# Finite Satisfiability of Keys and Foreign Keys for XML Data

**Wenfei Fan**
Temple University
fan@joda.cis.temple.edu

**Leonid Libkin**
Bell Laboratories
libkin@bell-labs.com

**Abstract**

Key and foreign key constraints are useful for XML [5] data in semantic specification, query optimization and more importantly, for information preservation in data exchange. Several XML proposals, e.g., XML Schema [28] and XML Data [21], support key and foreign key specifications. These constraints, however, may not be finitely satisfiable in the XML context. More specifically, given a DTD $D$ and a finite set $\Sigma$ of key and foreign key constraints, there may not exist any finite XML documents that both conform to $D$ and satisfy $\Sigma$. This paper investigates finite satisfiability problems associated with key and foreign key constraints for XML. We show that DTDs impose a class of cardinality constraints on XML documents, and these cardinality constraints interact with keys and foreign keys. We investigate the interaction and in particular, establish complexity results for the analysis of finite satisfiability of keys and foreign keys defined in two constraint languages. One language, $L_u$, defines unary keys and foreign keys in terms of XML attributes, and the other, $L$, defines multi-attribute keys and foreign keys. We show that the finite satisfiability problem for $L_u$ constraints is NP-complete, and the finite satisfiability problem for $L$ constraints is undecidable. We also establish PTIME decidability for several special cases of the finite satisfiability problems.

## 1 Introduction

Keys and foreign keys have proved useful for databases. They describe an essential part of the semantics of data and are fundamental to conceptual database design. In relational databases, they provide the means by which one tuple may refer to another tuple. They are also useful in update anomaly prevention, query optimization and index design [1, 29].

Key and foreign key constraints are also important for XML [5] data. They provide a robust mechanism of identifying and referring to portions of an XML document. The current reference mechanism of the XML standard relies on ID and IDREF attributes. It is rather weak since, as opposed to keys and foreign keys, ID and IDREF attributes are neither typed nor scoped. In other words, one has little control over what a reference points to. This mechanism fails to capture the semantics of native XML documents. Worse still, it is not capable of preserving the semantics of data in data exchange. Typically XML documents are generated from persistent databases and then sent over a network to an application. As mentioned earlier, keys and foreign keys convey a fundamental part of the semantic of data in databases that one cannot afford to lose. These constraints, however, cannot be precisely expressed by ID and IDREF attributes, and as a result, in data exchange databases may suffer loss of information when translated into an XML encoding. To overcome this limitation, several XML proposals, e.g., XML Schema [28] and XML Data [21], support key and foreign key specifications.

An important question in connection with keys and foreign keys for XML is the question of finite satisfiability: given a finite set $\Sigma$ of keys and foreign keys and a DTD (Document Type Definition [5]) $D$, whether there exists a finite XML document that both satisfies $\Sigma$ and conforms to $D$? XML documents are typically modeled as ordered node-labeled trees, referred to as *XML trees*, in which vertices are labeled with element or attribute tags. There is an one to one mapping between XML documents and XML trees when the order of attributes is disregarded. The tree model has been adopted by query languages, specifications and APIs for XML, e.g., XSL [14, 30], XQL [27], XML Schema [28] and DOM [3]. In the tree model, the finite satisfiability problem is to determine whether there exists a finite XML tree that both satisfies $\Sigma$ and conforms to $D$. In contrast to the analysis of finite satisfiability of keys and foreign keys for relational databases, the analysis for XML is not trivial: one may not be able to find any finite XML tree that both conforms to $D$ and satisfies $\Sigma$, although when considered separately, $D$ has some finite

XML tree $T_1$ conforming to it and $\Sigma$ can be satisfied by a finite tree $T_2$. This is because keys and foreign keys interact with DTDs. More specifically, DTD grammars impose a class of constraints, referred to as *cardinality constraints*, and there is interaction between cardinality constraints and key, foreign key constraints. Moreover, the interaction is not simple. To answer the question of finite satisfiability, it is important to understand the interaction. One wants to know whether a set of natural constraints and a perfectly natural DTD, when put together, could specify any finite XML documents. The problem, however, has received little attention. To the best of our knowledge, no previous work has studied the interaction and the finite satisfiability problem for XML.

This paper investigates finite satisfiability of key and foreign key constraints for XML data. We explore cardinality constraints imposed by XML DTDs and study their interaction with key and foreign key constraints. We also establish complexity results for determining finite satisfiability of key and foreign key constraints in the XML setting.

**Examples.** To illustrate the interaction between XML DTDs and key, foreign key constraints, let us consider a few concrete examples. Below is a DTD, $D_1$, which specifies a collection of teachers:

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

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

Assume each teacher has an attribute `name` and each subject has an attribute `taught_by`. Recall that XML attributes have string values. Here we consider single-valued attributes. That is, if an attribute $l$ is defined for an element type $\tau$ in a DTD, then in a document valid w.r.t. the DTD, each element of type $\tau$ must have a unique $l$ attribute that has a string value. Consider a set of simple unary key and foreign key constraints, $\Sigma_1$:

$$\begin{array}{ll} \textit{teacher.name} & \to \textit{teacher}, \\ \textit{subject.taught\_by} & \to \textit{subject}, \\ \textit{subject.taught\_by} & \subseteq \textit{teacher.name}. \end{array}$$

That is, `name` is a key of `teacher` elements, `taught_by` is a key of `subject` elements and it is also a foreign key referencing `name` of `teacher` elements. More specifically, referring to an XML tree $T$, the first constraint asserts that two distinct `teacher` nodes in $T$ cannot have the same `name` attribute value. In other words, for any nodes $x$ and $y$ in $T$ labeled `teacher`, if the (string) values of their `name` attributes are equal, then $x$ and $y$ must be the same node. It should be mentioned that two notions of equality are used in the definition of keys: we assume string *value* equality when comparing `name` attribute values of $x$ and $y$, and *vertex* equality when it comes to comparing $x$ and $y$. This provides a finer sharing and identifying semantics than that found in relational databases. The second constraint states that no two distinct `subject` nodes in $T$ can have the same `taught_by` attribute value. The third constraint, a foreign key, asserts that for any node $x$ in $T$ labeled `subject`, there is a node $y$ in $T$ labeled `teacher` such that the value of the `taught_by` attribute of $x$ equals to the value of the `name` attribute of $y$. In addition, observe that `name` is a key of `teacher`. Thus by the foreign key, the value of the `taught_by` attribute of a `subject` node uniquely identifies a `teacher` node.

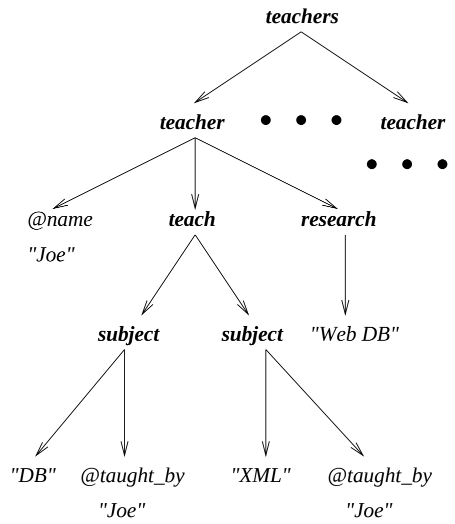


**Figure 1**: An XML tree conforming to $D_1$

Obviously, there exists a finite XML tree conforming to $D_1$, as shown in Fig. 1. However, there is no finite XML tree that both conforms to $D_1$ and satisfies $\Sigma_1$. To see this, let us first define some notations. Given an XML tree $T$ and an element type $\tau$, we use $ext(\tau)$ to denote the set of all the nodes labeled $\tau$ in $T$. Similarly, given an attribute $l$ of $\tau$, we use $ext(\tau.l)$ to denote the set of $l$ attribute values of all $\tau$ elements. Let us denote the cardinality of a set $S$ by $|S|$. Then immediately from $\Sigma_1$ follows a set of dependencies:

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

Therefore,

$$|ext(subject)| \leq |ext(teacher)|. \qquad (1)$$

On the other hand, the DTD $D_1$ requires that each teacher must teach two subjects. Since no sharing of nodes is allowed in XML trees, from $D_1$ follows:

$$2\,|ext(teacher)| = |ext(subject)|. \qquad (2)$$

Thus $|ext(teacher)| < |ext(subject)|$. Obviously, (1) and (2) contradict with each other and therefore, there exists no finite XML tree that both satisfies $\Sigma_1$ and conforms to $D_1$. In particular, the XML tree in Fig. 1 violates the key constraint $subject.taught\_by \rightarrow subject$.

This example demonstrates that a DTD may impose on XML trees dependencies among cardinalities of certain sets of objects. Let us refer to the dependencies as *cardinality constraints*. These cardinality constraints interact with key and foreign key constraints. More specifically, key and foreign key constraints also enforce a class of cardinality constraints, and there is interaction between these two classes of cardinality constraints. This makes the analysis of finite satisfiability of keys and foreign keys for XML far more intriguing than that for relational databases. Because of the interaction, simple key and foreign key constraints (e.g., $\Sigma_1$) may not be satisfiable by finite XML trees conforming to certain DTDs (e.g., $D_1$).

As another example, let us consider a DTD $D_2$ for states and capitals:

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

It specifies that a database consists of a non-empty sequence of states and a non-empty sequence of capitals, and a state has a capital and other cities. Here again we omit the descriptions of elements whose type is string. Assume that `state` has an attribute `name` and `capital` has an attribute `in_state`. It is natural to consider $\Sigma_2$, a set of simple unary key and foreign key constraints:

$$\begin{array}{ll} state.name & \rightarrow state, \\ capital.in\_state & \rightarrow capital, \\ capital.in\_state & \subseteq state.name. \end{array}$$

These assert that `name` is a key of `state` elements, `in_state` is a key of `capital` elements and is a foreign key referring to `name` of `state` elements. From these follows immediately:

$$|ext(capital)| \leq |ext(state)|. \qquad (3)$$

Taking the DTD $D_2$ alone, there exists a finite XML tree conforming to $D_2$, as shown in Fig.2. However,

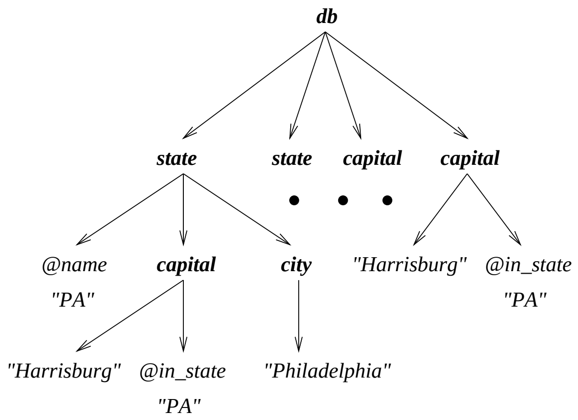


**Figure 2**: An XML tree conforming to $D_2$

observe that $D_2$ requires that each `state` node must have a `capital` as a child and in addition, the `db` node must have a non-empty sequence of `capital` children that are not children of any `state` node. Therefore, the following cardinality constraint is enforced by $D_2$:

$$|ext(state)| < |ext(capital)|. \qquad (4)$$

Observe that (4) contradicts to (3) given above. Again, the DTD $D_2$ interacts with the set $\Sigma_2$ of key and foreign key constraints and as a result, $\Sigma_2$ cannot be satisfied by finite XML trees conforming to $D_2$. In particular, the XML tree given in Fig. 2 does not satisfy the key constraint $capital.in\_state \rightarrow capital$, because two distinct `capital` nodes have the same `in_state` attribute value.

Some XML DTDs do not have finite XML trees conforming to them even in the absence of key and foreign key constraints. For instance, there exists no finite trees conforming to the DTD $D_3$ given below:

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

In other words, even given an empty set $\Sigma_3$ of constraints, $D_3$ still does not have any finite XML tree conforming to it.

**Contributions.** We consider two classes of key and foreign key constraints defined in terms of XML attributes, and investigate their interaction with DTDs. One class, denoted by $L_u$, consists of unary keys and foreign keys, and the other, denoted by $L$, is composed of multi-attribute keys and foreign keys. The main contributions of the paper are the following:

1. We show that $L_u$ constraints can be captured by cardinality constraints. Given a finite set $\Sigma$ of $L_u$ constraints, we define a finite set $C_\Sigma$ of cardinality

constraints such that over finite XML trees, $\Sigma$ is satisfiable if and only if $C_\Sigma$ is satisfiable.

2. We explore the connection between DTDs and cardinality constraints. Given a DTD $D$, we define a finite set $\Psi_D$ of cardinality constraints that characterizes $D$: there exists a finite XML tree conforming to $D$ if and only if $\Psi_D$ is finitely satisfiable.

3. We investigate the finite satisfiability problem for $L_u$ constraints: given any DTD $D$ and any finite set $\Sigma$ of $L_u$ constraints over $D$, whether there exists a finite XML tree that both conforms to $D$ and satisfies $\Sigma$. We show the problem is NP-complete. More specifically, we show it is NP-hard by reduction from linear integer programming problem (see, e.g., [20, 25]). To show it is in NP, we use $\Psi(D, \Sigma)$, which is defined in terms of $\Psi_D$ and $C_\Sigma$, to characterize $D$ and $\Sigma$: there exists a finite XML tree conforming to $D$ and satisfying $\Sigma$ if and only if $\Psi(D, \Sigma)$ admits an integer solution. We show that there is a NP algorithm for guessing and checking integer solution of $\Psi(D, \Sigma)$.

4. We establish the undecidability of the finite satisfiability problem for $L$ constraints by reduction from an undecidable finite implication problem associated with keys and foreign keys for relational databases.

5. We show that several special cases of the finite satisfiability problems are decidable in PTIME. In particular, taking advantage of the characterization $\Psi(D, \Sigma)$, we show that the problem for $L_u$ can be decided in PTIME when the DTD $D$ is fixed.

These complexity results are rather surprising. Recall that keys and foreign keys for relational databases are always satisfiable as long as there is no typing error in the given constraints. Indeed, given any relational schema $R$ and any set $\Sigma$ of keys and foreign keys over $R$, one can always construct a database instance $I$ of $R$ such that $I$ satisfies $\Sigma$ and moreover, $I$ contains one tuple for each relation in $R$. In contrast, the results of this paper show that the analysis of finite satisfiability of keys and foreign keys for XML is not trivial.

**Related work.** The interaction between cardinality constraints and database schemas has been studied for object-oriented databases [12, 13]. This interaction is quite different from what we explore in this paper because XML DTDs are defined in terms of extended context free grammars and they lead to a class of cardinality constraints more complex than those studied for object-oriented databases.

Key and foreign key specifications were proposed in XML Schema [28] and XML Data [21]. However, their associated finite satisfiability problem has not been addressed. A key constraint language for XML was recently proposed in [6] and its associated finite satisfiability and finite implication problems were investigated in [7]. However, these papers do not consider foreign keys and the interaction between DTDs and constraints. A class of simple key and foreign key constraints was also studied for a graph model of XML [19]. Their associated implication and finite implication problems were investigated there. However, in contrast to the XML tree model, the graph model allows sharing of nodes and thus trivializes the analysis of finite satisfiability of key and foreign key constraints. It should be mentioned that in general, there is no one to one mapping between XML graphs and XML documents. In other words, an XML graph may have multiple XML documents corresponding to it.

Keys, foreign keys and more general inclusion and functional dependencies have been well studied for relational databases (see, e.g., [1, 29] for detailed surveys). In particular, [16] investigates a class of unary inclusion and functional dependencies, which is related to unary key and foreign key constraints considered in this paper, and their associated (finite) implication problem. As mentioned earlier, in the relational setting, the finite satisfiability problem for keys and foreign keys is trivial.

A variety of path constraints have been studied for semistructured and XML data [2, 8, 10]. The interaction between path constraints and database schemas was investigated in [9]. Path constraints specify inclusions among certain sets of objects in edge-labeled graphs. They are not capable of expressing key constraints.

**Organization.** The reminder of the paper is organized as follows. Section 2 presents a formalism of XML DTDs, describes the tree model for XML, and defines two classes of key and foreign key constraints, namely, $L_u$ and $L$. Section 3 provides a characterization of $L_u$ constraints and a characterization of DTDs in terms of cardinality constraints. Section 4 shows that the finite satisfiability problem for $L_u$ constraints is NP-complete by using the characterizations of DTDs and $L_u$ constraints. Section 5 establishes the undecidability of the finite satisfiability problem for $L$ constraints. Finally, Section 6 summarizes our results and identifies directions for further work. Proofs of the results in the paper are given in an appendix.

## 2 DTDs, keys and foreign keys

In this section, we first present a formalism of XML DTDs [5] and the tree model of XML. We then define two classes of key and foreign key constraints for XML and describe their associated finite satisfiability problems.

### 2.1 DTDs and XML trees

In the literature (e.g., [4, 11, 24]), DTDs are often formalized as *Extended Context Free Grammars* (*ECFGs*), with elements as non-terminals, basic XML types as terminals and element type definitions (in terms of regular expressions) as productions. This formalism fails to capture XML attributes. We extend this formalism of DTDs by incorporating simple attributes.

**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*, ranged over by $\tau$;
- $A$ is a finite set of *attributes*, ranged over by $l$; $E$ and $A$ are disjoint, i.e., $E \cap A = \emptyset$;
- $P$ is a mapping from $E$ to *element type definitions*: $P(\tau) = \alpha$, where $\alpha$ is a regular expression defined as follows:

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

  where $\mathtt{S}$ denotes *string* type, $\tau' \in E$, $\epsilon$ denotes the empty sequence, and "|", "," and "$*$" stand for union, concatenation, and the Kleene closure, respectively;
- $R$ is a mapping from $E$ to $\mathcal{P}(A)$, the power-set of $A$; if $l \in R(\tau)$ then we say $l$ is *defined for* $\tau$;
- $r \in E$ and is called *the element type of the root*.

Without loss of generality, we assume that $r$ does not occur in $P(\tau)$ for any $\tau \in E$. ■

To simplify the discussion, we assume that all attributes are single-valued. That is, if $l \in R(\tau)$ then every element of type $\tau$ has a unique $l$ attribute and in addition, the value of the $l$ attribute is a string. We do not consider other XML attributes, e.g., set-valued attributes, in this paper.

As an example, let us consider the `teachers` DTD $D_1$ given in Section 1. In our formalism, $D_1$ can be represented as $(E_1, A_1, P_1, R_1, r_1)$, where

$$\begin{aligned}
&E_1 = \{teachers,\, teacher,\, teach,\, research,\, subject\}\\
&A_1 = \{name,\, taught_by\}\\
&P_1(teachers) = teacher,\, teacher^*\\
&P_1(teacher) \ = teach,\, research\\
&P_1(teach) \quad\ = subject,\, subject\\
&P_1(subject) \ = P_1(research) = \mathtt{S}\\
&R_1(teacher) \ = \{name\}\\
&R_1(subject) \ = \{taught_by\}\\
&R_1(teachers) = R_1(teach) = R_1(research) = \emptyset\\
&r_1 = teachers
\end{aligned}$$

Similarly, we represent the DTD $D_3$ given in Section 1 as $(E_3, A_3, P_3, R_3, r_3)$, where

$$\begin{aligned}
&E_3 = \{db,\, foo\}\\
&A_3 = \emptyset\\
&P_3(db) \quad = foo\\
&P_3(foo) \ = foo\\
&R_3(db) \quad = R_2(foo) = \emptyset\\
&r_3 = db
\end{aligned}$$

An XML document is typically modeled as a node-labeled ordered tree. Given a DTD, we define the notion of its valid documents, i.e., documents that conform to it, as follows.

**Definition 2.2:** Let $D = (E, A, P, R, r)$ be a DTD. A *data tree $T$ valid w.r.t. $D$* is defined to be

$$T = (V,\, lab,\, ele,\, att,\, val,\, root),$$

where

- $V$ is a set of *vertices* (*nodes*);
- $lab$ is a mapping from $V$ to $E \cup A \cup \{\mathtt{S}\}$;
- $ele$ is a partial function from $V$ to sequences of $V$ vertices such that for any $v \in V$, $ele(v)$ is defined iff $lab(v) = \tau$ and $\tau \in E$, and moreover, if $P(\tau)$ is $\alpha$ and $ele(v) = [v_1, ..., v_n]$, then $lab(v_1)...lab(v_n)$ must be in the regular language defined by $\alpha$;
- $att$ 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 a string iff $lab(v) = \mathtt{S}$ or $lab(v) \in A$;
- $root$ is a distinguished vertex in $V$ and is called *the root of $T$*; in addition, $lab(root) = r$;
- $T$ has a tree structure. More specifically, for any $v \in V$, there exists a unique path from $root$ to $v$. A path from $root$ is a sequence of labels, defined as follows:

– $\epsilon$ is a path from *root* to itself;

– if $\rho$ is a path from *root* to $v$ with $lab(v) = \tau$ in $E$, then for any $v'$ in $ele(v)$ with $lab(v') = \tau'$ and $\tau' \in E \cup \{\mathtt{S}\}$, $\rho.\tau'$ is a path from *root* to $v'$; and moreover, for any $l \in R(\tau)$, $\rho.l$ is a path from *root* to $att(v, l)$.

We assume that there is a unique node in $T$ labeled $r$.

A data tree $T$ is said to be *finite* if $V$ is finite. ■

Intuitively, $V$ is the set of vertices of the tree $T$. The mapping *lab* labels every node of $V$ with a symbol from $E \cup A \cup \{\mathtt{S}\}$. In general, vertices labeled with element types of $E$ are internal nodes of $T$, and those labeled with S or attributes of $A$ are leaves of $T$. More specifically, if a node $x$ is labeled $\tau$ in $E$, then functions *ele* and *att* define the children of $x$, which are partitioned into *sub-elements* and *attributes* according to $P(\tau)$ and $R(\tau)$ in DTD $D$. Sub-elements of node $x$ are ordered and their labels observe the regular expression $P(\tau)$. In contrast, attributes of node $x$ are unordered and are identified by their labels (names). The function *val* assigns string values to attributes of nodes and also to vertices labeled S. The last condition in the definition above assures that $T$ indeed has a tree structure. As a result, sharing of nodes is not allowed in $T$.

For example, Figures 1 and 2 depict data trees valid w.r.t. $D_1$ and $D_2$ given in Section 1, respectively.

There is an one to one mapping between data trees and XML documents valid w.r.t. a DTD $D$ when the order of attributes is ignored. Given a data tree $T$ valid w.r.t. $D$, it is easy to derive an XML document valid w.r.t. $D$. The document is unique regardless of the order of attributes. Conversely, given an XML document valid w.r.t. $D$, it is straightforward to depict a data tree representing the document. Again, the tree is unique when the order of attributes is disregarded.

Given a DTD $D$ and a data tree $T$ valid w.r.t. $D$, we define the following notations. For any $\tau \in E \cup \{\mathtt{S}\}$, we use $ext(\tau)$ to denote the set of all the nodes in $T$ labeled $\tau$. For any node $x$ in $T$ labeled $\tau$ in $E$ and for any $l \in R(\tau)$, we use $x.l$ to indicate $val(att(x, l))$, i.e., the *value* of the attribute $l$ of $x$. We define $ext(\tau.l)$ to be $\{x.l \mid x \in ext(\tau)\}$, which is a set of strings. For any finite set $S$, we use $|S|$ to denote the *cardinality* of $S$.

### 2.2 A constraint language $L_u$

As observed earlier, key and foreign key constraints are important for XML. Next, we define a class of unary key and foreign key constraints for XML, denoted by $L_u$.

Let $D = (E, A, P, R, r)$ be a DTD. A *constraint* $\varphi$ *of* $L_u$ *over* $D$ has one of the following forms:

- *key constraint:* $\tau.l \to \tau$, where $\tau \in E$ and $l \in R(\tau)$. It indicates that $l$ is a key of elements of $\tau$.
- *foreign key constraint:* $\tau_1.l_1 \subseteq \tau_2.l_2$, where $\tau_1, \tau_2$ are element types in $E$, $l_1 \in R(\tau_1)$, $l_2 \in R(\tau_2)$, and moreover, $\tau_2.l_2 \to \tau_2$. This constraint indicates that $l_1$ is a foreign key of $\tau_1$ elements referencing $l_2$ attributes of $\tau_2$ elements.

For example, the constraints of $\Sigma_1$ and $\Sigma_2$ given in Section 1 are $L_u$ constraints over DTDs $D_1$ and $D_2$, respectively.

Let $T$ be a data tree valid w.r.t. $D$. Then $T$ *satisfies* a $L_u$ constraint $\varphi$ over $D$, denoted by $T \models \varphi$, iff

- if $\varphi$ is a key constraint $\tau.l \to \tau$, then in $T$,

  $$\forall\, x\, y \in ext(\tau)\ (x.l = y.l \to x = y),$$

  i.e., no two distinct vertices in $T$ labeled $\tau$ can have the same $l$ attribute value;
- if $\varphi$ is a foreign key constraint $\tau_1.l_1 \subseteq \tau_2.l_2$, then in $T$,

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

  i.e., the set of $l_1$ attribute values of all $\tau_1$ vertices in $T$ is a subset of the set of $l_2$ attribute values of all $\tau_2$ vertices in $T$. In addition, $T \models \tau_2.l_2 \to \tau_2$.

Observe that two notions of equality are used to define keys: string value equality is assumed in $x.l = y.l$, and $x = y$ is true if and only if $x$ and $y$ are the same node. This is different from the meaning of keys in relational databases. A foreign key constraint asserts two things: $ext(\tau_1.l_1) \subseteq ext(\tau_2.l_2)$ and $\tau_2.l_2 \to \tau_2$. Thus the value of $l_1$ attribute of a $\tau_1$ element (node) can uniquely identify a $\tau_2$ element (node).

Let $\Sigma$ be a set of $L_u$ constraints. We use $T \models \Sigma$ to indicate that $T$ *satisfies* $\Sigma$, i.e., for any $\phi \in \Sigma$, $T \models \phi$.

The *finite satisfiability problem for* $L_u$ is the problem to determine, given any DTD $D$ and any set $\Sigma$ of $L_u$ constraints over $D$, whether there exists a finite data tree $T$ such that $T \models \Sigma$ and $T$ is valid w.r.t. $D$.

Finally, it is easy to verify the following, which asserts that, among others, any set of $L_u$ constraints over a DTD is finite.

**Proposition 2.1:** *Let* $D = (E, A, P, R, r)$ *be a DTD. The cardinality of the set of all* $L_u$ *constraints over* $D$ *is at most* $O(m^2\, n^2)$*, where* $m = |E|$ *and* $n = |A|$. ■

### 2.3 A constraint language $L$

We next extend $L_u$ to define a class of multi-attribute keys and foreign keys for XML, denoted by $L$. Multi-attribute keys and foreign keys are important in, among other things, data conversion for capturing keys and foreign keys commonly found in relational databases.

Let $D = (E, A, P, R, r)$ be a DTD. A *constraint* $\varphi$ *of* $L$ *over* $D$ has one of the following forms:

- *key:* $\tau[X] \to \tau$, where $\tau \in E$ and $X$ is a set of attributes in $R(\tau)$. It indicates that the set $X$ of attributes is a key of elements of $\tau$.
- *foreign key:* $\tau_1[X] \subseteq \tau_2[Y]$, where $\tau_1, \tau_2$ are element types in $E$, $X, Y$ are non-empty sequences of attributes in $R(\tau_1)$, $R(\tau_2)$, respectively, and moreover, $X$ and $Y$ have the same length. In addition, $\tau_2[Y] \to \tau_2$. This constraint indicates that $X$ is a foreign key of $\tau_1$ elements referencing key $Y$ of $\tau_2$ elements.

As an example, let us consider another DTD $D_4 = (E_4, A_4, P_4, R_4, r_4)$, where

$$
\begin{aligned}
E_4 &= \{\mathit{school}, \mathit{student}, \mathit{course}, \mathit{enroll},\\
&\qquad \mathit{name}, \mathit{gpa}, \mathit{subject}, \mathit{grade}\}\\
A_4 &= \{\mathit{student\_id}, \mathit{course\_no}, \mathit{dept}\}\\
P_4(\mathit{school}) &= \mathit{course}^*, \mathit{student}^*, \mathit{enroll}^*\\
P_4(\mathit{course}) &= \mathit{subject}\\
P_4(\mathit{student}) &= \mathit{name}, \mathit{gpa}\\
P_4(\mathit{enroll}) &= \mathit{grade}\\
P_4(\mathit{name}) &= P_4(\mathit{gpa}) = P_4(\mathit{subject})\\
&= P_4(\mathit{grade}) = \mathsf{S}\\
R_4(\mathit{course}) &= \{\mathit{dept}, \mathit{course\_no}\}\\
R_4(\mathit{student}) &= \{\mathit{student\_id}\}\\
R_4(\mathit{enroll}) &= \{\mathit{student\_id}, \mathit{dept}, \mathit{course\_no}\}\\
R_4(\mathit{school}) &= R_4(\mathit{name}) = R_4(\mathit{gpa})\\
&= R_4(\mathit{subject}) = R_4(\mathit{grade}) = \emptyset\\
r_4 &= \mathit{school}
\end{aligned}
$$

Typical $L$ constraints over $D_4$ include:

$$
\begin{aligned}
&\mathit{student}[\mathit{student\_id}] \to \mathit{student},\\
&\mathit{course}[\mathit{dept}, \mathit{course\_no}] \to \mathit{course},\\
&\mathit{enroll}[\mathit{student\_id}, \mathit{dept}, \mathit{course\_no}] \to \mathit{enroll},\\
&\mathit{enroll}[\mathit{student\_id}] \subseteq \mathit{student}[\mathit{student\_id}],\\
&\mathit{enroll}[\mathit{dept}, \mathit{course\_no}] \subseteq \mathit{course}[\mathit{dept}, \mathit{course\_no}].
\end{aligned}
$$

The first three constraints are keys in $L$, while the last two are foreign keys in $L$.

Observe that $L_u$ is a sub-language of $L$, in which the sequences of attributes in a constraint are of length 1.

Let $T$ be a data tree valid w.r.t. $D$. For each $\tau$ element $x$ in $T$ and a sequence $X = [l_1, \ldots, l_n]$ of attributes in $R(\tau)$, we use $x[X]$ to denote the sequence of $X$ attribute values of $x$, i.e., $x[X] = [x.l_1, \ldots, x.l_n]$.

A tree $T$ *satisfies* a $L$ constraint $\varphi$ over a DTD $D$, denoted by $T \models \varphi$, iff

- if $\varphi$ is a key $\tau[X] \to \tau$, then in $T$,
$$\forall\, x\, y \in \mathit{ext}(\tau)\ (\bigwedge_{l \in X} (x.l = y.l) \to x = y).$$
That is, two distinct $\tau$ nodes in $T$ cannot have the same $X$ attribute values;
- if $\varphi$ is a foreign key $\tau_1[X] \subseteq \tau_2[Y]$, then in $T$,
$$\forall\, x \in \mathit{ext}(\tau_1)\ \exists\, y \in \mathit{ext}(\tau_2)\ (x[X] = y[Y]).$$
That is, the sequence of $X$ attribute values of every $\tau_1$ node in $T$ must match the sequence of $Y$ attribute values of some $\tau_2$ node in $T$. In addition, $T \models \tau_2[Y] \to \tau_2$, i.e., $Y$ is a key of $\tau_2$.

The *finite satisfiability problem for* $L$ is defined in the same way as the finite satisfiability problem for $L_u$.

The central technical problems investigated in the paper are the finite satisfiability problems for $L_u$ and $L$. We investigate these problems in Sections 4 and 5, respectively.

## 3 Cardinality constraints, DTDs and $L_u$ constraints

In this section, we develop a class of cardinality constraints and provide characterizations of $L_u$ constraints and DTDs in terms of these cardinality constraints. More specifically, given a DTD $D$ and a set $\Sigma$ of $L_u$ constraints over $D$, we define a finite set $C_\Sigma$ of cardinality constraints such that there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$ if and only if there exists a finite data tree valid w.r.t. $D$ and satisfying $C_\Sigma$. Similarly, given a DTD $D$, we define a finite set $\Psi_D$ of cardinality constraints such that there exists a finite data tree valid w.r.t. $D$ if and only if $\Psi_D$ is finitely satisfiable. These characterizations are not only interesting in their own right, they are also important in the analysis of finite satisfiability and finite implication of $L_u$ constraints.

### 3.1 A Characterization of $L_u$ constraints

Consider a DTD $D = (E, A, P, R, r)$ and a set $\Sigma$ of $L_u$ constraints over $D$. For any finite data tree $T$ valid w.r.t. $D$, we derive from $\Sigma$ a class of cardinality constraints on $T$, denoted by $C_\Sigma$, as follows. For any $\varphi \in \Sigma$,

- if $\varphi$ is a key constraint $\tau.l \to \tau$, then

  $$|ext(\tau.l)| = |ext(\tau)|$$

  is in $C_\Sigma$;

- if $\varphi$ is a foreign key constraint $\tau_1.l_1 \subseteq \tau_2.l_2$, then

  $$\begin{array}{rcl} |ext(\tau_1.l_1)| & \leq & |ext(\tau_2.l_2)|, \\ |ext(\tau_2.l_2)| & = & |ext(\tau_2)| \end{array}$$

  are in $C_\Sigma$.

Moreover, for any $\tau \in E$ and $l \in R(\tau)$ that occur in $\Sigma$, $|ext(\tau.l)| \leq |ext(\tau)|$ and $0 \leq |ext(\tau.l)|$ are in $C_\Sigma$. We refer to $C_\Sigma$ as *the set of cardinality constraints determined by* $\Sigma$. We use $T \models C_\Sigma$ to denote that $T$ satisfies all constraints of $C_\Sigma$.

For example, recall the set $\Sigma_1$ of $L_u$ constraints over DTD $D_1$ given in Section 1. The set of cardinality constraints determined by $\Sigma_1$, denoted by $C_{\Sigma_1}$, consists of:

$$\begin{array}{rcl} |ext(teacher.name)| & = & |ext(teacher)| \\ |ext(subject.taught\_by)| & = & |ext(subject)| \\ |ext(subject.taught\_by)| & \leq & |ext(teacher.name)| \\ 0 & \leq & |ext(teacher.name)| \\ 0 & \leq & |ext(subject.taught\_by)| \end{array}$$

The theorem below shows that $C_\Sigma$ characterizes finite satisfiability of $\Sigma$. It allows us to consider $C_\Sigma$ instead of $\Sigma$ when studying finite satisfiability of $\Sigma$.

**Theorem 3.1:** *Let $D$ be a DTD, $\Sigma$ be a set of $L_u$ constraints over $D$, and $C_\Sigma$ be the set of cardinality constraints determined by $\Sigma$. Then the following two statements are equivalent.*

1. *There exists a finite data tree $T_1$ such that $T_1$ is valid w.r.t. $D$ and $T_1 \models \Sigma$.*
2. *There exists a finite data tree $T_2$ such that $T_2$ is valid w.r.t. $D$ and $T_2 \models C_\Sigma$.*

*In addition, any finite data tree valid w.r.t. $D$ and satisfying $\Sigma$ also satisfies $C_\Sigma$.* ∎

Indeed, given a finite data tree $T_1$ valid w.r.t. $D$ and satisfying $\Sigma$, one can verify $T_1 \models C_\Sigma$. Conversely, given a finite tree $T_2$ valid w.r.t. $D$ and satisfying $C_\Sigma$, one can construct a finite tree $T_1$ by modifying function *val* in $T_2$ such that $T_1$ is valid w.r.t. $D$ and $T_1 \models \Sigma$.

It is straightforward to verify the following.

**Proposition 3.2:** *Given any set $\Sigma$ of $L_u$ constraints over a DTD $D$, the set $C_\Sigma$ of cardinality constraints determined by $\Sigma$ can be computed in linear time.* ∎

As an immediate result of Proposition 3.2, the size of $C_\Sigma$ is linear in the size of $\Sigma$.

### 3.2 Eliminating Kleene closures

Element type definitions in a DTD are defined in terms of regular expressions. To simplify the discussion when it comes to the study of cardinality constraints enforced by DTDs, we want to rewrite element type definitions to eliminate occurrences of the Kleene star. Moreover, the rewriting should not affect finite satisfiability of $L_u$ constraints. More specifically, given a DTD $D$, we define another DTD $D^-$ such that $D^-$ does not contain the Kleene star and in addition, for any set $\Sigma$ of $L_u$ constraints over $D$, there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$ if and only if there exists a finite data tree valid w.r.t. $D^-$ and satisfying $\Sigma$.

We next present a rewriting that preserves finite satisfiability of $L_u$ constraints.

Let $D = (E, A, P, R, r)$ be a DTD. For each element type $\tau \in E$, the definition of $\tau$, $P(\tau)$, is a regular expression $\alpha$. We define another function $P^-$ for element type definitions: $P^-(\tau) = f(\alpha)$, where $f(\alpha)$ is given as follows:

$$f(\alpha) = \begin{cases} \alpha & \text{if } \alpha \text{ is } \mathtt{S},\ \tau' \text{ or } \epsilon \\ f(\alpha_1), f(\alpha_2) & \text{if } \alpha = \alpha_1, \alpha_2 \\ f(\alpha_1) \mid f(\alpha_2) & \text{if } \alpha = \alpha_1 \mid \alpha_2 \\ \tau_\alpha & \text{if } \alpha = \alpha_1^* \end{cases}$$

Here $\tau_\alpha$ is a new element type and

$$P^-(\tau_\alpha) = \epsilon \mid \alpha_1, \tau_\alpha.$$

That is, we eliminate occurrences of the Kleene star by introducing new element types and by rewriting Kleene closures in terms of direct recursion. We refer to the function $f$ as *the transition function*.

Formally, we define *the $*$-free DTD of $D$*, denoted by $D^-$, to be $D^- = (E^-, A, P^-, R^-, r)$, where

- $E^- = E \cup E_a$, and $E_a$ is the set of new element types $\tau_\alpha$'s introduced in the process of eliminating occurrences of the Kleene star, as described above;
- $P^-$ is defined in terms of $P$ and the transition function $f$ as described above;
- $R^-(\tau) = R(\tau)$ for all $\tau \in E$ and $R^-(\tau) = \emptyset$ for all $\tau \in E_a$.

Observe that the element type $r$ of the root and the set $A$ of attributes remain unchanged in $D^-$.

For example, the $*$-free DTD of $D_1$ given in Section 2 is $D_1^- = (E_1^-, A_1, P_1^-, R_1^-, r)$, where

$$\begin{array}{l} E_1^- = \{teachers, teacher, teach, research, subject, \tau_t\} \\ A_1 = \{name,\ taught\_by\} \end{array}$$

$$
\begin{aligned}
P_1^-(teachers) &= teacher,\ \tau_t \\
P_1^-(\tau_t) &= \epsilon \mid teacher,\ \tau_t \\
P_1^-(teacher) &= teach,\ research \\
P_1^-(teach) &= subject,\ subject \\
P_1^-(subject) &= P_1^-(research) = \mathtt{S} \\
R_1^-(teacher) &= \{name\} \\
R_1^-(subject) &= \{taught\_by\} \\
R_1^-(teachers) &= R_1^-(teach) = R_1^-(research) \\
&= R_1^-(\tau_t) = \emptyset \\
r_1 &= teachers
\end{aligned}
$$

Here $\tau_t$ is the only new element type introduced.

Observe that the $*$-free DTD $D_3^-$ of $D_3$ in Section 1 is $D_3$ itself, because $D_3$ does not contain Kleene star.

Obviously, any set $\Sigma$ of $L_u$ constraints over $D$ is also a set of $L_u$ constraints over the $*$-free DTD $D^-$ of $D$. The next theorem shows that $D^-$ preserves finite satisfiability of $\Sigma$. It allows us to consider $*$-free DTDs only, without loss of generality.

**Theorem 3.3:** *Let $D$ be a DTD, $D^-$ the $*$-free DTD of $D$, and $\Sigma$ a set of $L_u$ constraints over $D$. Then the following two statements are equivalent.*

1. *There exists a finite data tree $T_1$ such that $T_1$ is valid w.r.t. $D$ and $T_1 \models \Sigma$.*
2. *There exists a finite data tree $T_2$ such that $T_2$ is valid w.r.t. $D^-$ and $T_2 \models \Sigma$.*

■

Theorem 3.3 can be verified by induction on the structures of element type definitions in $D$. Suppose that there is a finite data tree $T_1$ valid w.r.t. $D$ and satisfying $\Sigma$. Then $T_1 \models C_\Sigma$ by Theorem 3.1, where $C_\Sigma$ is the set of cardinality constraints determined by $\Sigma$. We can construct a finite tree $T_2'$ by modifying $T_1$, following the definition of the transition function $f$, such that $T_2'$ is valid w.r.t. $D^-$ and $T_2' \models C_\Sigma$. Again by Theorem 3.1, there is a finite data tree $T_2$ such that $T_2$ is valid w.r.t. $D^-$ and $T_2 \models \Sigma$. Conversely, given a finite data tree $T_2$ valid w.r.t. $D^-$ and satisfying $\Sigma$, we have $T_2 \models C_\Sigma$ by Theorem 3.1. In addition, by modifying $T_2$ one can construct a finite tree $T_1'$, directed by the definition of $f$, such that $T_1'$ is valid w.r.t. $D$ and $T_1' \models C_\Sigma$. Thus by Theorem 3.1, there is a finite data tree $T_1$ valid w.r.t. $D$ and satisfying $\Sigma$. See Appendix for a detailed proof.

Finally, it is easy to verify the following.

**Proposition 3.4:** *Given any DTD $D$, its $*$-free DTD $D^-$ can be computed in linear time.* ■

As an immediate result of Proposition 3.4, the size of $D^-$ is linear in the size of $D$.

### 3.3 A characterization of DTDs

Next, we show that cardinality constraints can also be used to characterize DTDs. Without loss of generality by Theorems 3.3, we need to consider $*$-free DTDs only.

Consider a DTD $D = (E, A, P, R, r)$. To define cardinality constraints to characterize $D$, we first describe unknowns to be used in these constraints. For each symbol $\tau \in E \cup \{\mathtt{S}\}$, we assume a set of unknowns $X_\tau = \{x_\tau^i \mid i \in \mathbb{N}\}$, where $\mathbb{N}$ is the set of natural numbers. For each $\tau \in E$, consider the regular expression $\alpha = P(\tau)$. We define *the set of regular expressions in* $P(\tau)$, denoted by $S_\tau$, to be the set consisting of (1) $\alpha$, (2) regular expressions in $\alpha_1$ or $\alpha_2$, if $\alpha = (\alpha_1, \alpha_2)$ or $\alpha = (\alpha_1 \mid \alpha_2)$. For the regular expressions in $S_\tau$, we also assume a set of unknowns $Y_\tau = \{y^\tau_{\beta,k} \mid \beta \in S_\tau,\ k \in \mathbb{N}\}$.

Using these unknowns, we define a function $g$ from $S_\tau$ and $Y_\tau$ to sets of cardinality constraints as follows:

$$
g(\beta,\ y^\tau_{\beta,k}) \;=\; \begin{cases}
\{y^\tau_{\beta,k} = x^i_{\tau'}\} & \text{if } \beta \text{ is } \tau' \\
\{y^\tau_{\beta,k} = x^i_S\} & \text{if } \beta \text{ is } \mathtt{S} \\
\{y^\tau_{\beta,k} = y^\tau_{\epsilon,j}\} & \text{if } \beta \text{ is } \epsilon \\
\{y^\tau_{\beta,k} = y^\tau_{\beta_1,k_1},\ y^\tau_{\beta,k} = y^\tau_{\beta_2,k_2}\} & \\
\quad \cup\ g(\beta_1,\ y^\tau_{\beta_1,k_1}) & \\
\quad \cup\ g(\beta_2,\ y^\tau_{\beta_2,k_2}) & \text{if } \beta = \beta_1, \beta_2 \\
\{y^\tau_{\beta,k} = y^\tau_{\beta_1,k_1} + y^\tau_{\beta_2,k_2}\} & \\
\quad \cup\ g(\beta_1,\ y^\tau_{\beta_1,k_1}) & \\
\quad \cup\ g(\beta_2,\ y^\tau_{\beta_2,k_2}) & \text{if } \beta = \beta_1 \mid \beta_2
\end{cases}
$$

Here $x^i_{\tau'} \in X_{\tau'}$, $x^i_S \in X_{\mathtt{S}}$, and $y^\tau_{\epsilon,j}$, $y^\tau_{\beta_1,k_1}$, $y^\tau_{\beta_2,k_2} \in Y_\tau$ are fresh variables.

*The set of cardinality constraints determined by* $\tau$, denoted by $\psi_\tau$, consists of the following:

- $|ext(\tau)| = y^\tau_{\alpha,1}$, where $\alpha = P(\tau)$;
- constraints in $g(\alpha,\ y^\tau_{\alpha,1})$;
- $x^i_{\tau'} \geq 0$ for all $x^i_{\tau'}$ occurring in $g(\alpha,\ y^\tau_{\alpha,1})$;
- $y^\tau_{\beta,k} \geq 0$ for all $y^\tau_{\beta,k}$ occurring in $g(\alpha,\ y^\tau_{\alpha,1})$.

Intuitively, constraints of $\psi_\tau$ describe the grammar $P(\tau)$ in a quantitative way. Referring to a finite data tree $T$ valid w.r.t. $D$, let us consider the sub-elements $\vec{v} = [v_1, ..., v_n]$ of a node in $T$ labeled $\tau$. The labels of the sub-elements, $lab(\vec{v}) = lab(v_1) ... lab(v_n)$, must observe $P(\tau)$. Moreover, a subsequence of $lab(\vec{v})$ may correspond to a certain occurrence of a regular expression $\beta$ in $S_\tau$. For example, if $P(\tau) = (\beta_1,\ \beta_2)$, then there is $m \in [1, n]$ such that $lab(v_1) ... lab(v_m)$ and $lab(v_{m+1}) ... lab(v_n)$ are in the regular languages defined by $\beta_1$ and $\beta_2$, respectively. We say that $[v_1, ..., v_m]$ and $[v_{m+1}, ..., v_n]$ correspond to $\beta_1$ and $\beta_2$, respectively. In particular, if $\beta = \tau'$ for some $\tau' \in E \cup \{\mathtt{S}\}$, an occurrence of $\beta$ may correspond to a node in $T$ labeled $\tau'$. The unknown $x^i_{\tau'}$ intends to indicate the number of all $\tau'$ nodes in $T$ that correspond to a certain occurrence of $\tau'$ in $P(\tau)$. Similarly, $y^{\tau}_{\beta,k}$ keeps track of the number of sub-element sequences that correspond to the $k$th occurrence of $\beta$ in $P(\tau)$. Constraints of $\psi_\tau$ specify dependencies among these numbers. To illustrate this, let us consider the structure of $\alpha = P(\tau)$. Suppose that in $T$, there are $y^{\tau}_{\alpha,1}$ many nodes labeled $\tau$. Suppose $\alpha$ is some $\tau' \in E \cup \{\mathtt{S}\}$, then each $\tau$ node has a distinct sub-element labeled $\tau'$. Thus there must be $y^{\tau}_{\alpha,1}$ many nodes labeled $\tau'$ that are sub-elements of $\tau$ nodes. This is exactly what the constraint $y^{\tau}_{\alpha,1} = x^i_{\tau'}$ says. Next, suppose $\alpha = (\beta_1, \beta_2)$. The sub-elements $\vec{v}$ of a $\tau$ node can be partitioned into $\vec{v} = \vec{v_1}\,\vec{v_2}$ such that $\vec{v_1}$ and $\vec{v_2}$ correspond to $\beta_1$ and $\beta_2$, respectively. Assume that the numbers of sub-element sequences of $\tau$ nodes corresponding to $\beta_1$ and $\beta_2$ are $y^{\tau}_{\beta_1,k_1}$ and $y^{\tau}_{\beta_2,k_2}$, respectively. Then both $y^{\tau}_{\beta_1,k_1}$ and $y^{\tau}_{\beta_2,k_2}$ must equal to $y^{\tau}_{\alpha,1}$. The constraints $y^{\tau}_{\alpha,1} = y^{\tau}_{\beta_1,k_1}$ and $y^{\tau}_{\alpha,1} = y^{\tau}_{\beta_2,k_2}$ describe this quantitative relation. Finally, suppose $\alpha = (\beta_1 \,|\, \beta_2)$. The sub-elements of a $\tau$ node correspond to either $\beta_1$ or $\beta_2$. Assume that the numbers of sub-elements of $\tau$ nodes corresponding to $\beta_1$ and $\beta_2$ are $y^{\tau}_{\beta_1,k_1}$ and $y^{\tau}_{\beta_2,k_2}$, respectively. Then the sum of $y^{\tau}_{\beta_1,k_1}$ and $y^{\tau}_{\beta_2,k_2}$ must equal to $y^{\tau}_{\alpha,1}$. This is what the constraint $y^{\tau}_{\alpha,1} = y^{\tau}_{\beta_1,k_1} + y^{\tau}_{\beta_2,k_2}$ says.

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

- $|ext(r)| = 1$;
- constraints of $\psi_\tau$ for each $\tau \in E$;
- $|ext(\tau)| = \sum_{i \in [1, m^\tau]} x^i_\tau$ for each $\tau \in E \cup \{\mathtt{S}\}$, where without loss of generality, we assume $x^1_\tau$, ..., $x^{m^\tau}_\tau$ are all unknowns of $X_\tau$ occurring in some $\psi_{\tau'}$.

Intuitively, referring to a finite data tree $T$, the constraint $|ext(r)| = 1$ asserts that $T$ has a unique root. For each $\tau \in E$, $\psi_\tau$ assures that the grammar $P(\tau)$ is observed. Consider $\tau \in E \cup \{\mathtt{S}\}$. Observe that each node of $T$ labeled $\tau$ corresponds to a certain occurrence of $\tau$ in $P(\tau')$ for some element type $\tau'$. Obviously, the set $ext(\tau)$ should include all $\tau$ vertices in $T$ no matter what occurrences of $\tau$ they correspond to. This is exactly what the constraint $|ext(\tau)| = \sum_{1 \leq i \leq m^\tau} x^i_\tau$ says, where $x^i_\tau$ indicates the number of all $\tau$ nodes in $T$ that correspond to a certain occurrence of $\tau$.

By treating $|ext(\tau)|$ in $\Psi_D$ as an unknown for each $\tau \in E \cup \{\mathtt{S}\}$, we can view $\Psi_D$ as a system of linear equations and disequations. We say that $\Psi_D$ is *finitely satisfiable* if and only if $\Psi_D$ admits an integer solution. That is, there is an integer assignment to each unknown $|ext(r)|$, $x^i_{\tau'}$ and $y^{\tau}_{\beta,k}$ that satisfies all the equations and disequations in $\Psi_D$.

For example, let us consider DTDs $D_1^-$ and $D_3^-$ given in Section 3. The cardinality constraints determined by these DTDs are given below:

$\Psi_{D_1}$:

$$
\begin{array}{lll}
\psi_{teachers}: & |ext(teachers)| & = x^1_{teacher} \\
 & |ext(teachers)| & = x^1_{\tau_t} \\
\psi_{teacher}: & |ext(teacher)| & = x^1_{teach} \\
 & |ext(teacher)| & = x^1_{research} \\
\psi_{\tau_t}: & |ext(\tau_t)| & = y^{\tau_t}_{\epsilon} + y^{\tau_t}_{\beta} \\
 & y^{\tau_t}_{\beta} = x^2_{teacher} & \\
 & y^{\tau_t}_{\beta} = x^2_{\tau_t} & \\
\psi_{teach}: & |ext(teach)| & = x^1_{subject} \\
 & |ext(teach)| & = x^2_{subject} \\
\psi_{subject}: & |ext(subject)| & = x^1_{\mathtt{S}} \\
\psi_{research}: & |ext(research)| & = x^2_{\mathtt{S}} \\
\text{moreover,} & |ext(teachers)| & = 1 \\
 & |ext(teacher)| & = x^1_{teacher} + x^2_{teacher} \\
 & |ext(\tau_t)| & = x^1_{\tau_t} + x^2_{\tau_t} \\
 & |ext(teach)| & = x^1_{teach} \\
 & |ext(subject)| & = x^1_{subject} + x^2_{subject} \\
 & |ext(research)| & = x^1_{research} \\
 & |ext(S)| & = x^1_{\mathtt{S}} + x^2_{\mathtt{S}} \\
 & \text{all unknowns} \geq 0. &
\end{array}
$$

Observe, for example, $x^1_{teacher}$ indicates the number of `teacher` nodes that are children of `teachers` nodes, and $x^2_{teacher}$ stands for the number of `teacher` children of nodes labeled $\tau_t$. The cardinality of $ext(teacher)$ equals to the sum of $x^1_{teacher}$ and $x^2_{teacher}$. Obviously, there is a unique node labeled `teachers`, i.e., the root. Hence $x^1_{teacher} = 1$ since the root has a unique `teacher` child. Thus $|ext(teacher)| = 1 + x^2_{teacher}$.

$\Psi_{D_3}$:

$$
\begin{array}{lll}
\psi_{db}: & |ext(db)| & = x^1_{foo} \\
\psi_{foo}: & |ext(foo)| & = x^2_{foo} \\
\text{moreover,} & |ext(db)| & = 1 \\
 & |ext(foo)| & = x^1_{foo} + x^2_{foo} \\
 & \text{all unknowns} \geq 0. &
\end{array}
$$

It is easy to check that $\Psi_{D_1}$ is finitely satisfiable, whereas $\Psi_{D_3}$ is not. Intuitively, the unknowns in the constraints provide a finer labeling that keeps track of how vertices correspond to certain occurrences of regular expressions in a DTD.

We next show that $\Psi_D$ indeed characterizes $D$.

**Theorem 3.5:** *Let $D$ be a (∗-fee) DTD and $\Psi_D$ be the set of cardinality constraints determined by $D$. Then $\Psi_D$ is finitely satisfiable if and only if there is a finite data tree valid w.r.t. $D$.* ∎

Furthermore, given an integer solution of $\Psi_D$, one can construct a finite tree $T$ valid w.r.t. $D$ such that there is certain quantitative connection between $ext(\tau)$ in $T$ and the corresponding unknown $|ext(\tau)|$ in $\Psi_D$.

**Corollary 3.6:** *Let $D$ be a (∗-fee) DTD and $\Psi_D$ be the set of cardinality constraints determined by $D$. If $\Psi_D$ admits an integer solution such that $|ext(\tau)|$ is assigned a value $n_\tau$ for each $\tau \in E$, then there is a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and in addition, $|ext(\tau)|$ in $T$ is equal to $n_\tau$.* ∎

To prove Theorem 3.5 and Corollary 3.6, we use unknowns and the function $g$ to label vertices in a data tree. Given a finite data tree $T$ valid w.r.t. $D$, one can mark vertices (and vertex sequences) of $T$ with unknowns following the definition of $g$. Given the labeling, one can derive an integer assignment to unknowns in $\Psi_D$: each unknown is assigned the number of nodes (or node sequences) in $T$ marked with the unknown. In can be verified that the assignment is indeed a solution of $\Psi_D$. Conversely, given an integer solution of $\Psi_D$, one can construct a data tree $T$ such that for each $\tau \in E \cup \{\mathtt{S}\}$, the cardinality of $ext(\tau)$ in $T$ is the value of the unknown $|ext(\tau)|$ in the solution. Moreover, sequences of vertices in $T$ can be marked with unknowns, directed by the definition of $g$. This labeling yields a definition of the function $ele$ in $T$, which defines edges in $T$. It can be verified that $T$ is indeed finite and valid w.r.t. $D$, since $\Psi_D$ describes element type definitions in a quantitative way. See Appendix for a detailed proof.

The proof of the next proposition is straightforward.

**Proposition 3.7:** *Given any (∗-free) DTD $D$, the set $\Psi_D$ of cardinality constraints determined by $D$ can be computed in linear time.* ∎

Immediately from Proposition 3.7 it follows that the size of $\Psi_D$ is linear in the size of $D$.

## 4 Finite satisfiability of $L_u$ constraints

In this section, we show that the finite satisfiability problem for $L_u$ is NP-complete. That is, it is NP-complete to determine, given any DTD $D$ and any set $\Sigma$ of $L_u$ constraints over $D$, whether there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$. In addition, we show that several special cases of the finite satisfiability problem can be decided in PTIME.

### 4.1 Characterizing DTDs and $L_u$ constraints

To establish our complexity results, we first define a system of linear equations and disequations to characterize DTD and $L_u$ constraints. More specifically, given a DTD $D$ and a finite set $\Sigma$ of $L_u$ constraints over $D$, we define a finite set $\Psi(D, \Sigma)$ of cardinality constraints. The set $\Psi(D, \Sigma)$ can be treated as a system of linear equations and disequations. We show that there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$ if and only if $\Psi(D, \Sigma)$ admits a (legal) integer solution.

We proceed in two steps: we first consider ∗-free DTDs and then show that the results established for ∗-free DTDs can be extended to DTDs that are not necessarily ∗-free.

Let $D = (E, A, P, R, r)$ be a ∗-free DTD, and $\Sigma$ be a finite set of $L_u$ constraints over $D$. We define *the set of cardinality constraints determined by $D$ and $\Sigma$*, denoted by $\Psi(D, \Sigma)$, to be

$$\Psi(D, \Sigma) \;=\; \Psi_D \cup C_\Sigma,$$

where $C_\Sigma$ is the set of cardinality constraints determined by $\Sigma$, and $\Psi_D$ is the set of cardinality constraints determined by $D$, as defined in Sections 3.1 and 3.3, respectively.

For each $\tau \in E \cup \{\mathtt{S}\}$, we treat $|ext(\tau)|$ in $\Psi(D, \Sigma)$ as an unknown. Similarly, for each $\tau \in E$ and $l \in R(\tau)$, we treat $|ext(\tau.l)|$ also as an unknown. Thus $\Psi(D, \Sigma)$ can be viewed as a system of linear equations and disequations. We say that $\Psi(D, \Sigma)$ is *finitely satisfiable* if and only if $\Psi(D, \Sigma)$ admits an *legal* integer solution. By a legal solution we mean a solution that satisfies the following condition: if $|ext(\tau)| > 0$ then $|ext(\tau.l)| > 0$ for each $|ext(\tau)|$ and $|ext(\tau.l)|$ appearing in $\Psi(D, \Sigma)$. This is to ensure that every $\tau$ element has an $l$ attribute. Observe that $|ext(\tau.l)| \leq |ext(\tau)|$ is in $C_\Sigma$.

For example, recall DTDs $D_1^-$ and $D_3^-$ given in Section 3 and constraint sets $\Sigma_1$ and $\Sigma_3$ in Section 1. It is easy to verify that neither $\Psi(D_1^-, \Sigma_1)$ nor $\Psi(D_3^-, \Sigma_3)$ is finitely satisfiable. This is consistent with the observations made in Section 1.

We next show that $\Psi(D, \Sigma)$ characterizes $D$ and $\Sigma$.

**Theorem 4.1:** *Let $D$ be a (∗-free) DTD, $\Sigma$ a finite set of $L_u$ constraints over $D$, and $\Psi(D, \Sigma)$ the set of cardinality constraints determined by $D$ and $\Sigma$. Then $\Psi(D, \Sigma)$ is finitely satisfiable if and only if there is a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and $T \models \Sigma$.* ∎

To prove this theorem, we use Theorems 3.1 and 3.5. Given a finite data tree $T$ valid w.r.t. $D$ and satisfying $\Sigma$, we have $T \models C_\Sigma$ by Theorem 3.1, where

$C_\Sigma$ is the set of cardinality constraints determined by $\Sigma$. By Theorem 3.5, from $T$ we can derive an integer assignment to the unknowns in $\Psi_D$ such that it satisfies $\Psi_D$. We extend the assignment as follows: for each $\tau \in E$ and $l \in R(\tau)$, let the unknown $|ext(\tau.l)|$ in $\Psi(D, \Sigma)$ have as its value the cardinality of $ext(\tau.l)$ in $T$. Clearly, $|ext(\tau.l)| > 0$ if $|ext(\tau)| > 0$ since every $\tau$ element in $T$ has an $l$ attribute. By $T \models C_\Sigma$, it can be verified that this extended assignment is a legal solution of $\Psi(D, \Sigma)$. Conversely, given a legal integer solution of $\Psi(D, \Sigma)$, there exists a finite data tree $T_1$ valid w.r.t. $D$ by Theorem 3.5 and Corollary 3.6. In addition, for each $\tau \in E$, the cardinality of $ext(\tau)$ in $T_1$ equals to the value of the unknown $|ext(\tau)|$ in the solution. Observe that the solution also satisfies equations and disequations in $C_\Sigma$. That is, for all $\tau \in E$ and $l \in R(\tau)$ that occur in $\Sigma$, the values of the unknowns $|ext(\tau)|$ and $|ext(\tau.l)|$ in $\Psi(D, \Sigma)$ satisfy $C_\Sigma$. Thus we can construct a finite data tree $T_2$ by modifying the *val* function in $T_1$, directed by the solution, such that $T_2$ is valid w.r.t. $D$ and $T_2 \models C_\Sigma$. Observe that since the solution is legal we have $|ext(\tau.l)| > 0$ if $|ext(\tau)| > 0$. This makes it possible for every $\tau$ element in $T_2$ to have an $l$ attribute. Hence by Theorem 3.1, there exists a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and $T \models \Sigma$. A detailed proof can be found in Appendix.

Next, consider DTDs that are not necessarily $*$-free. The corollary below shows that for any DTD $D$ and finite set $\Sigma$ of $L_u$ constraints over $D$, $\Psi(D^-, \Sigma)$ also characterizes whether there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$, where $D^-$ is the $*$-free DTD of $D$ as defined in Section 3.

**Corollary 4.2:** *Let $D$ be a DTD, $D^-$ the $*$-free DTD of $D$, $\Sigma$ a finite set of $L_u$ constraints over $D$, and $\Psi(D^-, \Sigma)$ the set of cardinality constraints determined by $D^-$ and $\Sigma$. Then $\Psi(D^-, \Sigma)$ is finitely satisfiable if and only if there is a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and $T \models \Sigma$.* ■

The corollary follows from Theorems 3.3 and 4.1. To see this, first assume that there is a finite data tree $T$ valid w.r.t. DTD $D$ and satisfying $\Sigma$. By Theorem 3.3, there is a finite data tree $T'$ such that $T'$ is valid w.r.t. $D^-$ and $T' \models \Sigma$. Thus by Theorem 4.1, $\Psi(D^-, \Sigma)$ is finitely satisfiable. Conversely, suppose that $\Psi(D^-, \Sigma)$ admits a legal integer solution. Then by Theorem 4.1, there is a finite data tree $T'$ such that $T'$ is valid w.r.t. $D^-$ and $T' \models \Sigma$. Thus by Theorem 3.3, there is a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and $T \models \Sigma$.

## 4.2 Complexity analysis

Next, we establish one of the main complexity results of the paper:

**Theorem 4.3:** *The finite satisfiability problem for $L_u$ constraints is NP-complete.* ■

Before we prove the theorem, we first review the *linear integer programming* (*LIP*) problem: Given an $m \times n$ matrix $A$ of integers and a column vector $\vec{b}$ of $m$ integers, does there exist a column vector $\vec{x}$ of $n$ integers such that $A\,\vec{x} \geq \vec{b}$? That is, for $i \in [1, m]$,

$$\sum_{j \in [1,n]} a_{ij}\, x_j \geq b_i,$$

where $a_{ij}$ is the $j$th element of the $i$th row of $A$, $x_j$ is the $j$th entry of $\vec{x}$ and $b_i$ is the $i$th entry of $\vec{b}$. It is known that LIP is NP-complete in the strong sense [20]. In particular, when non-negative integer solutions are considered, [25] has shown that if the problem has a solution, then it has another solution in which for all $j \in [1, n]$, $x_j$ is no larger than $n\,(m\,a)^{2m+1}$, where $a$ is the largest absolute value of elements in $A$ and $\vec{b}$.

To show the finite satisfiability problem for $L_u$ is in NP, we encode an instance of the problem as an instance of LIP. By Corollary 4.2, given any DTD $D$ and any set $\Sigma$ of $L_u$ constraints over $D$, there is a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$ if and only if $\Psi(D^-, \Sigma)$ has a legal integer solution. Note that $\Psi(D^-, \Sigma)$ can be encoded as an instance of LIP. More specifically, observe that an equation $F_1 = F_2$ in $\Psi(D^-, \Sigma)$ is equivalent to disequations $F_1 \geq F_2$ and $F_2 \geq F_1$. Thus without loss of generality, we assume $\Psi(D^-, \Sigma)$ consists of only disequations. We enumerate the unknowns in $\Psi(D^-, \Sigma)$ as $x_1, ..., x_n$ and disequations as $F_1, ..., F_n$. Thus we can define an $m \times n$ matrix $A_\Psi$ of integers and a vector $\vec{b}_\Psi$ of $m$ integers such that for $i \in [1, m]$, $\sum_{j \in [1,n]} a_{ij}\, x_j \geq b_i$ is $F_i$. Hence $\Psi(D^-, \Sigma)$ admits an integer solution if and only if there exists a vector $\vec{x}$ such that $A_\Psi\,\vec{x} \geq \vec{b}_\Psi$. It should be noted that for all unknowns $x$ appearing in $\Psi(D^-, \Sigma)$, we have $x \geq 0$ in $\Psi(D^-, \Sigma)$. In other words, we are interested in non-negative solutions to $A_\Psi\,\vec{x} \geq \vec{b}_\Psi$ only.

We now consider the problem of determining whether $\Psi(D^-, \Sigma)$ admits a legal integer solution. A legal solution requires that $|ext(\tau.l)| > 0$ if $|ext(\tau)| > 0$. To ensure this, consider a system $\Psi$ of linear disequations that includes all the disequations in $\Psi(D^-, \Sigma)$ and moreover, $|ext(\tau.l)| \geq 1$ and $|ext(\tau)| \geq 1$ for each $|ext(\tau.l)|$ and $|ext(\tau)|$ appearing in $\Psi(D^-, \Sigma)$. Let $X$ be some set of pairs $(|ext(\tau.l)|, |ext(\tau)|)$ that appear in $\Psi(D^-, \Sigma)$, and $\Psi_X$ be a system of linear disequations that is the same as $\Psi$ except that it includes $|ext(\tau.l)| = 0$ and $|ext(\tau)| = 0$ for each $(|ext(\tau.l)|, |ext(\tau)|)$ in $X$, instead of $|ext(\tau.l)| \geq 1$ and $|ext(\tau)| \geq 1$. We can encode $\Psi_X$ as an instance $A_{\Psi_X}\,\vec{x} \geq \vec{b}_{\Psi_X}$ of LIP, as described above. It is easy to see that $\Psi(D^-, \Sigma)$ admits a legal integer solution if and only if there is some $A_{\Psi_X}\,\vec{x} \geq \vec{b}_{\Psi_X}$ that admits an integer solution. More specifically, an integer solution of any $A_{\Psi_X}\,\vec{x} \geq \vec{b}_{\Psi_X}$ is a legal integer solution of $\Psi(D^-, \Sigma)$, and conversely, a legal integer solution of $\Psi(D^-, \Sigma)$ is an integer solution of some $A_{\Psi_X}\,\vec{x} \geq \vec{b}_{\Psi_X}$. By the result of [25] cited above, if $\Psi_X$ has an integer solution, then it has one in which each unknown has a length in binary of at most $1+\lceil \log n+(2m+1)\cdot\log(ma)\rceil$ bits, where $m, n$ and $a$ are determined by $\Psi(D^-, \Sigma)$ as described above. In other words, the bounds on solutions for all $\Phi_x$'s are the same. Therefore, if the encoding $A_\Psi\,\vec{x} \geq \vec{b}_\Psi$ of $\Psi(D^-, \Sigma)$ has a legal integer solution, then it has one in which each unknown has a length in binary of at most $1 + \lceil \log n + (2m + 1) \cdot \log(ma)\rceil$ bits. Let $c$ be a number that in binary notation has $1 + \lceil \log n + (2m + 1) \cdot \log(ma)\rceil$ many 1's. Observe that the number $c$ can be computed in PTIME given $\Psi(D^-, \Sigma)$. In light of these, we define a new system $\Phi$ of linear disequations that is the same as $\Psi(D^-, \Sigma)$ except it also includes:

$$c\,|ext(\tau.l)| \geq |ext(\tau)|$$

for every $|ext(\tau.l)|$ and $|ext(\tau)|$ appearing in $\Psi(D^-, \Sigma)$. It is easy to verify that $\Psi(D^-, \Sigma)$ has a legal integer solution if and only $\Phi$ has an integer solution. Indeed, if $\Psi(D^-, \Sigma)$ has a legal integer solution then it has one bounded by $c$. Thus the solution satisfies $c\,|ext(\tau.l)| \geq |ext(\tau)|$. In other words, it is an integer solution of $\Phi$. Conversely, if $\Phi$ has an integer solution, then it is also an integer solution of $\Psi(D^-, \Sigma)$ and moreover, if $|ext(\tau)| > 0$ then $|ext(\tau.l)| > 0$ by $c\,|ext(\tau.l)| \geq |ext(\tau)|$ in $\Phi$. In other words, it is a legal solution of $\Psi(D^-, \Sigma)$. Therefore, by Corollary 4.2, there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$ iff $\Phi$ has an integer solution. By Propositions 3.4, 3.2 and 3.7, $\Psi(D^-, \Sigma)$ can be computed in linear time and its size is linear in the sizes of $D$ and $\Sigma$. Thus $\Phi$ can be computed in PTIME in the sizes of $D$ and $\Sigma$. It should be noted that from the discussions above follows that if $\Phi$ has an integer solution, then it has another solution in which all unknowns are no larger than $c$. Hence determining whether $\Phi$ admits an integer solution is in NP. Therefore, the finite satisfiability problem for $L_u$ is in NP.

We show that the finite satisfiability problem is NP-hard by reduction from a variant of LIP, namely,

$$A\,\vec{x} = \vec{b},$$

where for all $i \in [1, m]$, $j \in [1, n]$, $a_{ij}$ coefficients are in $\{0, 1\}$, all $b_i$ elements are 1, and all $x_j$ components are binary, i.e., in $\{0, 1\}$. It is known that the variant is also NP-complete [20, 23]. Given an instance $A\,\vec{x} = \vec{b}$ of the variant of LIP, we define a DTD $D$ and a set $\Sigma$ of $L_u$ constraints over $D$ such that there is a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$ if and only if $A\,\vec{x} = \vec{b}$ admits a binary solution. For $i \in [1, m]$, we use $F_i$ to denote $\sum_{j \in [1,n]} a_{ij}\, x_j$. We define $D$ to be $(E, A, P, R, r)$, where

$$\begin{array}{lll}
E = \{r\} \cup \{F_i \mid i \in [1, m]\} & & \\
\quad \cup \{b_i \mid i \in [1, m]\} & & \\
\quad \cup \{VF_i \mid i \in [1, m]\} & & \\
\quad \cup \{X_{ij} \mid i \in [1, m], j \in [1, n]\} & & \\
\quad \cup \{Z_{ij} \mid i \in [1, m], j \in [1, n]\} & & \\
A = \{v\} \cup \{A_{ij} \mid i \in [1, m], j \in [1, n]\} & & \\
P(r) & = F_1, \ldots, F_m, b_1, \ldots, b_m & \\
P(F_i) & = X_{ij_1}, \ldots, X_{ij_l} & \text{for } i \in [1, m] \\
 & \text{where } X_{ij_1}, \ldots, X_{ij_l} \text{ is a subsequence of} & \\
 & X_{i1}, \ldots, X_{im} \text{ such that } X_{ij} \text{ is in } P(F_i) & \\
 & \text{iff } a_{ij} \text{ in A is 1} & \\
P(X_{ij}) & = Z_{ij} \mid \epsilon & \text{for } i \in [1, m] \text{ and } j \in [1, n] \\
P(Z_{ij}) & = VF_i & \text{for } i \in [1, m] \text{ and } j \in [1, n] \\
P(VF_i) & = P(b_i) = \epsilon & \text{for } i \in [1, m] \\
R(Z_{ij}) & = \{A_{ij}\} & \text{for } i \in [1, m] \text{ and } j \in [1, n] \\
R(VF_i) & = R(b_i) = \{v\} & \text{for } i \in [1, m] \\
R(r) & = R(F_i) = R(X_{ij}) = \emptyset &
\end{array}$$

Intuitively, $X_{ij}$ indicate $x_j$ in $F_i$, $Z_{ij}$ denotes the value of $X_{ij}$: $X_{ij}$ has value 1 if and only if $X_{ij}$ has a $Z_{ij}$ child. The attribute $A_{ij}$ of $Z_{ij}$ is used to ensure that all occurrences of $x_j$ have the same value. The element type $VF_i$ indicates the value of $F_i$, and its attribute $v$ is to ensure that the value of $F_i$ is 1. More specifically, these are captured by the set $\Sigma$ of $L_u$ constraints over $D$. To ensure that all occurrences of $x_j$ have the same value, the following are in $\Sigma$: for $j \in [1, n]$ and $i, l \in [1, m]$,

$$\begin{array}{l}
Z_{ij}.A_{ij} \rightarrow Z_{ij} \\
Z_{ij}.A_{ij} \subseteq Z_{lj}.A_{lj}
\end{array}$$

These assert that $X_{ij}$ has value 1 if and only if $X_{lj}$ equals to 1. To ensure $F_i = b_i$, we include the following in $\Sigma$: for $i \in [1, m]$,

$$\begin{array}{ll}
VF_i.v \rightarrow VF_i & \qquad b_i.v \rightarrow b_i \\
VF_i.v \subseteq b_i.v & \qquad b_i.v \subseteq VF_i.v
\end{array}$$

These assert that $F_i$ node has a unique $VF_i$ descendent, and thus has value 1. A data tree valid w.r.t. $D$ has the form shown in Figure 3.

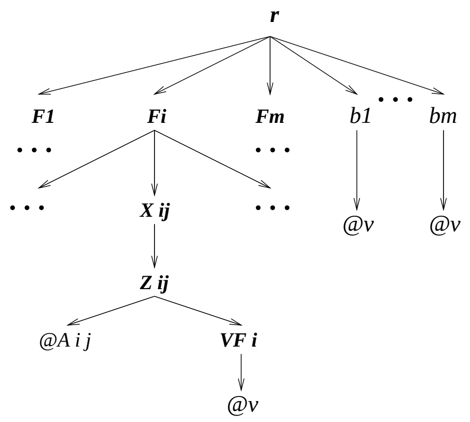


**Figure 3**: A tree used in the proof of Theorem 4.3

It is easy to verify that the encoding can be done in PTIME in $m$ and $n$. Moreover, $A\vec{x} = \vec{b}$ admits a binary solution if and only if there is a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$. Thus what given above is indeed a PTIME reduction from the variant of LIP. This completes the proof of Theorem 4.3.

In relational databases it is common to consider primary keys. That is, for each relation one can specify at most one key, namely, the primary key of the relation. In the XML setting, the *primary key restriction* requires that for each element type $\tau \in E$, one can specify at most one key, i.e., there is at most one $l \in R(\tau)$ such that $\tau.l \to \tau$. Under the primary key restriction, the finite satisfiability problem for $L_u$ is to determine, given any DTD $D$ and finite set $\Sigma$ of $L_u$ constraints in which there is at most one key for each element type (either given as keys or implied by foreign keys), whether there is a finite data tree that satisfies $\Sigma$ and is valid w.r.t. $D$.

One might think that the primary key restriction will simplify the analysis of finite satisfiability of $L_u$ constraints. Unfortunately, this is not true.

**Corollary 4.4:** *Under the primary key restriction, the finite satisfiability problem for $L_u$ is NP-complete.* ∎

To show this, observe that the reduction from LIP given above defines at most one key for each element type. Therefore, it is also a reduction to the finite satisfiability problem under the primary key restriction. As a result, under the primary key restriction the problem remains to be NP-hard.

We next consider special cases of the finite satisfiability problem for $L_u$. We first show that given a fixed DTD $D$, it can be decided in PTIME whether an arbitrary set $\Sigma$ of $L_u$ constraints over $D$ is satisfiable by a finite tree valid w.r.t. $D$. Given a fixed DTD $D$, the number of unknowns in $C_\Sigma$ is bounded by the size of $D$ by Proposition 2.1, and the number of unknowns in $\Psi_{D^-}$ is determined by $D$ and fixed. Thus the number of unknowns in $\Psi(D^-, \Sigma)$ is bounded. Recall the system $\Phi$ given above in the proof of NP-membership of the finite satisfiability problem for $L_u$. Observe that the number of unknowns in $\Phi$ is the same as that in $\Psi(D^-, \Sigma)$, and thus is also bounded given a fixed DTD. It is known that when the number of unknowns in a system of linear disequations is bounded, checking whether the system admits an integer solution can be done in PTIME [22, 26]. As shown above, $\Sigma$ can be satisfied by a finite data tree valid w.r.t. $D$ if and only if $\Phi$ admits an integer solution. Moreover, $\Phi$ can be computed in PTIME in the sizes of $D$ and $\Sigma$. Thus we have:

**Corollary 4.5:** *Given a fixed DTD $D$, it can be determined in PTIME whether an arbitrary set of $L_u$ constraints over $D$ can be satisfied by any finite data tree valid w.r.t. $D$.* ∎

A natural question is: What is the complexity of checking whether a DTD has a finite data tree valid w.r.t. it? This is a special case of the finite satisfiability problem, namely, when the given set of $L_u$ constraints is empty. The theorem below answers the question:

**Theorem 4.6:** *Given any DTD $D$, it can be decided in linear time whether there exists a finite data tree valid w.r.t. $D$.* ∎

To prove this theorem, we first define the following notion. Let $D = (E, A, P, R, r)$ be a DTD and $\alpha$ be a regular expression over $E \cup \{\mathtt{S}\}$. We say that $\alpha$ *yields a finite tree* if one the following is satisfied:

- $\alpha$ is one of $\epsilon$, $\mathtt{S}$ or $\alpha_1^*$;
- $\alpha$ is some $\tau$ in $E$ and $P(\tau)$ yields a finite tree;
- $\alpha = (\alpha_1, \alpha_2)$ and both $\alpha_1$ and $\alpha_2$ yield finite trees;
- $\alpha = (\alpha_1 \mid \alpha_2)$ and either $\alpha_1$ or $\alpha_2$ yields finite tree.

One can verify the following (see Appendix for a proof):

1. There exists a finite data tree valid w.r.t. $D$ iff $P(r)$ yields a finite tree.
2. Given an arbitrary DTD $D$, it can be decided in linear time whether $P(r)$ yields a finite tree.

From these follows immediately Theorem 4.6.

The next corollary answers another natural question: What is the complexity of the finite satisfiability problem for key constraints only?

**Corollary 4.7:** *The problem to determine, given any DTD $D$ and any set $\Sigma$ of key constraints in $L_u$ over $D$, whether there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$ can be decided in linear time in the size of $D$.* ∎

This follows from Theorem 4.6 and Proposition 4.8:

**Proposition 4.8:** *Let $D$ be a DTD and $\Sigma$ be a set of key constraints in $L_u$ over $D$. Then $\Sigma$ can be satisfied by a finite data tree valid w.r.t. $D$ if and only if there exists a finite data tree valid w.r.t. $D$.* ∎

To prove the proposition, observe that given any finite data tree $T_1$ valid w.r.t. $D$, we can always construct another finite data tree $T_2$ by modifying the *val* function in $T_1$ such that $|ext(\tau)| = |ext(\tau.l)|$ for any constraint $\tau.l \to \tau$ in $\Sigma$. Thus $T_2 \models C_\Sigma$. By Theorem 3.1, $T_2 \models \Sigma$. In addition, it is easy to verify that $T_2$ is valid w.r.t. $D$ since $T_1$ is valid w.r.t. $D$.

Observe that the corollary no longer holds if $\Sigma$ consists of only foreign key constraints in $L_u$. This is because the presence of $\tau_1.l_1 \subseteq \tau_2.l_2$ requires that $l_2$ is a key of $\tau_2$, i.e., $\tau_2.l_2 \to \tau_2$. It is easy to show that this special case has the same complexity as that of the finite satisfiability problem for $L_u$.

## 5 Finite satisfiability of $L$ constraints

In this section, we establish the undecidability of the finite satisfiability problem for the class $L$ of multi-attribute keys and foreign keys. To do this, we first show that in relational databases, a finite implication problem associated with keys and foreign keys is undecidable, and then present a reduction from the finite implication problem to the finite satisfiability problem for $L$ constraints.

We first review keys, foreign keys and their associated finite implication problems in relational databases. Let $\mathbf{R} = (R_1, \ldots, R_n)$ be a relational schema. For each relation (schema) $R_i$ in $\mathbf{R}$, we use $Att(R_i)$ to denote the set of all attributes of $R_i$, and $Inst(R_i)$ to denote the set of instances of $R_i$. Observe that a database instance $\mathbf{I}$ of $\mathbf{R}$ has the form $(I_1, \ldots, I_n)$, where $I_i \in Inst(R_i)$ for all $i \in [1, n]$. For an instance $I_i \in Inst(R_i)$, a tuple $t \in I_i$ and an attribute $l \in Att(R_i)$, we use $t.l$ to denote the $l$ attribute value of $t$. Keys and foreign keys over $\mathbf{R}$ are defined as follows:

- *key:* $R[l_1, ..., l_k] \to R$, where $R \in \mathbf{R}$, and for $i \in [1, k]$, $l_i \in Att(R)$. An instance $\mathbf{I}$ of $\mathbf{R}$ *satisfies* a key constraint $\varphi$, denoted by $\mathbf{I} \models \varphi$, if

$$\forall\, t_1\, t_2 \in I\ (\bigwedge_{1 \le i \le k} (t_1.l_i = t_2.l_i) \to \bigwedge_{l \in Att(R)} (t_1.l = t_2.l)),$$

  where $I$ is the instance of $R$ in $\mathbf{I}$;

- *foreign key:* $R[l_1, ..., l_k] \subseteq R'[l'_1, ..., l'_k]$, where $R$, $R'$ are in $\mathbf{R}$, $[l_1, ..., l_k]$ is a sequence of attributes in $Att(R)$, and $[l'_1, ..., l'_k]$ is a sequence of attributes in $Att(R')$. In addition, $R'[l'_1, ..., l'_k] \to R'$, i.e., the set consisting of $l'_1, ..., l'_k$ is a key of $R'$. An instance $\mathbf{I}$ of $\mathbf{R}$ *satisfies* a foreign key constraint $\varphi$, denoted by $\mathbf{I} \models \varphi$, if $\mathbf{I} \models R'[l'_1, ..., l'_k] \to R'$ and moreover,

$$\forall\, t_1 \in I\ \exists\, t_2 \in I'\ (\bigwedge_{1 \le j \le k} t_1.l_j = t_2.l'_j),$$

  where $I$ and $I'$ are the instances of $R$ and $R'$ in $\mathbf{I}$, respectively.

Let $\Sigma \cup \{\varphi\}$ be a set of keys and foreign keys over $\mathbf{R}$. We use $\Sigma \models \varphi$ to denote that $\Sigma$ *finitely satisfies* $\varphi$, i.e., for any finite instance $\mathbf{I}$ of $\mathbf{R}$, if $\mathbf{I} \models \Sigma$, then $\mathbf{I} \models \varphi$.

In relational databases, the *finite implication problem for keys and foreign keys* is the problem of determining, given a relational schema $\mathbf{R}$ and a set $\Sigma \cup \{\varphi\}$ of keys and foreign keys over $\mathbf{R}$, whether $\Sigma \models \varphi$. A special case of the problem is the *finite implication problem for keys by keys and foreign keys*, which is to determine whether $\Sigma \models \varphi$ where $\varphi$ is a key and $\Sigma$ is a set of keys and foreign keys over $\mathbf{R}$.

It was shown in [19] that the finite implication problem for keys and foreign keys in relational databases is undecidable. The theorem below improves the result. A proof of the theorem can be found in Appendix.

**Theorem 5.1:** *In relational databases, the finite implication problem for keys by keys and foreign keys is undecidable.* ∎

One might also want to consider finite implication of foreign keys by keys and foreign keys. That is the problem to determine $\Sigma \models \varphi$ where $\varphi$ is a foreign key and $\Sigma$ is a set of keys and foreign keys over a schema $\mathbf{R}$. However, it is easy to show that the problem has the same complexity as that of the finite implication problem for keys and foreign keys because a foreign key $R[l_1, ..., l_k] \subseteq R'[l'_1, ..., l'_k]$ requires that the set consisting of $l'_1, ..., l'_k$ is a key of $R'$.

From Theorem 5.1 follows immediately that the complement of the finite implication problem for keys by keys and foreign keys is also undecidable. That is the problem to determine, given a relational schema $\mathbf{R}$, a set $\Sigma$ of keys and foreign keys over $\mathbf{R}$ and a key $\varphi$ over $\mathbf{R}$, whether there is a finite instance of $\mathbf{R}$ satisfying $\bigwedge \Sigma \wedge \neg\varphi$. Using this we show another main complexity result of the paper:

**Theorem 5.2:** *The finite satisfiability problem for L constraints is undecidable.* ∎

We prove this theorem by reduction from the complement of the finite implication problem for keys by keys and foreign keys. Let $\mathbf{R} = (R_1, \ldots, R_n)$ be a relational schema, $\Theta$ be a set of keys and foreign keys over $\mathbf{R}$, and $\varphi = R[X] \to R$ be a key over $\mathbf{R}$. Let $Y = Att(R) \setminus X$. We encode $\mathbf{R}$, $\Theta$ and $\varphi$ in terms of a DTD $D$ and a set $\Sigma$ of $L$ constraints over $D$ as follows. Let $D = (E, A, P, R_A, r)$, where

$$
\begin{array}{lll}
E = \{R_i \mid i \in [1,n]\} \cup \{t_i \mid i \in [1,n]\} & & \\
\quad\; \cup \{r,\ D_Y,\ E_X\} & & \\
A = \bigcup_{i\in[1,n]} Att(R_i) & & \\
P(r) & = R_1, \ldots, R_n, D_Y, D_Y, E_X & \\
P(R_i) & = t_i^* & \text{for } i \in [1,n] \\
P(t_i) & = \epsilon & \text{for } i \in [1,n] \\
P(D_Y) & = P(E_X) = \epsilon & \\
R_A(t_i) & = Att(R_i) & \text{for } i \in [1,n] \\
R_A(D_Y) & = X \cup Y & \\
R_A(E_X) & = X & \\
R_A(r) & = R_A(R_i) = \emptyset & \text{for } i \in [1,n]
\end{array}
$$

In particular, we denote $P(R) = t_\varphi^*$ for the relation $R$ in $\varphi$. Note that $R = R_s$ and $t_\varphi = t_s$ for some $s \in [1,n]$.

We encode $\Theta$ and $\varphi$ with $\Sigma = \Sigma_\Theta \cup \Sigma_\varphi$, where $\Sigma_\Theta$ is defined as follows:

- For every key $R_i[Z] \to R_i$ in $\Theta$, $t_i[Z] \to t_i$ is in $\Sigma_\Theta$.
- For every foreign key $R_i[Z] \subseteq R_j[Z']$ in $\Theta$, $\Sigma_\Theta$ includes $t_i[Z] \subseteq t_j[Z']$.

The set $\Sigma_\varphi$ consists of the following:

$$
\begin{array}{rcl}
D_Y[Y] & \to & D_Y, \\
E_X[X] & \to & E_X, \\
D_Y[X] & \subseteq & E_X[X], \\
D_Y[X,Y] & \subseteq & t_\varphi[X,Y], \\
t_\varphi[XY] & \to & t_\varphi,
\end{array}
$$

where $[X,Y]$ stands for the concatenation of list $X$ and list $Y$, and $t_\varphi$ is the grammar symbol in $P(R) = t_\varphi^*$. Observe that $Att(R) = X \cup Y$ and thus we can require that $XY$ is a key of $t_\varphi$.

As depicted in Figure 4, in any finite data tree valid w.r.t. $D$, there are two distinct $D_Y$ nodes $d_1$ and $d_2$ that have all the attributes in $X \cup Y$, and a single $E_X$ node that has all attributes in $X$. If $T \models \Sigma_\varphi$, then

- $d_1[X] = d_2[X]$ by $D_Y[X] \subseteq E_X[X]$ and the fact $|ext(E_X)| = 1$,
- $d_1[Y] \neq d_2[Y]$ by $D_Y[Y] \to D_Y$.

These nodes will serve as a witness for $\neg\varphi$.

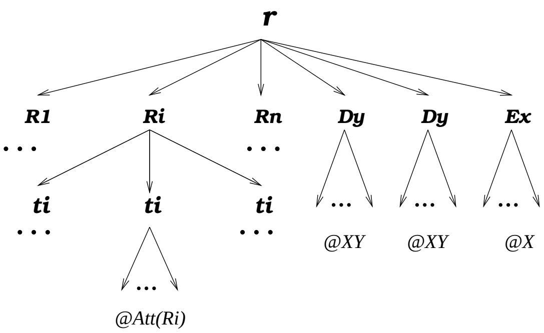


**Figure 4**: A tree used in the proof of Theorem 5.2

Given these, we show that $\bigwedge \Theta \wedge \neg\varphi$ can be satisfied by a finite instance of $\mathbf{R}$ if and only if $\Sigma$ can be satisfied by a finite data tree valid w.r.t. $D$. Assume that there is a finite instance $\mathbf{I}$ of $\mathbf{R}$ satisfying $\bigwedge \Theta \wedge \neg\varphi$. We construct a finite data tree $T$ from $\mathbf{I}$ as follows. Let $T$ have a root node $r$ and a $R_i$ node for each $R_i$ in $\mathbf{R}$. For any $R_i \in \mathbf{R}$ and each tuple $p$ in the instance of $R_i$ in $\mathbf{I}$, we create a distinct $t_i$ node $x$ such that $p.l = x.l$ for all $l \in Att(R_i)$. Since $\mathbf{I} \models \neg\varphi$, there are two tuples $p$ and $p'$ in the instance of $R$ in $\mathbf{I}$ such that $p[X] = p'[X]$ and $p[Y] \neq p'[Y]$. We create two distinct $D_Y$ nodes $d_1$ and $d_2$ such that $d_1.l = p.l$ and $d_2.l = p'.l$ for all $l \in Att(R)$. In addition, we create a single $E_X$ node $e$ such that $e.l = p.l$ for all $l \in X$. We define the edge relation of $T$ such that $T$ has the form shown in Figure 4. It is easy to verify that $T$ is a finite tree valid w.r.t. $D$. By $\mathbf{I} \models \Theta$ it is easy to verify that $T \models \Sigma_\Theta$. By the definition of $T$, it is also easy to see that $T \models \Sigma_\varphi$. In particular, since $Att(R) = X \cup Y$, we have $T \models t_\varphi[XY] \to t_\varphi$, where $t_\varphi$ is a symbol in $P(R) = t_\varphi^*$. Therefore, $T \models \Sigma$. Conversely, suppose that there is a finite data tree $T$ valid w.r.t. $D$ and satisfying $\Sigma$. We define a finite instance $\mathbf{I}$ of schema $\mathbf{R}$ as follows. For each $t_i$ node $x$, let $(l_1 = x.l_1, \ldots, l_m = x.l_m)$ be a tuple in the instance of $R_i$ in $\mathbf{I}$, where $l_1, \ldots, l_m$ are an enumeration of $Att(R_i)$. Obviously $\mathbf{I}$ is a finite instance of $\mathbf{R}$. By $T \models \Sigma_\Theta$, it is easy to verify that $\mathbf{I} \models \Theta$. Moreover, by $T \models \Sigma_\varphi$ and the definition of $\mathbf{I}$, we have $\mathbf{I} \models \neg\varphi$ since there must be two tuples $d_1$ and $d_2$ in the instance of $R$ in $\mathbf{I}$ such that $d_1[X] = d_2[X]$ but $d_1[Y] \neq d_2[Y]$. Thus the encoding is indeed a reduction from the complement of the finite implication problem for keys by keys and foreign keys. From this follows Theorem 5.2.

Observe that given a fixed DTD $D$, there are only finitely many $L$ constraints over $D$. Thus when DTD $D$ is fixed, finite satisfiability of arbitrary sets of $L$ constraints over $D$ is decidable [17].

A more interesting special case of the finite satisfiability problem for $L$ constraints is the *finite satisfiability problem for keys in* $L$. That is the problem to determine, given any DTD $D$ and any set $\Sigma$ of key constraints in $L$ over $D$, whether there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$. The corollary below shows that this problem is decidable.

**Corollary 5.3:** *The finite satisfiability problem for keys in $L$ is decidable in linear time.* ∎

The decidability of the problem follows from Theorem 4.6. A proof of Corollary 5.3 is given in Appendix.

One might be tempted to think that finite satisfiability of foreign keys in $L$ would be simpler than finite satisfiability of general $L$ constraints. That is the problem to determine, given any DTD $D$ and any set $\Sigma$ of foreign keys in $L$ over $D$, whether there exists a finite data tree valid w.r.t. $D$ and satisfying $\Sigma$. It is not hard to show that this problem, however, has the same complexity as that of the finite satisfiability problem for $L$ and therefore, is undecidable. This is because a foreign key $\tau[X] \subseteq \tau'[Y]$ in $\Sigma$ requires $\tau'[Y] \to \tau'$, i.e., the presence of keys.

## 6 Conclusion

We have investigated the interaction between DTDs and key and foreign key constraints for XML. We have demonstrated that DTDs impose a form of cardinality constraints on XML documents and in the presence of these cardinality constraints, the analysis of finite satisfiability of key and foreign key for XML becomes far more intriguing than that for relational databases. In particular, we have introduced and studied a class $L_u$ of unary keys and foreign keys, and a class $L$ of multi-attribute keys and foreign keys for XML. We have shown that both DTDs and $L_u$ constraints can be characterized by cardinality constraints. Taking advantage of these characterizations, we have shown that the finite satisfiability problem for $L_u$ constraints is NP-complete for XML. We have also established PTIME decidability for several special cases of finite satisfiability of $L_u$ constraints. In addition, we have shown that the finite satisfiability problem for $L$ is undecidable.

This work is the first step towards understanding the interaction between DTDs and integrity constraints. A number of questions remain open. First, we have only considered key and foreign key constraints defined in terms of XML attributes. More expressive key and foreign key constraints, such as those defined in terms of XPath [15] expressions (or regular path expressions), have been proposed for XML (e.g., XML Schema [28]). We expect to extend some of the complexity results established in this paper to these more powerful constraint languages. Second, the finite implication problem for key and foreign key constraints needs to be studied. This problem has been investigated for XML data modeled as graphs [19], but has not been addressed for the tree model of XML that is typically used to represent XML data. We defer to another paper [18] for a full treatment of finite implication of keys and foreign keys. Third, other constraints commonly found in databases, such as inverse constraints, have not been studied for the tree model of XML.

**Acknowledgements**. We thank Michael Benedikt, Peter Buneman, Frank Neven and Jérôme Siméon for valuable suggestions and discussions.

## 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. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 122–133, Tucson, Arizona, May 1997.

[3] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, Oct. 1998. `http://www.w3.org/TR/REC-DOM-Level-1/`.

[4] C. Beeri and T. Milo. Schemas for integration and translation of structured and semi-structured data. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 296–313, Jerusalem, Israel, Jan. 1999.

[5] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, Feb. 1998. `http://www.w3.org/TR/REC-xml/`.

[6] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. Draft manuscript, July 2000.

[7] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. Technical Report TUCIS-TR-2000-005, Department of Computer and Information Sciences, Temple University, 2000.

[8] P. Buneman, W. Fan, and S. Weinstein. Path constraints on semistructured and structured data. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 129–138, Seattle, Washington, June 1998.

[9] P. Buneman, W. Fan, and S. Weinstein. Interaction between path and type constraints. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 56–67, Philadelphia, Pennsylvania, May 1999.

[10] P. Buneman, W. Fan, and S. Weinstein. Query optimization for semistructured data using path constraints in a deterministic data model. In *Proceedings of International Workshop on Database Programming Languages (DBPL)*, 1999.

[11] D. Calvanese, G. De Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *Journal of Logic and Computation*, 9(3):295–318, June 1999.

[12] D. Calvanese and M. Lenzerini. Making object-oriented schemas more expressive. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 1994.

[13] D. Calvanese and M. Lenzerini. On the interaction between isa and cardinality constraints. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, 1994.

[14] J. Clark. XSL Transformations (XSLT). W3C Recommendation, Nov. 1999. `http://www.w3.org/TR/xslt`.

[15] J. Clark and S. DeRose. XML Path Language (XPath). W3C Working Draft, Nov. 1999. `http://www.w3.org/TR/xpath`.

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

[17] B. Dreben and W. D. Goldfarb. *The Decision Problem: Solvable Classes of Quantificational Formulas.* Addison-Wesley, 1979.

[18] W. Fan and L. Libkin. Finite implication of keys and foreign keys for XML data. Technical Report TUCIS-TR-2000-004, Department of Computer and Information Sciences, Temple University, 2000.

[19] W. Fan and J. Siméon. Integrity constraints for XML. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 23–34, Dallas, Texas, May 2000.

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

[21] A. Layman, E. Jung, E. Maler, H. S. Thompson, J. Paoli, J. Tigue, N. H. Mikula, and S. De Rose. XML-Data. W3C Note, Jan. 1998. `http://www.w3.org/TR/1998/NOTE-XML-data`.

[22] H. W. Lenstra. Integer programming in a fixed number of variables. *Math. Oper. Res.*, 8:538–548, 1983.

[23] H. R. Lewis and C. H. Papadimitrious. *Elements of the Theory of Computation.* Prentice-Hall, 1981.

[24] F. Neven. Extensions of attribute grammars for structured document queries. In *Proceedings of International Workshop on Database Programming Languages (DBPL)*, 1999.

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

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

[27] J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). Workshop on XML Query Languages, Dec. 1998.

[28] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. W3C Working Draft, Apr. 2000. `http://www.w3.org/TR/xmlschema-1/`.

[29] J. D. Ullman. *Database and Knowledge Base Systems.* Computer Science Press, 1988.

[30] P. Wadler. A Formal Semantics for Patterns in XSL. Technical report, Computing Sciences Research Center, Bell Labs, Lucent Technologies,

2000. http://www.cs.bell-labs.com/~wadler/topics/xml#xsl-semantics.

## Appendix

**Proof of Theorem 3.1.** We show (1) $\Leftrightarrow$ (2).

(1) $\Rightarrow$ (2). This direction is immediate. Given a finite data tree $T_1$ such that $T_1$ is valid w.r.t. $D$ and $T_1 \models \Sigma$, it is easy to verify that $T_1 \models C_\Sigma$. Thus (2) holds.

(2) $\Rightarrow$ (1). Let $T_2 = (V, lab, ele, att, val, root)$ be a finite data tree valid w.r.t. $D$ and satisfying $C_\Sigma$. We construct a data tree $T_1 = (V, lab, ele, att, val', root)$ such that $T_1 \models \Sigma$. Observe that the only difference between $T_1$ and $T_2$ is the definition of the function $val'$. The function $val'$ has the following property: for any $v \in V$,

$$val'(v) \;=\; \begin{cases} val(v) & \text{if } lab(v) \notin A \\ \text{a string} & \text{otherwise} \end{cases}$$

This suffices since $T_1 \models \Sigma$ and in addition, it is easy to verify that $T_1$ is a finite data tree valid w.r.t. $D$ by the assumption about $T_2$ and Definition 2.2.

We next define the function $val'$ such that $T_1 \models \Sigma$. To do so, let

$$S \;=\; \{\tau.l \mid \tau \in E,\, l \in R(\tau)\}.$$

We define a partial order $\preceq$ on $S$ as follows:

- $\tau.l \preceq \tau.l$,
- $\tau_1.l_1 \preceq \tau_2.l_2$ if $T_2 \models |ext(\tau_1.l_1)| \leq |ext(\tau_2.l_2)|$,
- $\tau_1.l_1 \preceq \tau_3.l_3$ if $\tau_1.l_1 \preceq \tau_2.l_2$ and $\tau_2.l_2 \preceq \tau_3.l_3$.

It is easy to verify that $\preceq$ is indeed a partial order on $S$. Using $\preceq$ we define an equivalence relation $\sim$ on $S$ as follows:

$$\tau_1.l_1 \sim \tau_2.l_2 \quad \text{iff} \quad \tau_1.l_1 \preceq \tau_2.l_2 \;\text{ and }\; \tau_2.l_2 \preceq \tau_1.l_1$$

Let $[\tau.l]$ denote the equivalence class of $\tau.l$ with respect to $\sim$, and let

$$S_{[\,]} \;=\; \{[\tau.l] \mid \tau.l \in S\}.$$

We now extend $\preceq$ to $S_{[\,]}$ such that $[\tau_1.l_1] \preceq [\tau_2.l_2]$ iff $\tau_1.l_1 \preceq \tau_2.l_2$. We then topologically sort $S_{[\,]}$ based on $\preceq$. For each $[\tau.l]$, let $s[\tau.l]$ be a distinct set. Based on the topological order, we assign a set of string values to each $s[\tau.l]$ using the algorithm below:

1. for each $s[\tau.l] \in S_{[\,]}$ do
   $s[\tau.l] := \emptyset$;
2. for each $s[\tau.l] \in S_{[\,]}$ in the topological order do
   (1) add distinct string values to $s[\tau.l]$ such that
   $|s[\tau.l]|$ is equal to $|ext(\tau.l)|$ in $T_2$;
   (2) for each $s[\tau'.l'] \in S_{[\,]}$ such that
   $[\tau.l] \preceq [\tau'.l']$ do
   $s[\tau'.l'] := s[\tau'.l'] \cup s[\tau.l]$;

Now for each $\tau \in E$, $l \in R(\tau)$ and $x \in ext(\tau)$, let $val'(att(x,l))$ be a string value in $s[\tau.l]$ such that in $T_1$, $|s[\tau.l]| = |ext(\tau.l)|$. This is possible since $T_2 \models C_\Sigma$ and moreover, $ext(\tau)$ in $T_1$ is the same as $ext(\tau)$ in $T_2$ by the definition of $T_1$. Given the definitions of $T_1$ and $val'$ above, it is easy to verify that $T_1 \models C_\Sigma$. In addition, it can be verified that $T_1 \models \Sigma$. More specifically, for any $\varphi \in \Sigma$, (1) if $\varphi$ is $\tau.l \to \tau$, then for any distinct $x, y \in ext(\tau)$, $x.l \neq y.l$ by the cardinality constraint $|ext(\tau.l)| = |ext(\tau)|$ and the definition of $val'$; (2) if $\varphi$ 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 $T_2 \models C_\Sigma$. Thus for any $x \in ext(\tau_1)$, we have that $x.l_1 \in ext(\tau_2.l_2)$ by the definition of $val'$. Therefore, $T_1$ is indeed a finite data tree such that $T_1$ is valid w.r.t. $D$ and $T_1 \models \Sigma$. Thus (1) holds. ■

**Proof of Theorem 3.3.** We first show the following:

Claim: Let $D$ be a DTD and $D^-$ be the $*$-free DTD of $D$. Then there exists a finite data tree $T_1$ valid w.r.t. $D$ iff there exists a finite data tree $T_2$ valid w.r.t. $D^-$.

To verify the claim, first assume that there exists a finite data tree $T_1 = (V_1, lab_1, ele_1, att, val, root)$ such that $T_1$ is valid w.r.t. $D$. We construct a data tree

$$T_2 = (V_2, lab_2, ele_2, att, val, root)$$

by modifying $T_1$ such that $T_2$ is valid w.r.t. $D^-$. We proceed to construct $T_2$ as follows. Initially, let $V_2 = V_1$, $lab_2 = lab_1$ and $ele_2 = ele_1$. Note that $att$, $val$ and $root$ are the same in $T_1$ and $T_2$. We traverse $T_2$ starting from $root$. For each $v \in V_2$ such that $lab(v) = \tau$ and $\tau \in E$, we increment $V_2$, $lab_2$ and $ele_2$ as follows. Assume $P(\tau) = \alpha$ and $ele(v) = [v_1, ..., v_n]$. Let $w = lab(v_1) ... lab(v_n)$. Since $T_1$ is valid w.r.t $D$, by Definition 2.2, $w$ must be in the regular language defined by $\alpha$. By the definitions of the transition function $f$ and the function $P^-$ for element type definitions in $D^-$, it is easy to verify that $w$ can be derived from the rule $P^-(\tau) = f(\alpha)$ and the related rules for new element types in $E_a$ introduced by $f(\alpha)$. Let $t$ be a derivation tree of $w$. Without loss of generality, assume the root of $t$ is $v$, the leaves of $t$ are $v_1, ..., v_n$, and internal nodes of $t$ are $x_1, ..., x_m$. Observe that these internal nodes are labeled with element types in $E_a$, i.e., $E^- \setminus E$. We modify $T_2$ by substituting $t$ for $v$. More specifically, we (i) increment $V_2$ by including $x_1, ..., x_m$; (ii) increment $lab_2$ by defining $lab_2(x_i)$ to be the label of $x_i$ in $t$, for each $i \in [1, m]$; and (iii) increment $ele_2$ according to the structure of $t$, i.e., by substituting the children of $v$ in $t$ for $[v_1, ..., v_n]$ and by defining $ele_2(x_i)$ to be the children of $x_i$ in $t$, for $i \in [1, m]$. Let $T_2$ be the tree obtained at the end of the traversal. It is easy to verify that $T_2$ is finite and in addition, $T_2$ is valid w.r.t $D^-$.

Conversely, let $T_2 = (V_2, lab_2, ele_2, att, val, root)$ such that $T_2$ is valid w.r.t. $D^-$. We construct a data tree

$$T_1 = (V_1, lab_1, ele_1, att, val, root)$$

by modifying $T_2$ such that $T_1$ is valid w.r.t. $D$. We construct $T_1$ as follows. For any node $v \in V_2$ with $lab(v) = \tau$ and $\tau \in E_a$, i.e., $E^- \setminus E$, we substitute the sub-elements of $v$ for $v$ in $ele(v')$, where $v'$ is the parent of $v$. In addition, we delete (i) $v$ from $V_2$, (ii) $lab_2(v)$ from $lab_2$, and (iii) $ele_2(v)$ from $ele_2$. We repeat the process until there is no node labeled with element type in $E^- \setminus E$. Now let $V_1$, $lab_1$ and $ele_1$ be $V_2$, $lab_2$ and $ele_2$ obtained at the end of the process, respectively. Observe that $att$, $val$ and $root$ remain unchanged. It is easy to verify that $T_1$ is indeed a finite data tree valid w.r.t $D$.

Using the claim, we next show Theorem 3.3.

(1) $\Rightarrow$ (2). Assume that there exists a finite data tree $T_1$ such that $T_1$ is valid w.r.t. $D$ and $T_1 \models \Sigma$. We show that there exists a finite date tree $T_2$ such that $T_2$ is valid w.r.t. $D^-$ and $T_2 \models \Sigma$. By Theorem 3.1, there is a finite data tree $T_1'$ such that $T_1'$ is valid w.r.t. $D$ and $T_1' \models C_\Sigma$, where $C_\Sigma$ is the set of cardinality constraints determined by $\Sigma$. By the claim, we can construct a finite data tree $T_2'$ such that $T_2'$ is valid w.r.t. $D^-$. Moreover, observe that the construction given in the proof of the claim does not alter $ext(\tau)$ and $ext(\tau.l)$ for any $\tau \in E$ and $l \in R(\tau)$. Therefore, by $T_1' \models C_\Sigma$ we have $T_2' \models C_\Sigma$. Thus again by Theorem 3.1, there is a finite data tree $T_2$ such that $T_2$ is valid w.r.t. $D^-$ and $T_2 \models \Sigma$. That is, (2) holds.

(2) $\Rightarrow$ (1). Suppose that there exists a finite date tree $T_2$ such that $T_2$ is valid w.r.t. $D^-$ and $T_2 \models \Sigma$. We show that there exists a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and $T \models \Sigma$. The proof is similar to the one given above. Consider the construction of $T_1$ valid w.r.t. $D$ from $T_2$ valid w.r.t. $D^-$ given in the proof of the claim. This construction does not alter $ext(\tau)$ and $ext(\tau.l)$ for any $\tau \in E$ and $l \in R(\tau)$. Thus $T_1 \models C_\Sigma$ iff $T_2 \models C_\Sigma$. Hence by Theorem 3.1, there exists a finite data tree $T$ valid w.r.t. $D$ and satisfying $\Sigma$. Therefore, (1) holds if (2) holds. ■

**Proof of Theorem 3.5.** First, assume

$$T = (V, lab, ele, att, val, root)$$

is a finite data tree valid w.r.t. $D$. We show that $\Psi_D$ admits an integer solution. It suffices to define an integer assignment to unknowns in $\Psi_D$ that satisfies equations and disequations in $\Psi_D$. To do so, for each $\tau \in E$, let the unknown $|ext(\tau)|$ be the number of vertices in $V$ labeled $\tau$. Assume $P(\tau) = \alpha$. Since $T$ is valid w.r.t. $D$, for any vertex $v \in \tau$ with $lab(v) = \tau$ and $ele(v) = [v_1, ..., v_n]$, $lab(v_1)...lab(v_n)$ must be in the regular language defined by $\alpha$. In other words, $[v_1, ..., v_n]$ observes the regular expression $\alpha$. Thus for $i \in [1, n]$, we can mark $v_i$ with $x^i_{\tau'}$ if $v_i$ corresponds to a certain occurrence of $\tau'$ in $\alpha$, following the definition of the function $g$. Similarly, we can mark a sequence of $[v_1, ..., v_n]$ with $y^\tau_{\beta,k}$ to indicate that the sequence corresponds to a certain occurrence of the regular expression $\beta$ in $\alpha$. More specifically, we mark $\vec{v} = [v_1, ..., v_n]$ with $y^\tau_{\alpha,1}$, and consider the structure of $\alpha$ as follows. If $\alpha$ is some $\tau' \in E \cup \{\mathtt{S}\}$, then $n$ must equal to 1. We mark $v_1$ with $x^i_{\tau'}$, where $x^i_{\tau'}$ is the corresponding variable in $g(\alpha, y^\tau_{\alpha,1})$. If $\alpha$ is $\epsilon$, then $n = 0$. We mark the empty sequence with $y^\tau_{\epsilon,k}$, where $y^\tau_{\epsilon,k}$ is the corresponding variable in $g(\alpha, y^\tau_{\alpha,1})$. If $\alpha$ is $(\beta_1, \beta_2)$, then $\vec{v}$ can be partitioned into two sub-sequences $\vec{v_1} = [v_1, ..., v_m]$ and $\vec{v_2} = [v_{m+1}, ..., v_n]$ such that $lab(v_1) \ldots lab(v_m)$ and $lab(v_{m+1}) \ldots lab(v_n)$ are in the regular languages defined by $\beta_1$ and $\beta_2$, respectively. We mark $\vec{v_1}$ with $y^\tau_{\beta_1,k_1}$ and mark $\vec{v_2}$ with $y^\tau_{\beta_2,k_2}$, where $y^\tau_{\beta_1,k_1}$ and $y^\tau_{\beta_2,k_2}$ are the corresponding variables in $g(\alpha, y^\tau_{\alpha,1})$. We then proceed to mark sub-sequences of $\vec{v_1}$ and $\vec{v_2}$ in the same way. If $\alpha$ is $(\beta_1 \mid \beta_2)$, then $\vec{v}$ corresponds to either $\beta_1$ or $\beta_2$. We mark the sequence with the corresponding variable in $g(\alpha, y^\tau_{\alpha,1})$ and proceed to mark its sub-sequences as above. These marks provide a finer labeling that keeps track of how $[v_1, ..., v_n]$ corresponds to regular expressions in $P(\tau)$. Let the unknown $x^i_{\tau'}$ be assigned as its value the number of all vertices in $V$ marked $x^i_{\tau'}$, and similarly, let $y^\tau_{\beta,k}$ be assigned the number of all vertex sequences marked $y^\tau_{\beta,k}$. This is indeed an integer assignment since $T$ is finite. This assignment must satisfy $\psi_\tau$ for each $\tau \in E$, since $T$ is valid w.r.t. $D$ and $\psi_\tau$ is determined by $P(\tau)$ in $D$. Formally, this can be verified by induction on the structure of $P(\tau)$. In addition, it is easy to verify that the assignment satisfies the other constraints in $\Psi_D$. More specifically, $|ext(r)| = 1$, all unknowns $\geq 0$, and $ext(\tau)$ is the collection of all vertices corresponding to some occurrences of $\tau$ in element type definitions of $D$. Therefore, the assignment is indeed an integer solution of $\Psi_D$. That is, $\Psi_D$ is finitely satisfiable.

Conversely, assume that $\Psi_D$ admits an integer solution. That is, there is an integer assignment to unknowns $|ext(\tau)|$, $x^i_{\tau'}$ and $y^\tau_{\beta,k}$ satisfying equations and disequations in $\Psi_D$. Observe that all these unknowns are equal to or greater than 0 because of disequations in $\Psi_D$. We show that there is finite data tree

$$T = (V,\, lab,\, ele,\, att,\, val,\, root)$$

valid w.r.t. $D$. To do so, for each $\tau \in E \cup \{\mathtt{S}\}$, we create $|ext(\tau)|$ many distinct vertices and label them with $\tau$. We refer to this set of vertices as $ext(\tau)$. In addition, for each $\tau \in E$, each $v \in ext(\tau)$ and $l \in R(\tau)$, we create a distinct vertex and label it with $l$. We refer to this vertex as $v_l$. Let

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

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

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

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

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

It is easy to verify that these functions are well defined. Let $root$ be the vertex labeled $r$. This vertex is unique since $|ext(r)| = 1$ is in $\Psi_D$. Finally, we define the function $ele$. To define $ele$, we do the following:

(1) For each $x^i_\tau$ in $\Psi_D$, we choose $x^i_\tau$ many distinct vertices labeled $\tau$ and mark them with $x^i_\tau$. We mark a vertex at most once. Observe that every vertex in $V$ labeled $\tau$ can be marked once and only once since the assignment to unknowns is an integer solution of $\Psi_D$.

(2) For each $y^\tau_{\beta,k}$ in $\Psi_D$, we choose $y^\tau_{\beta,k}$ many distinct sequences of vertices in $V$ and mark them with $y^\tau_{\beta,k}$, following the definition of the function $g$. More specifically, if $y^\tau_{\beta,k} = x^i_{\tau'}$ is in $\Psi_D$ for some $\tau' \in E \cup \{\mathtt{S}\}$, then we pick $x^i_{\tau'}$ many distinct vertices labeled $x^i_{\tau'}$ and mark them also with $y^\tau_{\beta,k}$. If $y^\tau_{\beta,k} = y^\tau_{\epsilon,j}$ is in $\Psi_D$, then we create $y^\tau_{\beta,k}$ many empty sequences and mark them with $y^\tau_{\beta,k}$. If $y^\tau_{\beta,k} = y^\tau_{\beta_1,k_1}$ and $y^\tau_{\beta,k} = y^\tau_{\beta_2,k_2}$ are in $\Psi_D$, then we choose $y^\tau_{\beta,k}$ many distinct sequences marked with $y^\tau_{\beta_1,k_1}$ and $y^\tau_{\beta,k}$ many distinct sequences marked with $y^\tau_{\beta_2,k_2}$. We create $y^\tau_{\beta,k}$ many distinct sequences of the form $\vec{v} = \vec{v_1}\vec{v_2}$, where $\vec{v_1}$ and $\vec{v_2}$ are sequences chosen as above and marked with $y^\tau_{\beta_1,k_1}$ and $y^\tau_{\beta_2,k_2}$, respectively. We mark $\vec{v}$ with $y^\tau_{\beta,k}$. If $y^\tau_{\beta,k} = y^\tau_{\beta_1,k_1} + y^\tau_{\beta_2,k_2}$ is in $\Psi_D$, then we choose $y^\tau_{\beta_1,k_1}$ many distinct sequences marked

with $y^{\tau}_{\beta_1,k_1}$ and $y^{\tau}_{\beta_2,k_2}$ many distinct sequences marked with $y^{\tau}_{\beta_2,k_2}$. We mark these sequences also with $y^{\tau}_{\beta,k}$. Again this is possible by the definition of constraints in $\Psi_D$.

For each $v \in V$, if $lab(v) = \tau$, $\tau \in E$ and $P(\tau) = \alpha$, then we choose a distinct sequence of vertices $[v_1, ..., v_n]$ marked with $y^{\tau}_{\alpha,1}$. This is possible by constraints of $\Psi_D$. We define $ele(v) = [v_1, ..., v_n]$. It can be shown that $lab(v_1)...lab(v_n)$ is indeed in the regular language defined by $\alpha$, by the definition of $\Psi_D$. Formally, this can be verified by induction on the structure of $\alpha$, i.e., considering the cases when $\alpha$ is one of $\mathtt{S}$, $\tau$, $\epsilon$, $(\beta_1 \mid \beta_2)$ and $(\beta_1,\ \beta_2)$. It is also easy to verify that $T$ defined in this way is indeed a finite data tree valid w.r.t. $D$. Therefore, there exists a finite data tree valid w.r.t. $D$. ∎

**Proof of Corollary 3.6.** This follows immediately from the proof of Theorem 3.5. Indeed, given an integer solution of $\Psi_D$, the proof of Theorem 3.5 constructs a finite data tree $T$ valid w.r.t. $D$. In addition, for any $\tau \in E$, the number of vertices in $T$ labeled $\tau$ is exactly the non-negative integer value assigned to the unknown $|ext(\tau)|$ in $\Psi_D$. Thus Corollary 3.6 holds. ∎

**Proof of Theorem 4.1.** Suppose that there exists a finite data tree $T$ such that $T$ is valid w.r.t. DTD $D$ and $T \models \Sigma$. We show that $\Psi(D, \Sigma)$ admits a legal integer solution. By Theorem 3.1, we have $T \models C_\Sigma$, where $C_\Sigma$ is the finite set of cardinality constraints determined by $\Sigma$. As in the proof of Theorem 3.5, we can define an integer assignment to the unknowns in $\Psi(D, \Sigma)$ such that equations and disequations in $\Psi_D$ are satisfied. Observe that the assignment assures that for each $\tau \in E$, the unknown $|ext(\tau)|$ equals to the number of all vertices in $T$ labeled $\tau$. We extend the assignment as follows: for each $\tau \in E$ and $l \in R(\tau)$, let the unknown $|ext(\tau.l)|$ be the number of distinct $l$ attribute values of all vertices in $T$ labeled $\tau$. Thus by $T \models C_\Sigma$, this extended assignment satisfies all the equations and disequations in $C_\Sigma$. In addition, if $|ext(\tau)| > 0$ then $|ext(\tau.l)| > 0$ since every $\tau$ element in $T$ has an $l$ attribute. Hence the assignment is indeed a legal solution of $\Psi(D, \Sigma)$. This is, $\Psi(D, \Sigma)$ is finitely satisfiable.

Conversely, suppose that $\Psi(D, \Sigma)$ admits a legal integer solution. We show that there is a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and $T \models \Sigma$. As in the proofs of Theorem 3.5 and Corollary 3.6, given an integer solution of $\Psi(D, \Sigma)$, we can construct a finite data tree

$$T' = (V, lab, ele, att, val, root)$$

such that $T'$ is valid w.r.t. $D$. Moreover, for each $\tau \in E$, if the unknown $|ext(\tau)|$ is assigned $n_\tau$ by the integer assignment, then $n_\tau$ is number of all vertices in $T'$ labeled $\tau$. We modify the definition of the function $val$ such that in $T'$, for each $\tau \in E$ and $l \in R(\tau)$ that occur in $\Sigma$, the number of distinct $l$ attribute values of all vertices labeled $\tau$ equals to the value assigned to the unknown $|ext(\tau.l)|$ by the assignment. This is possible since $|ext(\tau.l)| \leq |ext(\tau)|$ is in $C_\Sigma$, and the assignment is also a solution of $C_\Sigma$. Therefore, there is a finite data tree $T''$ such that $T'' \models C_\Sigma$ and $T''$ is valid w.r.t. $D$. Observe that since the solution is legal, we have $|ext(\tau.l)| > 0$ if $|ext(\tau)| > 0$, and thus every $\tau$ element in $T''$ can have an $l$ attribute. Hence by Theorem 3.1, there exists a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and moreover, $T \models \Sigma$. ∎

**Proof of Corollary 4.2.** First assume that there exists a finite data tree $T$ such that $T$ is valid w.r.t. DTD $D$ and $T \models \Sigma$. We show that $\Psi(D^-, \Sigma)$ admits a legal integer solution, where $D^-$ is the $*$-free DTD of $D$. By Theorem 3.3, there exists a finite data tree $T'$ such that $T'$ is valid w.r.t. $D^-$ and $T' \models \Sigma$. Thus by Theorem 4.1, $\Psi(D^-, \Sigma)$ is finitely satisfiable.

Conversely, suppose that $\Psi(D^-, \Sigma)$ admits a legal integer solution. We show that there is a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and $T \models \Sigma$. By Theorem 4.1, there exists a finite data tree $T'$ such that $T'$ is valid w.r.t. $D^-$ and $T' \models \Sigma$. Thus by Theorem 3.3, there exists a finite data tree $T$ such that $T$ is valid w.r.t. $D$ and $T \models \Sigma$. ∎

**Proof of Theorem 4.3.** The proof is outlined in Section 4.2. In particular, a reduction from a variant of the linear integer programming (LIP) problem to the finite satisfiability problem for $L_u$ constraints is given there. It shows that the finite satisfiability problem is NP-hard since the variant of LIP is NP-complete [20, 23]. ∎

**Proof of Corollary 4.4.** The proof is given in Section 4.2.

**Proof of Corollary 4.5.** The proof is given in Section 4.2.

**Proof of Theorem 4.6.** It suffices to verify:

*Claim 1.* There exists a finite data tree valid w.r.t. $D = (E, A, P, R, r)$ iff $P(r)$ yields a finite tree.

*Claim 2.* Given an arbitrary DTD $D = (E, A, P, R, r)$, it can be decided in linear time whether $P(r)$ yields a finite tree.

We first show Claim 1. If $P(r)$ yields a finite tree, one can easily construct a finite data tree valid w.r.t. $D$. Conversely, suppose that there exists a finite data tree $T$ valid w.r.t. $D$. We define the *depth* of $T$ to be the number of nodes in the longest path from the root to a leaf of $T$. Since $T$ is finite, its depth is some natural number $n$. We show by induction on $n$ that $P(r)$ yields a finite tree.

*Basis:* $n = 1$. Since $T$ is valid w.r.t. $D$, $\epsilon$ must be in the regular language $L(P(r))$ defined by $P(r)$. By induction on the structure of a regular expression $\alpha$, we can verify that $\alpha$ yields a finite tree if $\epsilon \in L(\alpha)$.

*Inductive step:* Assume the statement when $n = m$. We show that it also holds for $n = m + 1$. Let $[v_1, ..., v_l]$ be the sub-elements of the root of $T$. Since $T$ is valid w.r.t. $D$, $lab(v_1) \dots lab(v_l)$ must be in $L(P(r))$. Observe that for each $i \in [1, l]$, the subtree of $T$ rooted at $v_i$ has a depth at most $m$. Given these, by induction on structure of $P(r)$, we can verify that $P(r)$ yields a finite tree.

We next show Claim 2. We use a boolean recursive function *finite-tree*$(\alpha)$ to check whether a regular expression $\alpha$ over $E \cup \{\mathtt{S}\}$ yields a finite tree. For each $\tau \in E$, we use two global boolean variables: *checked*$(\tau)$ to indicate whether $\tau$ has been checked by the function, and $finite(\tau)$ to indicate whether $\tau$ yields a finite tree. These are used to ensure that each $\tau$ in $E$ is checked at most once by the function. Initially, *checked*$(\tau)$ and $finite(\tau)$ have value $false$ for each $\tau \in E$.

```
function finite-tree(α)
  if α is one of ε, S or α1*
        then return true
  else if α = (α1, α2)
        then return (finite-tree(α1) and finite-tree(α2))
  else if α = (α1 | α2)
        then return (finite-tree(α1) or finite-tree(α2))
  else if α = τ in E
        then (1) if checked(τ)
                      then return finite(τ);
             (2) checked(τ) := true;
             (3) finite(τ)  := finite-tree(P(τ));
             (4) return finite(τ);
  else return false;
```

It is easy to verify that $P(r)$ yields a finite tree if and only if *finite-tree*$(P(r))$ returns true. In addition, the function call *finite-tree*$(P(r))$ takes $O(m)$ time, where $m$ is the size of the function $P$. This is because for each $\tau$ in $E$, $P(\tau)$ is checked at most once by the function. ∎

**Proof of Proposition 4.8.** Suppose that there exists a finite data tree $T_1 = (V, lab, ele, att, val, root)$ valid w.r.t. $D$. We construct another finite data tree $T_2$ by modifying the $val$ function in $T_1$ such that for any constraint $\tau.l \to \tau$ in $\Sigma$, $|ext(\tau)| = |ext(\tau.l)|$. More specifically, let $T_2 = (V, lab, ele, att, val', root)$. Observe that the only difference between $T_1$ and $T_2$ is the definition of the function $val'$. For any $v_1, v_2$ in $V$ with $lab(v_1) = \tau$ and $lab(v_2) = \tau$, we can make $val'(att(v_1, l)) \neq val'(att(v_2, l))$. Let $val'(v) = val(v)$ for all other vertices in $V$. Thus $T_2 \models C_\Sigma$. By Theorem 3.1, $T_2 \models \Sigma$. In addition, it is easy to verify that $T_2$ is valid w.r.t. $D$ since $T_1$ is valid w.r.t. $D$. The other direction is immediate. ∎

**Proof of Theorem 5.1.** We prove this by reduction from the finite implication problem for functional dependencies by functional and inclusion dependencies, which is known to be undecidable (see, e.g., [1] for a proof). Before we give the reduction, we first review functional and inclusion dependencies in relational databases. Let $\mathbf{R}$ be a relational schema. Functional dependencies (FDs) and inclusion dependencies (IDs) over $\mathbf{R}$ are defined as follows.

- *FD.* $R : X \to Y$, where $R \in \mathbf{R}$, and $X$ and $Y$ are subsets of attributes in $Att(R)$. An instance $\mathbf{I}$ of $\mathbf{R}$ *satisfies* the FD $\theta$, denoted by $\mathbf{I} \models \theta$, if

  $$\forall\, t_1\, t_2 \in I \;(\bigwedge_{l \in X} (t_1.l = t_2.l) \to \bigwedge_{l' \in Y} (t_1.l' = t_2.l')),$$

  where $I$ is the instance of $R$ in $\mathbf{I}$. Observe that keys are a special case of FDs in which $Y = Att(R)$.

- *ID.* $R[l_1, ..., l_k] \subseteq R'[l'_1, ..., l'_k]$, where $R, R' \in \mathbf{R}$, $[l_1, ..., l_k]$ is a sequence of attributes in $Att(R)$, and similarly, $[l'_1, ..., l'_k]$ is a sequence of attributes in $Att(R')$. It should be mentioned that in contrast to foreign keys, $l'_1, ..., l'_k$ are not necessarily a key for $R'$. An instance $\mathbf{I}$ of $\mathbf{R}$ *satisfies* the ID $\theta$, denoted by $\mathbf{I} \models \theta$, if

  $$\forall\, t_1 \in I\; \exists\, t_2 \in I' \;(\bigwedge_{1 \leq j \leq k} t_1.l_j = t_2.l'_j),$$

  where $I$ and $I'$ are the instances of $R$ and $R'$ in $\mathbf{I}$, respectively.

Let $\Sigma \cup \{\theta\}$ be a set of FDs and IDs over $\mathbf{R}$. We use $\Sigma \models \theta$ to denote that $\Sigma$ *finitely satisfies* $\theta$ as for keys and foreign keys. The *finite implication problem for FDs by FDs and IDs* is the problem to determine, given any relational schema $\mathbf{R}$, any set $\Sigma$ of FDs and IDs over $\mathbf{R}$ and a FD $\theta$ over $\mathbf{R}$, whether $\Sigma \models \theta$. This is a well-known undecidable problem [1].

We encode FDs and IDs in terms of keys and foreign keys as follows.

(1) FD $\theta = R : X \to Y$.

Let $Z$ be a key for $R$, i.e., $R[Z] \to R$ (note that every relation schema has a key). We define a new (fresh) relation schema $R_{new}$ such that $Att(R_{new}) = XYZ$, i.e., the union of $X$, $Y$ and $Z$. Observe that $XYZ$ is a key for both $R_{new}$ and $R$. We encode $\theta$ with two key constraints $\phi_1$ and $\phi_4$, and two foreign key constraints $\phi_2$ and $\phi_3$:

$$\begin{aligned} \phi_1 &= R_{new}[X] \to R_{new} \\ \phi_2 &= R[XY] \subseteq R_{new}[XY] \\ \phi_3 &= R_{new}[XYZ] \subseteq R[XYZ] \\ \phi_4 &= R_{new}[XY] \to R_{new} \end{aligned}$$

(2) ID $\theta = R_1[X] \subseteq R_2[Y]$.

Let $Z$ be a key for $R_2$, i.e., $R_2[Z] \to R_2$. We define a new relation schema $R_{new}$ such that $Att(R_{new}) = YZ$. Observe that $YZ$ is a key for $R_2$. We encode $\theta$ with a key $\phi_1$ and two foreign keys $\phi_2$ and $\phi_3$:

$$\begin{aligned} \phi_1 &= R_{new}[Y] \to R_{new} \\ \phi_2 &= R_1[X] \subseteq R_{new}[Y] \\ \phi_3 &= R_{new}[YZ] \subseteq R_2[YZ] \end{aligned}$$

We next show that the encoding is indeed a reduction from the finite implication problem for FDs by FDs and IDs to the finite implication problem for keys by keys and foreign keys. Given a relational schema $\mathbf{R}$, a set $\Sigma$ of FDs and IDs over $\mathbf{R}$, and a FD $\theta = R_\theta : X \to Y$ over $\mathbf{R}$, as described above we encode $\Sigma$ with a set $\Sigma_1$ of keys and foreign keys, and encode $\theta$ with

$$\begin{aligned} \phi_1 &= R^\theta_{new}[X] \to R^\theta_{new}, \\ \phi_2 &= R_\theta[XY] \subseteq R^\theta_{new}[XY], \\ \phi_3 &= R^\theta_{new}[XYZ] \subseteq R_\theta[XYZ], \\ \phi_4 &= R^\theta_{new}[XY] \to R^\theta_{new}. \end{aligned}$$

Let $\Sigma' = \Sigma \cup \{\phi_2, \phi_3, \phi_4\}$. It suffices to show that

$$\Sigma \models \theta \quad \text{iff} \quad \Sigma' \models \phi_1.$$

Let **R'** be the relational schema that includes all relation schemas in $\mathbf{R}$ as well as new relations created in the encoding. We show the claim as follows.

(1) Suppose that there is a finite instance $\mathbf{I}$ of $\mathbf{R}$ such that $\mathbf{I} \models \bigwedge \Sigma \wedge \neg\theta$. We show that there is a finite instance **I'** of **R'** such that **I'** $\models \bigwedge \Sigma' \wedge \neg\phi_1$ We construct **I'** such that for any $R$ in $\mathbf{R}$, the instance of $R$ in **I'** is the same as the instance of $R$ in $\mathbf{I}$. We populate instances of new relations $R_{new}$ created in the encoding as follows.

- If $R_{new}$ is introduced in the encoding of a FD $R : X \to Y$ as described above, then we let the instance $I_{new}$ of $R_{new}$ in **I'** be a subset of $\Pi_{XYZ}(I)$ such that $\Pi_{XY}(I) = \Pi_{XY}(I_{new})$ and moreover, $I_{new} \models R_{new}[XY] \to R_{new}$, where $I$ is the instance of $R$ in $\mathbf{I}$ and $\Pi_W(I)$ is the projection of $I$ on attributes $W$.
- If $R_{new}$ is introduced in the encoding of an ID $R_1[X] \subseteq R_2[Y]$ as described above, then let the instance $I_{new}$ of $R_{new}$ in **I'** be a subset of $\Pi_{YZ}(I_2)$ such that $\Pi_Y(I_2) = \Pi_Y(I_{new})$ and in addition, $I_{new} \models R_{new}[Y] \to R_{new}$, where $I_2$ is the instance of $R_2$ in $\mathbf{I}$.

It is easy to verify that **I'** $\models \bigwedge \Sigma' \wedge \neg\phi_1$.

(2) Suppose that there is a finite instance **I'** of **R'** such that **I'** $\models \bigwedge \Sigma' \wedge \neg\phi_1$. We construct a finite instance $\mathbf{I}$ of $\mathbf{R}$ by removing from **I'** all instances of new relations introduced in the encoding. It is easy to verify that $\mathbf{I} \models \bigwedge \Sigma \wedge \neg\theta$.

Therefore, the encoding is indeed a reduction from the finite implication problem for FDs by FDs and IDs. This shows that the finite implication problem for keys by keys and foreign keys is undecidable. ■

**Proof of Theorem 5.2.** A proof is given in Section 5.

**Proof of Corollary 5.3.** Given any DTD $D$ and any set $\Sigma$ of keys in $L$ over $D$, it suffices to show that $\Sigma$ can be satisfied by a finite data tree valid w.r.t. $D$ if and only if there exists a finite data tree valid w.r.t. $D$. For if it holds, then the corollary follows immediately from Theorem 4.6.

We now show the claim. Suppose that there exists a finite data tree $T_1 = (V, lab, ele, att, val, root)$ valid w.r.t. $D$. We construct another finite data tree $T_2$ by modifying the $val$ function in $T_1$ such that for any constraint $\tau[X] \to \tau$ in $\Sigma$, $|ext(\tau)| = |ext(\tau.l)|$ for every $l \in X$. That is, $T_2 \models \tau.l \to \tau$ for all $l \in X$. More specifically, let $T_2 = (V, lab, ele, att, val', root)$. Observe that the only difference between $T_1$ and $T_2$ is the definition of the function $val'$. For any $v_1, v_2$ in $V$ with $lab(v_1) = \tau$ and $lab(v_2) = \tau$, we can make

$val'(att(v_1, l)) \neq val'(att(v_2, l))$ for any $l \in X$. Let $val'(v) = val(v)$ for all other vertices in $V$. It is easy to verify that $T_2$ is valid w.r.t. $D$ since $T_1$ is valid w.r.t. $D$. In addition, $T_2 \models \tau[X] \rightarrow \tau$ since for any $x, y \in ext(\tau)$, $x[X] \neq y[X]$. The other direction is immediate. ■