it has the following properties [14]:

- if Ψ_D has a nonnegative integer solution, then there exists an XML tree T valid w.r.t. 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;
- if there exists an XML tree T valid w.r.t. D then Ψ_D has a nonnegative integer solution such that the value of the variable $|ext(\tau)|$ in the solution equals the cardinality of $ext(\tau)$ in T.

We now combine this coding with the coding for $\mathcal{AC}^{*,1}_{PK,FK}$ constraints. Given a DTD D and a finite 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

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

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 $(|ext(\tau)| > 0) \rightarrow (|ext(\tau.l)| > 0)$ are to ensure that every τ element has an l attribute. Note that $|ext(\tau.l)| \leq |ext(\tau)|$ 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.

We say that $\Psi(D, \Sigma)$ is *consistent* if and only if $\Psi(D, \Sigma)$ admits a nonnegative integer solution. That is, there is a nonnegative integer assignment to the variables in $\Psi(D, \Sigma)$ such that all the constraints in $\Psi(D, \Sigma)$ are satisfied. The system is computable in linear time in the size of D and Σ.

Lemma 2 *Let D be a DTD, Σ be a finite set of $\mathcal{AC}^{*,1}_{PK,FK}$ constraints over D, and $\Psi(D, \Sigma)$ be the set of cardinality constraints determined by D and Σ. Then $\Psi(D, \Sigma)$ is consistent if and only if there is an XML tree T such that $T \models D$ and $T \models \Sigma$.* □

Proof. Suppose that there exists an XML tree T such that $T \models D$ and $T \models \Sigma$. 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. By Lemma 1 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 $|ext(\tau.l)|$ 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 the constraints in C_Σ. In addition, if $|ext(\tau)| > 0$ then $|ext(\tau.l)| > 0$ since every τ element in T has an l attribute. Hence the assignment is indeed a nonnegative solution to $\Psi(D, \Sigma)$. This is, $\Psi(D, \Sigma)$ 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. We construct a new tree T' from T by modifying the definition of the function val 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 $|ext(\tau.l)|$ in the solution. This is possible since $|ext(\tau.l)| \leq |ext(\tau)|$ is in C_Σ, and the assignment is also a solution to C_Σ. Thus $T' \models D$ and $T' \models C_\Sigma$. Observe that since it is a solution of $\Psi(D, \Sigma)$, we have $|ext(\tau.l)| > 0$ if $|ext(\tau)| > 0$, and thus every τ element in T'' can have an l attribute. Hence by Lemma 1, there exists an XML tree T'' such that $T'' \models D$ and $T'' \models \Sigma$. □

We now conclude the proof of reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE. By Lemma 2, given any DTD D and any finite set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$ constraints, there exists an XML tree T such that $T \models D$ and $T \models \Sigma$ iff $\Psi(D, \Sigma)$ has a nonnegative integer solution. Such a solution requires $|ext(\tau.l)| > 0$ if $|ext(\tau)| > 0$. To ensure this, let Φ

be a system that includes all linear integer constraints and prequadratic constraints in $\Psi(D, \Sigma)$ and moreover, $|ext(\tau)| \leq |ext(\tau.l)| \cdot |ext(\tau)|$ for each constraint $(|ext(\tau)| > 0) \rightarrow (|ext(\tau.l)| > 0)$ in $\Psi(D, \Sigma)$. Observe that Φ is a prequadratic Diophantine system. In addition, $\Psi(D, \Sigma)$ has a nonnegative integer solution iff Φ has a nonnegative integer solution. To see this, observe that for any nonnegative integer assignment to $|ext(\tau)|$ and $|ext(\tau.l)|$, $(|ext(\tau)| > 0) \rightarrow (|ext(\tau.l)| > 0)$ iff $|ext(\tau)| \leq |ext(\tau.l)| \cdot |ext(\tau)|$. Indeed, if a nonnegative solution satisfies $|ext(\tau)| \leq |ext(\tau.l)| \cdot |ext(\tau)|$ then it cannot assign 0 to the variable $|ext(\tau.l)|$ if $|ext(\tau)| > 0$. Conversely, if a solution satisfies $(|ext(\tau)| > 0) \rightarrow (|ext(\tau.l)| > 0)$ then it is obvious that $|ext(\tau)| \leq |ext(\tau.l)| \cdot |ext(\tau)|$ holds as long as the solution consists of nonnegative integers. Thus D and Σ are consistent iff the prequadratic Diophantine system Φ has a nonnegative integer solution. Observe that Φ can be computed in linear time in the size of $\Psi(D, \Sigma)$, and $\Psi(D, \Sigma)$ can be computed in linear time in the size of D and Σ (denoted by n). Hence it takes $O(n)$ time to compute Φ. Therefore, there is a PTIME reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE.

We now move to the other direction, a reduction from PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. Without loss of generality, we show it for the case of PDE instances with a nonnegative matrix A and a nonnegative vector $\vec{b}$, since we can code any linear inequality in the same way and thus the proof extends straightforwardly to arbitrary instance of PDE.

Given an instance of PDE, i.e., a system S consisting of a set S_l of linear equations/inequalities on nonnegative integers and a set S_p of prequadratic constraints of the form $x_i \leq x_j \cdot x_k$, 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 the variables in S. Assume that $S_l = \{e_j \mid j \in [1, m]\}$ and e_j has one of the following forms:

$$a^j_1\ x_1 + \ldots + a^j_n\ x_n\ \ op\ \ b_j,$$

where op is either "$\leq$" or "$\geq$", and b_j and a^j_i are nonnegative integers.

We define the DTD $D = (E,\ A,\ P,\ R,\ r)$ as follows:

(1) For each variable x_i, we define a distinct element type X_i and moreover, distinct element types $CX_{i,j}, DX_{i,j}$ for each $j \in [1, m]$. In addition, for each $p = x_i \leq x_s \cdot x_k$ in S_p, we define distinct element types X^p_i, NX^p_i, and distinct element types $CX^p_{i,j}, DX^p_{i,j}$ for each $j \in [1, m]$. That is, for each prequadratic equation $x_i \ \leq\ x_s \cdot x_k$ in S_p, we create a distinct copy of X_i. For each linear constraint e_j, we define distinct element types E_j, U_j, B_j and a distinct element type $U_{i,j}$ for each $i \in [1, n]$. We use r to denote the root element type. That is,

$$\begin{aligned} E \ = \ & \{r\} \\ & \cup \{E_j, U_j, B_j \mid j \in [1, m]\} \\ & \cup \{X_i \mid i \in [1, n]\} \\ & \cup \{CX_{i,j}, DX_{i,j}, U_{i,j} \mid i \in [1, n],\ j \in [1, m]\} \\ & \cup \{X^p_i, NX^p_i \mid p = x_i \leq x_j \cdot x_k \in S_p\} \\ & \cup \{CX^p_{i,j}, DX^p_{i,j} \mid j \in [1, m],\ p = x_i \leq x_s \cdot x_k \in S_p\} \end{aligned}$$

(2) $A \ = \ \{l\} \cup \{l_{i,j}, l_{i,k} \mid (x_i \leq x_j \cdot x_k) \in S_p\}$. That is, A has an attribute l and moreover, for each prequadratic constraint $x_i \leq x_j \cdot x_k$ in S_p, A includes two extra attributes $l_{i,j}$ and $l_{i,k}$.

(3) We define production rules as follows:

$$\begin{array}{lll} P(r) \ = \ E_1, \ldots, E_m,\ (X_1)^*, \ldots, (X_n)^*, (X^{p_1}_1)^*, \ldots, (X^{p_n}_n)^*, & \text{/* for each } p_i = x_i \ \leq\ x_j \cdot x_k \text{ in } S_p \\ P(E_j) \ = \ B_j, \ldots, B_j,\ (U_{1,j})^*, \ldots, (U_{n,j})^* & \text{/* } b_j \text{ many occurrences of } B_j\text{; for each } e_j \in S_l \\ P(U_{i,j}) \ = U_j & \text{/* for all } i \in [1, n] \text{ and } j \in [1, m] \\ P(U_j) \ = \ \epsilon & \text{/* for all } j \in [1, m] \\ P(X_i) \ = \ CX_{i,1}, \ldots, CX_{i,m} & \text{/* for all } i \in [1, n] \\ P(CX_{i,j}) \ = \ DX_{i,j}, \ldots, DX_{i,j} & \text{/* } a^i_j \text{ many occurrences of } DX_{i,j}\text{;} \\ & \text{/* for all } i \in [1, n] \text{ and } j \in [1, m] \\ P(DX_{i,j}) \ = \ \epsilon & \text{/* for all } i \in [1, n] \text{ and } j \in [1, m] \\ P(X^p_i) \ = \ CX^p_{i,1}, \ldots, CX^p_{i,m},\ NX^p_i & \text{/* for all } p = x_i \ \leq\ x_j \cdot x_k \text{ in } S_p \\ P(CX^p_{i,j}) \ = \ DX^p_{i,j}, \ldots, DX^p_{i,j} & \text{/* } a^i_j \text{ many occurrences of } DX_{i,j}, \end{array}$$

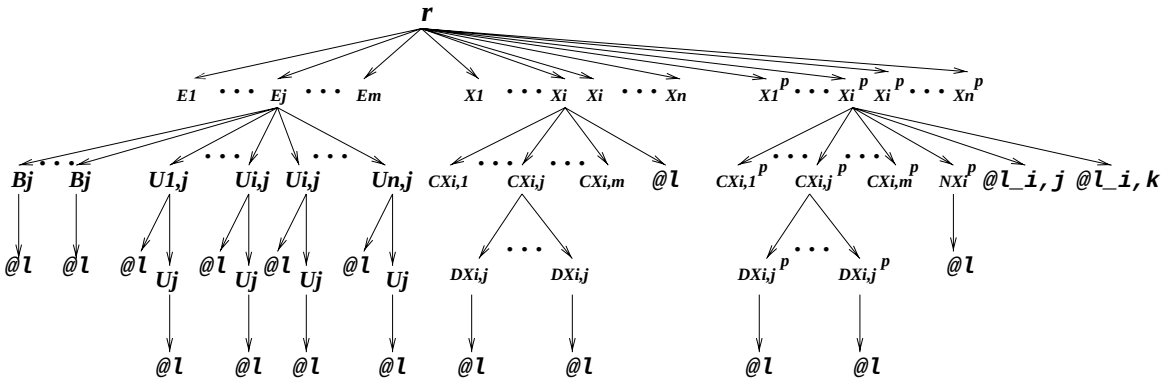


Figure 5: A tree used in the proof of Theorem 3.1

$P(DX^p_{i,j}) = \epsilon$ /* for all $p = x_i \leq x_s \cdot x_k$ in S_p and $j \in [1, m]$;
/* for all $p = x_i \leq x_s \cdot x_k$ in S_p and $j \in [1, m]$

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. We encode the coefficient a^j_i with $DX_{i,j}$ and the term $a^j_i\ x_i$ with $CX_{i,j}$. The value of $a^j_i\ x_i$ is encoded with $|ext(DX_{i,j})|$ under X_i. Moreover, for each prequadratic equation $p = x_i \leq x_j \cdot x_k$ in S_p, we give the same treatment to the copy X^p_i of X_i. We encode $|ext(X^p_i)|$ with $|ext(NX^p_i)|$. Indeed, by $P(X^p_i)$, in any XML tree valid w.r.t. D, $|ext(X^p_i)| = |ext(NX^p_i)|$. The reason to use X^p_i and NX^p_i is to ensure that the set Σ of $\mathcal{AC}^{*,1}_{K,FK}$ constraints defined below is primary. We use E_j to code the linear constraint e_j in S_l. More specifically, we encode the value of $a^j_i\ x_i$ with $|ext(U_{i,j})|$, b_j with $|ext(B_j)|$ and the value of $a^j_1\ x_1 + \ldots + a^j_n\ x_n$ with $|ext(U_j)|$.

(4) We define the attribute function R as follows: for each of X_i, NX^p_i, U_j, B_j, $U_{i,j}$ and $DX_{i,j}$, let l be its attribute. In addition, for each $p = x_i \leq x_j \cdot x_k$ in S_p, let $R(X^p_i)$ consist of $l_{i,j}$ and $l_{i,k}$. For all other element type τ, let $R(\tau)$ be empty. Intuitively, we shall define l attribute as a key and use $l_{i,j}$ and $l_{i,k}$ to code prequadratic constraint $x_i \leq x_j \cdot x_k$.

An XML tree conforming to D has the form shown in Figure 5.

Given the DTD D, we define a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$ constraints over D.

(1) For each τ ranging over X_i, NX^p_i, U_j, B_j, $U_{i,j}$, $DX_{i,j}$ and $DX^p_{i,j}$, Σ includes a key:

$$\tau.l \rightarrow \tau.$$

That is, l is a key of τ.

(2) For each $i \in [1, n]$, $j \in [1, m]$ and $p = x_i \leq x_s \cdot x_k$ in S_p, Σ includes

$$U_{i,j}.l \subseteq DX_{i,j}.l, \qquad DX_{i,j}.l \subseteq U_{i,j}.l, \qquad U_{i,j}.l \subseteq DX^p_{i,j}.l, \qquad DX^p_{i,j}.l \subseteq U_{i,j}.l.$$

These ensure that the encoding of the term $a^j_i\ x_i$ are consistent.

(3) For each $e_j = a^j_1\ x_1 + \ldots + a^j_n\ x_n\ op\ b_j$, if op is "$\leq$" then Σ includes

$$U_j.l \subseteq B_j.l,$$

and if op is "$\geq$" then Σ includes

$$B_j.l \subseteq U_j.l.$$

This constraint captures the linear inequality e_j.

(4) For each $p = x_i \leq x_j \cdot x_k$ in S_p, Σ includes

$$X^p_i[l_{i,j}, l_{i,k}] \rightarrow X^p_i, \qquad X^p_i.l_{i,j} \subseteq X_j.l, \qquad X^p_i.l_{i,k} \subseteq X_k.l.$$

Intuitively, these constraints encode the prequadratic constraint $x_i \leq x_j \cdot x_k$.

(5) For each $p = x_i \leq x_j \cdot x_k$ in S_p, Σ includes

$$X_i.l \subseteq NX_i^p.l, \qquad NX_i^p.l \subseteq X_i.l.$$

These constraints ensure $|ext(X_i)| = |ext(NX_i^p)|$. Since by the DTD D, $|ext(X_i^p)| = |ext(NX_i^p)|$, these constraints in fact ensure that $|ext(X_i)| = |ext(X_i^p)|$, i.e., X_i^p is indeed a copy of X_i.

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 X_i^p of X_i just to ensure that Σ is primary.

We next show that the encoding is indeed a PTIME 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 5, and moreover, for each $i \in [1, n]$ and each $p = x_i \leq x_j \cdot x_k$ in S_p, let $|ext(X_i)|$ and $|ext(X_i^p)|$ in T equal the value of the variable x_i in the solution. By the definitions of D and Σ, it is easy to verify $T \models \Sigma$. In particular, for each $p = x_i \leq x_j \cdot x_k$ in S_p, T satisfies the $\mathcal{AC}^{*,1}_{K,FK}$ constraints coding the inequality given above, by Lemma 1. Conversely, suppose that there exists an XML tree T such that $T \models D$ and $T \models \Sigma$. We construct a solution of S by letting variable x_i equal $|ext(X_i)|$ in T. Again by Lemma 1 and the definitions of D and Σ, it is easy to verify that it is indeed a nonnegative integer solution. In particular, each $p = x_i \leq x_j \cdot x_k$ in S_p holds because of $|ext(X_i^p)| \leq |ext(X_j)| \cdot |ext(X_k)|$ and $|ext(X_i^p)| = |ext(X_i)|$. Finally, observe that the encoding can be computed in PTIME in the size of S. Thus there is a PTIME reduction from PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$.

This completes the proof of Theorem 3.1. □

Proof of Theorem 3.4

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 $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ [14]: we code both the DTD and the constraints with linear inequalities over integers. However, compared to the proof of [14], the current proof is considerably harder due to the following. First, regular expressions in DTDs ("horizontal" regular expressions) interact in a certain way with regular 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 [14] by introducing exponentially many variables that account for 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 [2], and tag all the variables in the coding of DTDs with states of a certain automaton (the product automaton for all the automata corresponding to the regular expressions used in constraints).

Putting everything together, 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 exponential size; hence the NEXPTIME bound.

And now it is time to fill in all the details. First, we need some additional notation. For any regular expression β over a DTD D, any element type $\tau \in E$ and any attribute $l \in R(\tau)$, we write $values(\beta.\tau.l)$ to denote the set $\{val(y.l) \mid y \in nodes(\beta.\tau)\}$. Observe that for any $\tau \in E$, and $l \in R(\tau)$, $values(r.\_^*.\tau.l) = ext(\tau.l)$

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 $D = (E, A, P, R, r)$. For each $\tau \in E$, $P(\tau)$ is a regular expression α. A DTD is basically an extended regular grammar (cf. [8, 23]); 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:

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

where τ, τ_1, τ_2 are element types in $E \cup N$, $\tau' \in E$, 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, 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, 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

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

Note that the root element type 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 of E are of the form $\tau \to \tau'$, where $\tau' \in E$. It is easy to see that D_N is computable in linear time.

Observe that an XML tree T valid w.r.t. D may not conform to D_N and vice versa. Furthermore, an $\mathcal{AC}^{reg}_{K,FK}$ constraint φ over D may be satisfied by T but it may not be satisfied by any XML trees conforming to D_N. To explore the connection between XML trees valid w.r.t. D and those valid w.r.t. D_N, we interpret regular expressions and $\mathcal{AC}^{reg}_{K,FK}$ constraints over D in XML trees valid w.r.t. D_N.

The *restriction of a word* ρ *to* D is the list of E symbols in ρ. Let $\beta.\tau$ be a regular expression over D and T be an XML tree. We say that a node x in T is reachable by *following* $\beta.\tau$ *w.r.t.* D from the root of T, denoted by $T \models_D \beta.\tau(root,\ x)$, iff the restriction of $\rho(root, x)$ to D is in the language defined by $\beta.\tau$, where $\rho(root, x)$ is the path from *root* to x in T. Similarly, we use $T \models_D \beta.\tau(x,\ y)$ to denote that a node y in T is reachable from a node x by following $\beta.\tau$ w.r.t. D. We use $nodes_D(\beta.\tau)$ to denote the set of nodes of T reachable from the root by following $\beta.\tau$ w.r.t. D, i.e., the set $\{y \mid T \models_D \beta.\tau(root,\ y)\}$. For each $l \in R(\tau)$, $values_D(\beta.\tau.l)$ stands for the set $\{y.l \mid y \in nodes_D(\beta.\tau)\}$. Observe that if $T \models D$ then $nodes_D(\beta.\tau) = nodes(\beta.\tau)$ and $nodes_D(\beta.\tau) = nodes(\beta.\tau)$ in T.

An XML tree T *satisfies an* $\mathcal{AC}^{reg}_{K,FK}$ *constraint* φ *w.r.t. a DTD* D, denoted by $T \models_D \varphi$, iff (1) if φ is a key $\beta.\tau.l \to \beta.\tau$, then for any two nodes $x, y \in nodes_D(\beta.\tau)$ in T, $x.l = y.l$ implies $x = y$; (2) if φ is a foreign key: $\beta_1.\tau_1.l_1 \subseteq \beta_2.\tau_2.l_2$ and $\beta_2.\tau_2.l_2 \to \beta_2.\tau_2$, then $T \models_D \beta_2.\tau_2.l_2 \to \beta_2.\tau_2$ and for every $x \in nodes_D(\beta_1.\tau_1)$ there exists $y \in nodes_D(\beta_2.\tau_2)$ in T such that $x.l_1 = y.l_2$. Again, if $T \models D$ then $T \models_D \varphi$ iff $T \models \varphi$.

Using these 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 *Let* D *be a DTD,* D_N *be the narrowed DTD of* D *and* Σ *be a set of* $\mathcal{AC}^{reg}_{K,FK}$ *constraints over* D. *Then there exists an XML tree* T_1 *such that* $T_1 \models D$ *and* $T_1 \models \Sigma$ *iff there exists an XML tree* T_2 *such that* $T_2 \models D_N$ *and* $T_2 \models_D \Sigma$.

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 $l \in R(\tau)$, $|nodes_D(\beta.\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1, and $values_D(\beta.\tau.l)$ in T_2 equals $values(\beta.\tau.l)$ in T_1.

For 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, there is $\varphi \in \Sigma$ such that $T_2 \not\models \varphi$. If φ is a key $\beta.\tau.l \rightarrow \beta.\tau$, then there two distinct nodes $x, y \in nodes_D(\beta.\tau)$ in T_1 such that $x.l = y.l$. In other words, $|values_D(\beta.\tau.l)| < |nodes_D(\beta.\tau)|$ in T_2. Since $T_1 \models \varphi$, it must be the case that $|values(\beta.\tau.l)| = |nodes(\beta.\tau)|$ in T_1 since the value $x.l$ of each $x \in nodes(\beta.\tau)$ uniquely identifies x among $nodes(\beta.\tau)$. This contradicts the claim that $|nodes_D(\beta.\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1 and $values_D(\beta.\tau.l)$ in T_2 equals $values(\beta.\tau.l)$ in T_1. If φ is a foreign key: $\beta_1.\tau_1.l_1 \subseteq \beta_2.\tau_2.l_2$ and $\beta_2.\tau_2.l_2 \rightarrow \beta_2.\tau_2$, then either $T_2 \not\models_D \beta_2.\tau_2.l_2 \rightarrow \beta_2.\tau_2$ or there is $x \in nodes_D(\beta_1.\tau_1)$ such that for all $y \in nodes_D(\beta_2.\tau_2)$ in T_2, $x.l_1 \neq y.l_2$. If it is the first case, then the argument for keys shows that it leads to a contradiction. If it is the second case, then we have $x.l_1 \notin values_D(\beta_2.\tau_2.l_2)$. By the claim, $x.l_1 \in values(\beta_1.\tau_1.l_1)$ in T_1. Since $T_1 \models \varphi$, $x.l_1 \in values(\beta_2.\tau_2.l_2)$ in T_1. Again by the claim, we have $x.l_1 \in values_D(\beta_2.\tau_2.l_2)$ in T_2, which contradicts the assumption. The proof for the other direction is similar.

We next verify the claim. Given an XML tree $T_1 = (V_1, lab_1, ele_1, att, val, root)$ 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 $ele_1(v) = [v_1, ..., v_n]$ and $w = lab_1(v_1) \ldots lab_1(v_n)$. Recall N_τ and P_τ, the set of nonterminals and the set of production rules generated when narrowing $\tau \rightarrow P(\tau)$. Let Q_τ be the set of E symbols that appear in P_τ plus 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. 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 $T(w)$ of 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 $T_2 = (V_2, lab_2, ele_2, att, val, root)$. 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 $E_N \setminus E$. If $lab_2(x)$ is some element type τ, let $ele_2(x)$ be the list of its children in the parse tree. Note that att and val remain unchanged. By the construction of T_2 it can be verified that $T_2 \models D_N$; and moreover, for any regular expression $\beta.\tau$ over D and $l \in R(\tau)$, $|nodes_D(\beta.\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1 and $values_D(\beta.\tau.l)$ in T_2 equals $values(\beta.\tau.l)$ 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 ancestor-descendent relation on T_1 elements is not changed in T_2.

Conversely, assume that there is $T_2 = (V_2, lab_2, ele_2, att, val, root)$ 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 substitute the subelements of v for v in $ele(v')$, 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 $T_1 = (V_1, lab_1, ele_1, att_1, val_1, root)$, 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, val and $root$ remain unchanged. By the definition of T_1 it can be verified that $T_1 \models D$; and in addition, for any regular expression $\beta.\tau$ over D and $l \in R(\tau)$, $|nodes(\beta.\tau)|$ in T_1 equals $|nodes_D(\beta.\tau)|$ in T_2, and $values(\beta.\tau.l)$ in T_1 equals $values_D(\beta.\tau.l)$ in T_2, because, among other things, none of the nodes removed is labeled with a type of E and the attribute function att is unchanged. □

We now move to encoding $\mathcal{AC}^{reg}_{K,FK}$ constraints in terms of integer constraints. Let D be a DTD (E, A, P, R, r) and Σ be a set of $\mathcal{AC}^{reg}_{K,FK}$ constraints over D. Let D_N be the narrowed DTD of D. To encode Σ, let $\beta_1.\tau_1.l_1, \ldots, \beta_k.\tau_k.l_k$ be an enumeration of all regular expressions and attributes that appear in Σ, and Θ be the set of functions $\theta : \{1, \ldots, k\} \rightarrow \{0, 1\}$ which are not identically 0. For every θ, we introduce a new variable z_θ. In any XML tree valid w.r.t. D_N, the intended interpretation of z_θ is the cardinality of

$$\bigcap_{i:\theta(i)=1} values_D(\beta_i \tau_i.l_i) \quad \setminus \quad \bigcup_{j:\theta(j)=0} values_D(\beta_j.\tau_j.l_j).$$

That is, the variables z_θ describe all possible intersections of $values_D(\beta_i.\tau_i.l_i)$ and their complements. Note that there are exponentially many new variables in total. Using these variables, we define the set of *the cardinality constraints determined by* Σ, denoted by C_Σ, which consists of the following:

- for each $i \in [1, k]$, $|values_D(\beta_i.\tau_i.l_i)| = \sum_{\theta:\theta(i)=1} z_\theta$;

- for each $i, j \in [1, k]$, $\sum_{\theta:\theta(i)=1,\theta(j)=0} z_\theta = 0$ if $\beta_i \subseteq \beta_j$ (for each τ and l such that $\beta_i.\tau.l$ and $\beta_j.\tau.l$ appear in Σ), or if $\beta_i.\tau_i.l_i \subseteq \beta_j.\tau_j.l_j$ is in Σ; these encode inclusion constraints as well as containment of regular expressions;
- for each key $\beta_i.\tau_i.l_i \rightarrow \beta_i.\tau_i$ in Σ, $|values_D(\beta_i.\tau_i.l_i)| = |nodes_D(\beta_i.\tau_i)|$;
- for each $i \in [1, k]$, $|values_D(\beta_i.\tau_i.l_i)| \leq |nodes_D(\beta_i.\tau_i)|$ and $0 \leq |values_D(\beta_i.\tau_i.l_i)|$.

We use $T \models C_\Sigma$ to denote that there is an integer solution to C_Σ such that in T, for each $i \in [1, k]$, $|values_D(\beta_i.\tau_i.l_i)|$ and $|values_D(\beta_i.\tau_i)|$ equal the values assigned to these variables by the solution. Note that C_Σ can be computed in exponential time in $|\Sigma|$, and its size is exponential in $|\Sigma|$ as well.

The lemma below shows that C_Σ characterizes the consistency of Σ.

Lemma 4 *Let D_N be the narrowed DTD of D, Σ be a set of $\mathcal{AC}^{reg}_{K,FK}$ constraints over D, and C_Σ be the set of cardinality constraints determined by Σ. Then there exists an XML tree T_1 such that $T_1 \models D_N$ and $T_1 \models_D \Sigma$ iff there exists an XML tree T_2 such that $T_2 \models D_N$ and $T_2 \models C_\Sigma$. In addition, any XML tree valid w.r.t. D_N and satisfying Σ w.r.t. D also satisfies C_Σ.*

Proof. It is easy to see that for any XML tree T_1, if $T_1 \models_D \Sigma$, it must be the case that $T_1 \models C_\Sigma$. Indeed, given $values_D(\beta_i.\tau_i.l_i)$ and $nodes_D(\beta_i.\tau_i)$ in T_1 for all $i \in [1, k]$, we can compute an assignment to the variables z_θ by the equations in C_Σ. It is easy to verify that the assignment is a solution of integer constraints in C_Σ and moreover, $T_1 \models C_\Sigma$.

Conversely, we show that if there exists an XML tree $T_2 = (V, lab, ele, att, val, root)$ such that $T_2 \models D_N$ and $T_2 \models C_\Sigma$, then we can construct an XML tree T_1 such that $T_1 \models D$ and $T_1 \models_D \Sigma$. We construct T_1 from T_2 by modifying the function val while leaving V, lab, ele, att and $root$ unchanged. More specifically, we modify $val(v)$ if v is an attribute of a node in $nodes_D(\beta.\tau)$ for some regular expression $\beta.\tau$ in Σ, and leave $val(v)$ unchanged otherwise. To do this, for each z_θ we create a set s_θ of distinct string values such that $|s_\theta| = z_\theta$ and $s_\theta \cap s_{\theta'} = \emptyset$ if $\theta \neq \theta'$. For each $i \in [1, k]$, we define a set S_i by $S_i = \bigcup_{\theta:\theta(i)=1} s_\theta$ such that in T_2, $values_D(\beta_i.\tau_i.l_i) = S_i$. That is, for each τ_i element x in $nodes_D(\beta_i.\tau_i)$ in T_2, let $val'(att(x, l_i))$ be a string value in S_i such that $values_D(\beta_i.\tau_i.l_i)$ in T_2 equals S_i. This is possible because by the definition of T_1, (1) $nodes_D(\beta_i.\tau_i)$ in T_1 equals $nodes_D(\beta_i.\tau_i)$ in T_2; (2) $|values_D(\beta_i.\tau_i.l_i)|$ in T_1 equals $|values_D(\beta_i.\tau_i.l_i)|$ in T_2 by C_Σ constraint $|values_D(\beta_i.\tau_i.l_i)| = \sum_{\theta:\theta(i)=1} z_\theta$ and by $|S_i| = \sum_{\theta:\theta(i)=1} z_\theta$; (3) $0 \leq |values_D(\beta_i.\tau_i.l_i)|$ and $|values_D(\beta_i.\tau_i.l_i)| \leq |nodes_D(\beta_i.\tau_i)|$ are in C_Σ; and (4) $T_2 \models C_\Sigma$.

We next show that T_1 has the desired properties. It is easy to verify $T_1 \models D_N$ given the construction of T_1 from T_2 and the assumption $T_2 \models D_N$. From the discussion above follows that $T_1 \models C_\Sigma$. To show $T_1 \models_D \Sigma$, we consider $\varphi \in \Sigma$ in the following cases. (1) If φ is a key $\beta_i.\tau_i.l_i \rightarrow \beta_i.\tau_i$, it is immediate from the definition of T_1 that $T_1 \models_D \varphi$ since $T_1 \models |values_D(\beta_i.\tau_i.l_i)| = |nodes_D(\beta_i.\tau_i)|$. That is, each $x \in nodes_D(\beta_i.\tau_i)$ in T_1 has a distinct l_i attribute value and thus the value of its l_i attribute uniquely identifies x among nodes in $nodes_D(\beta_i.\tau_i)$. (2) If φ is $\beta_i.\tau_i.l_i \subseteq \beta_j.\tau_j.l_j$, then it is easy to see that $values_D(\beta_i.\tau_i.l_i) \setminus values_D(\beta_j.\tau_j.l_j) = \bigcup_{\theta:\theta(i)=1} s_\theta \setminus \bigcup_{\theta:\theta(j)=1} s_\theta = \bigcup_{\theta:\theta(i)=1,\theta(j)=0} s_\theta$. Since $s_\theta \cap s_{\theta'} = \emptyset$ if $\theta \neq \theta'$, $|values_D(\beta_i.\tau_i.l_i) \setminus values_D(\beta_j.\tau_j.l_j)| = \sum_{\theta:\theta(i)=1,\theta(j)=0} z_\theta$. Since $T_1 \models C_\Sigma$ and $\sum_{\theta:\theta(i)=1,\theta(j)=0} z_\theta = 0$ is in C_Σ, we have $|values_D(\beta_i.\tau_i.l_i) \setminus values_D(\beta_j.\tau_j.l_j)| = 0$. That is, $values_D(\beta_i.\tau_i.l_i) \subseteq values_D(\beta_j.\tau_j.l_j)$ in T_1. Thus $T_1 \models_D \varphi$. Therefore, T_1 is indeed an XML tree such that $T_1 \models D_N$ and $T_1 \models_D \Sigma$. □

Next, we move to encoding of DTDs, more specifically, narrow DTDs. Let $D = (E, A, P, R, r)$ and Σ be a set of $\mathcal{AC}^{reg}_{K,FK}$ constraints over D. We encode the narrowed DTD $D_N = (E_N, A, P_N, R_N, r)$ of D with a system Ψ^Σ_D of integer constraints such that there exists an XML tree valid w.r.t. D_N iff Ψ^Σ_D admits an integer solution.

The coding is developed w.r.t. Σ. More specifically, to capture the interaction between D_N and constraints of Σ, the system Ψ_D^Σ accommodates certain variables in C_Σ, i.e., $|nodes_D(\beta.\tau)|$ for each regular expression $\beta.\tau$ in Σ. In other words, Ψ_D^Σ specifies the dependencies imposed by D_N on the number of τ elements reachable by following $\beta.\tau$ w.r.t. D.

To capture $|nodes_D(\beta.\tau)|$ in Ψ_D^Σ, consider, for each regular expression in Σ, an automaton that recognizes that expression. Let M be the deterministic automaton equivalent to the product of all these automata. We refer to M as the DFA for Σ. Let s_M be the start state of M and δ be its transition function. Given an XML tree T valid w.r.t. D_N, for each τ element x in T we define

$$state(x) = \begin{cases} s & \text{if there is a simple path } \rho.\tau \text{ over } D \text{ such that} \\ & T \models_D \rho.\tau(root, x), \text{ and } s = \delta(s_M, \rho.\tau) \\ undefined & \text{otherwise} \end{cases}$$

The connection between M and T w.r.t. $\beta.\tau$ is described by the following lemma:

Lemma 5 *Let D be a DTD, D_N be the narrowed DTD of D, Σ be a set of $\mathcal{AC}^{reg}_{K,FK}$ constraints over D, M be the DFA for Σ, and $\beta.\tau$ be a regular expression in Σ. Then for any XML tree T valid w.r.t. D_N and any τ element x in T, $x \in nodes_D(\beta.\tau)$ in T iff $state(x)$ contains some final state $f_{\beta.\tau}$ of the automaton for $\beta.\tau$.*

In other words, $nodes_D(\beta.\tau)$ in T consists of all τ elements x such that $state(x)$ (which is a tuple of states of automata corresponding to regular expressions in Σ) contains some final state $f_{\beta.\tau}$ of the automaton for $\beta.\tau$. A similar idea was exploited in [2].

Proof. Since T is a tree, there exists a unique simple path ρ over D such that $T \models_D \rho.\tau(root,\ x)$. Thus $x \in nodes_D(\beta.\tau)$ in T iff $\rho.\tau \in \beta.\tau$. If $\rho.\tau \in \beta.\tau$, then there must be a final state $f_{\beta.\tau}$ in the automaton for $\beta.\tau$ and a state s in M such that $s = \delta(s_M, \rho.\tau)$ and s contains $f_{\beta.\tau}$. Thus $s = state(x)$. Conversely, if $state(x)$ contains a final state $f_{\beta.\tau}$ in the automaton for $\beta.\tau$, then $\rho.\tau \in \beta.\tau$ since $s = \delta(s_M, \rho.\tau)$. Therefore, $x \in nodes_D(\beta.\tau)$ in T. □

We next define a system Ψ_D^Σ of integer constraints. The variables used in the constraints of Ψ_D^Σ are described as follows. For each element type $\tau \in E$ and each state s in M such that $s = \delta(s_M, \rho.\tau)$ for some simple path $\rho \in E^*$, we create a distinct variable x^s_τ. Intuitively, in an XML tree T valid w.r.t. D_N, we use x^s_τ to keep track of the number of τ elements with state s. Recall N_τ, the set of element types introduced in the narrowing process of $\tau \to P(\tau)$. Note that N_τ and E are disjoint. For each $\tau' \in N_\tau$ and each state s for τ, we also create a distinct variable $x^s_{\tau'}$. In the XML tree T, we use $x^s_{\tau'}$ to keep track of the number of τ' descendants of a τ element with state s. There are exponentially many variables (in the size of D and Σ) in total since M is a DFA. Using these, we define an integer constraint to specify $\tau' \to P_N(\tau')$ at state s for each $\tau' \in N_\tau \cup \{\tau\}$ as follows. Let us use Ψ^s_τ to denote the set of integer constraints defined for $N_\tau \cup \{\tau\}$ at s.

- If $P_N(\tau') = \tau_1$ for some element type $\tau_1 \in E$, then Ψ^s_τ includes $x^s_{\tau'} = x^{s'}_{\tau_1}$, where s' is the state $\delta(s, \tau')$.
- If $P_N(\tau') = (\tau_1, \tau_2)$, then Ψ^s_τ includes $x^s_{\tau'} = x^s_{\tau_1}$ and $x^s_{\tau'} = x^s_{\tau_2}$, where $\tau_1, \tau_2 \in N_\tau$ by the narrowing process. 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 $P_N(\tau') = (\tau_1|\tau_2)$, then Ψ^s_τ includes $x^s_{\tau'} = x^s_{\tau_1} + x^s_{\tau_2}$, where $\tau_1, \tau_2 \in N_\tau$. Referring to the tree T, 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 number of these τ_1 subelements and the number of τ_2 subelements equals the number of τ' elements.
- If $P_N(\tau') = \tau_1^*$, then Ψ^s_τ includes $(x^s_{\tau_1} > 0) \to (x^s_{\tau'} > 0)$, where $\tau_1 \in N_\tau$. Referring to the XML tree T, this assures that each τ_1 subelement in T must have a τ' element as its parent.

In addition, Ψ^s_τ includes $x^s_{\tau'} \geq 0$ for each $\tau' \in N_\tau \cup \{\tau\}$. These assure that the numbers of elements in an XML tree cannot be negative. Note that in the coding, under a τ element at state s, the descendants of any type in N_τ do not change state as they do not appear in the DFA M, whereas descendants of type $\tau' \in E$ change state. Note that the coding of Kleene closure here is quite different from its coding in [14].

By our restriction on regular expressions regarding element type r, there is a unique variable x^s_r associated with r, where s is the start state s_M of M. We write x_r for x^s_r.

Given these, 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}, to be the set consisting of the following:

- all the constraints in Ψ^{s}_{τ} for each $\tau \in E$ and each state s given above;
- $x_r = 1$; i.e., there is a unique root in each XML tree valid w.r.t. D_N;
- $|nodes_D(\beta.\tau)| = \sum_{f_{\beta.\tau} \ in \ s} x^{s}_{\tau}$ for each regular expression $\beta.\tau$ in Σ, where x^{s}_{τ}'s are all those variables associated with τ such that s contains some final state in the automaton for $\beta.\tau$.

Observe that by Lemma 5, in an XML tree T valid w.r.t. D_N, $\sum_{f_{\beta.\tau} \ in \ s} x^{s}_{\tau}$ is indeed $|nodes_D(\beta.\tau)|$, the cardinality of the set of τ elements in T that are reachable from the root by following $\beta.\tau$ w.r.t. D. Note that Ψ^{Σ}_{D} can be computed in $EXPTIME$ in $|D|$ and $|\Sigma|$, and the size of Ψ^{Σ}_{D} is exponential in $|D|$ and $|\Sigma|$.

We say that Ψ^{Σ}_{D} is *consistent* iff it has an integer solution. We next show that Ψ^{Σ}_{D} indeed characterizes the DTD D_N.

Lemma 6 *Let D_N be the narrowed DTD of D and Ψ^{Σ}_{D} be the set of cardinality constraints determined by D w.r.t. a set Σ of $\mathcal{AC}^{reg}_{K,FK}$ constraints over D. Then Ψ^{Σ}_{D} is consistent if and only if there is an XML tree T such that $T \models D_N$. In addition, for each regular expression $\beta.\tau$ in Σ, $|nodes_D(\beta.\tau)|$ in T equals the value of the variable $|nodes_D(\beta.\tau)|$ given by the solution to Ψ^{Σ}_{D}.*

Proof. First, assume that there is an XML tree $T = (V, lab, ele, att, val, root)$ valid w.r.t. D_N. We define an integer solution of Ψ^{Σ}_{D} as follows. For each element x of type $\tau \in E$, let S_x be the set of its descendants of some type $\tau' \in N_\tau$. More specifically, S_x is computed as follows. Initially, let x be in S_x. For any $y \in S_x$ with type $\tau' \in N_\tau \cup \{\tau\}$, consider $\tau' \to P_N(\tau')$. If $P_N(\tau') = (\tau_1, \tau_2)$, then τ_1, τ_2 must be in N_τ by the narrowing process, and y has a τ_1 subelement y_1 and a τ_2 subelement y_2 by $T \models D_N$. Let S_x also include y_1 and y_2. If $P_N(\tau') = (\tau_1|\tau_2)$, then y has a subelement y_1 which is either of type τ_1 or τ_2. Let S_x also include y_1. If $P_N(\tau') = \tau_1^*$, then y has a list of τ_1 subelements. Let S_x also include these subelements. We repeat the process until no further change can be made to S_x. For each $y \in S_x$, let $state(y) = state(x)$. Given this, we define the integer solution. For each variable x^{s}_{τ} in Ψ^{Σ}_{D}, let its value be the number of τ elements x in T such that $state(x) = s$, where s is the state in x^{s}_{τ}. For each regular expression $\beta.\tau$ in Σ, let $|nodes_D(\beta.\tau)|$ be the sum of the variables x^{s}_{τ} in the constraint $|nodes_D(\beta.\tau)| = \sum_{f_{\beta.\tau} \ in \ s} x^{s}_{\tau}$. This defines an integer assignment since T is finite. It can be verified that the assignment is an integer solution of Ψ^{Σ}_{D}. Indeed, it satisfies the constraint $x_r = 1$ and constraints of the form $|nodes_D(\beta.\tau)| = \sum_{f_{\beta.\tau} \ in \ s} x^{s}_{\tau}$ by the definition of the assignment. Moreover, one can verify that it also satisfies constraints of Ψ^{s}_{τ} for each element type τ and each state s, by a straightforward induction on the structure of $P_N(\tau)$. In particular, it satisfies constraints of the form $(x^{s}_{\tau_1} > 0) \to (x^{s}_{\tau'} > 0)$ for each $\tau' \to \tau_1^*$ in P_N, since each τ_1 subelement in T has a τ' parent. Therefore, Ψ^{Σ}_{D} is consistent. Moreover, by Lemma 5, for each regular expression $\beta.\tau$ in Σ, the value of the variable $|nodes_D(\beta.\tau)|$ in the solution is indeed $|nodes_D(\beta.\tau)|$ in T.

Conversely, assume that Ψ^{Σ}_{D} admits an integer solution. Observe that the solution assigns a nonnegative integer to each variable because of inequalities in Ψ^{Σ}_{D}. We show that there exists an XML tree $T = (V, lab, ele, att, val, root)$ such that $T \models D_N$. To do so, for each element type τ and state s for τ, we create x^{s}_{τ} many distinct τ elements and label them with τ. Let $ext(\tau)$ denote the set of all τ elements created above. In addition, for each $\tau \in E_N$ such that $\tau \to \mathtt{S}$ is in P_N, we create x^{s}_{τ} many distinct string elements and label them with $\mathtt{S}$. Let $ext(\mathtt{S})$ denote the set of all the string elements created. Finally, for any $\tau \in E$, $v \in ext(\tau)$ and $l \in R(\tau)$, we create a distinct node and label it with l. We refer to this node as v_l. Let

$$V = \bigcup_{\tau \in E} ext(\tau) \ \cup \ ext(\mathtt{S}) \ \cup \ \bigcup_{\tau \in E} \{v_l \mid v \in ext(\tau), l \in R(\tau)\}.$$

The functions lab, att and val are defined as follows: for each $v \in V$ and $l \in A$,

$$lab(v) = \begin{cases} \tau & \text{if } v \in ext(\tau) \text{ and } \tau \in E_N \cup \{\mathtt{S}\} \\ l & \text{if } v = v'_l \text{ for some } v'_l \end{cases}$$

$$att(v, l) = \begin{cases} v_l & \text{if } v_l \in V \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$val(v) = \begin{cases} \text{empty string} & \text{if } lab(v) \text{ is } \mathtt{S} \text{ or } l, \text{ where } l \in A \\ \text{undefined} & \text{otherwise} \end{cases}$$

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 the function ele, we do the following. For each x_τ^s in Ψ_D^Σ, we choose x_τ^s many distinct vertices labeled τ and mark them with x_τ^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_τ^s, consider $P_N(\tau)$ and constraints of Ψ_D^Σ. If $P_N(\tau)$ is $\tau' \in E$, then we choose a distinct τ' element y marked with $x_{\tau'}^{s'}$ and let $ele(x) = [y]$, where $x_\tau^s = x_{\tau'}^{s'}$ is Ψ_D^Σ. If $P_N(\tau) = \mathtt{S}$, then we choose a string element y and let $ele(x) = [y]$. If $P_N(\tau) = (\tau_1, \tau_2)$, then we choose a τ_1 element y_1 marked with $x_{\tau_1}^s$ and a τ_2 element y_2 marked with $x_{\tau_2}^s$, and let $ele(x) = [y_1, y_2]$. If $P_N(\tau) = (\tau_1|\tau_2)$, then we choose an element y of either type τ_1 or τ_2, and let $ele(x) = [y]$. If $P_N(\tau) = \tau_1^*$, then we choose a list $\vec{v}$ of τ_1 elements and let $ele(x) = \vec{v}$. By the constraints in Ψ_D^Σ, each element of V can be chosen once and only once. By induction on the structure of $P_N(\tau)$, one can verify that T defined in this way is indeed an XML tree and $T \models D_N$. Therefore, there exists an XML tree valid w.r.t. D_N.

Finally, to see that $|nodes_D(\beta.\tau)|$ in T equals the value $n_{\beta.\tau}$ of the variable $|nodes_D(\beta.\tau)|$ in the solution for each regular expression $\beta.\tau$ in Σ, it suffices to show that for each $\tau \in E$ and each τ element x in T, if x is marked with x_τ^s in the construction, then $state(x) = s$ where s is the state in x_τ^s. For if it holds, by Lemma 5, we have $|nodes_D(\beta.\tau)|$ in T equals $n_{\beta.\tau}$ by the constraint $|nodes_D(\beta.\tau)| = \sum_{f_{\beta.\tau} \text{ in } s} x_\tau^s$ in Ψ_D^Σ for each $\beta.\tau$ in Σ. Since T is a tree, there is a unique simple path $\rho \in E^*$ such that $T \models_D \rho(root, x)$. We show the claim by induction on the length $|\rho|$ of ρ. If $|\rho| = 0$, i.e., $\rho = \epsilon$, then x is the root and obviously, $state(x) = s_M = s$. 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 $T \models_D \rho(root, y)$ and $T \models_D \tau(y, x)$. Suppose that y is marked with $x_{\tau'}^{s'}$ in the construction. By the induction hypothesis, we have $state(y) = s'$, where s' is the state in $x_{\tau'}^{s'}$. It is easy to see $state(x) = \delta(s', \tau)$. By the definition of $\Psi_{\tau'}^{s'}$ constraints, s is precisely the state $\delta(s', \tau)$. Thus $state(x) = s$. This proves the claim and thus the lemma. □

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

$$\Psi_D^\Sigma \cup C_\Sigma \cup \{|nodes_D(\beta.\tau)| > 0 \rightarrow |nodes_D(\beta.\tau.l)| > 0 \mid \beta.\tau \text{ is a regular expression in } \Sigma, l \in R(\tau)\},$$

where C_Σ is the set of cardinality constraints determined by Σ, and Ψ_D^Σ 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 an integer solution that satisfies all of its constraints.

Observe that $\Psi(D, \Sigma)$ can be partitioned into two sets: $\Psi(D, \Sigma) = \Psi^l(D, \Sigma) \cup \Psi^c(D, \Sigma)$, where $\Psi^l(D, \Sigma)$ consists of linear integer constraints, and $\Psi^c(D, \Sigma)$ consists of constraints of the form $(x > 0 \rightarrow y > 0)$, which come from two sources:

- For each regular expression $\beta.\tau$ in Σ, $(|nodes_D(\beta.\tau)| > 0) \rightarrow (|nodes_D(\beta.\tau.l)| > 0)$ is in $\Psi^c(D, \Sigma)$. That is, if there is a τ element reachable by following $\beta.\tau$ w.r.t. D, then the τ element must have an l attribute.
- Recall $P_{\tau'}$, the set of production rules generated in the narrowing process of $\tau' \rightarrow P(\tau')$ for $\tau' \in E$. For each $\tau \rightarrow \tau_1^*$ in $P_{\tau'}$ for some $\tau' \in E$ and each state s for τ', $\Psi^c(D, \Sigma)$ includes $(x_{\tau_1}^s > 0) \rightarrow (x_\tau^s > 0)$. That is, each τ_1 subelement must have a parent τ element.

It should be noted that the sizes of $\Psi^c(D, \Sigma)$ and $\Psi(D, \Sigma)$ are exponential in $|D|$ and $|\Sigma|$.

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

Lemma 7 *Let D be a DTD, Σ be a finite set of $\mathcal{AC}^{reg}_{K,FK}$ constraints over D, and $\Psi(D, \Sigma)$ be the set of cardinality constraints determined by D and Σ. Then $\Psi(D, \Sigma)$ is consistent if and only if there is an XML tree T such that $T \models D$ and $T \models \Sigma$.*

Proof. Let D_N be the narrow DTD corresponding to D. By Lemma 3, it suffices to show that $\Psi(D, \Sigma)$ is consistent if and only if there is an XML tree T such that $T \models D_N$ and $T \models_D \Sigma$.

Suppose that there exists an XML tree T such that $T \models D_N$ and $T \models_D \Sigma$. We show that $\Psi(D, \Sigma)$ has an integer solution. By Lemma 4, we have $T \models C_\Sigma$. By Lemma 6, one can define an integer assignment to the variables in Ψ^Σ_D such that all the constraints in Ψ^Σ_D are satisfied. Moreover, the assignment assures that for each regular expression $\beta.\tau$ in Σ, the value of the variable $|nodes_D(\beta.\tau)|$ equals the number of all τ elements in T reachable by following $\beta.\tau$ w.r.t. D. We extend the assignment as follows: for each regular expression $\beta.\tau$ in Σ, let the value of the variable $|values_D(\beta.\tau.l)|$ be the number of distinct l attribute values of all nodes in $nodes_D(\beta.\tau)$ in T. Thus by $T \models C_\Sigma$, this extended assignment satisfies all the equalities and inequalities in C_Σ. In addition, if $|nodes_D(\beta.\tau)| > 0$ then $|values_D(\beta.\tau.l)| > 0$ since every τ element in T has an l attribute. Hence the assignment is indeed a solution to $\Psi(D, \Sigma)$. Thus $\Psi(D, \Sigma)$ is consistent.

Conversely, suppose that $\Psi(D, \Sigma)$ has an integer solution. We show that there is an XML tree T such that $T \models D_N$ and $T \models_D \Sigma$. By Lemma 6, given an integer solution to $\Psi(D, \Sigma)$, we can construct an XML tree $T' = (V, lab, ele, att, val, root)$ such that $T' \models D_N$. Moreover, for each regular expression $\beta.\tau$ in Σ, there are exactly $n_{\beta.\tau}$ many τ elements in T' reachable by following $\beta.\tau$ w.r.t. D, where $n_{\beta.\tau}$ is the value of the variable $|nodes_D(\beta.\tau)|$ in $\Psi(D, \Sigma)$. We modify the definition of the function val such that for each regular expression $\beta.\tau$ in Σ and each $l \in R(\tau)$ that occurs in Σ, the number of distinct l attribute values of all nodes of $nodes_D(\beta.\tau)$ in T' equals the value of the variable $|values_D(\beta.\tau.l)|$ in the assignment. This is possible since $|values_D(\beta.\tau.l)| \leq |nodes_D(\beta.\tau)|$ is in C_Σ, and the assignment is also a solution to C_Σ. Moreover, since $(|nodes_D(\beta.\tau)| > 0) \rightarrow (|values_D(\beta.\tau.l)| > 0)$ is in $\Psi(D, \Sigma)$, each τ element in $nodes_D(\beta.\tau)$ can have an attribute. Let T'' denote the XML tree obtained by modifying the val function of T'. Then it is easy to verify that $T'' \models C_\Sigma$ and $T'' \models D$. Hence by Lemma 4, there is an XML tree T such that $T \models D_N$ and $T \models_D \Sigma$. □

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

Lemma 8 *Given a system $A\vec{x} \leq \vec{b}$ of linear integer constraints together with conditions of the form $(x_i > 0) \rightarrow (x_j > 0)$, where A is an $n \times m$ matrix on integers, $\vec{b}$ is an n-vector on integers and $1 \leq i, j \leq n$, the problem of determining whether the system admits a nonnegative integer solution is in NP.*

Proof. Let $c_1, \ldots, c_p$ enumerate the conditions of the form $(x > 0) \rightarrow (y > 0)$, c_k being $(x^1_k > 0) \rightarrow (x^2_k > 0)$. Consider 2^p instances of the integer linear programming $\mathcal{I}_j$ obtained by adding, for each $k \leq p$, either $x^1_k = x^2_k = 0$, or $x^1_k, x^2_k > 0$ 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 [24], $\mathcal{I}_j$ has a solution iff it has a solution whose size is polynomial in $A, \vec{b}$. Hence, to check if the original system of constraints has a solution, it suffices to guess a polynomial size solution; thus, the problem is in NP. □

The proof of Theorem 3.4 a) now follows from Lemmas 7 and 8. Given any DTD D and any finite set Σ of $\mathcal{AC}^{reg}_{K,FK}$ constraints over D, we compute, in exponential time in $|D|$ and $|\Sigma|$, the system $\Psi(D, \Sigma)$ whose size is also exponential in $|D|$ and $|\Sigma|$. By Lemma 8, one can check in NEXPTIME whether $\Psi(D, \Sigma)$ has a nonnegative integer solution, which, by Lemma 7, happens iff there exists an XML tree T such that $T \models D$ and $T \models \Sigma$.

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 [25].

Let θ be a formula of the form

$$Q_1 x_1 \cdots Q_m x_m \psi, \tag{5}$$

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 if and only if 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.

$$\begin{aligned} E &= \{r, C\} \cup \bigcup_{i=1}^{m} \{x_i, \bar{x}_i, N_{x_i}, P_{x_i}\}, \\ A &= \{l\}. \end{aligned}$$

To define the function P, first we consider the quantifiers of θ. We consider Q_1 to define P on the root r:

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

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 the 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 $y_1| \cdots |y_p|\bar{z}_1| \cdots |\bar{z}_q$. We define P on the element types N_{x_m} and P_{x_m} as $P(N_{x_m}) = P(x_{x_m}) = tr(C_1), \ldots, tr(C_n)$. Finally, for the rest of the elements in E, we define P as ϵ. We define the function R as follows.

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

Finally, Σ_θ contains the following foreign keys:

$$\begin{array}{ll} r.\_^*.N_{x_i}.\_^*.x_i.l \subseteq r.C.C.l, & r.C.C.l \to r.C.C \quad i \in [1, m] \\ r.\_^*.P_{x_i}.\_^*.\bar{x}_i.l \subseteq r.C.C.l, & r.C.C.l \to r.C.C \quad i \in [1, m] \end{array}$$

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 6. 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} has 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 $\forall x_3$.

In figure 6, 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 these assignments satisfy the formula $x_1 \vee x_2 \vee \neg x_3$ we use the set of constraint Σ_θ.

We will to prove that θ, defined in (5), is valid if and only if there is an XML tree T conforming to D_θ and satisfying Σ_θ. We will show only the "if" direction. The "only if" direction is similar.

Suppose that there is an XML tree T such that $T \models D_\theta$ and $T \models \Sigma_\theta$. 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 one of these paths and let n be the node reachable from the root by following p. We define the truth assignment σ_p as follows.

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

We have to prove that $\sigma_p(C_i) = 1$ for each $i \in [1, n]$. Given that $T \models D_\theta$, n has as a child a node n' whose type is in $tr(C_i)$. If the type of n' is x_j, then given that $T \models r.\_^*.N_{x_j}.\_^*.x_j.l \subseteq r.C.C.l$ and there are no nodes in T reachable by following the path $r.C.C$, we conclude that p contains a node of type P_{x_j}, and, therefore, $\sigma(C_i) = 1$ since $\sigma_p(x_j) = 1$. If the type of n' is $\bar{x}_j$,then given that $T \models r.\_^*.P_{x_j}.\_^*.\bar{x}_j.l \subseteq r.C.C.l$ we conclude that p contains a node of type N_{x_j}, and, therefore, $\sigma_p(C_i) = 1$ since $\sigma_p(\neg x_j) = 1$. Therefore, θ is a valid formula. $\square$

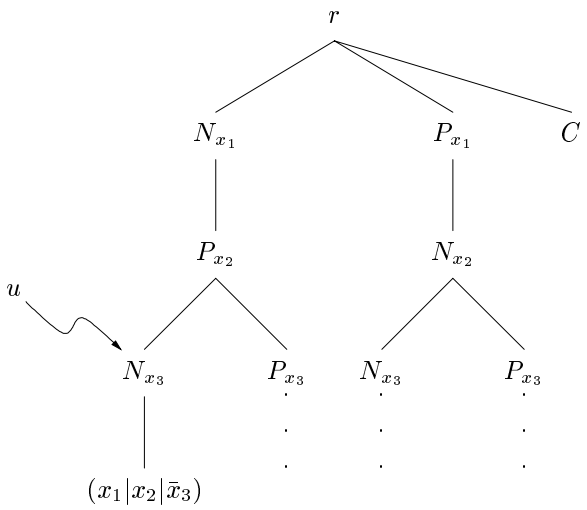


Figure 6: An XML tree conforming to the DTD constructed from $\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee \neg x_3)$.

Proof of Theorem 3.5

Proof of a) First, we will prove the NP-hardness of depth-d $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ ($d \geq 2$). It suffices to prove the NP-hardness of depth-2 $\mathsf{SAT}(\mathcal{AC}_{K,FK})$. We do this by reduction from SAT-CNF.

Let φ be a propositional formula of the form $C_1 \wedge \cdots \wedge C_n$, where each C_i ($1 \leq i \leq n$) is a clause. Assume that φ mentions propositional variables $x_1, \ldots, x_m$. We construct a DTD D_φ and a set of $\mathcal{AC}_{K,FK}$-constraints Σ_φ such that $Depth(D_\varphi) = 2$ and φ is satisfiable if and only if there is an XML tree T conforming to D_φ and satisfying Σ_φ. Define a DTD $D_\varphi = (E, A, P, R, r)$ as follows.

$$E = \{r, x_1, \ldots, x_m, \bar{x}_1, \ldots, \bar{x}_m\} \cup \{C_{i,j} \mid C_i \text{ mentions literal } x_j\} \cup \{\bar{C}_{i,j} \mid C_i \text{ mentions literal } \neg x_j\},$$
$$A = \{l\}.$$

In order to define P, first we define a function for translating clauses into regular expressions. If the set of literals mentioned in C_i ($1 \leq i \leq n$) is $\{x_{j_1}, \ldots, x_{j_p}, \neg x_{k_1}, \ldots, \neg x_{k_q}\}$, then

$$tr(C_i) \quad = \quad C_{i,j_1}| \cdots |C_{i,j_p}|\bar{C}_{i,k_1}| \cdots |\bar{C}_{i,k_q}.$$

We define the function P on the root as follows.

$$P(r) \quad = \quad tr(C_1), \ldots, tr(C_n), (x_1|\bar{x}_1), \ldots, (x_m|\bar{x}_m).$$

For the rest of the element types in E, we define P as ϵ. We define R as follows: $R(r) = \emptyset$ and $R(x) = \{l\}$ for each $x \in E \setminus \{r\}$. Finally, the set of constraints Σ_φ is defined as follows. For each $C_{i,j} \in E$, the foreign key $C_{i,j}.l \subseteq x_j.l$, $x_j.l \rightarrow x_j$ is contained in Σ_φ. For each $\bar{C}_{i,j} \in E$, the foreign key $\bar{C}_{i,j}.l \subseteq \bar{x}_j.l$, $\bar{x}_j.l \rightarrow \bar{x}_j$ is contained in Σ_φ.

For example, if $\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$, then D_φ is of the form shown in figure 7 and $\Sigma_\varphi = \{C_{1,1} \subseteq x_1.l$, $x_1.l \rightarrow x_1$, $\bar{C}_{1,2}.l \subseteq \bar{x}_2.l$, $\bar{x}_2.l \rightarrow \bar{x}_2$, $\bar{C}_{2,1}.l \subseteq \bar{x}_1.l$, $\bar{x}_1.l \rightarrow \bar{x}_1.l$, $C_{2,3}.l \subseteq x_3.l$, $x_3.l \rightarrow x_3\}$.

We will prove that φ is satisfiable if and only if there is an XML tree T conforming to D_φ and satisfying Σ_φ. If φ is satisfiable, then there is a truth assignment σ such that $\sigma(C_i) = 1$ for each $i \in [1, n]$. By considering σ we construct an XML tree T conforming to D_φ and satisfying Σ_φ. For each $i \in [1, n]$, there is a literal l_j in C_i such that $\sigma(l_j) = 1$. If $l_j = x_j$, then the root of T has a child of type $C_{i,j}$. If $l_j = \neg x_j$, then it has a child of type $\bar{C}_{i,j}$. Moreover, for each $i \in [1, m]$, if $\sigma(x_i) = 1$, then the root of T has a child of type x_i, otherwise it has a child of type $\bar{x}_i$. Finally, we give value 1 to each attribute in T. It is straightforward to prove that T conforms to D_φ and satisfies Σ_φ.

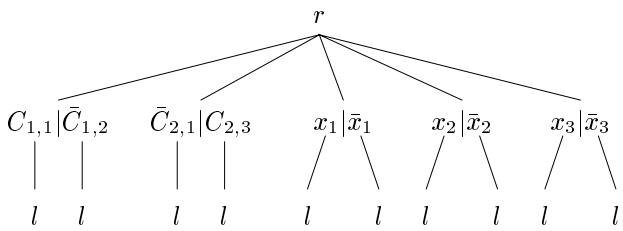


Figure 7: DTD D_φ for $\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$.

If there is an XML tree T conforming to D_φ and satisfying Σ_φ, then we define a truth assignment σ as follows. For each $i \in [1, m]$, if T has a node of type x_i, then $\sigma(x_i) = 1$, otherwise $\sigma(x_i) = 0$. We have to prove that $\sigma(C_i) = 1$ for each $i \in [1, n]$. Given that T conforms to D_φ, its root has as a child a node n whose type is in the regular expression $tr(C_i)$. If the type of n is $C_{i,j}$, then C_i mentions literal x_j and T has a child of type x_j, since T must satisfy the foreign key $C_{i,j}.l \subseteq x_j.l$, $x_j.l \to x_j$. Thus, we conclude that $\sigma(C_i) = 1$ because $\sigma(x_j) = 1$. If the type of n is $\bar{C}_{i,j}$, then C_i mentions literal $\neg x_j$ and T has a child of type $\bar{x}_j$, since T must satisfy the foreign key $\bar{C}_{i,j}.l \subseteq \bar{x}_j.l$, $\bar{x}_j.l \to \bar{x}_j$. Thus, we conclude that $\sigma(C_i) = 1$ because $\sigma(x_j) = 0$.

Therefore, we conclude that depth-2 $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is NP-hard.

Next, we prove the NP-hardness of 2-constraint $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ by reduction from SUBSET SUM.

An instance of the SUBSET SUM problem is a number a and a set of numbers S, all given in binary. The problem is to determine whether there is $S' \subseteq S$ such that $\sum_{x \in S'} x = a$. It is known to be NP-complete [18].

Suppose that $S = \{a_1, \ldots, a_n\}$. We construct a DTD $D_{a,S}$ and a set of $\mathcal{AC}_{K,FK}$-constraints $\Sigma_{a,S}$ containing two constraints such that there is an XML tree T conforming to $D_{a,S}$ and satisfying $\Sigma_{a,S}$ if and only if there is $S' \subseteq S$ such that $\sum_{x \in S'} x = a$. We define the DTD $D_{a,S} = (E, A, P, R, r)$ as follows.

We introduce new element types $V, V_1, \ldots, V_n$ for coding the values of $a, a_1, \ldots, a_n$, respectively. Define $P(r) = V, (V_1|\epsilon), (V_2|\epsilon), \ldots, (V_n|\epsilon)$. If $a = \sum_{i=1}^{m} 2^{k_i}$, where $0 \leq k_1 < k_2 < \ldots < k_m$, then $P(V) = X_{k_1}, X_{k_2}, \ldots, X_{k_m}$. Define P on $\{X_0, X_1, \ldots, X_{k_m - 1}, X_{k_m}\}$ as follows:

$$P(X_i) = \begin{cases} \tau & i = 0 \\ X_{i-1}, X_{i-1} & \text{Otherwise} \end{cases}$$

If $V_j = \sum_{i=1}^{m} 2^{l_i}$, where $j \in [1, n]$ and $0 \leq l_1 < l_2 < \ldots < l_m$, then $P(V_j) = Y_{l_1}, Y_{l_2}, \ldots, Y_{l_m}$. Define P on $\{Y_0, Y_1, \ldots, Y_{l_m - 1}, Y_{l_m}\}$ as follows:

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

The set E contains all the element types mentioned in the previous paragraph. Moreover, $A = \{l\}$, $R(\tau) = R(\tau') = \{l\}$ and R is defined as $\emptyset$ for the rest of the elements in E. Finally, the set of constraints $\Sigma_{a,S}$ contains two foreign keys: $\tau.l \subseteq \tau'.l$, $\tau'.l \to \tau'$ and $\tau'.l \subseteq \tau.l$, $\tau.l \to \tau$. Observe that the size of $D_{a,s}$ and $\Sigma_{a,S}$ is polynomial in the size of a and S.

For instance, if $a = 101$ and $S = \{11, 110\}$, then $P(r) = V, (V_1|\epsilon), (V_2|\epsilon)$. An XML tree conforming to $D_{a,S}$ is shown in figure 8. Notice that the number of τ-elements that are descendants of V is 5, since we use the node V for coding the binary value 101. Besides, notice that the number of τ'-elements that are descendants of V_1 is 3, since we use the node V_1 for coding the binary value 11. The tree shown in figure 8 does not contain a node of type V_2. Given that $P(r) = V, (V_1|\epsilon), (V_2|\epsilon)$, we can choose whether a node of type V_i is going to be included ($i \in [1, 2]$). Thus, we use the regular expression $(V_1|\epsilon), (V_2|\epsilon)$ for coding all the possible subsets of S.

We will prove that there is an XML tree T conforming to $D_{a,S}$ and satisfying $\Sigma_{a,S}$ if and only if there is $S' \subseteq S$ such that $\sum_{x \in S'} x = a$. If there is an XML tree conforming to $D_{a,S}$ and satisfying $\Sigma_{a,S}$, then define $S' = \{a_i \mid T \text{ contains a node of type } V_i\}$. Given that $T \models \Sigma_{a,S}$, we conclude that $\sum_{x \in S'} x = a$.

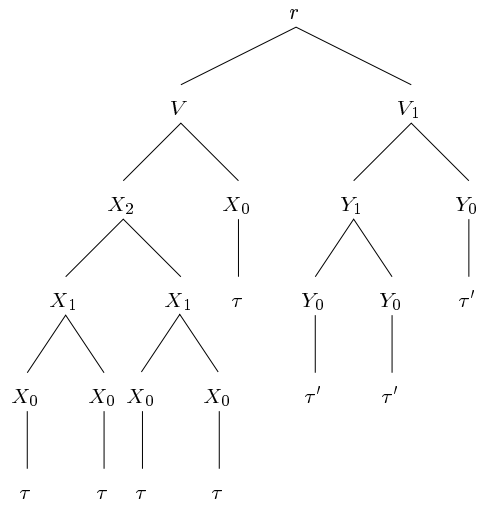


Figure 8: An XML tree conforming to the DTD $D_{a,S}$ for $a = 101$ and $S = \{11, 110\}$.

If there is $S' \subseteq S$ such that $\sum_{x \in S'} x = a$, then define an XML tree T as follows. For each $i \in [1, n]$, the root of T has a child of type V_i if and only if $a_i \in S'$. For each child of the root, define its descendant in such a way that they conform to $D_{a,S}$. Given that $\sum_{x \in S'} x = a$, there is an integer k such that $k = |\{x \mid x \text{ is a node in } T \text{ of type } \tau\}| = |\{y \mid y \text{ is a node in } T \text{ of type } \tau'\}|$. Assign the set of values 1, 2, ..., k to $\{x.l \mid x \text{ is a node in } T \text{ of type } \tau\}$ and $\{y.l \mid y \text{ is a node in } T \text{ of type } \tau'\}$. It is straightforward to verify that T conforms to $D_{a,S}$ and satisfies $\Sigma_{a,S}$.

Therefore, we conclude that 2-constraint $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is NP-hard.

Proof of b) In this proof we will use the following lemma.

Lemma 9 ([14]) *Given a DTD D_1 and a set Σ_1 of $\mathcal{AC}_{K,FK}$-constraints, there is a set of cardinality constraints C_{Σ_1} such that the following statements are equivalent:*

1. *There is a tree T_1 such that $T_1 \models D_1$ and $T_1 \models \Sigma_1$.*
2. *There is a tree T_2 such that $T_2 \models D_1$ and $T_2 \models C_{\Sigma_1}$.*

These cardinality constraints are constructed as follows. For each key $\tau.l \to \tau \in \Sigma_1$, $|ext(\tau)| = |ext(\tau.l)|$ is in C_{Σ_1}. For each inclusion dependency $\tau_1.l_1 \subseteq \tau_2.l_2 \in \Sigma_1$, $|ext(\tau_1.l_1)| \leq |ext(\tau_2.l_2)|$ is in C_{Σ_1}.

We use this result to construct a nondeterministic algorithm that verifies whether a given specification (D, Σ) is consistent. This algorithm works in space $O(|\Sigma| \cdot \mathit{Depth}(D) \cdot \log(|D|))$. Thus, for any fixed $k, d > 0$, the k-constraint depth-d $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is solvable in NLOGSPACE.

Let $D = (E, A, P, R, r)$ be a non-recursive no-star DTD and Σ a set of $\mathcal{AC}_{K,FK}$-constraints. Let $\mathit{Restricted}(\Sigma)$ be the set

$$\{\tau \mid \tau \in E \text{ and } \tau \text{ is mentioned in some constraint in } \Sigma\} \cup \{\tau.l \mid \tau \in E, l \in R(\tau) \text{ and } \tau.l \text{ is mentioned in some constraint in } \Sigma\}.$$

To verify if (D, Σ) is consistent, we execute a simple nondeterministic algorithm: Guess an XML tree T conforming to D and verify whether T satisfies C_Σ. To implement this algorithm, it is not necessary to store the entire tree T; it suffices to store the value of $|ext(x)|$ in T, for each $x \in \mathit{Restricted}(\Sigma)$, since we only need to check whether these values satisfy C_Σ. The following procedure implements this idea.

```
Count(s: regular expression)
  if s ∈ E then
    if s ∈ Restricted(Σ) then
       |ext(s)| := |ext(s)| + 1
       for each s.l ∈ Restricted(Σ) do
           if |ext(s.l)| = 0 then |ext(s.l)| := 1
           else
             flip a coin
             if "head" then |ext(s.l)| := |ext(s.l)| + 1
    Count(P(s))
  else if s = s1, s2 then
    Count(s1)
    Count(s2)
  else if s = s1|s2 then
    flip a coin
    if "head" then Count(s1)
    else Count(s2)
```

Initially, $|ext(x)| = 0$ for each $x \in Restricted(\Sigma)$. The algorithm is invoked as $\texttt{Count}(r)$, where r is the type of the root. In each step, it verifies whether the regular expression s is an element type. If this is the case, then it verifies whether $s \in Restricted(\Sigma)$. If this holds, it increments $|ext(s)|$, since it found another node of type s, and for each $s.l \in Restricted(\Sigma)$, it flips a coin for deciding whether to assign a new value to the attribute l (and then increment $|ext(s.l)|$) or to copy this value from attribute l of another node of type s (and leave $|ext(s.l)|$ intact). Notice that if $|ext(s.l)| = 0$, then it is not possible to copy the value of l from another node of type s, and, therefore, the algorithm must assign the value 1 to $|ext(s.l)|$. Finally, for each $s \in E$, it recursively invokes to $\texttt{Count}(P(s))$.

If $s \notin E$, then $s = (s_1, s_2)$, $s = (s_1|s_2)$ or $s = \epsilon$, where s_1 and s_2 are regular expressions. If $s = (s_1, s_2)$, then in order to generate an XML tree conforming to D the algorithm invokes to $\texttt{Count}(s_1)$ and $\texttt{Count}(s_2)$. If $s = (s_1|s_2)$, then in order to generate an XML tree conforming to D, the algorithm nondeterministically decides to invoke to $\texttt{Count}(s_1)$ or $\texttt{Count}(s_2)$. Finally, if $s = \epsilon$, then the algorithm does not execute any instruction.

When $\texttt{Count}(r)$ finishes, it guessed a tree T conforming to D and for each element type $x \in Restricted(\Sigma)$ it stored the value of $|ext(x)|$. Thus, to verify whether (D, Σ) is consistent, we check whether these values satisfy the set of cardinality constraint C_Σ.

Given that D is a non-recursive no-star DTD, for each $x \in Restricted(\Sigma)$, $|ext(x)| \leq |D|^{Depth(D)}$. Thus, the previous algorithm works in space $O(|\Sigma| \cdot \log(|D|^{Depth(D)})) = O(|\Sigma| \cdot Depth(D) \cdot \log(|D|))$, since $|Restricted(\Sigma)|$ is $O(|\Sigma|)$. Therefore, if $|\Sigma|$ and $Depth(D)$ are fixed, then $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is solvable in NLOGSPACE. □

Proof of Proposition 3.6

It suffices to provide a DLOGSPACE reduction from $\mathsf{SAT}(\mathcal{C})$ to the complement of $\mathsf{Impl}(\mathcal{C})$. Given any DTD $D = (E, A, P, R, r)$ and any set Σ of $\mathcal{C}$ constraints, we define another DTD D' and a key φ and a foreign key ϕ in $\mathcal{AC}_{K,FK}$ such that there exists an XML tree conforming to D and satisfying Σ iff $(D, \Sigma \cup \{\phi\}) \not\vdash \varphi$.

We define $D' = (E', A', P', R', r)$, where E' is E plus two new element types E_X and D_Y, A' is A plus a new attribute K, P' is identical to P except:

$P'(r) = P(r), D_Y, D_Y, E_X$ /* $P(r)$ followed by two D_Y elements and an E_X element
$P'(D_Y) = P'(E_X) = \epsilon$

and R' is the same as R except $R'(D_Y) = R'(E_X) = \{K\}$. We define a unary key φ and a unary inclusion constraint ϕ as follows:

$$\varphi : D_Y.K \to D_Y, \qquad \phi : D_Y.K \subseteq E_X.K,\ E_X.K \to E_X.$$

Clearly this can be done in DLOGSPACE. We next show that this is indeed a reduction. First, assume that there exists a tree $T \models D'$ and $T \models \bigwedge \Sigma \wedge \phi \wedge \neg\varphi$, then we construct another tree T' by removing D_Y, E_X

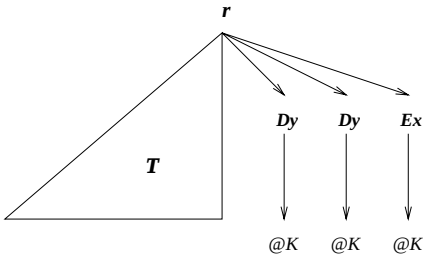


Figure 9: An XML tree conforming to D' in the proof of Proposition 3.6

elements from T. Obviously, $T' \models D$ and $T' \models \Sigma$. Conversely, suppose that there is a tree $T \models D$ and $T \models \Sigma$. We construct another tree T' from T as shown in Figure 9. Let us refer to the two D_Y elements in T' as d_1, d_2, and the E_X element as e. Let $d_1.K = d_2.K = e.K$. Then it is easy to see that $T' \models D'$, $T' \models \Sigma$ and $T' \models \phi \wedge \neg\varphi$. □

Proof of Theorem 4.1

We establish the undecidability of the consistency problem for unary relative keys and foreign keys by reduction from the Hilbert's 10th problem [21]. To do this, we consider a variation of the Diophantine problem, referred as the *positive Diophantine quadratic system problem*. An instance of the problem is

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

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

Thus, if we reduce the positive Diophantine quadratic system problem to the consistency problem for unary relative keys and foreign keys, we prove that the second problem is undecidable. We will do this in the rest of this proof. More precisely, given a quadratic equation we will 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

$$\sum_{i=1}^{m} a_i x_{\alpha_i} + \sum_{i=m+1}^{n} a_i x_{\alpha_i} x_{\beta_i} = \sum_{i=1}^{p} b_i x_{\gamma_i} + \sum_{i=p+1}^{q} b_i x_{\gamma_i} x_{\delta_i} + o. \tag{6}$$

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}, x_{β_j} represent variables, where $\alpha_i \in [1, k]$ and $\beta_i \in [1, k]$. Moreover, for every $i \in [1, q]$ and $j \in [p+1, q]$, b_i is a positive integer and x_{γ_i}, x_{δ_j} represent variables, where $\gamma_i \in [1, k]$ and $\delta_i \in [1, k]$. Finally, o is a positive integer.

To codify the previous equation, we need to define a DTD $D = (E, A, P, R, r)$ and a set of constraints Σ. 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. Each element n_i is used to store the value of the variable x_i. In the XML trees that we consider in this proof, each node of type n_i has an attribute v. Two different nodes of type n_i contain different values in v and the total number of values of v in these nodes represents the value of x_i, that is, $|ext(n_i.v)|$ is equal to the value of x_i. Moreover, nodes of type either X or Y also have an attribute v, and they are used, like in the previous case, to store the values of the left hand side and the right hand side of (6), respectively.

We define $P(r)$ as follows:

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

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

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

In order to codify (6) we need to represent the multiplication between two variables. To do this, we use α_i' and γ_i'.

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

For the rest of the elements of the DTD, P is defined as ϵ.

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

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 represent equation (6) we need to add a set of constraints Σ. This set contains the following absolute keys:

$r(X.v \to X)$		$r(Y.v \to Y)$	
$r(\alpha_i.v \to \alpha_i)$	for every $1 \leq i \leq n$	$r(\gamma_i.v \to \gamma_i)$	for every $1 \leq i \leq q$
$r(\beta_i.v \to \beta_i)$	for every $m+1 \leq i \leq n$	$r(\delta_i.v \to \delta_i)$	for every $p+1 \leq i \leq q$
$r(c_i.v \to c_i)$	for every $m+1 \leq i \leq n$	$r(f_i.v \to f_i)$	for every $p+1 \leq i \leq q$
$r(d_i.v \to d_i)$	for every $m+1 \leq i \leq n$	$r(g_i.v \to g_i)$	for every $p+1 \leq i \leq q$
$r(e_i.v \to e_i)$	for every $m+1 \leq i \leq n$	$r(h_i.v \to h_i)$	for every $p+1 \leq i \leq q$
$r(n_i.v \to n_i)$	for every $1 \leq i \leq k$		

Σ contains the following absolute foreign keys:

$r(X.v \subseteq Y.v)$,	$r(Y.v \to Y)$	
$r(Y.v \subseteq X.v)$,	$r(X.v \to X)$	
$r(n_s.v \subseteq \alpha_i.v)$,	$r(\alpha_i.v \to \alpha_i)$	$1 \leq i \leq n$ and the value of α_i is equal to s
$r(\alpha_i.v \subseteq n_s.v)$,	$r(n_s.v \to n_s)$	$1 \leq i \leq n$ and the value of α_i is equal to s
$r(n_s.v \subseteq e_i.v)$,	$r(e_i.v \to e_i)$	$m+1 \leq i \leq n$ and the value of β_i is equal to s
$r(e_i.v \subseteq n_s.v)$,	$r(n_s.v \to n_s)$	$m+1 \leq i \leq n$ and the value of β_i is equal to s
$r(n_s.v \subseteq \gamma_i.v)$,	$r(\gamma_i.v \to \gamma_i)$	$1 \leq i \leq q$ and the value of γ_i is equal to s
$r(\gamma_i.v \subseteq n_s.v)$,	$r(n_s.v \to n_s)$	$1 \leq i \leq q$ and the value of γ_i is equal to s
$r(n_s.v \subseteq h_i.v)$,	$r(h_i.v \to h_i)$	$p+1 \leq i \leq q$ and the value of δ_i is equal to s
$r(h_i.v \subseteq n_s.v)$,	$r(n_s.v \to n_s)$	$p+1 \leq i \leq q$ and the value of δ_i is equal to s

Finally, Σ contains the following relative foreign keys:

$\alpha_i(\beta_i.v \subseteq d_i.v)$	$\alpha_i(d_i.v \to d_i)$	$m+1 \leq i \leq n$
$\alpha_i(d_i.v \subseteq \beta_i.v)$	$\alpha_i(\beta_i.v \to \beta_i)$	$m+1 \leq i \leq n$
$\alpha'_i(\beta_i.v \subseteq c_i.v)$	$\alpha'_i(c_i.v \to c_i)$	$m+1 \leq i \leq n$
$\alpha'_i(c_i.v \subseteq \beta_i.v)$	$\alpha'_i(\beta_i.v \to \beta_i)$	$m+1 \leq i \leq n$
$\gamma_i(\delta_i.v \subseteq g_i.v)$	$\gamma_i(g_i.v \to g_i)$	$p+1 \leq i \leq q$
$\gamma_i(g_i.v \subseteq \delta_i.v)$	$\gamma_i(\delta_i.v \to \delta_i)$	$p+1 \leq i \leq q$
$\gamma'_i(\delta_i.v \subseteq f_i.v)$	$\gamma'_i(f_i.v \to f_i)$	$p+1 \leq i \leq q$
$\gamma'_i(f_i.v \subseteq \delta_i.v)$	$\gamma'_i(\delta_i.v \to \delta_i)$	$p+1 \leq i \leq q$

We need to prove that there is an XML tree T such that $T \models D$ and $T \models \Sigma$ if and only if there exists a non-negative integer solution for (6). To do this, we will prove that every XML tree T satisfying D and Σ codifies equation (6). More precisely, if the value of every variable x_i is v_i and $|ext(n_i.v)| = v_i$, for $i \in [1,k]$, then

$$|ext(X.v)| = \sum_{i=1}^{m} a_i v_{\alpha_i} + \sum_{i=m+1}^{n} a_i v_{\alpha_i} v_{\beta_i}, \tag{7}$$

$$|ext(Y.v)| = \sum_{i=1}^{p} b_i v_{\gamma_i} + \sum_{i=p+1}^{q} b_i v_{\gamma_i} v_{\delta_i} + o. \tag{8}$$

If an XML tree $T \models D$, then there exists n children of the root $x_1, \ldots, x_n$ of types $\alpha_1, \ldots, \alpha_n$, respectively. Every node of type X appears as a descendant of some x_i. Thus, given that $r(X.v \to X) \in \Sigma$, in order to prove (7) it suffices to prove that the number of values of v in the X-nodes that are descendant of x_i ($1 \leq i \leq n$) is equal to the ith term of the left hand of (7), that is,

$$\begin{array}{lcll} |\{ext(x.v) \mid x_i \prec x \text{ and } x \text{ is of type } X\}| & = & a_i v_{\alpha_i} & 1 \leq i \leq m, \\ |\{ext(x.v) \mid x_i \prec x \text{ and } x \text{ is of type } X\}| & = & a_i v_{\alpha_i} v_{\beta_i} & m+1 \leq i \leq n. \end{array}$$

In order to prove (8), we also have to prove that a set of equations is valid. But, this set is analogous to the set of equations for the type X. Thus, we will show only that the equations for nodes of type X are valid.

If $1 \leq i \leq m$, it is easy to see that the total number of X-nodes that are descendant of x_i is equal to $a_i v_{\alpha_i}$. Thus, let i be an integer in $[m+1, n]$, let s be the value of α_i in (6) and let t be the value of β_i in (6).

Next, we prove that $|\{ext(x.v) \mid x_i \prec x \text{ and } x \text{ is of type } X\}| = a_i v_s v_t$ by induction on v_s. Here, we will show only the case $v_s = 2$. It is straightforward to extend this idea for any value of v_s.

Given that $\{r(\alpha_i.v \to \alpha_i),\ r(n_s.v \to n_s),\ r(\alpha_i.v \subseteq n_s.v),\ r(n_s.v \subseteq \alpha_i.v)\} \subseteq \Sigma$, we can deduce that $|ext(\alpha_i.v)| = |ext(n_s.v)| = v_s$. Thus, we know that in T there are exactly v_s nodes of type α_i, since v is a key for α_i. Each of them has a child of type α'_i. Then, there are exactly v_s nodes of type α'_i, and the last one must have children of type e_i, as we show in figure 10.

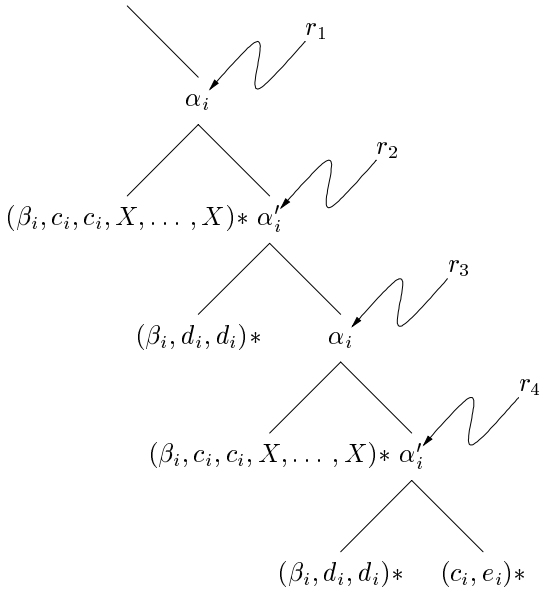


Figure 10: Part of the tree T.

We know that $r(c_i.v \to c_i)$ and $r(e_i.v \to e_i)$ are elements of Σ. Therefore, we can conclude that in the subtree T_4 of T whose root is r_4, $|\{ext(x.v) \mid x \text{ is a child of } r_4 \text{ of type } c_i\}| = |\{ext(x.v) \mid x \text{ is a child of } r_4 \text{ of type } e_i\}|$, since $P(\alpha'_i) = (\beta_i, d_i, d_i)^*, (\alpha_i|(c_i, e_i)^*)$. But, $\{r(n_t.v \to n_t),\ r(e_i.v \to e_i),\ r(n_t.v \subseteq e_i.v),\ r(e_i.v \subseteq n_t.v)\} \subseteq \Sigma$, and, therefore, $|\{ext(x.v) \mid x \text{ is a child of } r_4 \text{ of type } c_i\}| = |ext(n_t.v)|$. Thus, given that r_4 is a node of type α'_i we can use constraints $\alpha'_i(\beta_i.v \to \beta_i)$, $\alpha'_i(c_i.v \to c_i)$, $\alpha'_i(\beta_i.v \subseteq c_i.v)$, $\alpha'_i(c_i.v \subseteq \beta_i.v)$, to conclude that $|\{ext(x.v) \mid x \text{ is a child of } r_4 \text{ of type } \beta_i\}| = |ext(n_t.v)| = v_t$. In addition to this, the number of children of r_4 of type d_i is $2v_t$, because $\alpha'_i(d_i.v \to d_i) \in \Sigma$ and $P(\alpha'_i) = (\beta_i, d_i, d_i)^*, (\alpha_i|(c_i, e_i)^*)$.

Given that r_3 is a node of type α_i we can use constraints $\alpha_i(\beta_i.v \to \beta_i)$, $\alpha_i(d_i.v \to d_i)$, $\alpha_i(\beta_i.v \subseteq d_i.v)$, $\alpha_i(d_i.v \subseteq \beta_i.v)$ to conclude that $|\{ext(x.v) \mid x \text{ is a child of } r_3 \text{ of type } \beta_i\}| = v_t$, since there are $2v_t$ descendants of r_3 of type d_i and v_t children of r_4 of type β_i. In addition to this, 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 $2v_t$, because of the definition of $P(\alpha_i)$.

We can use the argument given in the previous paragraph to conclude, by induction, that the number of children of r_2 of type β_i and the number of its children of type 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 $2a_i v_t$. In general, if we continue with this process we will infer that the number of descendants of x_i of type X is $v_s a_i v_t$, since there are v_s nodes of type α_i. This conclude our proof, since $|\{ext(x.v) \mid x_i \prec x \text{ and } x \text{ is of type } X\}| = a_i v_s v_t$. □

Proof of Theorem 4.3

To prove this theorem we need to introduce some terminology. Let $D = (E, A, P, R, r)$ be a non-recursive DTD, Σ a set of $\mathcal{RC}_{K,FK}$-constraint over D and T an XML tree conforming to D and satisfying Σ. A string $w_1 \cdots w_n$ over E is a path in T if there is a sequence of vertices $v_1, \cdots, v_{n-1}, v_n$ in V such that $v_1 = root$, v_{i+1} is a child of v_i ($i \in [1, n-1]$) and $lab(v_i) = w_i$ ($i \in [1, n]$). $Paths_D(T)$ is the set of paths in T.

Let x be a node in T of type τ. The node x is a *restricted node* if $\tau = r$ or τ is the context type of some

constraint in Σ. If x is a restricted node, then

$$\begin{aligned} \mathit{Scope}(x) \;=\; \{y \mid\; & y \text{ is a node in } T \text{ and } y = x \text{ or there is a path } w = \tau_1.\tau_2.\cdots.\tau_n \text{ such that} \\ & y \text{ is reachable from } x \text{ by following path } w \text{ and for every } i \in [2, n-1], \\ & \tau_i \text{ is not the context type of some constraint in } \Sigma\}. \end{aligned}$$

Moreover, if (D, Σ) is hierarchical, then for each $\tau \in E$ such that $\tau = r$ or τ is the context type of some constraint in Σ, a restricted DTD $D_\tau = (E_\tau, A_\tau, P_\tau, R_\tau, \tau)$ is defined as follows.

1. E_τ is the set of elements types:

$$\begin{aligned} \{\tau' \mid\; & \tau' = \tau \text{ or there is a path } w = \tau_1.\tau_2.\cdots.\tau_n \text{ in } D \text{ from } \tau \text{ to } \tau' \text{ such that} \\ & \text{for every } i \in [2, n-1],\ \tau_i \text{ is not the context type of some constraint in } \Sigma\}. \end{aligned}$$

2. $A_\tau = \{l \in A \mid \text{there is } \tau' \in E_\tau \setminus \{\tau\} \text{ such that } l \in R(\tau')\}$.
3. $P_\tau(\tau) = P(\tau)$. For each $\tau' \in E_\tau \setminus \{\tau\}$, if τ' is the context type of some constraint in Σ, then $P_\tau(\tau') = \epsilon$, otherwise $P_\tau(\tau') = P(\tau')$.
4. For each $\tau' \in E_\tau \setminus \{\tau\}$, $R_\tau(\tau') = R(\tau')$. Moreover, $R_\tau(\tau) = \emptyset$.

It is straightforward to prove the following lemma.

Lemma 10 *Let $D = (E, A, P, R, r)$ be a non-recursive DTD, Σ a set of $\mathcal{RC}_{K,FK}$-constraints over D and T an XML tree conforming to D and satisfying Σ. For each node y in T there is at most one restricted node x in T such that $y \neq x$ and $y \in \mathit{Scope}(x)$.*

To prove theorem 4.3, first we need to prove the following lemma.

Lemma 11 *If there exists a function $f : \mathbb{N} \to \mathbb{N}$ satisfying the following condition:*

1. *For each $(D, \Sigma) \in \mathcal{HRC}_{K,FK}$, if (D, Σ) is consistent, then there is an XML tree T conforming to D and satisfying Σ such that for each restricted node x in T, $|\mathit{Scope}(x)| \leq f(|D_\tau|)$, where τ is the type of x.*

Then $\mathsf{SAT}(\mathcal{HRC}_{K,FK})$ is in NSPACE($\log(f)$).

Proof. In order to prove the lemma, first we need to introduce some terminology. Let $D = (E, A, P, R, r)$ be a non-recursive DTD, Σ a set of $\mathcal{RC}_{K,FK}$-constraints over D, $\tau \in E$ and w a path in D from r to τ. Then, a set of $\mathcal{AC}_{K,FK}$-constraints Σ_w over D_τ is defined as follows.

- For every key $\tau_1(\tau_2.l_2 \to \tau_2) \in \Sigma$, if τ_1 is a symbol in w and $\tau_2 \in E_\tau$, then $\tau_2.l_2 \to \tau_2$ is in Σ_w.
- For every inclusion dependency $\tau(\tau_1.l_1 \subseteq \tau_2.l_2)$, $\tau_1.l_1 \subseteq \tau_2.l_2$ is in Σ_w.

Assume that $(D, \Sigma) \in \mathcal{HRC}_{K,FK}$. To verify whether (D, Σ) is consistent we have to check if there is an XML tree T conforming to D and satisfying Σ such that for each restricted node x in T, $|\mathit{Scope}(x)| \leq f(|D_\tau|)$, where τ is the type of x. In order to do this, we can use a simple non-deterministic algorithm:

```
CheckConsistency(w: path, visited: array)
  Let τ := last(w)
  Generate D_τ
  for each w' ∈ paths(D) do  /* creates a new copy of visited */
    visited'[w'] = visited[w'].
  T := GenerateTree(D_τ, w, visited')
  if T ⊭ Σ_w then return(false)
  else
    for each w' such that visited'[w'] = 1 and P(last(w')) ≠ P_τ(last(w')) do
      if CheckConsistency(w', visited') = false then return(false)
    return(true)
```

In this algorithm, $last(w)$ represents the last element type of the path w and *visited* (*visited'*) is an array indexed by the paths in the DTD D. Function `GenerateTree` guesses and returns an XML tree conforming to D_τ of size at most $f(|D_\tau|)$. Initially, we set up all the values of *visited* as 0 and invoke `CheckConsistency`(r, *visited*). `CheckConsistency` construct the DTD D_r and invokes `GenerateTree`(D_r, r, *visited'*). This function guesses an XML tree T_r of size at most $f(|D_r|)$ conforming to D_r. Besides, for each path $w' \in Paths_{D_r}(T_r)$ it changes the value of *visited'*$[w']$ to 1, indicating that this path has been visited. If $T_r \not\models \Sigma_r$, then it returns *false*. Otherwise, it has guessed the part of an XML tree T that could conforms to D and satisfy Σ. This part correspond to the scope of the root. In order to continue guessing T, the algorithm has to consider all the leaves τ of T_r for which $P(\tau) \neq P_r(\tau)$. For each of them, we recursively apply the same algorithm.

The previous algorithm works in space $O(f)$, since `GenerateTree` guesses XML trees of size at most $f(|D|)$. It is possible to improve this by using the same idea considered in the proof of Theorem 3.5 (b). By using Lemma 9, we can replace the function `GenerateTree` by a new function `Count`.

```
Count(D_1: DTD, w: path, s: regular expression, visited: array)
    if s = ε then return(values of ext)
    else if s ∈ E_1 then     /* D_1 = (E_1, A_1, P_1, R_1, r_1) */
        |ext(s)| := |ext(s)| + 1
        visited[w.s] := 1
        for each l ∈ R_1(s) do
            if |ext(s.l)| = 0 then |ext(s.l)| := 1
            else
                flip a coin
                if "head" then |ext(s.l)| := |ext(s.l)| + 1
        Count(D_1, w.s, P_1(s), visited)
    else if s = s_1, s_2 then
        Count(D_1, w, s_1, visited)
        Count(D_1, w, s_2, visited)
    else if s = s_1|s_2 then
        flip a coin
        if "head" then Count(D_1, w, s_1, visited)
        else Count(D_1, w, s_2, visited)
    else if s = s_1^* then
        guess n ∈ [0, f(|D_1|)]
        for i := 1 to n do
            Count(D_1, w, s_1, visited)
```

This algorithm is invoked as `Count`(D_τ, w', τ, *visited*). It guesses an XML tree T conforming to D_τ and instead of storing this tree, for each element node $s \in E_\tau$ and $l \in R_\tau(s)$ it stores $|ext(s)|$ and $|ext(s.l)|$. For each path $w \in Paths(D_\tau)$, it changes the value of *visited*$[w'.w]$ to 1, where w' is a path from the root to an element node that has at least one child of type τ.

Initially, $|ext(x)| = 0$ for each $x \in E_\tau$. In each step, `Count` verifies whether the regular expression s is an element type. If this is the case, it increments $|ext(s)|$, since it found another node of type s, and for each $l \in R_\tau(s)$, it flips a coin for deciding whether to assign a new value to the attribute l (and then increment $|ext(s.l)|$) or to copy this value from attribute l of another node of type s (and leave $|ext(s.l)|$ intact). Notice that if $|ext(s.l)| = 0$, then it is not possible to copy the value of l from another node of type s, and, therefore, the algorithm must assign the value 1 to $|ext(s.l)|$. Finally, for each $s \in E$, it recursively invokes to `Count`(D_τ, $w.s$, $P_\tau(s)$, *visited*).

If $s \notin E$, then $s = (s_1, s_2)$, $s = (s_1|s_2)$ or $s = \epsilon$, where s_1 and s_2 are regular expressions. If $s = (s_1, s_2)$, then in order to generate an XML tree conforming to D the algorithm invokes to `Count`(D_τ, w, s_1, *visited*) and `Count`(D_τ, w, s_2, *visited*). If $s = (s_1|s_2)$, then in order to generate an XML tree conforming to D, the algorithm nondeterministically decides to invoke to `Count`(D_τ, w, s_1, *visited*) or `Count`(D_τ, w, s_2, *visited*). Finally, if $s = \epsilon$, then the algorithm returns the values of $|ext(s)|$ for each $s \in E_\tau$ and $|ext(s.l)|$ for each $s \in E_\tau$ and $l \in R_\tau(s)$.

A new version of `CheckConsistency` that uses `Count` is defined as follows.

```
CheckConsistency(w: path, visited: array)
  Let τ := last(w) and w_1 be the path obtained from w by removing its last element
  Generate D_τ
  for each w' ∈ paths(D) do
    visited'[w'] = visited[w'].
  values of ext := Count(D_τ, w_1, τ, visited')
  if values of ext do not satisfy C_{Σ_w} then return(false)
  else
    for each w' such that visited'[w'] = 1 and P(last(w')) ≠ P_τ(last(w')) do
      if CheckConsistency(w', visited') = false then return(false)
    return(true)
```

Notice that the previous algorithm works in space $O(log(f))$, since `Count` works in space $O(log(f))$ (all the values stored by this function are less than or equal to $f(|D|)$). Thus, we conclude that $\mathsf{SAT}(\mathcal{HRC}_{K,FK})$ is in NSPACE(log(f)). □

In order to finish the proof of theorem 4.3, we have to find the function f mentioned in the previous lemma. We do this in the following lemmas.

Lemma 12 *Given a non-recursive DTD D and a set Σ of $\mathcal{AC}_{K,FK}$-constraints over D, there is a fixed polynomial p such that if there is an XML tree T conforming to D and satisfying Σ, then there is an XML tree T' conforming to D and satisfying Σ such that $|T'| \leq 2^{2^{p(|D|)}}$ and $Paths_D(T') \subseteq Paths_D(T)$.*

Proof. In [14] it is shown that the consistency problem for DTDs and $\mathcal{AC}_{K,FK}$-constraints is in NP. Given a DTD $D = (E, A, P, R, r)$ and a set Σ of $\mathcal{AC}_{K,FK}$-constraints, [14] shows how to construct a polynomial size system of linear integer equations $\Phi(D, \Sigma)$ including, among others, variables $\{x_\tau \mid \tau \in E\}$ and $\{x_{\tau.l} \mid \tau \in E \text{ and } l \in R(\tau)\}$. The system admits a solution if and only if there is an XML tree T conforming to D and satisfying Σ such that $|ext(\tau)|$ is equal to the value of x_τ, for each $\tau \in E$, and $|ext(\tau.l)|$ is equal to the value of $x_{\tau.l}$, for each $\tau \in E$ and $l \in R(\tau)$. We will use this result in order to prove the lemma.

Let D_0 be a DTD (E, A, P, R, r) and Σ_0 be a set of $\mathcal{AC}_{K,FK}$-constraints. In order to prove this lemma, we need to define from D_0 a new DTD $D_1 = (E_1, A, P_1, R_1, r)$ such that

- $E_1 = E \cup Paths(D_0)$.
- For each $\tau \in E \setminus \{r\}$, $P_1(\tau) = \epsilon$. For each $w \in Paths(D_0)$, $P_1(w) = f_w(P(last(w)))$, where $last(w)$ is the last symbol of w and f_w is a homomorphism defined as $f_w(\tau') = \tau', w.\tau'$ ($\tau', w.\tau'$ is a string with two symbols: the element type τ' and the path $w.\tau'$), for each τ' in the alphabet of $P(last(w))$.
- For each $\tau \in E$, $R_1(\tau) = R(\tau)$. Moreover, for each $w \in Paths(D_0)$, $R_1(w) = \emptyset$.

It is straightforward to verify the following lemma.

Lemma 13 *There is an XML tree T_0 conforming to D_0 and satisfying Σ_0 iff there is an XML tree T_1 conforming to D_1 and satisfying Σ_0 such that:*

1. *For each $\tau \in E$, $|ext(\tau)|$ in T_0 is equal to $|ext(\tau)|$ in T_1.*
2. *For each $\tau \in E$ and $l \in R(\tau)$, $|ext(\tau.l)|$ in T_0 is equal to $|ext(\tau.l)|$ in T_1.*
3. *For each $w \in Paths(D_0)$, $|ext(w)|$ in T_1 is equal to the number of nodes in T_0 reachable from the root by following the path w.*

Suppose that T_0 is an XML tree conforming to D_0 and satisfying Σ_0. Then, there is an XML tree T_1 conforming to D_1, satisfying Σ_0 and satisfying the conditions mentioned in Lemma 13. Thus, the system of linear equations $\Phi(D_1, \Sigma_0)$ admits a solution. Moreover, the system of linear equations

$$\Phi'(D_1, \Sigma_0) \quad = \quad \Phi(D_1, \Sigma_0) \cup \{x_w = 0 \mid w \in Paths(D_0) \setminus Paths_{D_0}(T_0)\}$$

admits a solution since T_1 satisfies the third condition of the lemma. But, if this system has a solution, it has a polynomial size solution [24]. Thus, given that the size of $\Phi'(D_1, \Sigma_0)$ is polynomial in the size of D_1 and Σ_0, we conclude that there is a fixed polynomial s and a solution of $\Phi'(D_1, \Sigma_0)$ such that for every variable x in $\Phi'(D_1, \Sigma_0)$, the value of x is at most $2^{s(|D_1|+|\Sigma_0|)}$. Moreover, for each $w \in Paths(D_0) \setminus Paths_{D_0}(T_0)$ the value of x_w is equal to 0.

Therefore, by using the result in [14] we conclude that there is an XML tree T_1' conforming to D_1 and satisfying Σ_0 such that $|ext(\tau)| \leq 2^{s(|D_1|+|\Sigma_0|)}$ for each $\tau \in E$, $|ext(\tau.l)| \leq 2^{s(|D_1|+|\Sigma_0|)}$ for each $\tau \in E$ and $l \in R(\tau)$, $|ext(w)| \leq 2^{s(|D_1|+|\Sigma_0|)}$ for each $w \in Paths(D_0)$, and $|ext(w)| = 0$ for each $w \in Paths(D_0) \setminus Paths_{D_0}(T_0)$. The size of D_1 is exponential in the size of D_0 and the size of Σ_0 is polynomial in the size of D_0 (it is easy to prove that $|\Sigma_0|$ is $O(|E|^3|A|^2)$). Therefore, by using Lemma 13 we conclude that there is fixed polynomial p and an XML tree T_0' conforming to D_0 and satisfying Σ_0 such that $|T_0'| \leq 2^{2^{p(|D_0|)}}$. Moreover, given that $|ext(w)| = 0$ in T_1' for each $w \in Paths(D_0) \setminus Paths_{D_0}(T_0)$, we deduce that $Paths_{D_0}(T_0') \subseteq Paths_{D_0}(T_0)$. This proves the lemma. □

Lemma 14 *Given a non-recursive DTD $D = (E, A, P, R, r)$ and a set Σ of $\mathcal{RC}_{K,FK}$-constraints over D, if (D, Σ) is consistent and hierarchical, then there is an XML tree T conforming to D and satisfying Σ such that for each restricted node x in T, $|Scope(x)| \leq 2^{2^{p(|D_\tau|)}}$, where τ is the type of x and p is the polynomial mentioned in Lemma 12.*

Proof. Let $D = (E, A, P, R, r)$ be a non-recursive DTD and Σ be a set of $\mathcal{RC}_{K,FK}$-constraints over D such that (D, Σ) is hierarchical. Suppose that $T = (V, lab, ele, att, val, root)$ is an XML tree such that $T \models D$ and $T \models \Sigma$. We will prove that there exists an XML tree T' conforming to D and satisfying Σ such that for each restricted node x in T' of type τ, $|Scope(x)| \leq 2^{2^{p(|D_\tau|)}}$. To do this, we will inductively define T' from T.

Let $D_r = (E_r, A_r, P_r, R_r, r)$ and Σ_r be the following set of $\mathcal{AC}_{K,FK}$-constraints over D_r:

- For every key $r(\tau_1.l_1 \rightarrow \tau_1) \in \Sigma$, if $\tau_1 \in E_r$, then $\tau_1.l_1 \rightarrow \tau_1 \in \Sigma_r$.
- For every inclusion dependency $r(\tau_1.l_1 \subseteq \tau_2.l_2) \in \Sigma$, $\tau_1.l_1 \subseteq \tau_2.l_2 \in \Sigma_r$.

Notice that in the last rule we do not have to impose the condition $\tau_1, \tau_2 \in E_r$, since (D, Σ) is hierarchical.

Given that (D, Σ) is consistent, we can conclude that (D_r, Σ_r) is consistent. Let T_r be a subtree of T whose element nodes are $Scope(root)$. In this subtree we do not include the attributes of r. Moreover, for each node $x \in Scope(root) \setminus \{root\}$, we also include all the children of x whose label is S and all the attributes of x. It is straightforward to prove that T_r conforms to D_r and satisfies Σ_r. Thus, we can use Lemma 12 to conclude that there is an XML tree T_r' conforming to D_r and satisfying Σ_r such that $|T_r'| \leq 2^{2^{p(|D_r|)}}$ and $Paths_{D_r}(T_r') \subseteq Paths_{D_r}(T_r)$.

In order to construct the first levels of the tree T', we give a fresh set of values to the attributes of *root* and we include T_r' as a subtree of T' rooted at note *root*, as it is shown in figure 11. We call this subtree T_1.

Notice that T_1 does not necessarily satisfy the DTD D, since in order to construct D_r for some elements types τ we redefine $P(\tau)$ as $P_r(\tau) = \epsilon$. These elements correspond to some of the leaves of the tree T_1.

Let x be a node in T_1 of type τ, where τ is an element type such that $P(\tau) \neq P_r(\tau)$. Let w be the path in T_1 such that x is reachable from the root by following w. We define Σ_w as a set $\mathcal{AC}_{K,FK}$-constraints over the restricted DTD D_τ:

- For every key $\tau_1(\tau_2.l_2 \rightarrow \tau_2) \in \Sigma$, if τ_1 is a symbol in w and $\tau_2 \in E_\tau$, then $\tau_2.l_2 \rightarrow \tau_2$ is in Σ_w.
- For every inclusion dependency $\tau(\tau_1.l_1 \subseteq \tau_2.l_2)$, $\tau_1.l_1 \subseteq \tau_2.l_2$ is in Σ_w.

Notice that if there is an element type τ_1 such that $\tau_1(\tau_2.l_2 \subseteq \tau_3.l_3)$ is an inclusion dependency in Σ, τ_1 is an ancestor of τ and τ_1 appears as a symbol in w, then neither τ_2 nor τ_3 can be a descendant of τ, since (D, Σ) is hierarchical. Thus, we need only to consider inclusion dependencies of the form $\tau(\tau_2.l_2 \subseteq \tau_3.l_3)$ in order to define Σ_w.

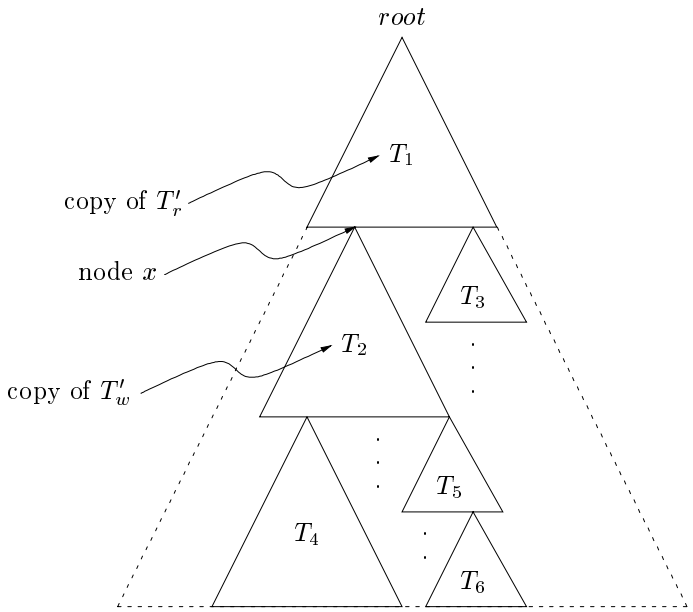


Figure 11: Construction of tree T'.

Given that $Paths_{D_r}(T'_r) \subseteq Paths_{D_r}(T_r)$, there is a node y in T of type τ such that y is reachable from the root of T by following the path w. It is easy to verify that the subtree T_w of T whose nodes are $Scope(y)$ conforms to D_τ and satisfies Σ_w, since $T \models \Sigma$ and $T \models D$. Thus, we can apply Lemma 12 to conclude that there is a tree T'_w conforming to D_τ and satisfying Σ_w such that $|T'_w| \leq 2^{2^{p(|D_\tau|)}}$. We define a new tree T_2 by giving a fresh set of values to the attributes of T'_w. We put this tree as the subtree of T' rooted at node x, as it is shown in figure 11.

We continue defining T' until we reach the last level in T. We get a tree T' conforming to D and satisfying Σ such that for every restricted node x in T', $|Scope(x)| \leq 2^{2^{p(|D_\tau|)}}$, where τ is the type of x. Notice that T' satisfies all the keys in Σ, since it locally satisfies these constraints, for the set of nodes in the scope of a restricted node, and every time that we copy a subtree to T' we use different values for the attributes. It also satisfies all the foreign keys in Σ since each of them is applicable only to the scope of some restricted node. □

Proof of Theorem 4.4

It is convenient to reformulate the notion of d-locality using the DTDs D_τ, for τ a context type of a Σ constraint, introduced in the proof of Theorem 4.3. Namely, (D, Σ) is d-local, if

1. $Depth(D_r) \leq d$.
2. For each $\tau \in E$, if τ is the context type of some constraint in Σ, then $Depth(D_\tau) \leq d$.

Now the membership in PSPACE is a consequence of lemma 11. We establish the PSPACE-hardness of $\mathsf{SAT}(2\text{-}\mathcal{HRC}_{K,FK})$ by reduction from the QBF-CNF problem: Given a quantified boolean formula θ in prenex conjunctive normal form, determine whether θ is valid. It is a known result that QBF-CNF is PSPACE-complete.

Let θ be a formula of the form

$$Q_1 x_1 \cdots Q_m x_m \psi, \tag{9}$$

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 only variables $x_1, \ldots, x_m$. We construct a DTD $D = (E, A, P, R, r)$ representing this formula as follows.

- $E = \{r, C\} \cup \bigcup_{i=1}^{m} \{x_i, \bar{x}_i, 1_{x_i}, 0_{x_i}, N_{x_i}, P_{x_i}, A_{x_i}, B_{x_i}\}$
- $A = \{v\}$.
- In order to define the function P, firstly we consider the quantifiers of θ. Thus, we consider Q_1 in order to define P on the root r:

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

  In general, for each $1 \leq i \leq m-1$, we consider quantifier Q_{i+1} in order 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 need to represent the formula ψ as a regular expression, in order to define P on the type elements N_{x_m} and P_{x_m}. Given a clause $C_j = \bigvee_{i=1}^{p} y_i \vee \bigvee_{i=1}^{q} \neg z_i$, $t(C_j)$ is defined as the regular expression $y_1| \cdots |y_p|\bar{z}_1| \cdots |\bar{z}_q$. We use t to define $P(N_{x_m})$ and $P(P_{x_m})$:

$$P(N_{x_m}) = P(x_{x_m}) = C, (0_{x_1}, A_{x_1}, A_{x_1} \mid 1_{x_1}, B_{x_1}, B_{x_1}), \ldots , \\ (0_{x_m}, A_{x_m}, A_{x_m} \mid 1_{x_m}, B_{x_m}, B_{x_m}), t(C_1), \ldots , t(C_n).$$

  Finally, for the rest of the elements in E, we define P as follows:

$$P(C) = \epsilon \\ P(0_{x_i}) = P(1_{x_i}) = P(\bar{x}_i) = P(x_i) = P(A_{x_i}) = P(B_{x_i}) = \epsilon \quad 1 \leq i \leq m$$

- We define the function R as:

$$R(r) = \emptyset \\ R(P_{x_i}) = R(N_{x_i}) = \emptyset \quad 1 \leq i \leq m \\ R(C) = \{v\} \\ R(0_{x_i}) = R(1_{x_i}) = R(\bar{x}_i) = R(x_i) = R(A_{x_i}) = R(B_{x_i}) = \{v\} \quad 1 \leq i \leq m.$$

It is also necessary to define a set of constraints Σ such that (D, Σ) is hierarchical and 2-local. In this case, Σ contains the following constraints:

$N_{x_i}(B_{x_i}.v \to B_{x_i})$		$1 \leq i \leq m$
$P_{x_i}(A_{x_i}.v \to A_{x_i})$		$1 \leq i \leq m$
$N_{x_m}(x_i.v \subseteq 1_{x_i}.v)$	$N_{x_m}(1_{x_i}.v \to 1_{x_i})$	$1 \leq i \leq m$
$N_{x_m}(\bar{x}_i.v \subseteq 0_{x_i}.v)$	$N_{x_m}(0_{x_i}.v \to 0_{x_i})$	$1 \leq i \leq m$
$N_{x_m}(A_i.v \subseteq C.v)$	$N_{x_m}(C.v \to C)$	$1 \leq i \leq m$
$N_{x_m}(B_i.v \subseteq C.v)$	$N_{x_m}(C.v \to C)$	$1 \leq i \leq m$
$P_{x_m}(x_i.v \subseteq 1_{x_i}.v)$	$P_{x_m}(1_{x_i}.v \to 1_{x_i})$	$1 \leq i \leq m$
$P_{x_m}(\bar{x}_i.v \subseteq 0_{x_i}.v)$	$P_{x_m}(0_{x_i}.v \to 0_{x_i})$	$1 \leq i \leq m$
$P_{x_m}(A_i.v \subseteq C.v)$	$P_{x_m}(C.v \to C)$	$1 \leq i \leq m$
$P_{x_m}(B_i.v \subseteq C.v)$	$P_{x_m}(C.v \to C)$	$1 \leq i \leq m$

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 12. 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} has 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 $\forall x_3$.

In figure 12, 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 $\sigma_u(x_3) = 0$. It is necessary to verify that all these assignments satisfy the formula

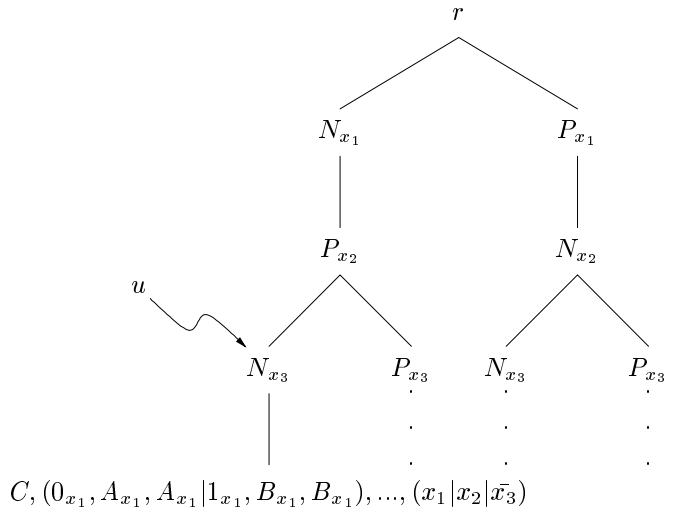


Figure 12: An XML tree conforming to the DTD constructed from $\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee \neg x_3)$.

$x_1 \vee x_2 \vee \neg x_3$. In order to do this, firstly we use the set of constraint Σ to enforce that for every node n of type N_{x_3} or P_{x_3}, its children of type either 0_{x_i} or 1_{x_i} represents the truth assignment from the root to n. For example, u must have a children of type 0_{x_1} ($\sigma_u(x_1) = 0$) and this is enforced by constraints $N_{x_1}(B_{x_1}.v \rightarrow B_{x_1})$, $N_{x_3}(B_{x_1}.v \subseteq C.v)$. If u has a child of type 1_{x_1}, then it has two children of type B_{x_1}, by definition of $P(N_{x_3})$. But, v is a key for the children of u of type B_{x_1} and it has only one child of type C, a contradiction. Secondly, we use the constraints in Σ of the form $N_{x_3}(x_i.v \subseteq 1_{x_i}.v)$, $N_{x_3}(\bar{x}_i.v \subseteq 0_{x_i}.v)$, $P_{x_3}(x_i.v \subseteq 1_{x_i}.v)$, $P_{x_3}(\bar{x}_i.v \subseteq 0_{x_i}.v)$ $(1 \leq i \leq 3)$ to check whether the truth assignments defined by the tree satisfy $x_1 \vee x_2 \vee \neg x_3$. For instance, σ_u satisfies this formula if and only if it is possible to choose a child of u from $(x_1|x_2|\bar{x}_3)$ and it is possible to give a value to the attribute v that satisfies constraints $N_{x_3}(x_i.v \subseteq 1_{x_i}.v)$, $N_{x_3}(\bar{x}_i.v \subseteq 0_{x_i}.v)$ $(1 \leq i \leq 3)$. In this case, we can choose either x_2 or $\bar{x}_3$ as child of u in order to satisfy these constraints.

In general, we need to prove that θ, defined in (9), is valid if and only if there is an XML tree T such that $T \models D$ and $T \models \Sigma$. We will show only the "if" direction. The "only if" direction is similar.

Suppose that there is an XML tree T such that $T \models D$ and $T \models \Sigma$. Let n_1 be a node of T of type N_{x_i} and n_2 be a descendant of n_1 of type N_{x_m} or P_{x_m}. Given that $T \models \Sigma$, v must be a key for all the descendants of n_1 of type B_{x_i}, since $N_{x_i}(B_{x_i}.v \rightarrow B_{x_i}) \in \Sigma$. In particular, v must be a key for the elements of type B_{x_i} that are children of n_2. Thus, if 1_{x_i} is a child of n_2, then n_2 has two children of type B_{x_i}, since $T \models D$, with different values in the attribute v. But, in this case T cannot satisfy the constraints $N_{x_m}(B_i.v \subseteq C.v)$, $P_{x_m}(B_i.v \subseteq C.v)$, since n_2 has only one child of type C, a contradiction. Therefore, n_2 has a child of type 0_{x_i}.

By using an argument analogous to the previous one, it can be verified that if n_1 is a node of type P_{x_i} and n_2 is a descendant of n_1 of type N_{x_m} or P_{x_m}, then n_2 has a child of type 1_{x_i}. Thus, a path from the root to a node n of type N_{x_m} or P_{x_m} represents a truth assignment for the variables x_1, ..., x_m, say σ. But, $T \models \Sigma$ and this set of constraints contains the formulas $N_{x_m}(x_i.v \subseteq 1_{x_i}.v)$, $N_{x_m}(\bar{x}_i.v \subseteq 0_{x_i}.v)$, $P_{x_m}(x_i.v \subseteq 1_{x_i}.v)$, $P_{x_m}(\bar{x}_i.v \subseteq 0_{x_i}.v)$. Thus, σ must satisfy ψ, since $T \models D$ and, therefore, for every clause in ψ it is possible to pick up a literal which is satisfied by this truth assignment.

By the previous paragraph, we know that every path in T, from the root to a node of type N_{x_m} or P_{x_m}, represents a truth assignment for the variables x_1, ..., x_m that satisfies ψ. By definition of D, these paths represent the quantifiers of θ: For every existential quantifier Q_i we choose either the value 0 (represented by N_{x_i}) or 1 (represented by P_{x_i}) and for every universal quantifier Q_j we choose both values (N_{x_j} and P_{x_j}). Thus, we can conclude that θ is valid. $\square$