Integrity Constraints for XML

Wenfei Fan
Bell Laboratories

E-mail: wenfei@research.bell-labs.com

and

Jérôme Siméon
Bell Laboratories

E-mail: simeon@research.bell-labs.com

Integrity constraints have proved fundamentally important in database management. The ID/IDREF mechanism provided by XML DTDs relies on a simple form of constraints to describe references. Yet, this mechanism is sufficient neither for specifying references in XML documents, nor for expressing semantic constraints commonly found in databases. In this paper, we extend XML DTDs with several classes of integrity constraints and investigate the complexity of reasoning about these constraints. The constraints range over keys, foreign keys, inverse constraints as well as ID constraints for capturing the semantics of object identities. They improve semantic specifications and provide a better reference mechanism for native XML applications. They are also useful in information exchange and data integration for preserving the semantics of data originating in relational and object-oriented databases. We establish complexity and axiomatization results for the (finite) implication problems associated with these constraints. In addition, we study implication of more general constraints, such as functional, inclusion and inverse constraints defined in terms of navigation paths.

Key Words: keys, foreign keys, inverse constraints, constraint implication, XML.

1. INTRODUCTION

XML (eXtensible Markup Language [9]) has emerged as the standard for information exchange between Web applications. It offers a convenient syntax for representing data from heterogeneous sources, but provides little semantic information. To specify the semantics of XML data, a variety of approaches have been proposed: type systems [6, 18, 21, 31, 35], description logics [14], meta-data descriptions [30], etc. As some of these proposals [21, 31, 35] point out, integrity constraints are important for semantic specifications of XML data. In addition, they are useful for query optimization [23, 26], update anomaly prevention [2], and for information preservation in data integration [1, 18]. Integrity constraints are also used to model references in relational databases, through keys and foreign keys.

Integrity constraints are traditionally part of a schema specification. Document Type Definitions [9] (DTDs) offer the so-called ID and IDREF attributes to identify and reference an element within an XML document, in a way similar to relational keys and foreign keys. However, ID and IDREF attributes are not expressive enough to capture semantic constraints such as keys, foreign keys and inverse constraints commonly found in databases, or to model object-style references. XML Schema [35] supports a complex form of keys and foreign keys defined with XPath [16] expressions. However, XPath is rather complex and as a result, reasoning about constraints defined with XPath is highly intricate, if not impossible.

In response to these problems, we propose several constraint languages for XML that are both expressive enough for practical applications and simple enough to allow efficient reasoning. More specifically, we make the following contributions:

- We introduce a model for XML data with schema and integrity constraints. We propose L, L_{id} and L_u, three basic constraint languages that provide both a reference mechanism and better semantic specifications. Language L_u is a minimal extension of the original ID/IDREF mechanism. Constraints of L_{id} and L are to capture semantic constraints when data originates in object-oriented and relational databases, respectively.
- We study the implication and finite implication problems for these three languages. For each language, we provide complexity results and axiomatization when one exists. The results for L extend relational dependency theory. Notably, the implication and finite implication problems for arbitrary keys and foreign keys are shown to be undecidable, but they become decidable when only primary keys and foreign keys are considered.
- We investigate implication of more general forms of constraints, including functional, inclusion and inverse constraints defined in terms of navigation paths, by basic constraints of L_{id}. Such path constraints have a variety of practical applications, ranging from query optimization to verification of the correctness of integration/transformation programs.

As this work is motivated by the need for integrity constraints in practical XML applications, we first illustrate several important application contexts and the limitations of the current ID/IDREF mechanism.

The ID/IDREF mechanism, constraints and references. As a concrete example, consider an XML document about books:

```
<?XML version = "1.0">
<bib>
  <book>
    <entry isbn="1-55860-622-X">
      <title>Data on the Web</title>
      <publisher>Morgan Kaufmann</publisher>
    </entry>
    <author>Serge Abiteboul</author>
    <author>Peter Buneman</author>
    <author>Dan Suciu</author>
    <section sid="1">
      <title>Introduction</title>
      ...
     <section sid="11"><title>Audience</title>...</section>
    ...
    <ref to="0-201-53771-0 1-55860-463-4"/>
  </book>
  <book>
    <entry isbn="0-201-53771-0">
      <title>Foundations of Databases</title>
      <publisher>Addison Wesley</publisher>
    </entry>
    <author>Serge Abiteboul</author>
    <author>Richard Hull</author>
    <author>Victor Vianu</author>
    ...
  </book>
</bib>
```

For each book, its isbn, title and publisher are given in an entry element, followed by a list of author elements, the book content and a set of bibliographical references (in a ref element). This document conforms to the following DTD:

```
<!ELEMENT book      (entry, author*, section*, ref)>
<!ELEMENT entry     (title, publisher)>
<!ATTLIST entry
          isbn      ID      #required>
<!ELEMENT section   (title, (#PCDATA|section)*)>
<!ATTLIST section
          sid       ID      #required>
<!ELEMENT ref       EMPTY>
<!ATTLIST ref
          to        IDREFS  #implied>
```

A DTD defines element types and attributes for each element. We omit the descriptions of the elements whose type is string (e.g., #PCDATA in XML). An ID annotation indicates that the corresponding attribute should uniquely identify an element in the entire document, i.e., it is unique among all ID attributes. An IDREF(S) annotation indicates a reference, i.e., it should contain a (set of) value(s) of some ID attribute(s) present in the document.

Observe that the ID/IDREF mechanism is similar to both the object-identity based notion of references from object-oriented databases [3] and to keys/foreign keys from relational databases. On the one hand, like object identifiers, ID attributes uniquely identify elements within the whole document. On the other hand, as XML has a textual format, the reference semantics is achieved with implicit constraints that must hold on attribute values, in the spirit of relational keys and foreign keys. Yet, it captures neither the complete semantics of relational keys and foreign keys nor that of object-style references. For instance, isbn should be a key for entry. Its representation as an ID attribute indeed makes it unique, but among all the ID attributes in the document. This is too strong an assumption, preventing other elements, e.g., books, from using the same isbn number as a key. Worse still, the scope and type of an ID/IDREF attribute are not clear. The to attribute, for instance, could contain a reference to a section or an author element. One has no control over what an IDREF reference points to. Obviously, we would like to constrain such references to entry elements only.

We can resolve these problems by changing slightly the constraints on the attributes involved. More specifically, we can (i) treat `isbn` (resp. `sid`) attribute as a key for `entry` (resp. `section`) elements, which uniquely identifies an element among the elements of `entry` (resp. `section`), as opposed to all elements in the entire document; (ii) add an inclusion constraint as part of a foreign key, asserting that `ref.to` is a subset of `entry.isbn`, where $\tau.l$ stands for the set of l attribute values of all τ elements in a document. That is, for any `ref` element x and each value v of the `to` attribute of x, there is an `entry` element y such that v matches the `isbn` attribute value of y. Observe that `isbn` is a key of `entry` and thus the `to` attribute is a foreign key of `ref` that references `entry` elements. These constraints can be expressed in our language L_u.

Capturing the semantics of legacy repositories. XML is mainly used for data exchange. As a consequence, a large amount of XML data originates in relational or object-oriented databases, for which keys, foreign keys and inverse relationships [2, 15] convey a fundamental part of the information. Consider, for instance, the following object-oriented schema (in ODL syntax [15]):

```
class Person
(key  name)
      { attribute String name;
        attribute String address;
        relationship set<Dept> in_dept
             inverse  Dept::has_staff; }
class Dept
(key  dname)
      { attribute String dname;
        attribute Person manager;
        relationship set<Person> has_staff
             inverse  Person::in_dept; }
```

On top of the structure specified by the schema, we have the following: (1) `name` and `dname` are keys for the `Person` and `Dept` classes respectively, and (2) there is an inverse relationship between `Person.in_dept` and `Dept.has_staff`. That is, (1) no distinct `Person` objects can have the same `name` attribute value; similarly for `dname` of `Dept`; and (2) for any `Person` object p and `Dept` object d, if d is one of the departments in which p works, (i.e., d is in the (set) attribute `in_dept` of p), then p is a staff of d (i.e., p is in the set `has_staff` of d), and vice versa.

When exporting this object-oriented database to XML, the following DTD could be generated, in an attempt to preserve the semantics of the original schema:

```
<!ELEMENT db (person*, dept*)>
<!ELEMENT person (name, address)
<!ATTLIST person
          oid         ID       #required
          in_dept     IDREFS   #implied>
<!ELEMENT dept (dname)>
<!ATTLIST dept
          oid         ID       #required
          manager     IDREF    #required
          has_staff   IDREFS   #implied>
```

Here the original ID semantics is appropriate to capture the notion of object identifiers [3] (the `oid` attributes). However, references through IDREF are rather weak: as IDREF attributes are "untyped", we no longer know that the `in_dept` attribute of a `person` element should reference a department (a `dept` element). In addition, as in the previous example, keys are not precisely captured (here we cannot even specify `name` and `dname` with ID attributes as XML only allows a single ID attribute for each element type). Furthermore, we have no way to express the inverse relationships in a DTD. One wants to overcome these limitations while preserving the semantics of the original notion of object identities. To do so, we specify the following constraints: (i) `name` and `dname` as keys for persons and departments, in addition to `oid` as an ID constraint to capture the original semantics of ID attributes; (ii) foreign keys to assert that `person.in_dept` (resp. `dept.has_staff`) refers to departments (resp. persons) only; (iii) an inverse constraint between `person.in_dept` and `dept.has_staff`. These can be expressed in our language L_{id}.

As another example, consider a DTD which is translated from a relational schema:

```
<!ELEMENT university (student*, course*, enroll*)>
<!ELEMENT student    (SSN, name, GPA)>
<!ELEMENT course     (dept, number, credit)>
<!ELEMENT enroll     (sid, dept, number, grade)>
```

We would like to capture the following semantic constraints of the relational database: (i) `SSN` is a key of `student`, $\{$ `dept`, `number`$\}$ is a key of `course`, $\{$ `sid`, `dept`, `number`$\}$ is a key of `enroll`; (ii) `sid` of `enroll` is a foreign key referencing (`SSN` of) `student`, and (`dept`, `number`) is a foreign key of `enroll` referencing `course`. That is, one wants to express typical relational keys and foreign keys by means of multi-attributes, as well as constraints on certain sub-elements. These can be specified in our language L.

Implication problems for XML constraints and related work. A key question [2] in connection with integrity constraints for XML is about their (finite) implication: given any finite set Σ of constraints, is it the case that any (finite) document that satisfies Σ must also satisfy some other constraint? The (finite) implication problem is important if we want to reason about XML constraints as we do in databases. Among other things, it is useful in query optimization and data integration [26, 33].

There is a large body of work on integrity constraints and their associated (finite) implication problems in the relational context [2, 36] that we can try to exploit. As ID and IDREF attributes are unary, the work on unary inclusion and functional dependencies by Cosmadakis, Kanellakis and Vardi [20] is particularly relevant. However, because of the semantics of ID attributes, results from [20] are not directly applicable in the XML context. In addition, because different notions of equality are used to define relational and XML keys, the analysis of key constraints for XML is a little different from their relational counterpart. More specifically, relational keys are simply defined with equality on values, whereas keys for XML are defined in terms of two notions of equality, namely, *value equality* and *node identity*. Recall that an XML document is typically modeled as a node-labeled tree. In such a tree distinct nodes may have the same "value", i.e., node identity and value equality are different notions. In contrast, it is impossible for distinct tuples in a relation to have the same value. By saying, e.g., that l is a key of τ elements in an XML document, it means that for any τ elements x and y, if they agree on their l attribute values (value equality), then x and y are the same node (node identity). This complicates the analysis of XML constraints. Another difference is due to the more complex structure of XML documents, for instance the presence of set-valued attributes (see the IDREFS attribute in our first example). Our results address these issues. As language L is designed to capture semantic constraints from relational databases, our results for implication and finite implication of general/primary keys and foreign keys of L also hold in the relational setting, which, to the best of our knowledge, have not been addressed before.

XML documents can have arbitrarily nested structures (note that the `section` elements in the `book` DTD have a recursive definition). It is therefore natural to consider both (unrestricted) implication and finite implication problems. This also highlights the importance of path constraints in this context. As an example, we would like to know that `isbn` is not only a key for `entry`, but also a key for the outer `book` elements. This never occurs in the relational setting.

There have been a number of proposals for adding constraints to XML, e.g., XML Schema [35], XML Data [31] and a recent proposal [10]. These proposals consider keys and foreign keys for XML, but stop short of addressing inverse constraints, which are common in object-oriented databases [15]. In addition, the implication and finite implication problems for the constraints of [35, 31] are, as far as the authors are aware, unresolved. Recently implication of XML keys of [10] was studied in [11], but in the absence of foreign keys and other constraints that may interact with keys in a nontrivial way. Most of these constraint languages are defined in terms of regular path expressions or XPath [16]. While these powerful path expressions yield expressive constraint languages, they complicate the analysis of implication and finite implication of these constraints. The high complexity of reasoning about these constraints may compromise their practical applicability. Finally, most of these constraints are defined in a rich schema definition language for XML. In other words, they can only be embedded within some types for XML documents. In practice, however, XML documents are commonly specified with DTDs instead of those complicated type systems. These considerations suggest us to adopt a different approach: we consider XML constraints that are both powerful enough to express important database constraints and are simple enough to be reasoned about efficiently, and we add these constraints as a minimum extension to XML DTDs. We investigate more expressive constraints that can be derived from by basic XML constraints, such as constraints defined in terms of navigation paths.

There has been work on the study of path constraints. The path constraint languages introduced in [4, 13] specify inclusions among certain sets of objects, and were studied for semistructured data and XML. They are generalizations of (unary) inclusion dependencies. Inverse constrains are also expressible in the languages of [13]. However, these languages cannot express keys. Generalizations of functional dependencies have also been studied [28, 29]. These constraints are capable of expressing neither foreign keys nor inverse constraints. Furthermore, they were studied in the database context (structured data).

Finally, we address the connection between our XML constraints and bounded variable logics, in particular, two-variable first-order logic (FO^2). FO^2 is the fragment of first-order logic consisting of all relational sentences with at most two distinct variables [27]. Many constraints considered here are not expressible in FO^2, including foreign keys of L, inverse constraints of L_{id}, and (unary) keys of all the three languages. These can be verified by using the 2-pebble Ehrenfeucht-Fraïssé (EF) style game [5]. As shown by Borgida [8], FO^2 has equivalent expressive power as the language $DL \setminus \{trans, compose, at\_least, at\_most\}$, i.e., description logic omitting the transitive closure and composition constructors as well as counting quantifiers. As an immediate result, many XML constraints considered in this paper are not expressible in $DL \setminus \{trans, compose, at\_least, at\_most\}$. [8] has also shown that description logic with the composition constructor, i.e., $DL \setminus \{trans, at\_least, at\_most\}$, is equally expressive as FO^3, the fragment of first-order logic with at most three distinct variables and with monadic and binary relations. It is known that FO^3 possesses undecidable (finite) implication problems [7]. In contrast, we shall show that most of the implication and finite implication problems associated with our constraint languages are decidable. Therefore, results about description logics are not much of help for studying implication of our constraints.

Organization. The rest of the paper is organized as follows. Section 2 presents an XML data model with schema and constraints, defines the languages L, L_{id} and L_u, and shows how they capture constraints in practice. Section 3 investigates implication, finite implication and axiomatization of these constraints. Section 4 introduces path constraints, i.e., constraints defined in terms of navigation paths, and studies implication of path constraints by basic constraints of L_{id}. Section 5 identifies directions for further work.

2. CAPTURING XML DOCUMENT SEMANTICS

In this section, we present a data model and a formalization of DTDs [9]. The data model represents the content of XML documents, and DTDs specify the syntactic structure and the semantics of the data. A DTD is formalized as a combination of a structural specification and a set of integrity constraints. We shall use several classes of constraints to specify various extensions over DTD structures.

2.1. Documents

We begin with the data model for representing XML documents. We assume the existence of three pairwise disjoint sets: **E** of element names, **A** of attribute names, and **Str** of string values. We assume that all atomic values are of the string type, denoted by S_τ. We also assume an infinite set **V** of vertices. Given a set X, we use $F(X)$ and $P(X)$ to denote the set of all lists built over elements of X and the power-set of X, respectively. We represent XML documents as ordered annotated trees with labels on the nodes.

DEFINITION 2.1. A *data tree* is denoted by $(V, elem, att, root)$, where

- V is a set of vertices (nodes), i.e., a subset of **V**;
- $elem$ is a mapping from vertices to their labels and children, i.e., a function from V to **E** $\times$ $F($**Str** $\cup V)$; for any node v' of V that occurs in $elem(v)$, v' is called a *child* of v and v is called the *parent node of* v'; we say that there is a *parent-child edge* from v to v';
- att is a partial function from vertices and attribute names to a set of atomic values, i.e., from $V \times$ **A** to $P($**Str**$)$;
- $root$ is a distinguished element of V, called the root of the tree.

A data tree has a tree structure. More specifically, for any $v \in V$, there is a unique path of parent-child edges from $root$ to v. A data tree is *finite* if V is finite.

Intuitively, V is the set of (internal) nodes of the data tree that represent elements. The function $elem$ defines for each node its label (element name) and its list of children (either string values or sub-elements). The string and sub-element children of the node are ordered. The function att defines the attributes of each node. In XML, the attributes of an element

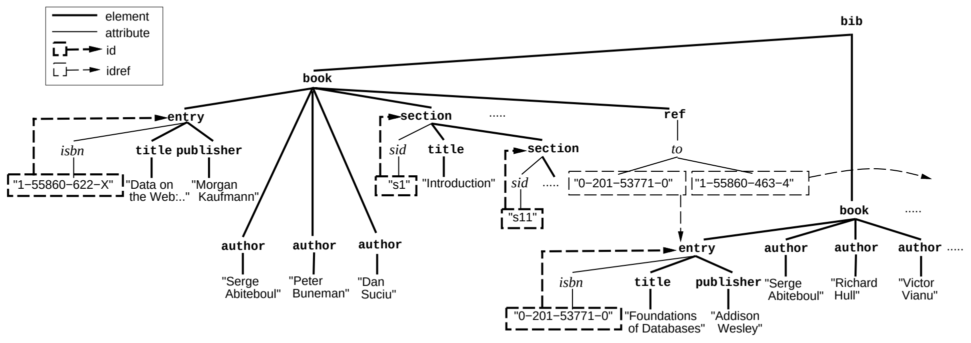

FIG. 1. Graph representation of an XML document

are unordered and each contains a set of atomic values. A node v in V is called a *text node* if $elem(v) = (\tau, [s])$, where τ is an element name and s is a string. To model XML precisely, we assume that $root$ and text nodes do not have attributes.

We shall use DTDs to specify the structure and semantics of data trees. Figure 1 shows a data tree representing our book document given in Section 1. Note that the indications about the semantics of ID/IDREF attributes assume that the corresponding DTD is available.

We shall use the following notations. For any $\tau \in E$, we use $ext(\tau)$ to denote the set of nodes labeled τ in V. For any $x \in V$ and $l \in \mathbf{A}$, we use $x.l$ to indicate $att(x, l)$, i.e., the value of the attribute l of x. We define $ext(\tau).l$ to be $\bigcup_{x \in ext(\tau)} x.l$. Furthermore, let $[X]$ be a list of attributes $[l_1, ..., l_n]$. We use $x[X]$ to denote $[x.l_1, ..., x.l_n]$. For any finite set S, we use $|S|$ to denote the cardinality of S.

2.2. Document Type Definitions

We extend DTDs with constraints to capture the semantics of XML documents. We first describe the structural specifications, and then introduce the constraint languages.

Document Structure

In the literature [6, 14, 32], DTDs are often modeled as *Extended Context Free Grammars* (*ECFGs*), with elements as non-terminals, basic XML types as terminals and element definitions specified in terms of regular expressions as production rules. While ECFGs can specify the syntactic structure of elements, they fail to describe attributes, notably the ID/IDREF mechanism. We extend ECFGs [14] to specify attributes.

DEFINITION 2.2. A *DTD structure* is denoted by $S = (E, P, R, kind, r)$, where:

- E is a finite set of *element types* in **E**, ranged over by τ;
- P is a function from element types to *element type definitions*: for any $\tau \in E$, $P(\tau) = \alpha$, where α is a regular expression, defined as follows:

$$\alpha \ ::= \ S_\tau \ | \ \tau \ | \ \epsilon \ | \ \alpha + \alpha \ | \ \alpha , \alpha \ | \ \alpha^*$$

where S_τ is the (string) type of atomic values given above, $\tau \in E$, ϵ denotes the empty word, "$+$" stands for union, "," for concatenation, and "$*$" for the Kleene closure.

- R is a partial function from $E \times \mathbf{A}$ to *attribute type definitions*: $R(\tau, l) = \beta$, where τ is an element name in E, l is an attribute in **A** and β is either S_τ or S_τ^*.

We write $Att(\tau)$ for the set of attributes of τ, i.e., $\{l \in \mathbf{A} \mid R(\tau, l)$ is defined$\}$. An attribute l is called *set-valued* if $R(\tau, l) = S_\tau^*$, and *singled-valued* otherwise.

• $kind$ is a partial function from $E \times \mathbf{A}$ to $\{$ ID, IDREF $\}$, identifying the ID and IDREF attributes.

We assume that for any $\tau \in E$ and $l \in \mathbf{A}$, if $kind(\tau, l)$ is defined then so is $R(\tau, l)$. Moreover, there exists at most one attribute l_o such that $kind(\tau, l_o) = ID$. In addition, l_o must be single-valued. We use $\tau.id$ to denote the ID attribute $\tau.l_o$, when it exists.

• $r \in E$ is the element type of the root.

Without loss of generality, we assume that r does not occur in $P(\tau)$ for any $\tau \in E$. In addition, we assume that for each $\tau \in E \setminus \{r\}$, τ is *connected to* r, i.e., either τ occurs in $P(r)$, or it occurs in $P(\tau')$ for some τ' that is connected to r.

A DTD structure S specifies the syntactic structure of a document. That is, it imposes the following syntactic restrictions on a data tree: (1) there is a unique node, i.e., the root of the tree, labeled with r; (2) string values are labeled with S_τ; (3) for any element type τ, the regular grammar $P(\tau)$ restricts the children of each τ element v; that is, the labels of the children of v must be in the regular language defined by $P(\tau)$; and (4) a τ element has an attribute l if and only if $R(\tau, l)$ is defined in S. More precisely, these are described as follows.

DEFINITION 2.3. A data tree $(V, elem, att, root)$ *conforms to* a DTD structure $(E, P, R, kind, r)$ iff there is a mapping $\mu : V \cup \mathbf{Str} \to E \cup \{S_\tau\}$ such that:

• $\mu(root) = r$,

• for any s in **Str**, $\mu(s) = S_\tau$,

• for any $v \in V$, if $elem(v) = (\tau, [v_1, \ldots, v_n])$, then $\tau = \mu(v)$ and $[\mu(v_1), \ldots, \mu(v_n)]$ is in the regular language defined by $P(\tau)$,

• for any $v \in V$ and $l \in \mathbf{A}$, $att(v, l)$ is defined if and only if $R(\mu(v), l)$ is defined. Moreover, if l is a single-valued attribute of τ, then $att(v, l)$ must be a singleton set.

Document Constraints

Next, we introduce our three constraint languages.

Language L. The first language, L, intends to capture integrity constraints from relational databases. It defines the classical key and foreign key constraints [2]. Over a DTD structure $S = (E, P, R, kind, r)$, a constraint φ of L has one of the following forms:

• *Key:* $\tau[X] \to \tau$, where $\tau \in E$ and X is a *set* of single-valued attributes in $Att(\tau)$. A data tree G *satisfies* the key, denoted by $G \models \varphi$, iff in G,

$$\forall\, x\, y \in ext(\tau)\ (\bigwedge_{l \in X} (x.l = y.l) \to x = y).$$

• *Foreign key:* $\tau[X] \subseteq \tau'[Y]$ and $\tau'[Y] \to \tau'$, where $\tau, \tau' \in E$, X, Y are nonempty *lists* of single-valued attributes in $Att(\tau)$ and $Att(\tau')$, respectively, and X and Y have the same length. We write $G \models \varphi$ iff $G \models \tau'[Y] \to \tau'$ and in addition,

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

That is, $\tau[X] \to \tau$ indicates that X is a key of τ elements, i.e., the X-attribute value of a τ element v uniquely identifies v among all the elements in $ext(\tau)$. A foreign key is a combination of two constraints, namely, $\tau[X] \subseteq \tau'[Y]$ called an *inclusion constraint*, and a key $\tau'[Y] \to \tau'$. It indicates that X is a foreign key of τ elements referencing the key Y of τ' elements. Note that Y in $\tau'[Y] \to \tau'$ is treated as a set.

Observe that two notions of equality are used to define keys: string value equality is assumed when comparing attributes $x.l$ and $y.l$, and node equality when comparing elements x and y, i.e., $x = y$ if and only if x and y are the same node.

Language L_u. The purpose of language L_u is to provide a minimal extension of DTDs that supports keys and references. In L, a key may be composed of several attributes. In XML, references are always *unary*, i.e., via a single attribute. In addition, XML supports IDREFS attributes, that is, attributes that are set-valued. To be as close to the XML standard as possible, we consider L constraints in which the lists (sets) X, Y consist of a single attribute. We refer to such constraints as *unary* constraints, and write $x[l]$ as $x.l$. Moreover, we study set-valued foreign keys. As observed in the last section, we need to improve the current reference mechanism of XML with typing and scoping. Thus in contrast to the semantics of ID attributes, we assume that a key of an element is unique among elements of the same type, rather than within the entire document.

Based on these considerations we define L_u to contain unary constraints of L as well as set-valued foreign key constraints. More specifically, a constraint of L_u has one of the following forms:

- *Unary key constraint of* L*:* $\tau.l \to \tau$.
- *Unary foreign key constraint of* L*:* $\tau.l \subseteq \tau'.l'$ and $\tau'.l' \to \tau'$.
- *Set-valued foreign key constraint:* $\tau.l \subseteq_S \tau'.l'$ and $\tau'.l' \to \tau'$, where $\tau, \tau' \in E$, l is a set-valued attribute of τ and l' is a single-valued (key) attribute of τ'. A data tree G satisfies the foreign key iff $G \models \tau'.l' \to \tau'$ and in addition,

$$\forall\, x \in ext(\tau)\ (x.l \subseteq ext(\tau').l').$$

A constraint of the form $\tau.l \subseteq_S \tau'.l'$ is called a *set-valued inclusion constraint*.

A set-valued inclusion constraint $\tau.l \subseteq_S \tau'.l'$ asserts that for any τ element x, each value in the set-valued attribute l of x matches the single-valued l' attribute of some τ' element.

Constraints of L_u provide a simple reference mechanism for XML that overcomes the limitations of the original ID/IDREF mechanism by adopting the semantics of the relational key/foreign key mechanism.

It should be noted that the $kind$ function of the DTD structure is not used when defining L and L_u constraints. More specifically, in L and L_u, keys are not necessarily ID attributes and likewise, foreign keys do not have to be IDREF attributes.

Language L_{id}. Finally, we want a language that preserves the semantic of object identifiers of object-oriented databases. To do so, we keep the original semantics of ID attributes, whose value is unique within the whole document. Yet, we want to add key and inverse constraints. To capture these, we define the language L_{id} that consists of constraints of the following forms:

- *Unary key constraint of* L*:* $\tau.l \to \tau$.

- *ID constraint:* $\tau.id \to_{id} \tau$, where $\tau \in E$, $id \in Att(\tau)$ and $kind(\tau, id) = ID$. A data tree G satisfies the ID constraint iff in G,

$$\forall\, x \in ext(\tau)\ \exists\, s \in \mathbf{Str}\ (x.id\ = s\ \wedge\ \forall\, y\, (y.id = s \to x = y)).$$

- *Foreign key constraint:* $\tau.l \subseteq \tau'.id$ and $\tau'.id \to_{id} \tau'$, where $\tau, \tau' \in E$, l is a single-valued attribute of τ and $kind(\tau, l) = IDREF$. A data tree G satisfies the foreign key iff it satisfies the ID constraint $\tau'.id \to_{id} \tau'$ and in addition,

$$\forall\, x \in ext(\tau)\ (x.l \in ext(\tau').id).$$

- *Set-valued foreign key constraint:* $\tau.l \subseteq_S \tau'.id$ and $\tau'.id \to_{id} \tau'$, where $\tau, \tau' \in E$, l is a set-valued attribute of τ and $kind(\tau, l) = IDREF$. A data tree G satisfies the set-valued foreign key iff it satisfies the ID constraint $\tau'.id \to_{id} \tau'$ and in addition,

$$\forall\, x \in ext(\tau)\ (x.l \subseteq ext(\tau').id).$$

- *Inverse constraint:* $\tau.l \rightleftharpoons \tau'.l'$, where $\tau, \tau' \in E$, l, l' are set-valued attributes of τ, τ' respectively, $kind(\tau, l) = kind(\tau', l') = IDREF$, and moreover, τ and τ' have ID attributes, i.e., $\tau.id \to_{id} \tau$ and $\tau'.id \to_{id} \tau'$. It asserts that there is an inverse relationship between l and l'. A data tree G satisfies the inverse constraint iff it satisfies the two ID constraints, two set-valued inclusion constraints: $\tau.l \subseteq_S \tau'.id$ and $\tau'.l' \subseteq_S \tau.id$, and

$$\forall\, x \in ext(\tau)\, \forall\, y \in ext(\tau')\ (x.id \in y.l' \leftrightarrow y.id \in x.l).$$

Observe that two notions of equality are also used to define ID constraints: string value equality is assumed when comparing $x.id$ and $y.id$, and $x = y$ is true if and only if x and y are the same node. In a foreign key constraint $\tau.l \subseteq \tau'.id$ ($\tau.l \subseteq_S \tau'.id$) and $\tau'.id \to_{id} \tau'$ of L_{id}, the l attribute of τ refers to the ID attribute of τ'. An inverse constraint $\tau.l \rightleftharpoons \tau'.l'$ is actually a combination of three constraints, namely, two set-valued foreign keys and an inverse relationship specification, which asserts that for any τ element x and τ' element y, if a value in the set $y.l'$ references the id of x, then a value in $x.l$ references the id of y; and vice versa. Because each element type has at most one ID attribute, we assume that the ID attributes are known when specifying an inverse constraint of L_{id}.

Language L_{id} improves the original XML reference mechanism by imposing typing and scoping constraints on the attributes. It also supports inverse constraints and unary keys. In contrast to L_u and L, it implicitly uses the function $kind$ in the DTD structure to define its constraint.

From now on, we shall refer to the constraints from one of these three languages as the *basic XML constraints*. Observe that basic XML constraints are more complex than their relational and object-oriented counterparts.

We shall use the following notation. Let Σ be a set of constraints and G be a data tree. We write $G \models \Sigma$ if G satisfies all the constraints in Σ, i.e., for each $\phi \in \Sigma$, $G \models \phi$.

Finally, we are ready to define DTDs with constraints.

DEFINITION 2.4. A *Document Type Definition with constraints* (*DTD*^{C}) is defined to be $D = (S, \Sigma)$, where:

- S is a DTD structure,
- Σ is a set of basic XML constraints; that is, Σ is a set of C constraints, where C is either L_{id}, L_u or L.

Given a *DTD*^{C}, we define the notion of its valid documents as follows.

DEFINITION 2.5. A data tree G is *valid w.r.t. a* DTD^C (S, Σ) if and only if G conforms to the DTD structure S and $G \models \Sigma$.

2.3. Examples

We now reexamine the examples given in Section 1 and show how their semantics can be captured by a *DTD*^{C}, using the constraint languages L_{id}, L_u and L.

We start with the `book` document. To specify its structure, we define a *DTD*^{C} $D = ((E, P, R, kind, r), \Sigma)$ with constraints of Σ in L_u, as follows.

$$E = \{\ \texttt{book, entry, section, ref, author, title, publisher}\ \}$$

$$\begin{array}{lcl} P(\texttt{book}) & = & (\texttt{entry, author}^*\texttt{, section}^*\texttt{, ref}) \\ P(\texttt{entry}) & = & (\texttt{title, publisher}) \\ P(\texttt{section}) & = & (\texttt{title, } (S_\tau \texttt{ + section})^*) \\ P(\texttt{ref}) & = & \epsilon \\ P(\texttt{author}) & = & P(\texttt{title}) \ = \ P(\texttt{publisher}) \ = (S_\tau) \end{array}$$

$$\begin{array}{lcl} R(\texttt{entry}, \texttt{isbn}) & = & R(\texttt{section}, \texttt{sid}) \ = \ S_\tau \\ R(\texttt{ref}, \texttt{to}) & = & S_\tau^* \end{array}$$

$$r = \ \texttt{book}$$

$$\Sigma = \{\texttt{entry.isbn} \rightarrow \texttt{entry},\ \ \texttt{section.sid} \rightarrow \texttt{section},\ \ \texttt{ref.to} \subseteq_S \texttt{entry.isbn}\}$$

In Σ, `entry.isbn` $\rightarrow$ `entry` and `section.sid` $\rightarrow$ `section` are key constraints of L_u, `ref.to` $\subseteq_S$ `entry.isbn` is a set-valued inclusion constraint, and the inclusion constraint and the key `entry.isbn` $\rightarrow$ `entry` make up a set-valued foreign key of L_u. We can keep the function $kind$ empty as we do not use the original ID/IDREF semantics. Note also the use of a set-valued foreign key to capture the semantics of the set-valued `ref` attribute.

We next give a *DTD*^{C} with L_{id} constraints to describe the structure of our `person/dept` object-oriented database: $D_o = ((E_o, P_o, R_o, kind_o, r_o), \Sigma_o)$, where

$$E_o \ = \{\texttt{db, person, dept, name, address, dname}\}$$

$$\begin{array}{lcl} P_o(\texttt{db}) & = & (\texttt{person}^*\texttt{, dept}^*) \\ P_o(\texttt{person}) & = & (\texttt{name, address}) \\ P_o(\texttt{dept}) & = & (\texttt{dname}) \\ P_o(\texttt{name}) & = & P_o(\texttt{dname}) \ = \ P_o(\texttt{address}) \ = \ (S_\tau) \end{array}$$

$$\begin{array}{lcl} R_o(\texttt{person}, \texttt{oid}) & = & R_o(\texttt{dept}, \texttt{oid}) \ = \ R_o(\texttt{dept}, \texttt{manager}) \ = S_\tau \\ R_o(\texttt{person}, \texttt{in\_dept}) & = & R_o(\texttt{dept}, \texttt{has\_staff}) \ = \ S_\tau^* \end{array}$$

$$\begin{array}{lcl} kind_o(\texttt{person}, \texttt{oid}) & = & kind_o(\texttt{dept}, \texttt{oid}) \ = \texttt{ID} \\ kind_o(\texttt{person}, \texttt{in\_dept}) & = & kind_o(\texttt{dept}, \texttt{manager}) \ = kind_o(\texttt{dept}, \texttt{has\_staff}) \ = \texttt{IDREF} \end{array}$$

$$r_o \ = \ \texttt{db}$$

$$\begin{array}{lll} \Sigma_o \ = \{ & \texttt{person.oid} \rightarrow_{id} \texttt{person}, & \texttt{dept.oid} \rightarrow_{id} \texttt{dept}, \\ & \texttt{person.name} \rightarrow \texttt{person}, & \texttt{dept.dname} \rightarrow \texttt{dept}, \end{array}$$

```
person.in_dept ⊆_S dept.oid,      dept.manager ⊆ person.oid,
dept.has_staff ⊆_S person.oid,    dept.has_staff ⇌ person.in_dept }
```

The L_{id} constraints in the set Σ_o can categorized as follows: (1) two ID constraints: `person.oid` $\rightarrow_{id}$ `person`, `dept.oid` $\rightarrow_{id}$ `dept`; (2) two keys: `person.name` $\rightarrow$ `person`, `dept.dname` $\rightarrow$ `dept`; (3) two set-valued foreign keys: one is the combination of `person.in_dept` $\subseteq_S$ `dept.oid` and `dept.oid` $\rightarrow_{id}$ `dept`, and the other is the pair `dept.has_staff` $\subseteq_S$ `person.oid` and `person.oid` $\rightarrow_{id}$ `person`; (4) a single-valued foreign key: `dept.manager` $\subseteq$ `person.oid` and `person.oid` $\rightarrow_{id}$ `person`; and (5) an inverse constraint `dept.has_staff` $\rightleftharpoons$ `person.in_dept`. In Section 2.4, we shall extend L_{id} to specify constraints in terms of sub-elements. Thus we do not have to redefine `name`, `dname` as attributes. Note that here we specify keys in addition to object identities.

Finally, consider the `university` DTD given in Section 1. We use the following constraints in language L to specify that (`dept`, `number`) is a key of `course` and a foreign key of `enroll` referring to `course`:

```
course [dept, number] → course,
enroll [dept, number] ⊆ course[dept, number].
```

These are multi-attribute (sub-element) constraints. Below we shall see that L constraints can be defined in terms of sub-elements in addition to attributes.

2.4. Sub-elements as keys and foreign keys

In the XML standard [9], sub-elements are not allowed to participate in the reference mechanism. To be consistent with this approach, we have so far used only attributes in our constraints. A natural question here is whether sub-elements can also be used as keys and foreign keys. As an example, let us consider the element type definition of `person` given in Section 1:

```
<!ELEMENT person (name, address)>,
```

where the type of `name` is S_τ (string). It is reasonable to assume that `name` is a key for `person`. This was easily captured in our corresponding DTD specification (see D_o above) by including `person.name` $\rightarrow$ `person` in the constraint set (Σ_o). This suggests that we extend the definition of keys in L_{id}. In general, let τ be specified by an element type definition $P(\tau) = \alpha$ and K be a sub-element of τ. We may specify keys of the form: $\tau.K \rightarrow \tau$ if the following two conditions are satisfied: (1) K is *unique in* τ, i.e., for any $w \in L(\alpha)$, K occurs exactly once in w, where $L(\alpha)$ is the regular language defined by α; (2) the type of K is S_τ, i.e., $P(K) = S_\tau$ and therefore, in a data tree, a K element is represented as a text node. One can easily check these syntactic restrictions, for instance using the type checking algorithm of [25]. For any τ element x, we refer to its K element as $x.K$ and the value of $x.K$ as $val(x.K)$. This key constraint asserts that for any τ elements x and y, if they agree on the values of their K sub-elements, i.e., $val(x.K) = val(y.K)$, then x and y must be the same node, i.e., $x = y$. Observe that we again use two notions of equality here: value equality when comparing the values of the K sub-elements of x and y, and node equality when comparing x and y. We require $P(K)$ to be a simple type so that one can easily compare the values of K elements.

Along the same lines, we extend the definitions of key and foreign key constraints in L_u and L to incorporate sub-elements.

Sub-elements have also been used for key specifications in XML Schema [35] and the key constraint language of [10]. XML Schema also assumes that sub-elements used as keys are of simple types, while the general notion of value equality used in [10] no longer requires sub-elements to have simple types.

To simplify the discussion, the proofs given in the paper will assume basic XML constraints defined in terms of XML attributes, but all the results also hold for XML constraints defined in terms of sub-elements.

3. IMPLICATION OF BASIC XML CONSTRAINTS

In this section, we investigate the question of logical implication in connection with basic XML constraints: given that certain constraints are known to hold, does it follow that some other constraint necessarily holds? We examine the question for L_{id}, L_u and L. For each of these constraint languages, we establish complexity results for its implication and finite implication problems. We also provide axiomatization if one exists. These results are useful for, among others, studying XML semantics and query optimization. Some of these results are also applicable to relational databases.

We first give a formal description of implication of XML constraints. Let C be either L_{id}, L_u or L, and $\Sigma \cup \varphi$ be a finite set of C constraints. We use $\Sigma \models \varphi$ (resp. $\Sigma \models_f \varphi$) to denote that for any (resp. finite) data tree, if $G \models \Sigma$ then it must be the case that $G \models \varphi$.

The *implication problem for* C is to determine, given any finite set $\Sigma \cup \varphi$ of C constraints, whether $\Sigma \models \varphi$. The finite implication problem for C is to determine whether $\Sigma \models_f \varphi$.

In this paper, we consider constraint implication that generally holds for all XML documents. The documents may conform to any DTD, or do not have a DTD at all. In practice, it is common to find XML documents without DTDs. In the last section we define constraints as part of DTD^C to illustrate semantic specifications for XML data. In fact we can specify L_{id}, L_u and L constraints independent of any DTDs, while the constraints provide certain structural specification. More specifically, given a finite set Σ of constraints, let T be the set of all element types in Σ and for each $\tau \in T$, let $A(\tau)$ be the set of all attributes l's such that $\tau.l$ is in Σ. We assume that for any $\tau \in T$ and $l \in A(\tau)$, each τ element has a unique l attribute and moreover, we assume the following as specified in Section 2.

(1) When L_{id} is considered, we assume that each element type has at most one ID attribute and its elements can only be referenced with the ID attribute. In addition, if $\tau.l \subseteq \tau'.id$ is in Σ, then l must be a single-valued attribute, and if $\tau.l \subseteq_S \tau'.id$ is in Σ, then l must be a set-valued attribute. If $\tau.l \rightleftharpoons \tau'.l'$ is in Σ, then l, l' are set-valued attributes of τ and τ', respectively, and moreover, both τ and τ' have distinguished ID attributes. The attributes l and l' above are of IDREF kind. In a key $\tau.l \to \tau$, l is a single-valued attribute.

(2) When L_u is considered, given $\tau.l \subseteq \tau'.l'$ in Σ, we assume that l and l' are single-valued attributes of τ and τ', respectively. If $\tau.l \subseteq_S \tau'.l'$ is in Σ, then l is a set-valued attribute of τ and l' is a single-valued attribute of τ'. In a key $\tau.l \to \tau$, l is a single-valued attribute. An attribute cannot be both single-valued and set-valued.

(3) All attributes in L constraints are single-valued attributes.

3.1. Implication of L_{id} constraints

We first study the constraint language L_{id}. In L_{id}, an ID constraint asserts that an ID attribute value uniquely identifies an element within the entire document. An element has at most one ID, and is referred to by means of its ID attribute. As mentioned earlier, this reference mechanism is similar to the one used in object-oriented databases. Given the semantics of ID constraints and the reference mechanism, for any element types τ_1, τ_2, τ_3, we have neither $\tau_1.id \subseteq \tau_2.l$ nor $\tau_1.id \subseteq_S \tau_2.l$; and moreover, we cannot have both $\tau_1.l \subseteq \tau_2.id$ (or $\tau_1.l \subseteq_S \tau_2.id$) and $\tau_1.l \subseteq \tau_3.id$ (or $\tau_1.l \subseteq_S \tau_3.id$) if $\tau_2 \neq \tau_3$. As a result, $\subseteq$ and $\subseteq_S$ do not have transitivity in L_{id}. For the same reason, we cannot have both $\tau_1.l_1 \rightleftharpoons \tau_2.l_2$ and $\tau_1.l_1 \rightleftharpoons \tau_3.l_3$ if $\tau_2 \neq \tau_3$. Also observe that we do not allow $\tau.l \subseteq \tau.l$ (or $\tau.l \subseteq_S \tau.l)$ because by the definition of foreign keys in L_{id}, we can only reference elements with their ID attributes, and an attribute cannot be both ID and IDREF. These syntactic anomalies are easy to detect and thus without loss of generality, we assume that for any set $\Sigma \cup \{\varphi\}$ of L_{id} constraints considered in the (finite) implication problem, constraints with these anomalies do not occur. Recall that a foreign key always comes as a pair: an inclusion constraint and an ID constraint.

We give a finite axiomatization, denoted by $\mathcal{I}_{id}$, for implication and finite implication of L_{id} constraints as follows:

- ID-Key:
$$\frac{\tau.id \to_{id} \tau}{\tau.id \to \tau}$$
- Inv-SFK-ID:
$$\frac{\tau.l \rightleftharpoons \tau'.l'}{\tau.l \subseteq_S \tau'.id \quad \tau'.id \to_{id} \tau' \quad \tau'.l' \subseteq_S \tau.id \quad \tau.id \to_{id} \tau}$$
- Inv-commu:
$$\frac{\tau.l \rightleftharpoons \tau'.l'}{\tau'.l' \rightleftharpoons \tau.l}$$
- Inv-trans:
$$\frac{\tau.l_1 \rightleftharpoons \tau'.l'_1 \quad \tau'.l'_1 \rightleftharpoons \tau.l_2 \quad \tau.l_2 \rightleftharpoons \tau'.l'_2}{\tau.l_1 \rightleftharpoons \tau'.l'_2}$$

To see that Inv-trans is sound, consider a data tree G that satisfies the inverse constraints in the precondition of the rule. For any τ element d and τ' element d' in G, if $d'.id$ is in $d.l_1$, then $d.id$ must be in $d'.l'_2$ because (1) $d.id$ is in $d'.l'_1$ by $\tau.l_1 \rightleftharpoons \tau'.l'_1$; (2) $d'.id$ is in $d.l_2$ by $\tau'.l'_1 \rightleftharpoons \tau.l_2$; and thus (3) $d.id$ must be in $d'.l'_2$ by $\tau.l_2 \rightleftharpoons \tau'.l'_2$. Similarly, if $d.id$ is in $d'.l'_2$ then $d'.id$ must be in $d.l_1$. The other rules of $\mathcal{I}_{id}$ are intuitive.

We use $\Sigma \vdash_{\mathcal{I}_{id}} \varphi$ to denote that φ can be proved from Σ by using rules of $\mathcal{I}_{id}$, i.e., there is an $\mathcal{I}_{id}$-proof of φ from Σ.

THEOREM 3.1. *(1) $\mathcal{I}_{id}$ is sound and complete for both implication and finite implication of L_{id} constraints. (2) The implication and finite implication problems for L_{id} coincide and are decidable in linear time.*

Proof. Let $\Sigma \cup \{\varphi\}$ be a finite set of L_{id} constraints.
(1) Soundness of $\mathcal{I}_{id}$ can be verified by induction on the lengths of $\mathcal{I}_{id}$-proofs. For the proof of completeness, it suffices to construct a finite data tree $G = (V, elem, att, root)$ such that $G \models \Sigma$ and in addition, if $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$, then $G \not\models \varphi$. We construct G in two steps. We first define a finite data tree $G' = (V, elem, att', root)$ such that $G' \models \Sigma$. We then modify G' to construct G.

To construct G', we do the following. Let T be the set of all element types in $\Sigma \cup \{\varphi\}$. For each element type τ in T, we create a set of τ elements, denoted by $E(\tau)$. To ensure that $G \models \Sigma$, we need to enforce a certain relation on the cardinalities of these sets. To do so we define a relation $\leq$ on T as follows:

- $\tau \leq \tau$;
- $\tau_1 \leq \tau_2$ if $\Sigma \vdash_{\mathcal{I}_{id}} (\tau_1.l \to \tau_1) \wedge (\tau_1.l \subseteq \tau_2.id)$;
- $\tau_1 \leq \tau_3$ if $\tau_1 \leq \tau_2$ and $\tau_2 \leq \tau_3$.

Using $\leq$ we define an equivalence relation on T:

$$\tau_1 \sim \tau_2 \quad \text{iff} \quad \tau_1 \leq \tau_2 \text{ and } \tau_2 \leq \tau_1.$$

Let $[\tau]$ be the equivalence class of τ w.r.t. $\sim$. Intuitively, if we use $|E(\tau)|$ to denote the cardinality of $E(\tau)$, then we need to ensure $|E(\tau_1)| = |E(\tau_2)|$ if $\tau_1 \sim \tau_2$, and $|E(\tau_1)| \leq |E(\tau_2)|$ if $\tau_1 \leq \tau_2$. To make this happen, let us define a topological order $<$ on $[\tau]$'s having the following property: $[\tau_1] < [\tau_2]$ if there are $\tau_1' \in [\tau_1]$ and $\tau_2' \in [\tau_2]$ such that $\tau_1' \leq \tau_2'$. It is easy to see that this is well-defined. Assume that the topological order is $[\tau_1], ..., [\tau_k]$. For each $i \in [1, k]$ and each τ in $[\tau_i]$, we create $i + 1$ elements of τ. In addition, we create a distinct node $root$. Let V be the set consisting of all the elements created above. We define the *elem* function such that $elem(root) = (r, [F(\tau_1), ..., F(\tau_n)])$, where $\tau_1, ..., \tau_n$ are all the element types in T, and $F(\tau_i)$ is a list of all elements in $E(\tau_i)$. For each τ_i and each $d \in E(\tau_i)$, let $elem(d) = (\tau_i, \epsilon)$. These define a finite data tree in which the root has all the other elements as its children. Finally, we define the partial function att' as follows: for each $\tau \in T$,
(a) if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.id \to_{id} \tau$, then let $ext(\tau).id$ be a set of distinct string values;
(b) for each $l \in A(\tau)$, if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \subseteq \tau'.id$, then let $ext(\tau).l$ be a subset of $ext(\tau').id$. In particular, if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \to \tau$, then let $d.l \neq d'.l$ for all $d, d' \in E(\tau)$. This is possible because (i) by the definition of the $\leq$ relation, $|ext(\tau)| \leq |ext(\tau')|$ if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \subseteq \tau'.id$ and $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \to \tau$; and (ii) it is illegal to have both $\tau.l \subseteq \tau_1.id$ and $\tau.l \subseteq \tau_2.id$ if $\tau_1 \neq \tau_2$. It is to ensure these that we need to consider the cardinality dependencies in our definition of G'. If $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \subseteq_S \tau'.id$, then for each $d \in E(\tau)$, let $d.l = ext(\tau').id$. Observe that there is at most one τ' such that $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \subseteq_S \tau'.id$ by the semantics of ID attributes. Otherwise, let $\tau.l$ be a set consisting of arbitrary (distinct) string values.

It is easy to verify that $G' \models \Sigma$ given the definition of att'.

Next, we construct G from G'. Suppose $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$.
(a) φ is $\tau.id \to_{id} \tau$. By $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$, we have $\varphi \notin \Sigma$. Let d be a node in $E(\tau)$. We create another element d' of some fresh element type τ' such that $d'.id = d.id$. Let G be G' with the addition of d' as a child of $root$. It is easy to see that $G \models \Sigma$ and $G \models \neg\varphi$.
(b) φ is $\tau.l \to \tau$. By the definition of G', there are two distinct nodes $d_1, d_2 \in E(\tau)$. Let $d_1.l = d_2.l$ and G be G' with this modification. Since $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$, φ is not in Σ. Moreover, neither is $\tau.l \to_{id} \tau$ in Σ when l is id, by the ID-Key rule in $\mathcal{I}_{id}$. Thus Σ does not contain constraints of the form $\tau'.l' \subseteq \tau.l$ (or $\tau'.l' \subseteq_S \tau.l$). Given these, it is easy to verify that $G \models \Sigma$ and $G \models \neg\varphi$.
(c) φ is a foreign key $\tau.l \subseteq \tau'.id$ and $\tau'.id \to_{id} \tau'$. If $\Sigma \not\vdash_{\mathcal{I}_{id}} \tau'.id \to_{id} \tau'$, then the proof of (a) suffices. Otherwise we must have $\Sigma \not\vdash_{\mathcal{I}_{id}} \tau.l \subseteq \tau'.id$. In addition, we have $\Sigma \not\vdash_{\mathcal{I}_{id}} \tau'.id \subseteq \tau.l$, since otherwise we would also have $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \to_{id} \tau$ by the definition of foreign keys, and these and $\Sigma \vdash_{\mathcal{I}_{id}} \tau'.id \to_{id} \tau'$ contradict the fact that ID attributes are unique in the entire document. Hence we have both $\Sigma \not\vdash_{\mathcal{I}_{id}} \tau.l \subseteq \tau'.id$ and $\Sigma \not\vdash_{\mathcal{I}_{id}} \tau'.id \subseteq \tau.l$. Thus in the definition of G' we can ensure that there is $d \in E(\tau)$ such that $d.l \neq d'.id$ for all $d' \in E(\tau')$. Let G be this G'. Then $G \models \Sigma$ and $G \models \neg\varphi$.
(d) φ is a set-valued foreign key $\tau.l \subseteq_S \tau'.id$ and $\tau'.id \to_{id} \tau'$. The proof is similar to that of (c).

(e) φ is $\tau.l \rightleftharpoons \tau'.l'$. By $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$ and the Inv-commu rule in $\mathcal{I}_{id}$, neither φ nor $\tau'.l' \rightleftharpoons \tau.l$ is in Σ. If $\Sigma \not\vdash_{\mathcal{I}_{id}} (\tau.l \subseteq_S \tau'.id) \vee (\tau'.l' \subseteq_S \tau.id) \vee (\tau.id \rightarrow_{id} \tau) \vee (\tau'.id \rightarrow_{id} \tau')$, then the proof of (d) suffices by the Inv-SFK-ID rule in $\mathcal{I}_{id}$. Otherwise, by the definition of G', there exist $d \in E(\tau)$ and $d' \in E(\tau')$. By the construction of G', we have $d'.id \in d.l$ by $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \subseteq_S \tau'.id$. We modify G' by removing $d.id$ from $d'.l'$. Similarly, $d.id \in d'.l'$. This may violate certain inverse constraints of Σ, but not other forms of constraints. To ensure that inverse constraints of Σ are satisfied, if $\tau.l_1 \rightleftharpoons \tau'.l'$ is in Σ, then we also remove $d'.id$ from $d.l_1$, and so on. Observe that this will not lead to the removal of $d'.id$ from $d.l$ since otherwise we would have had $\Sigma \vdash_{\mathcal{I}_{id}} \varphi$ by the Inv-trans rule. Let G be the data tree obtained after the modifications. It is easy to verify that G satisfies all the constraints of Σ. However, it violates the inverse relationship between $\tau.l$ and $\tau'.l'$ because there are $x \in ext(\tau)$ and $y \in ext(\tau')$ such that $y.id \in x.l$, but $x.id \notin y.l'$. Thus $G \not\models \varphi$ and $G \models \Sigma$.

Note that G is finite. Thus $\mathcal{I}_{id}$ is sound and complete for both implication and finite implication of L_{id} constraints. As an immediate result, these two problems coincide.

(2) We next show that L_{id} constraint implication can be checked in linear time. Given a set $\Sigma \cup \{\varphi\}$ of L_{id} constraints, we first construct a graph G_D, and then inspect G_D to determine whether $\Sigma \models \varphi$. The nodes of G_D are element types τ of T and their attributes in $A(\tau)$. The edges of G_D are defined as follows. (i) For any $\tau \in T$ and $\tau.l \in A(\tau)$, there is an edge from τ to $\tau.l$. (ii) For any $\phi \in \Sigma$, we add edges to G_D as follows: (a) if ϕ is $\tau.l \rightarrow_{id} \tau$ (resp. $\tau.l \rightarrow \tau$), then add an edge labeled "$\rightarrow_{id}$" (resp. "$\rightarrow$") from $\tau.l$ to τ; (b) if ϕ is $\tau.l \subseteq \tau'.id$ (resp. $\tau.l \subseteq_S \tau'.id$), then add an edge labeled "$\subseteq$" (resp. "$\subseteq_S$") from $\tau.l$ to $\tau'.id$; (c) if ϕ is $\tau.l \rightleftharpoons \tau'.l'$, then add an edge labeled "$\rightleftharpoons$" between $\tau.l$ and $\tau'.l'$, an edge labeled "$\rightarrow_{id}$" from $\tau.id$ to τ and from $\tau'.id$ to τ', and an edge labeled "$\subseteq_S$" from $\tau'.l'$ to $\tau.id$ and from $\tau.l$ to $\tau'.id$. By the syntactic restrictions of L_{id} constraints mentioned above, $\tau.id$ and $\tau'.id$ are identified given an inverse constraints. By induction on the lengths of $\mathcal{I}_{id}$-proofs it is easy to verify the following:

Claim: $\Sigma \vdash_{\mathcal{I}_{id}} \varphi$ iff in G_D,

- if φ is $\tau.l \rightleftharpoons \tau'.l'$, then there is a path of edges labeled "$\rightleftharpoons$" between $\tau.l$ and $\tau'.l'$, which is length $2k+1$ for some natural number k;
- if φ is $\tau.l \rightarrow_{id} \tau$, then there is a "$\rightarrow_{id}$" edge from $\tau.l$ to τ;
- if φ is $\tau.l \rightarrow \tau$, then there is an edge labeled with either "$\rightarrow$" or "$\rightarrow_{id}$" from $\tau.l$ to τ;
- if φ is $\tau.l \subseteq \tau'.id$ (resp. $\tau.l \subseteq_S \tau'.id$) and $\tau'.id \rightarrow_{id} \tau'$, then there is a "$\rightarrow_{id}$" edge from $\tau'.id$ to τ' and a "$\subseteq$" (resp. "$\subseteq_S$") edge from $\tau.l$ to $\tau'.id$.

As the graph can be constructed in linear time and the conditions can also be checked in linear time, by (1), one can determine whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) in linear time. ■

From this one can see that it is rather straightforward to reason about L_{id} constraints.

3.2. Implication of L_u constraints

We next consider the constraint language L_u. In L_u, a key constraint states that a key is unique among the elements of the same type, rather than within the whole document. An element may have more than one key, and is referenced by means of any one of its keys. This constraint language provides an alternative reference mechanism for XML.

We present a finite axiomatization $\mathcal{I}_u$ for implication of L_u constraints as follows.

- UK-FK: $$\frac{\tau.l \to \tau}{\tau.l \subseteq \tau.l}$$
- UFK-trans: $$\frac{\tau_1.l_1 \subseteq \tau_2.l_2 \quad \tau_2.l_2 \to \tau_2 \quad \tau_2.l_2 \subseteq \tau_3.l_3 \quad \tau_3.l_3 \to \tau_3}{\tau_1.l_1 \subseteq \tau_3.l_3}$$
- USFK-trans: $$\frac{\tau_1.l_1 \subseteq_S \tau_2.l_2 \quad \tau_2.l_2 \to \tau_2 \quad \tau_2.l_2 \subseteq \tau_3.l_3 \quad \tau_3.l_3 \to \tau_3}{\tau_1.l_1 \subseteq_S \tau_3.l_3}$$

In contrast to L_{id}, we have transitivity rules (UFK-trans, USFK-trans) for $\subseteq$ and $\subseteq_S$ in L_u. However, observe that we do not have the rule: if $\tau_1.l_1 \subseteq \tau_2.l_2$ and $\tau_2.l_2 \subseteq_S \tau_3.l_3$ then $\tau_1.l_1 \subseteq_S \tau_3.l_3$. This is because key attributes cannot be set-valued.

Cosmadakis, Kanellakis and Vardi have shown [20] that in relational databases, implication and finite implication of unary inclusion and functional dependencies are different problems. In other words, (the complement of) implication of these dependencies does not have the finite model property. In addition, for any fixed integer k, there is no k-ary axiomatization for finite implication. Instead, there is a *cycle rule* for each odd positive integer. This is also the case for L_u. This is because keys and foreign keys impose dependencies on the cardinalities of finite sets of attribute values. More specifically, for any finite data tree G that satisfies an L_u constraint φ, if φ is a key $\tau.l \to \tau$, then $|ext(\tau)| = |ext(\tau).l|$, and if φ is a foreign key $\tau_1.l_1 \subseteq \tau_2.l_2$ and $\tau_2.l_2 \to \tau_2$, then $|ext(\tau_1).l_1| \leq |ext(\tau_2).l_2| = |ext(\tau_2)|$. As a result, for finite implication of L_u, we also have cycle rules: for each positive integer k, there is a cycle rule Ck:

$$\frac{\tau_1.l_1 \to \tau_1 \quad \tau_1.l_1 \subseteq \tau_2.l_2' \quad \tau_2.l_2' \to \tau_2 \quad \ldots \quad \tau_k.l_k \to \tau_k \quad \tau_k.l_k \subseteq \tau_1.l_1' \quad \tau_1.l_1' \to \tau_1}{\tau_1.l_1 \to \tau_1 \quad \tau_2.l_2' \subseteq \tau_1.l_1 \quad \ldots \quad \tau_k.l_k \to \tau_k \quad \tau_1.l_1' \subseteq \tau_k.l_k}$$

It is worth mentioning that the Ck rules are a little different from those in [20] since each element type τ_i involves two attributes l_i and l_i' in a Ck rule. As a result they do not require k to be odd.

Let $\mathcal{I}_u^f$ consist of $\mathcal{I}_u$ rules and Ck for each positive integer k. We use $\Sigma \vdash_{\mathcal{I}_u^f} \varphi$ to denote that there is an $\mathcal{I}_u^f$-proof of φ from Σ, and $\Sigma \vdash_{\mathcal{I}_u} \varphi$ to denote that there is an $\mathcal{I}_u$-proof of φ from Σ.

THEOREM 3.2. *(1) $\mathcal{I}_u^f$ is sound and complete for finite implication of L_u constraints. (2) $\mathcal{I}_u$ is sound and complete for implication of L_u constraints.*

Proof. Soundness of $\mathcal{I}_u^f$ ($\mathcal{I}_u$) can be verified by induction on the lengths of $\mathcal{I}_u^f$-proofs ($\mathcal{I}_u$-proofs). For the proof of completeness, given any finite set $\Sigma \cup \{\varphi\}$ of L_u constraints. we construct a finite data tree $G = (V, elem, att, root)$ such that $G \models \Sigma$ and in addition, if $\Sigma \not\vdash_{\mathcal{I}_u^f} \varphi$ ($\Sigma \not\vdash_{\mathcal{I}_u} \varphi$), then $G \not\models \varphi$. As in the proof of Theorem 3.1, we construct G in two steps: we first define a finite data tree $G' = (V, elem, att', root)$ such that $G' \models \Sigma$, and then modify G' to construct G. Let T be the set of element types that appear in $\Sigma \cup \{\varphi\}$, and S be the set $\{\tau.l \mid \tau \in T, l \in A(\tau)\}$.

(1) We first consider finite implication. For each $\tau \in T$, we create a finite set of τ elements, denoted by $E(\tau)$. As mentioned earlier, L_u constraints impose dependencies on

the cardinalities of these finite sets. To capture these, we first define an equivalence relation $\sim$ on the single-valued attributes of S:

$$\tau_1.l_1 \sim \tau_2.l_2 \quad \text{iff} \quad \Sigma \vdash_{\mathcal{I}_u^f} (\tau_1.l_1 \subseteq \tau_2.l_2) \wedge (\tau_2.l_2 \subseteq \tau_1.l_1).$$

Let $[\tau.l]$ denote the equivalence class of $\tau.l$ w.r.t. $\sim$. Intuitively, if $\tau_1.l_1 \sim \tau_2.l_2$, then we need to ensure $ext(\tau_1).l_1 = ext(\tau_2).l_2$, i.e., $\{x.l_1 \mid x \in E(\tau_1)\} = \{y.l_2 \mid y \in E(\tau_2)\}$. Observe that by $\mathcal{I}_u^f$ and the definition of foreign keys, if $\Sigma \vdash_{\mathcal{I}_u^f} \tau_1.l_1 \subseteq \tau_2.l_2$, then there must be $\Sigma \vdash_{\mathcal{I}_u^f} \tau_2.l_2 \to \tau_2$. Thus $|E(\tau_1)| = |ext(\tau_1).l_1| = |ext(\tau_2).l_2| = |E(\tau_2)|$ if $\tau_1.l_1 \sim \tau_2.l_2$ for some l_1 and l_2. This is, however, not enough to ensure that the cardinality dependencies are satisfied. Thus on $[\tau.l]$'s we define another equivalence relation $\approx$: $[\tau_1.l_1] \approx [\tau_2.l_2]$ iff there are $\tau.l \in [\tau_1.l_1]$, $\tau.l' \in [\tau_2.l_2]$ such that $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \to \tau$ and $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l' \to \tau$. That is, both l and l' are a key of τ elements, but $\tau.l$ and $\tau'.l'$ are in $[\tau_1.l_1]$ and $[\tau_2.l_2]$ respectively. As a result, $|ext(\tau).l| = |E(\tau)| = |ext(\tau).l'|$. Let $\{[\tau.l]\}$ be the equivalence class of $[\tau.l]$ w.r.t. $\approx$. Intuitively, if $[\tau_1.l_1] \approx [\tau_2.l_2]$, then we need to ensure $|ext(\tau_1).l_1| = |ext(\tau_2).l_2|$ since $ext(\tau_1).l_1 = ext(\tau).l$ and $ext(\tau_2).l_2 = ext(\tau).l'$. Finally, for each set-valued attribute l_1 of τ_1, let $\{[\tau_1.l_1]\}$ be a singleton set consisting of $\tau_1.l_1$. Given these, we define a topological order $<$ on $\{[\tau.l]\}$'s such that (a) if $\Sigma \vdash_{\mathcal{I}_u^f} \tau_1.l_1 \subseteq \tau_2.l_2$ or $\Sigma \vdash_{\mathcal{I}_u^f} \tau_1.l_1 \subseteq_S \tau_2.l_2$, then $\{[\tau_1.l_1]\} < \{[\tau_2.l_2]\}$ unless $\{[\tau_1.l_1]\} = \{[\tau_2.l_2]\}$; and (b) if $\Sigma \not\vdash_{\mathcal{I}_u^f} \tau.l \to \tau$ and $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l' \to \tau$, then $\{[\tau.l]\} < \{[\tau.l']\}$. It can be verified that the relations $\sim$, $\approx$ and $<$ are well-defined by using, among others, the Ck and UFK-trans rules in $\mathcal{I}_u^f$ and the syntax of L_u constraints. Assume that the topological order is $\{[\tau_1.l_1]\}, ..., \{[\tau_k.l_k]\}$. For each $i \in [1, k]$ and $[\tau.l] \in \{[\tau_i.l_i]\}$, we create a set $str[\tau.l]$ of distinct string values such that for any $\tau'.l \in [\tau.l]$, $ext(\tau').l' = str[\tau.l]$, and in addition, for any $[\tau.l], [\tau'.l'] \in \{[\tau_i.l_i]\}$, $|str[\tau.l]| = |str[\tau'.l']|$. To do so, we process $\{[\tau_i.l_i]\}$ for i from 1 to k as follows. Initially, let $str[\tau.l]$ be empty for all $\tau.l$. For each $[\tau.l] \in \{[\tau_i.l_i]\}$, we increment $str[\tau.l]$ such that it is a set of (at least two) distinct string values with at least one new string value not considered so far. This ensures that $str[\tau.l] \neq str[\tau.l']$ if $\tau.l \not\sim \tau.l'$. In addition, let $|str[\tau.l]| = |str[\tau'.l']|$ for any other $[\tau'.l'] \in \{[\tau_i.l_i]\}$. We then propagate $str[\tau.l]$ such that for any $[\tau'.l']$, if $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \subseteq \tau'.l'$ or $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \subseteq_S \tau'.l'$, then $str[\tau.l]$ is a subset of $str[\tau'.l']$. More specifically, by the definition of $<$, $\tau'.l'$ must be in some $\{[\tau_j.l_j]\}$ for some $j > i$, and thus we can increment $str[\tau'.l']$ by including $str[\tau.l]$ when processing $\tau.l$ in the loop. Given these, we define $E(\tau)$ for each $\tau \in T$. Let n_τ be the cardinality of the largest set $str[\tau.l]$ for all $l \in A(\tau)$. We create n_τ distinct elements of τ and let $E(\tau)$ be the set consisting of these τ elements. By the cardinality argument, it can be verified that $|str[\tau.l]| = n_\tau$ as long as $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \to \tau$, i.e., $|ext(\tau).l| = |E(\tau)|$. In addition, we create a distinct node $root$. We define G' as follows. Let V be the set consisting of all the elements created above. We define the *elem* function such that $elem(root) = (r, [F(\tau_1), ..., F(\tau_n)])$, where $\tau_1, ..., \tau_n$ are all the element types in T, and $F(\tau_i)$ is a list of all elements in $E(\tau_i)$. For each τ_i and each $d \in E(\tau_i)$, let $elem(d) = (\tau_i, \epsilon)$. These define a finite data tree in which the root has all the other elements as its children. Finally, we define the partial function att' as follows: for each $\tau \in T$ and $l \in A(\tau)$, if l is a single-valued attribute, then let $ext(\tau).l$ be $str[\tau.l]$ and for each $d \in E(\tau)$, let $d.l$ be a single string value. In particular, if $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \to \tau$, then for any $d, d' \in E(\tau)$, let $d.l \neq d'.l$. This is possible because the construction of G' satisfies the cardinality dependency $|ext(\tau).l| = |E(\tau)|$. If l is set-valued, then let $d.l = str[\tau.l]$, which is a set. Given the definition of G', it is easy to verify that $G' \models \Sigma$.

We next construct G from G'. Suppose $\Sigma \not\vdash_{\mathcal{I}_u^f} \varphi$.
(a) φ is $\tau.l \to \tau$. By the definition of G', there are two distinct nodes $d_1, d_2 \in E(\tau)$. We define G by changing $d_1.l$ in G' such that $d_1.l = d_2.l$. Since $\Sigma \not\vdash_{\mathcal{I}_u^f} \varphi$, φ is not in Σ. Moreover, for any $\tau'.l'$, $\Sigma \not\vdash_{\mathcal{I}_u^f} \tau'.l' \subseteq \tau.l$ and $\Sigma \not\vdash_{\mathcal{I}_u^f} \tau'.l' \subseteq_S \tau.l$ by the definition of foreign keys and $\mathcal{I}_u^f$ rules. Given this, it is easy to verify that $G \models \Sigma$ and $G \models \neg\varphi$.
(b) φ is a foreign key $\tau.l \subseteq \tau'.l'$ and $\tau'.l' \to \tau'$. If $\Sigma \not\vdash_{\mathcal{I}_u^f} \tau'.l' \to \tau'$, then the proof of (a) suffices. Otherwise we must have $\Sigma \not\vdash_{\mathcal{I}_u^f} \tau.l \subseteq \tau'.l'$ and $\Sigma \vdash_{\mathcal{I}_u^f} \tau'.l' \to \tau'$. We assume that $\tau.l$ and $\tau'.l'$ are single-valued since otherwise by the syntax of L_u constraints one cannot write $\tau.l \subseteq \tau'.l'$. We show that in G', $ext(\tau).l \not\subseteq ext(\tau').l'$. By the definition of G', this happens if and only if $\Sigma \not\vdash_{\mathcal{I}_u^f} \tau.l \subseteq \tau'.l'$. Indeed, in the population process above, if $\{[\tau.l]\} \neq \{[\tau'.l']\}$, then $str[\tau.l] \subseteq str[\tau'.l']$ if and only if $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \subseteq \tau'.l'$. If $\{[\tau.l]\} = \{[\tau'.l']\}$, then $str[\tau.l] \subseteq str[\tau'.l']$ if and only if either $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \subseteq \tau'.l'$ or $\Sigma \vdash_{\mathcal{I}_u^f} \tau'.l' \subseteq \tau.l$. But in both cases we would have $\tau.l \sim \tau'.l'$ by the Ck rules and UFK-trans, which implies $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \subseteq \tau'.l'$. Therefore, $ext(\tau).l \not\subseteq ext(\tau').l'$ in G'. Let G be G'. Thus $G \models \Sigma$ and $G \models \neg\varphi$.
(c) φ is a set-valued foreign key $\tau.l \subseteq_S \tau'.l'$ and $\tau'.l' \to \tau'$. If $\Sigma \not\vdash_{\mathcal{I}_u^f} \tau'.l' \to \tau'$, then the proof of (a) suffices. Otherwise we must have $\Sigma \not\vdash_{\mathcal{I}_u^f} \tau.l \subseteq_S \tau'.l'$. By the definition of G' above, $str[\tau.l]$ has a distinct string value that is included in $str[\tau_1.l_1]$ for any $\tau_1.l_1$ only if $\Sigma \vdash_{\mathcal{I}_u^f} \tau.l \subseteq_S \tau_1.l_1$. Hence it is not in $ext(\tau'.l')$. Let G be G'. Thus $G \models \Sigma$ but $G \not\models \varphi$.

(2) We next consider implication of L_u constraints. If we allow infinite data trees, then keys and foreign keys no longer impose dependencies on cardinalities of sets of attribute values. In other words, the cycle rules Ck no longer apply. As in the proof of (1), we define an equivalence relation $\sim$ on S by means of $\vdash_{\mathcal{I}_u}$. We define a topological order on $[\tau.l]$'s such that if $\Sigma \vdash_{\mathcal{I}_u} \tau_1.l_1 \subseteq \tau_2.l_2$ or $\Sigma \vdash_{\mathcal{I}_u} \tau_1.l_1 \subseteq_S \tau_2.l_2$, then $[\tau_1.l_1] < [\tau_2.l_2]$ unless $[\tau_1.l_1] = [\tau_2.l_2]$. As in the proof of (1), we define $str[\tau.l]$ for each $\tau.l \in S$, and let $str[\tau.l]$ be a subset of $str[\tau'.l']$ if $\Sigma \vdash_{\mathcal{I}_u} \tau.l \subseteq \tau'.l'$ or $\Sigma \vdash_{\mathcal{I}_u} \tau.l \subseteq_S \tau'.l'$. However, here we let $str[\tau.l]$ be a countably infinite set of distinct values, and we create a countably infinite set $E(\tau)$ of τ elements for each $\tau \in T$. We define a data tree G' such that for each $\tau \in T$, $ext(\tau) = E(\tau)$. The rest of the proof is similar to the proof of (1). ■

Using $\mathcal{I}_u$ and $\mathcal{I}_u^f$, we can develop a linear-time algorithm for testing implication of L_u constraints, and a linear-time algorithm for testing finite implication of L_u constraints.

THEOREM 3.3. *The implication and finite implication problems for L_u are both decidable in linear time, but these problems do not coincide.*

Proof. The cycle rules Ck in $\mathcal{I}_u^f$ show that (the complement of) L_u constraint implication does not have the finite model property, and thus is different from finite implication of L_u constraints. To prove this, consider $\Sigma = \{\tau.l_1 \to \tau,\ \tau.l_1 \subseteq \tau.l_2,\ \tau.l_2 \to \tau\}$ and let φ be $\tau.l_2 \subseteq \tau.l_1$ and $\tau.l_1 \to \tau$. By the Ck and UK-FK rules in $\mathcal{I}_u^f$ and Theorem 3.2, we have $\Sigma \models_f \varphi$. However, $\Sigma \not\models \varphi$. Indeed, let G be a data tree such that in G, $ext(\tau)$ has countably infinitely many τ elements: $d_1, d_2, \ldots$, and for each i, $d_i.l_1 = i + 1$ and $d_i.l_2 = i$. Thus $G \models \Sigma$ but $G \not\models \varphi$.

We next show that for any finite subset $\Sigma \cup \{\varphi\}$ of L_u constraints, $\Sigma \models_f \varphi$ can be determined in linear time. Implication of L_u constraints is similar and simpler.

By Theorem 3.2, it suffices to show that $\Sigma \vdash_{\mathcal{I}_u^f} \varphi$ can be determined in linear time. As in the proof of Theorem 3.1 (2), we first construct a graph, called the *dependency graph* of $\Sigma \cup \{\varphi\}$ and denoted by G_D. Let T be the set of all element types in $\Sigma \cup \{\varphi\}$. The vertices of G_D are element types of T and attributes of $A(\tau)$ for all $\tau \in T$. The edges of G_D are defined as follows. (i) For each $\tau \in T$ and $\tau.l \in A(\tau)$, there is an edge from τ to $\tau.l$ labeled with $\tau.l$. (ii) For each $\phi \in \Sigma$, we add the following edges to G_D: (a) if ϕ is $\tau.l \rightarrow \tau$, then add an edge labeled "$\rightarrow$" from $\tau.l$ to τ, and an edge labeled "$\supseteq$" from $\tau.l$ to itself; (b) if ϕ is $\tau.l \subseteq \tau'.l'$, then add an edge labeled "$\supseteq$" from $\tau'.l'$ to $\tau.l$; (c) if ϕ is $\tau.l \subseteq_S \tau'.l'$, then add an edge labeled "$\supseteq_S$" from $\tau'.l'$ to $\tau.l$. We can verify the following claim by an induction on the lengths of $\mathcal{I}_u^f$-proofs:

Claim: $\Sigma \vdash_{\mathcal{I}_u^f} \varphi$ iff in G_D,

- if φ is $\tau.l \rightarrow \tau$, then there is a "$\rightarrow$" edge from $\tau.l$ to τ;
- if φ is $\tau.l \subseteq \tau'.l'$ and $\tau'.l' \rightarrow \tau'$, then $\Sigma \vdash_{\mathcal{I}_u^f} \tau'.l' \rightarrow \tau'$ and moreover, one of the following conditions is satisfied: (a) either there is a path of "$\supseteq$" edges from $\tau'.l'$ to $\tau.l$, or (b) there is a path of "$\supseteq$" edges from $\tau.l$ to $\tau'.l'$ and moreover, there is a path from $\tau'.l'$ to $\tau.l$ in which all edges are labeled with "$\supseteq$", "$\rightarrow$" or $\tau_1.l_1$, where $\Sigma \vdash_{\mathcal{I}_u^f} \tau_1.l_1 \rightarrow \tau_1$;
- if φ is $\tau.l \subseteq_S \tau'.l'$ and $\tau'.l' \rightarrow \tau'$, then $\Sigma \vdash_{\mathcal{I}_u^f} \tau'.l' \rightarrow \tau'$ and moreover, there is $\tau_1.l_1$ such that $\Sigma \vdash_{\mathcal{I}_u^f} \tau_1.l_1 \subseteq \tau'.l'$ and there is a "$\supseteq_S$" edge from $\tau_1.l_1$ to $\tau.l$.

It takes $O(|\Sigma| + |\varphi|)$ time to construct the graph G_D. By the claim, we can test whether $\Sigma \vdash_{\mathcal{I}_u^f} \varphi$ by inspecting G_D. This also takes $O(|\Sigma| + |\varphi|)$ time. Thus by Theorem 3.2, whether $\Sigma \models_f \varphi$ can be determined in linear time. ■

To be even closer to the original XML semantics for ID attributes, we consider a *primary key restriction*. This restriction requires that for any element type τ, there is at most one attribute l such that l is a key of τ, i.e., $\tau.l \rightarrow \tau$. Elements of τ can only be referred to by using their l attributes. As a result, the cycle rules do not apply here. In addition, we cannot have both $\tau_1.l_1 \subseteq \tau.l$ and $\tau_2.l_2 \subseteq \tau.l'$ if $l \neq l'$. In relational databases, it is also common to consider primary keys.

The primary key restriction simplifies the analysis of L_u constraint implication. Indeed, the implication and finite implication problems for L_u coincide in this setting.

COROLLARY 3.1. *Under the primary key restriction, $\mathcal{I}_u$ is sound and complete for both implication and finite implication of L_u constraints.*

Proof. Under the primary key restriction, the cycle rules Ck in $\mathcal{I}_u^f$ no longer apply. Thus $\mathcal{I}_u^f$ and $\mathcal{I}_u$ become the same. The proof is similar to that of Theorem 3.2 (1), except that there is no need to define the equivalence relation $\approx$ here. ■

It is easy to verify that a similar result also holds for relational databases.

COROLLARY 3.2. *In relational databases, the implication and finite implication problems for primary unary keys and foreign keys coincide and are decidable in linear time.*

These results show that in the absence of DTDs, implication analysis of L_u constraints is similar to their relational counterpart and can be done efficiently.

3.3. Implication of L constraints

Next, we investigate the constraint language L. In L, one can express multi-attribute keys and foreign keys that are common in relational databases. As observed by [35], these constraints are also of practical interest for native XML data.

The analysis of L constraint implication is, however, nontrivial.

THEOREM 3.4. *The implication and finite implication problems for L are undecidable.*

This result also holds for relational databases.

THEOREM 3.5. *In relational databases, the implication and finite implication problems for key and foreign key constraints are undecidable.*

This can be proved by reduction from the implication and finite implication problems for inclusion and functional dependencies, which are well-known undecidable problems (see, e.g., [2] for a proof). Below we give a proof sketch of Theorem 3.5. Theorem 3.4 can be verified by reduction from the (finite) implication problem for keys and foreign keys in relational databases.

Proof. Let **R** be a relational schema, which is a collection of relation schemas. For a relation schema R, we use $Att(R)$ to denote the set of all attributes of R. Functional dependencies (FDs) and inclusion dependencies (IDs) over **R** are expressed as $R : X \to Y$ and $R_1[X] \subseteq R_2[Y]$, respectively, where R, R_1, R_2 are relation schemas in **R**, X, Y in the FD are sets of attributes in $Att(R)$, and $R_1[X], R_2[Y]$ denote lists of attributes of R_1, R_2, respectively. We use $R[X] \to R$ to denote that X is a key for R, where X is a subset of $Att(R)$. That is, $R : X \to Att(R)$. A foreign key is a pair consisting of an ID and a key: $R_1[X] \subseteq R_2[Y]$ and $R_2[Y] \to R_2$. It is known that it is undecidable to determine, given any relational schema **R**, any set Σ of FDs and IDs over **R** and a FD θ over **R**, whether $\Sigma \models \theta$ ($\Sigma \models_f \theta$. See, e.g., [2]). Let us refer to this problem as the (finite) implication problem for FDs and IDs. We prove Theorem 3.5 by reduction from this problem. To do so, 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 has a key, e.g., the set of all the attributes of a relation is a key of the relation). We define a new 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 R as it contains the key Z. We encode θ with:

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

(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 as it subsumes Z. We encode θ with:

$$\phi_1 = R_{new}[Y] \to R_{new}, \quad \phi_2 = R_1[X] \subseteq R_{new}[Y], \quad \phi_3 = R_{new}[YZ] \subseteq R_2[YZ].$$

We next show that the encoding is a reduction from the (finite) implication problem for FDs and IDs to the (finite) implication problem for keys and foreign keys. Given a relational schema **R**, a set Σ of FDs and IDs over **R**, and a FD $\theta = R_\theta : X \to Y$ over **R**, as described above we encode Σ with a set Σ_1 of keys and foreign keys, and encode θ with

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

where R^θ_{new} is the new relation introduced when coding θ. Let $\Sigma' = \Sigma_1 \cup \{\phi_2, \phi_3, \phi_4\}$. Observe that ϕ_1 is a key of R^θ_{new} and XYZ is a key of R_θ. Recall that a foreign key consists of a key and an ID. Thus 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 **R** as well as new relations created in the encoding. We show the claim as follows.
(1) Assume that there is a (finite) instance **I** of **R** such that **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 **R**, the instance of R in **I'** is the same as the instance of R in **I**. We populate instances of new relations R_{new} created in the encoding as follows. (a) If R_{new} is introduced in the encoding of a FD $R : X \to Y$ as 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 $I_{new} \models R_{new}[XY] \to R_{new}$, where I is the instance of R in **I** and $\Pi_W(I)$ denotes the projection of I on attributes W. (b) If R_{new} is introduced in the encoding of an ID $R_1[X] \subseteq R_2[Y]$ as 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 $I_{new} \models R_{new}[Y] \to R_{new}$, where I_2 is the instance of R_2 in **I**. One can easily verify that we indeed have **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 **I** of **R** by removing from **I'** all instances of new relations introduced in the encoding. It is easy to verify that **I** $\models \bigwedge \Sigma \wedge \neg\theta$.

Therefore, the encoding is indeed a reduction from the (finite) implication problem for FDs and IDs. This completes the proof of Theorem 3.5. ■

The undecidability result suggests that we consider the primary key restriction for L constraints. That is, in a set considered in L constraint implication, for any element type τ, one can specify at most one key, either as a key or as part of a foreign key. Under this restriction, one cannot specify two foreign keys $\tau_1[X_1] \subseteq \tau[Y]$, $\tau[Y] \to \tau$ and $\tau_2[X_2] \subseteq \tau[Y']$, $\tau[Y'] \to \tau$ at the same time if the two sets Y and Y' are not equal. Similarly, one cannot specify a foreign key $\tau'[X] \subseteq \tau[Y]$, $\tau[Y] \to \tau$ and a key $\tau[Y'] \to \tau$ if the two sets Y and Y' are not equal. When the primary key restriction is imposed, we refer to L constraints as primary keys and foreign keys. In the (finite) implication problem for primary keys and foreign keys, we consider finite set $\Sigma \cup \{\varphi\}$ of L constraints in which there is at most one key for each element type.

The primary key restriction simplifies reasoning about L constraints. Indeed, under this restriction, implication and finite implication of L constraints become axiomatizable. More specifically, we present an axiomatization $\mathcal{I}_p$ as follows:

- PK-FK:

$$\frac{\tau[X] \to \tau}{\tau[X] \subseteq \tau[X]}$$

- PFK-perm: for each list $i_1, i_2, ..., i_n$ of distinct integers in $[1, ..., n]$,

$$\frac{\tau[l_1, l_2, ..., l_n] \subseteq \tau'[l'_1, l'_2, ..., l'_n] \quad \tau'[l'_1, l'_2, ..., l'_n] \rightarrow \tau'}{\tau[l_{i_1}, l_{i_2}, ..., l_{i_n}] \subseteq \tau'[l'_{i_1}, l'_{i_2}, ..., l'_{i_n}]}$$

- PFK-trans: $$\frac{\tau_1[X] \subseteq \tau_2[Y] \quad \tau_2[Y] \rightarrow \tau_2 \qquad \tau_2[Y] \subseteq \tau_3[Z] \quad \tau_3[Z] \rightarrow \tau_3}{\tau_1[X] \subseteq \tau_3[Z]}$$

We use $\Sigma \vdash_{\mathcal{I}_p} \varphi$ to denote that φ can be proved from Σ by using rules of $\mathcal{I}_p$.

THEOREM 3.6. *Under the primary key restriction, $\mathcal{I}_p$ is sound and complete for both implication and finite implication of L constraints.*

By Theorem 3.6, the implication and finite implication problems for L coincide and are decidable under the primary key restriction.

Proof. Soundness of $\mathcal{I}_p$ can be verified by induction on the lengths of $\mathcal{I}_p$-proofs. To prove the completeness of $\mathcal{I}_p$, consider a finite set $\Sigma \cup \{\varphi\}$ of primary keys and foreign keys in L. It suffices to construct a finite data tree G such that $G \models \Sigma$ and moreover, if $\Sigma \not\vdash_{\mathcal{I}_p} \varphi$, then $G \not\models \varphi$. As in the proof of Theorem 3.1, we define T to be the set of all element types in $\Sigma \cup \{\varphi\}$. We construct G based on φ and Σ as follows. Assume that $\Sigma \not\vdash_{\mathcal{I}_p} \varphi$.

(1) φ is a key $\tau[X] \rightarrow \tau$. We create two distinct τ elements a and b such that $a[X]$ and $b[X]$ are a list of 0's, and thus $a[X] = b[X]$. In addition, for each $l \in A(\tau) \setminus X$, let $a.l = 0$ and $b.l = 1$. We use V to denote the set of vertices in G. Initially, let $V = \{a, b\}$. We add elements to V using the procedure given below.

```
repeat until no further change to $V$
  if $\Sigma \vdash_{\mathcal{I}_p} \tau_1[Y] \subseteq \tau_2[Z]$
    then for each $\tau_1$ element $d_1$ of in $V$
           (1) create a $\tau_2$ element $d_2$ such that
                 $d_2[Z] = d_1[Y]$ and $d_2.l = 1$ for all $l \in A(\tau) \setminus Z$;
           (2) $V := V \cup \{d_2\}$ if there is no $\tau_2$ element $d$ in $V$
                       such that $d.l = d_2.l$ for all $l \in A(\tau)$;
```

The procedure terminates since T is finite and moreover, only 0 and 1 are used as string values of the attributes involved. Finally, we add a distinct node *root* to V and define a finite data tree G such that the set of vertices in G is V and the root node of G is $root$. In addition, *root* has all the other elements in V as its children and any node except *root* does not have any children. Obviously $G \not\models \varphi$ because a, b are in V, $a \neq b$ and $a[X] = b[X]$. We next show that $G \models \Sigma$. Assume, by contradiction, $G \not\models \phi$ for some $\phi \in \Sigma$.

(i) If ϕ is a key $\tau_1[Y] \rightarrow \tau_1$, then there are two distinct τ_1 elements $d_1, d_2 \in V$ such that $d_1[Y] = d_2[Y]$. By the primary key restriction, if there exist τ_2 in T and a list Z of attributes in $A(\tau_2)$ such that $\Sigma \vdash_{\mathcal{I}_p} \tau_2[Z] \subseteq \tau_1[W]$, then the two sets W and Y must be equal. Indeed, by $\mathcal{I}_p$ and the definition of foreign keys, it is easy to see that if $\Sigma \vdash_{\mathcal{I}_p} \tau_2[Z] \subseteq \tau_1[W]$ then there must be $\Sigma \vdash_{\mathcal{I}_p} \tau_1[W] \rightarrow \tau_1$, and moreover, $\Sigma \vdash_{\mathcal{I}_p} \tau_1[W] \rightarrow \tau_1$ iff $\tau_1[W] \rightarrow \tau_1$ is in Σ. Thus by the primary key restriction, We have $W = Y$. If $\tau_1 \neq \tau$, then the procedure for

populating V vertices (by the condition in (2)) ensures that for any τ_1 elements $d_1, d_2 \in V$, $d_1[Y] \neq d_2[Y]$. This contradicts the assumption. If $\tau_1 = \tau$, consider the following cases. If $Y = X$, then this contradicts the assumption that $\Sigma \not\vdash_{\mathcal{I}_p} \varphi$. If $Y \neq X$, then this violates the primary key restriction, which again contradicts the assumption.
(ii) If ϕ is a foreign key $\tau_1[Y] \subseteq \tau_2[Z]$ and $\tau_2[Z] \to \tau_2$, consider the following cases. If $G \not\models \tau_2[Z] \to \tau_2$, then the proof of (i) suffices. Now assume $G \not\models \tau_1[Y] \subseteq \tau_2[Z]$. Then there exists a τ_1 element d_1 such that for any τ_2 element d_2, $d_1[Y] \neq d_2[Z]$. However, by $\Sigma \vdash_{\mathcal{I}_p} \tau_1[Y] \subseteq \tau_2[Z]$ and the procedure for populating V vertices, this cannot happen.

This shows that when φ is a key, $\mathcal{I}_p$ is complete for $\Sigma \models \varphi$ and $\Sigma \models_f \varphi$.

(2) φ is a foreign key $\tau_1[X] \subseteq \tau_2[Y]$ and $\tau_2[Y] \to \tau_2$. If $\Sigma \not\vdash_{\mathcal{I}_p} \tau_2[Y] \to \tau_2$, then the proof of (1) suffices. Assume $\Sigma \vdash_{\mathcal{I}_p} \tau_2[Y] \to \tau_2$ but $\Sigma \not\vdash_{\mathcal{I}_p} \tau_1[X] \subseteq \tau_2[Y]$. We create a τ_1 element a such that $a[X]$ is a list of 0's, and for any $l \in A(\tau_1) \setminus X$, $a.l = 1$. Let $V = \{a\}$ and we add elements to V using the procedure given in (1). Given V, we construct a finite data tree G as in (1). The proof of (1) can show that $G \models \Sigma$. We next show that $G \not\models \tau_1[X] \subseteq \tau_2[Y]$. If $\tau_1 \neq \tau_2$, then by $\Sigma \not\vdash_{\mathcal{I}_p} \tau_1[X] \subseteq \tau_2[Y]$ and the PFK-perm and PFK-trans rules in $\mathcal{I}_p$, no τ_2 element d is created by the procedure of (1) such that $a[X] = d[Y]$. If $\tau_1 = \tau_2$, consider the following cases. If $Y = X$, then this contradicts the assumption that $\Sigma \not\vdash_{\mathcal{I}_p} \varphi$ by the PFK-FK rule. If $Y \subset X$ or $X \subset Y$, then $\tau_1[X] \subseteq \tau_2[Y]$ are not syntactically correct because X and Y have different lengths. Thus $Y \not\subseteq X$ and $X \not\subseteq Y$. In this case, we have $a[X] \neq d[Y]$ for any τ_1 element d created by the procedure for populating V because $\Sigma \not\vdash_{\mathcal{I}_p} \tau_1[X] \subseteq \tau_1[Y]$. Thus we must have $G \not\models \tau_1[X] \subseteq \tau_2[Y]$. This shows $\mathcal{I}_p$ is also complete for $\Sigma \models \varphi$ and $\Sigma \models_f \varphi$ when φ is a foreign key.

This completes the proof of Theorem 3.6. ■

This result also holds for relational databases.

COROLLARY 3.3. *In relational databases, the implication and finite implication problems for primary keys and foreign keys coincide and are decidable.*

Proof. The rules of $\mathcal{I}_p$ are still sound for (finite) implication of primary keys and foreign keys in relational databases, with slight syntax modification. The only concern here is that in relational databases, only value equality is used for defining keys. More specifically, consider a relation schema R, a key $R[X] \to R$, and an instance I of R. Let $Att(R)$ be the set of all attributes of R. Then I satisfies the key iff for all tuples t_1, t_2 in I, if $t_1[X] = t_2[X]$ then $t_1.l = t_2.l$ for all $l \in Att(R) \setminus X$. In contrast, keys for XML are defined using two notions of equality, namely, value equality when comparing attributes values and node equality when comparing vertices, as described in Section 2. To cope with the semantics of keys in relational databases, we need to refine the definition of the primary restriction as follows: a set Σ of keys and foreign keys is said to be *primary* iff for any relation R, (1) there is at most one set $X \subseteq Att(R)$ such that $R[X] \to R$ is in Σ (either as a key or as part of a foreign key); (2) if $R[X] \to R$ is in Σ, then Σ does not include $R'[Y] \subseteq R[Z]$ for any relation schema R' and $Z = Att(R)$. We also need another rule: $R[Ztt(R)] \to R$ for any relation R. Let us refer to the set consisting of $\mathcal{I}_p$ rules and this rule as $\mathcal{I}_p^r$. Then $\mathcal{I}_p^r$ is sound and complete for implication and finite implication of primary keys and foreign keys in relational databases. The proof is similar to that of Theorem 3.6. ■

These results show that it is not feasible to reason about multi-attribute keys and foreign keys, even in the absence of DTDs. Although implication analysis of L constraints is decidable under the primary key restriction, its precise lower bound remains open.

To our knowledge, no previous work has studied the interaction between (primary) keys and foreign keys in relational databases, and the results established here extend relational dependency theory.

4. IMPLICATION OF PATH CONSTRAINTS

Navigation paths are commonly used in XML query languages (e.g., [17, 22, 34, 19]). Constraints defined in terms of paths are useful for, among other things, query optimization. Let us refer to such constraints as *path constraints*. In this section, we study implication of certain path constraints by basic XML constraints. More specifically, we examine three forms of path constraints, referred to as *path functional*, *inclusion* and *inverse constraints*. To do this, we first describe the notion of paths. We then define path constraints and investigate their implication by basic XML constraints. In this section we assume that basic XML constraints are expressed in L_{id}.

4.1. Paths

A *path* is a string in $(\mathbf{E} \cup \mathbf{A})^*$. In a data tree G, a path represents labels of nodes in a parent-child path. For example, paths in Figure 1 include `book.entry`, `book.author`, and `book.ref.to.author`. Observe that we treat attribute `to` as a reference from a `ref` element to `entry` elements.

To be precise, we give a formal definition of paths. Let Σ be a finite set of L_{id} constraints. Referring to data trees that satisfy Σ, we specify *the set of paths associated with an element type* τ, denoted by $paths(\tau)$. Moreover, for any $\alpha \in paths(\tau)$, we specify *the type of* α, denoted by $type(\tau.\alpha)$. These are defined as follows.

- The empty path ϵ is in $paths(\tau)$ and $type(\tau.\epsilon) = \tau$. We write $\tau.\epsilon$ simply as τ.
- Assume $\alpha \in paths(\tau)$ and $type(\tau.\alpha) = \tau_1$.
  - For any element type $\tau_2 \in \mathbf{E}$, $\alpha.\tau_2$ is in $paths(\tau)$ and $type(\tau.\alpha.\tau_2) = \tau_2$.
  - For any attribute $l \in \mathbf{A}$, $\alpha.l$ is in $paths(\tau)$. If there exists an element type τ_2 such that either $\Sigma \models \tau_1.l \subseteq \tau_2.id$ or $\Sigma \models \tau_1.l \subseteq_S \tau_2.id$, then $type(\tau.\alpha.l) = \tau_2$. Otherwise $type(\tau.\alpha.l) = S_\tau$.

Intuitively, if an attribute l is a foreign key of τ_1 elements referencing τ_2 elements, then we treat the l attribute of a τ_1 element as an “edge” (or “edges”) from the τ_1 element to τ_2 elements. As a result, a path may traverse across different subtrees of a data tree.

By the definition of L_{id}, one can easily verify that for any $\alpha \in paths(\tau)$, there is a unique τ' such that $type(\tau.\alpha) = \tau'$. That is, $type(\tau.\alpha)$ is well-defined. To show this, observe that for any element type τ_1 and attribute l, there is at most one element type τ_2 such that $\Sigma \models \tau_1.l \subseteq \tau_2.id$ or $\Sigma \models \tau_1.l \subseteq_S \tau_2.id$. This is because, by the definition of ID constraints in L_{id}, an ID value uniquely identifies an element within the entire document.

For example, referring to our `book` document depicted in Figure 1, assume that the following set Σ_0 of L_{id} constraints is given:

$$\texttt{entry.isbn} \to_{id} \texttt{entry}, \quad \texttt{ref.to} \subseteq_S \texttt{entry.isbn},$$

then we have $\texttt{entry.ref.to} \in paths(\texttt{book})$ and $type(\texttt{book.entry.ref.to}) = \texttt{entry}$.

The *length* of α in $paths(\tau)$, denoted by $|\alpha|$, is defined as follows: $|\alpha| = 0$ if $\alpha = \epsilon$; $|\alpha| = |\alpha_1| + 1$ if $\alpha = \alpha_1.\eta$, where η is either an element type or an attribute.

We call α in $paths(\tau)$ a *null path* if α is of the form $\alpha_1.\eta.\alpha_2$, where $\alpha_2 \neq \epsilon$ and $type(\tau.\alpha_1.l) = S_\tau$. Null paths are not meaningful when traversal following the paths is considered. Thus in the sequel we assume all paths considered are not null.

For any data tree G that satisfies Σ, any node $x \in ext(\tau)$ in G and path $\alpha \in paths(\tau)$, we define *the set of vertices reachable from x via α*, denoted by $nodes(x.\alpha)$, as follows.

- If $\alpha = \epsilon$, then $nodes(x.\alpha) = \{x\}$.
- If $\alpha = \beta.\tau_1$ and τ_1 is an element type, then for any $y \in nodes(x.\beta)$, the children of y labeled with τ_1 are in $nodes(x.\alpha)$.
- If $\alpha = \beta.l$ and l is an attribute, we consider the following cases.
  - If $type(\tau.\beta.l) = S_\tau$, then for any $y \in nodes(x.\beta)$ and any string value $z \in y.l$, z is in $nodes(x.\alpha)$.
  - If $type(\tau.\beta)$ is an element type τ_1 and $\Sigma \models \tau_1.l \subseteq \tau_2.id$ (resp. $\Sigma \models \tau_1.l \subseteq_S \tau_2.id$) for some element type τ_2, then for any $y \in nodes(x.\beta)$ and any vertex z labeled τ_2 such that $z.id = y.l$ (resp. $z.id \in y.l$), z is in $nodes(x.\alpha)$.

We use $ext(\tau.\alpha)$ to denote the set of nodes reachable from τ elements by following α, i.e, $ext(\tau.\alpha) = \bigcup_{x \in ext(\tau)} nodes(x.\alpha)$.

4.2. Path constraints

Next, we define path constraints and investigate their implication by L_{id} constraints. As in Section 3, we consider implication and finite implication that hold for any XML documents independent of DTDs. That is, given any finite set Σ of L_{id} constraints and a path constraint φ, whether for any (finite) data tree G, if $G \models \Sigma$ then $G \models \varphi$. Let us use $\Sigma \models \varphi$ and $\Sigma \models_f \varphi$ to denote implication and finite implication, respectively. We study the (finite) implication problem for path functional, inclusion and inverse constraints, defined as follows.

Let Σ be a finite set of L_{id} constraints and we consider data trees that satisfy Σ.

Path functional constraints. A *path functional constraint* φ is an expression of the form $\tau.\alpha \to \tau.\beta$, where τ is an element type, and $\alpha, \beta \in paths(\tau)$. A data tree G satisfies φ, denoted by $G \models \varphi$, iff in G,

$$\forall\, x, y \in ext(\tau)\ ((nodes(x.\alpha) \neq \emptyset \wedge nodes(x.\alpha) = nodes(y.\alpha)) \ \to \\ nodes(x.\beta) = nodes(y.\beta)).$$

That is, for all τ elements x and y, if they agree on the nodes reachable by following path α, then they must also agree on the nodes reached by following β. Note that only node equality is used when defining path functional constraints.

For example, referring to the `book` document given in Section 1, a path functional constraint φ_0 is $\texttt{book.entry.isbn} \to \texttt{book.author}$, which states that the isbn of an entry of a book determines the authors of the book.

Suppose that the `book` document satisfies the set Σ_0 of L_{id} constraints given in Section 4.1. A natural question is whether the document must also satisfy φ_0. In general, given a finite set Σ of L_{id} constraints, we want to know whether some path functional constraint φ is (finitely) implied by Σ. That is, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$). We refer to this question as (finite) implication of path functional constraints by L_{id} constraints. In our `book` example, $\Sigma_0 \models \varphi_0$ ($\Sigma_0 \models_f \varphi_0$) indeed holds, which is an instance of the (finite) implication problem for path functional constraints by L_{id} constraints.

About these implication and finite implication problems we have the following result.

THEOREM 4.1. *For any finite set Σ of L_{id} constraints and any path functional constraint φ, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) is decidable in $O(|\Sigma|\,|\varphi|)$ time, where $|\Sigma|$ and $|\varphi|$ are the lengths of Σ and φ, respectively.*

Proof. We first define a notion of key paths. Let Σ be a finite set of L_{id} constraints, τ be an element type and α be in $paths(\tau)$. We call α a *key path* of τ w.r.t. Σ if one of the following conditions is satisfied. (1) $\alpha = \epsilon$; (2) $\alpha = \alpha_1.\tau_1$, α_1 is a key path of τ and τ_1 is an element type; (3) $\alpha = \alpha_1.l$, α_1 is a key path of τ, $type(\tau.\alpha_1) = \tau_1$ and moreover, $\Sigma \models \tau_1.l \rightarrow \tau_1$, i.e., l is a key attribute of τ_1. Observe that by the ID-Key rule in $\mathcal{I}_{id}$ and Theorem 3.1, if l is the ID of τ_1, then it is also a key attribute of τ_1. For example, `entry.isbn` is a key path of `book` whereas `entry.refer.to` is not. Intuitively, for any $x, y \in ext(\tau)$ in a data tree G satisfying Σ, if x and y agree on some nodes reachable by following a key path, then x and y must be the same node, because in G, there is at most one parent-child path from a node to another. This can be easily verified by induction on the length of a key path.

To prove Theorem 4.1, it suffices to show the following claim.

Claim: Let $\varphi = \tau.\alpha \rightarrow \tau.\beta$. Then $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) iff either (1) $\beta = \alpha.\alpha'$, or (2) $\alpha = \rho.\alpha'$, $\beta = \rho.\beta'$, ρ is either a list of single-valued attributes or a key path of τ, and α' is a key path of $type(\tau.\rho)$ w.r.t. Σ.

For if it holds, then one can determine whether $\Sigma \models \varphi$ and $\Sigma \models_f \varphi$ by checking these conditions, which can be done in $O(|\varphi|\,|\Sigma|)$ time, by Theorem 3.1.

We next show the claim.

If $\beta = \alpha.\alpha'$, then by the definitions of path functional constraints and the notation $nodes(x.\beta)$, $G \models \varphi$ for any data tree G. Now suppose that there is ρ such that $\alpha = \rho.\alpha'$, $\beta = \rho.\beta'$, and α' is a key path of $type(\tau.\rho)$. Consider a data tree G that satisfies Σ. If ρ is a key path of τ, then α is also a key path of τ. In this case, for any $x, y \in ext(\tau)$, if $nodes(x.\alpha) = nodes(y.\alpha)$ in G, then $x = y$ by the discussion above. If ρ is a list of single-valued attributes, then for any $x, y \in ext(\tau)$, there exist unique x_1, y_1 in G such that $nodes(x.\rho) = \{x_1\}$ and $nodes(y.\rho) = \{y_1\}$. If $nodes(x.\alpha) = nodes(y.\alpha)$, then we have $x_1 = y_1$ since α' is a key path of $type(\tau.\rho)$. Thus $nodes(x.\beta) = nodes(x_1.\beta') = nodes(y_1.\beta') = nodes(y.\beta)$. Hence $\Sigma \models \varphi$ and $\Sigma \models_f \varphi$.

For the other direction of the claim, suppose that the conditions of the claim are not satisfied. We show that there is a finite data tree $G = (V, elem, att, root)$ such that $G \models \Sigma$ but $G \not\models \varphi$. That is, $\Sigma \not\models \varphi$ and $\Sigma \not\models_f \varphi$. To do so, we construct G as in the proof of Theorem 3.1, except the following. Let k be $|\alpha| + |\beta|$. For each element type τ' in $\Sigma \cup \varphi$, let V have at least $2k + 1$ elements of τ'. This is possible by the construction given in the proof of Theorem 3.1. Let x and y be two distinct τ elements, and assume $\alpha = \eta_1.\ldots.\eta_n$. If η_1 is an element type τ_1, then we choose two distinct τ_1 elements x_1 and y_1 in V and define $elem(x) = (\tau, [x_1])$ and $elem(y) = (\tau, [y_1])$. If η_1 is an attribute

l and $type(\tau.\eta_1) = \tau_1$, then we choose two distinct τ_1 elements x_1 and y_1 in V and let $att(x, l) = \{x_1.id\}$ and $att(y, l) = \{y_1.id\}$. By assumption, α is not a null path, thus η_1 has only the two cases discussed above. In the same way we process η_i and find x_i, y_i. Let x_n and y_n be the two nodes (or string values) finally reached. We let $x_n = y_n$. This may result in the collapse of x_{n-1} and y_{n-1} into the same node if η_n is an element type, and so on. But it will lead to the collapse of x and y into the same node as α is not a key path τ. In other words, there is $i \leq n$ such that $x_{i-1} \neq y_{i-1}$ where η_i is an attribute. We then process path β in the same way starting from x and y. Here we reuse nodes x_i, y_i created earlier only if necessary, i.e., only when β starts with $\eta_1 \ldots . \eta_j$ where η_i is a single-valued attribute. It is possible to find elements not used earlier since we have $2k + 1$ many τ elements for each element type τ. Let x'_m and y'_m be the two nodes (or string values) reached after processing β. Here we let $x'_m \neq y'_m$. This is possible only if none of the two conditions in the claim is satisfied, as we assumed. Finally, we create a distinct node $root$ and let $root$ has children x, y and all nodes that have not yet been assigned a parent. With slight modification to the construction given in the proof of Theorem 3.1, we can define G such that $G \models \Sigma$. By our construction, $nodes(x.\alpha) \neq \emptyset$ and $nodes(x.\alpha) = nodes(y.\alpha)$. However, Observe that $nodes(x.\beta) = \{x'_m\}$, $nodes(y.\beta) = \{y'_m\}$ and $x'_m \neq y'_m$. Thus $G \not\models \varphi$. Therefore, $\Sigma \not\models \varphi$ and $\Sigma \not\models_f \varphi$. This completes the proof of Theorem 4.1. ■

Path inclusion constraints. Along the same lines, we define a *path inclusion constraint* φ to be an expression of the form $\tau_1.\alpha_1 \subseteq \tau_2.\alpha_2$, where τ_1, τ_2 are element types, and $\alpha_1 \in paths(\tau_1)$, $\alpha_2 \in paths(\tau_2)$. A data tree G satisfies φ, denoted by $G \models \varphi$, iff in G,

$$ext(\tau_1.\alpha_1) \subseteq ext(\tau_2.\alpha_2).$$

That is, any nodes reachable by following path α_1 from τ_1 elements can also be reached by following α_2 from τ_2 elements. Again only node equality is needed here.

In particular, observe that when path α_2 is the empty path, path inclusion constraints have the form $\tau_1.\alpha_1 \subseteq \tau_2$. Constraints of this form describe typing information.

For example, for our `book` document, path inclusion constraints include:

$$\varphi_1 = \texttt{book.entry.ref.to} \subseteq \texttt{entry}, \;\; \varphi_2 = \texttt{book.entry.ref.to.title} \subseteq \texttt{entry.title}.$$

Recall the set Σ_0 of L_{id} constraints given above. We have $\Sigma_0 \models \varphi_1 \wedge \varphi_2$ ($\Sigma_0 \models_f \varphi_1 \wedge \varphi_2$), which are instances of the (finite) implication problem for path inclusion constraints by L_{id} constraints.

Implication and finite implication of path inclusion constraints by L_{id} constraints can also be determined efficiently.

THEOREM 4.2. *For any finite set Σ of L_{id} constraints Σ and any path inclusion constraint φ, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) is decidable in $O(|\varphi|\,|\Sigma|)$ time.*

Proof. Let $\varphi = \tau_1.\alpha_1 \subseteq \tau_2.\alpha_2$. First observe that if $type(\tau_1.\alpha_1) \neq type(\tau_2.\alpha_2)$, then as in the proof of Theorem 4.1, one can construct a finite data tree G such that $G \models \Sigma$ and $G \not\models \varphi$, following the construction given in the proof of Theorem 3.1. That is, $\Sigma \not\models \varphi$ and $\Sigma \not\models_f \varphi$. This can be determined in $O(|\varphi|\,|\Sigma|)$ time by Theorem 3.1. Below we assume $type(\tau_1.\alpha_1) = type(\tau_2.\alpha_2) = \tau$. Consider the following inference rules:

- IC1: if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \subseteq \tau'.id$ or $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \subseteq_S \tau'.id$, then $\tau.l \subseteq \tau'$.
- IC2: $\alpha.\tau \subseteq \tau$ for any path α and element type τ.
- IC3: if $\tau.\alpha \subseteq \tau'.\alpha'$ and $\tau'.\alpha' \subseteq \tau''.\alpha''$, then $\tau.\alpha \subseteq \tau''.\alpha''$.
- IC4: if $\tau.\alpha \subseteq \tau'.\alpha'$ then $\tau.\alpha.\beta \subseteq \tau'.\alpha'.\beta$ for any path β.
- IC5: if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \to \tau \wedge \tau'.l' \to \tau' \wedge \tau.l \subseteq \tau'.id \wedge \tau'.l' \subseteq \tau.id$, then $\tau \subseteq \tau'.l'$ and $\tau' \subseteq \tau.l$ in the finite case.

The condition of IC5 asserts that in the finite case, $|ext(\tau)| = |ext(\tau).l| = |ext(\tau').l'| = |ext(\tau')|$ and thus by $\tau'.l' \subseteq \tau.id$ and $\tau.l \subseteq \tau'.id$, we have $\tau \subseteq \tau'.l'$ and $\tau' \subseteq \tau.l$. Let $\mathcal{I}_{ic}$ denote the set consisting of IC1, IC2, IC3 and IC4, and $\mathcal{I}_{ic}^f$ be the set of all the rules above. To prove Theorem 4.2, it suffices to show the following claims.

Claim 1: $\Sigma \models \varphi$ iff $\Sigma \vdash_{\mathcal{I}_{ic}} \varphi$, and $\Sigma \models_f \varphi$ iff $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$, where $\Sigma \vdash_{\mathcal{I}_{ic}} \varphi$ ($\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$) denotes that there is an $\mathcal{I}_{ic}$-proof ($\mathcal{I}_{ic}^f$-proof) of φ from Σ.

Claim 2: $\Sigma \vdash_{\mathcal{I}_{ic}} \varphi$ iff $\alpha_1 = \alpha.\alpha_2$ and $type(\tau_1.\alpha) = \tau_2$. Moreover, $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$ iff there exist an element type τ' and paths α_1', α_2' and α such that $\alpha_1 = \alpha_1'.\alpha$, $\alpha_2 = \alpha_2'.\alpha$, $type(\tau_1.\alpha_1') = \tau'$ and $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau' \subseteq \tau_2.\alpha_2'$. In the latter case, we say that ρ is a *common suffix* of α_1 and α_2.

For if these hold, then whether $\Sigma \models \varphi$ can be determined by checking whether for some α, $\alpha_1 = \alpha.\alpha_2$ and $type(\tau_1.\alpha) = \tau_2$. By Theorem 3.1, this can be done in $O(|\varphi|\,|\Sigma|)$ time. To determine whether $\Sigma \models_f \varphi$, we only need to check the conditions for finite implication given in Claim 2. To do so, a dependency graph similar to the one given in the proof of Theorem 3.1 (2) can be constructed, and these conditions can be checked by inspecting the graph. Again this can be done in $O(|\varphi|\,|\Sigma|)$ time. Below we prove the claims for finite implication. The proof for implication is similar.

(1) We first show Claim 1. The soundness of $\mathcal{I}_{ic}^f$ can be verified by induction on the lengths of $\mathcal{I}_{ic}^f$-proofs. For completeness, we show by induction on $|\alpha_1|$ that if $\Sigma \models_f \varphi$ then $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$. More specifically, we show that if $\Sigma \not\vdash_{\mathcal{I}_{ic}^f} \varphi$, then there is a finite data tree G such that $G \models \Sigma$ but $G \not\models \varphi$.

Induction basis. We first consider $|\alpha_1| = 0$ and show the claim by induction on $|\alpha_2|$. If $|\alpha_2| = 0$, then for any data tree G, $G \models \varphi$ only if $\tau_1 = \tau_2$. But if $\tau_1 = \tau_2$ then $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$ by IC2, which contradicts our assumption. Assume the claim for $|\alpha_2| = k$. We show that the claim also holds for $\alpha_2 = \alpha_2'.\eta$, where η is an element type or an attribute. Assume $type(\tau_2.\alpha_2') = \tau'$. We can construct a finite data tree G in which there is a τ_1 element that is not in $ext(\tau_2.\alpha_2)$ except when (a) $\Sigma \models \tau' \subseteq \tau_2.\alpha_2'$, and (b) there exists an l attribute of τ_1 such that $\Sigma \vdash_{\mathcal{I}_{id}} \tau_1.l \to \tau' \wedge \tau'.\eta \to \tau' \wedge \tau_1.l \subseteq \tau'.id \wedge \tau'.\eta \subseteq \tau_1.id$. More specifically, G can be constructed by first creating an α_2 path emitting from a τ_2 element y, and then a τ_1 element x in the same way as in the proof of Theorem 4.1. One can ensure that $G \models \Sigma$ and $x \notin (y.\alpha_2)$ if either (a) or (b) is not satisfied. Given (a), by the induction hypothesis, $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau' \subseteq \tau_2.\alpha_2'$. Given (b), by IC5, we have $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1 \subseteq \tau'.\eta$. Thus by IC4 and IC3, we have $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$, which again contradicts our assumption. Hence Claim 1 holds when $|\alpha_1| = 0$.

Inductive step. Assume the statement for $|\alpha_1| = k$. We show that the statement also holds for $\alpha_1 = \alpha_1'.\eta_1$, where $|\alpha_1'| = k$ and η_1 is either an attribute or an element type. Assume $type(\tau_1.\alpha_1') = \tau_1'$. As in the proof of Theorem 4.1, with slight modification to the construction given in the proof of Theorem 3.1, one can construct a finite data tree G_1 such that $G_1 \models \Sigma$ and moreover, there exist a τ_1 element x, a τ_1' element y and a τ element z

such that $y \in nodes(x.\alpha_1')$ and $z \in nodes(y.\eta_1)$. If $|\alpha_2| = 0$, then $z \in ext(\tau_2.\alpha_2)$ only if either (a) $\eta_1 = \tau_2$, or (b) $\Sigma \vdash_{\mathcal{I}_{id}} \tau_1'.\eta_1 \subseteq \tau_2.id$ or $\Sigma \vdash_{\mathcal{I}_{id}} \tau_1'.\eta_1 \subseteq_S \tau_2.id$. However, if so then $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$ by IC2 in case (a) and by IC2, IC4, IC1 and IC3 in case (b). This contradicts our assumption. Next we assume $\alpha_2 = \alpha_2'.\eta_2$, where η_2 is either an attribute or an element type. We consider the following cases. (i) Assume $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1.\alpha_1' \subseteq \tau_2.\alpha_2'$. If $\eta_1 = \eta_2$, then by IC4, we have $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$, which contradicts the assumption. Assume $\eta_1 \neq \eta_2$. In G_1, we can make $z \notin ext(\tau_2.\alpha_2)$ except when $\Sigma \models \tau \subseteq \tau_2.\alpha_2$. By the induction hypothesis, if $\Sigma \models \tau \subseteq \tau_2.\alpha_2$ then $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau \subseteq \tau_2.\alpha_2$. Thus by IC2 and IC3, we have $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$. Again this contradicts the assumption. Thus the statement holds in this case. (ii) Assume $\Sigma \not\vdash_{\mathcal{I}_{ic}^f} \tau_1.\alpha_1' \subseteq \tau_2.\alpha_2'$. Then by the induction hypothesis, there exists a finite data tree G_2 such that $G_2 \models \Sigma$ and $G_2 \not\models \tau_1.\alpha_1' \subseteq \tau_2.\alpha_2'$. Hence there exist a τ_1 element x and a τ_1' element y such that $y \in nodes(x.\alpha_1')$ but $y \notin ext(\tau_2.\alpha_2')$. Again as in the proof of Theorem 4.1, with slight modification to the construction given in the proof of Theorem 3.1, one can construct a finite data tree G from G_2 such that $G \models \Sigma$, $G \models \tau_1.\alpha_1' \not\subseteq \tau_2.\alpha_2'$ and moreover, there is a τ element z such that $z \in nodes(y.\eta_1)$. We can make $z \notin ext(\tau_2.\alpha_2)$ except when $\Sigma \models \tau \subseteq \tau_2.\alpha_2$, which leads to contradiction as shown in the proof of (i). Thus the statement also holds for $|\alpha_1| = k+1$. This completes the proof of Claim 1.

(2) We next show Claim 2. First assume $\alpha_1 = \alpha_1'.\alpha$, $\alpha_2 = \alpha_2'.\alpha$, $type(\tau_1.\alpha_1') = \tau'$, and $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau' \subseteq \tau_2.\alpha_2'$. We show $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$. It suffices to show that if $type(\tau_1.\alpha_1') = \tau'$ then $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1.\alpha_1' \subseteq \tau'$; for if it holds, then $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$ by IC3 and IC4. This can be verified by induction on $|\alpha_1'|$. Indeed, if $|\alpha_1'| = 0$ then $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1.\alpha_1' \subseteq \tau'$ by IC2. Assume that it holds for $|\alpha_1'| = k$. We show the statement for $\alpha_1' = \alpha'.\eta$, where $|\alpha'| = k$ and η is an element type or an attribute. If η is an element type, then it must be τ' by the definition of $type(\tau_1.\alpha_1')$. Thus $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1.\alpha_1' \subseteq \tau'$ by IC2. If η is an attribute, assume $type(\tau_1.\alpha') = \tau_1'$. By the induction hypothesis, we have $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1.\alpha' \subseteq \tau_1'$. Again by the definition of $type(\tau_1.\alpha_1')$, we must have $\Sigma \vdash_{\mathcal{I}_{id}} \tau_1'.\eta \subseteq \tau'.id$ or $\Sigma \vdash_{\mathcal{I}_{id}} \tau_1'.\eta \subseteq_S \tau'.id$. By IC1, $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1'.\eta \subseteq \tau'$. Thus $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1.\alpha_1' \subseteq \tau'$ by IC4 and IC3. Conversely, we show if $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$ then there are paths α_1', α_2' and α such that $\alpha_1 = \alpha_1'.\alpha$, $\alpha_2 = \alpha_2'.\alpha$, $type(\tau_1.\alpha_1') = \tau'$, and $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau' \subseteq \tau_2.\alpha_2'$, by induction on the length of the $\mathcal{I}_{ic}^f$-proof. For induction basis, either IC1, IC2 or IC5 is applied in the proof and the statement obviously holds in these cases. Assume that the statement holds for $\mathcal{I}_{ic}^f$-proofs of lengths less than k. We next show that the statement also holds for $\mathcal{I}_{ic}^f$-proof of length k. Now suppose that $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$ is proved by using IC3, i.e., by first showing $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1.\alpha_1 \subseteq \tau_3.\alpha_3$, $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_3.\alpha_3 \subseteq \tau_2.\alpha_2$ and then by applying IC3. By the induction hypothesis, there are paths α_1', α_2', α_3', α_3'', α and α' such that $\alpha_1 = \alpha_1'.\alpha$, $\alpha_3 = \alpha_3'.\alpha$, $type(\tau_1.\alpha_1') = \tau_1'$, $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1' \subseteq \tau_3.\alpha_3'$, $\alpha_3 = \alpha_3''.\alpha'$, $\alpha_2 = \alpha_2'.\alpha'$, $type(\tau_3.\alpha_3'') = \tau_2'$ and $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_2' \subseteq \tau_2.\alpha_2'$. We need to consider the following cases. (a) $\alpha = \beta.\alpha'$. Observe that $type(\tau_1.\alpha_1'.\beta) = type(\tau_1'.\beta) = type(\tau_3.\alpha_3'') = \tau_2'$. Thus by $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_2' \subseteq \tau_2.\alpha_2'$, α' is a common suffix of α_1 and α_2, and thus the statement holds in this case. (b) $\alpha' = \beta.\alpha$. By $type(\tau_3.\alpha_3'') = \tau_2'$, we have $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_3.\alpha_3'' \subseteq \tau_2'$. Moreover, by $\alpha_3' = \alpha_3''.\beta$, $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1' \subseteq \tau_3.\alpha_3'$, IC4 and IC3, we have $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1' \subseteq \tau_2'.\beta$. Hence by $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_2' \subseteq \tau_2.\alpha_2'$, IC4 and IC3, we have $\Sigma \vdash_{\mathcal{I}_{ic}^f} \tau_1' \subseteq \tau_2.\alpha_2'.\beta$. Moreover, $type(\tau_1.\alpha_1') = \tau_1'$, $\alpha_1 = \alpha_1'.\alpha$ and $\alpha_2 = \alpha_2'.\beta.\alpha$, i.e., α is a common suffix of α_1 and α_2. Thus the statement also holds in this case. Next, suppose that $\Sigma \vdash_{\mathcal{I}_{ic}^f} \varphi$ is proved

by using IC4, i.e., by showing $\Sigma \vdash_{\mathcal{I}^f_{ic}} \tau_1.\alpha'_1 \subseteq \tau_2.\alpha'_2$, $\alpha_1 = \alpha'_1.\beta$, $\alpha_2 = \alpha'_2.\beta$ and then by applying IC4. By the induction hypothesis, there are paths α, α''_1 and α''_2 such that $\alpha'_1 = \alpha''_1.\alpha$, $\alpha'_2 = \alpha''_2.\alpha$, $type(\tau_1.\alpha''_1) = \tau'$ and $\Sigma \vdash_{\mathcal{I}^f_{ic}} \tau' \subseteq \tau_2.\alpha''_2$. Obviously, $\alpha_1 = \alpha''_1.\alpha.\beta$ and $\alpha_2 = \alpha''_2.\alpha.\beta$, i.e., $\alpha.\beta$ is a common suffix of α_1 and α_2. Thus the statement also holds for $\mathcal{I}^f_{ic}$-proofs of length k. This completes the proof of Claim 2. ■

Path inverse constraints. A more general form of inverse constraints is defined as follows. A *path inverse constraint* φ is an expression of the form $\tau_1.\alpha_1 \rightleftharpoons \tau_2.\alpha_2$, where τ_1, τ_2 are element types, $\alpha_1 \in paths(\tau_1)$ and $\alpha_2 \in paths(\tau_2)$. It states an inverse relationship between paths $\tau_1.\alpha_1$ and $\tau_2.\alpha_2$. We say that a data tree G satisfies φ, denoted by $G \models \varphi$, iff in G,

$$\forall\, x \in ext(\tau_1)\, \forall\, y \in ext(\tau_2)\ (y \in nodes(x.\alpha_1) \leftrightarrow x \in nodes(y.\alpha_2)).$$

As an example, let us consider element types `course`, `student`, and `teacher`. Suppose that `student` has an attribute `taking`, `teacher` has an attribute `teaching` and in addition, `course` has attributes `taken_by` and `taught_by`. Then φ given below is a path inverse constraint: `student.taking.taught_by` $\rightleftharpoons$ `teacher.teaching.taken_by`. Assume the following basic inverse constraints in L_{id}:

`student.taking` $\rightleftharpoons$ `course.taken_by`, `teacher.teaching` $\rightleftharpoons$ `course.taught_by`.

Then these (finitely) imply the path inverse constraint φ.

The complexity of implication and finite implication of path inverse constraints by L_{id} constraints is given as follows.

THEOREM 4.3. *For any finite set Σ of L_{id} constraints and any path inverse constraint φ, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) is decidable in $O(|\Sigma|\,|\varphi|)$ time.*

Proof. Assume that φ is $\tau_1.\alpha_1 \rightleftharpoons \tau_2.\alpha_2$, $type(\tau_1.\alpha_1) = \tau'_1$ and $type(\tau_2.\alpha_2) = \tau'_2$ w.r.t. Σ. First observe that if τ'_1, τ'_2 are some element types but $\tau'_1 \neq \tau_2$ and $\tau'_2 \neq \tau_1$, then $ext(\tau_1.\alpha_1)$ and $ext(\tau_2)$ are disjoint, and $ext(\tau_2.\alpha_2)$ and $ext(\tau_1)$ are disjoint. In this case, by the definition of path inverse constraints, for any data tree G that satisfies Σ, we must have $G \models \varphi$. Otherwise, if either $\tau'_1 \neq \tau_2$ or $\tau'_2 \neq \tau_1$, one can always construct a finite data tree G such that $G \models \Sigma$ but $G \not\models \varphi$. These cases can be easily determined by examining types of the paths involved, and can be done in $O(|\Sigma|\,|\varphi|)$ time by Theorem 3.1. Below we assume $\tau'_1 = \tau_2$ and $\tau'_2 = \tau_1$. In this case, let $\mathcal{I}_{inv}$ be the set consisting of the following inference rules:

- Inv1: $\tau \rightleftharpoons \tau$.
- Inv2: if $\tau_1.\alpha'_1 \rightleftharpoons \tau_2.\alpha'_2$ and $\Sigma \vdash_{\mathcal{I}_{id}} \tau_2.l_2 \rightleftharpoons \tau_3.l_3$, then $\tau_1.\alpha'_1.l_2 \rightleftharpoons \tau_3.l_3.\alpha'_2$.

To prove Theorem 4.3, it suffices to show the following claim:

Claim: $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) iff $\Sigma \vdash_{\mathcal{I}_{inv}} \varphi$, i.e., there is an $\mathcal{I}_{inv}$-proof of φ from Σ.

For if it holds, then one can determine whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) by examining whether each corresponding pair of labels in α_1 and α_2 has an inverse relationship. By Theorem 3.1, this can be done in $O(|\Sigma|\,|\varphi|)$ time.

We next show the claim. If $\Sigma \vdash_{\mathcal{I}_{inv}} \varphi$, then by induction on the length of the $\mathcal{I}_{inv}$-proof one can verify that $\Sigma \models \varphi$ and $\Sigma \models_f \varphi$. Conversely, assume $\Sigma \not\vdash_{\mathcal{I}_{inv}} \varphi$, we construct a finite tree G such that $G \models \Sigma$ but $G \not\models \varphi$. In other words, $\Sigma \not\models \varphi$ and $\Sigma \not\models_f \varphi$. We give the construction of G as follows. As in the proof of Theorem 4.1, we can generalize the construction given in the proof of Theorem 3.1 to define a finite data tree G_1 such that $G_1 \models \Sigma$ and in addition, there are τ_1 element x and τ_2 element y such that $nodes(x.\alpha_1) = \{y\}$ if $|\alpha_1| \neq 0$. We show that G can be constructed by modifying G_1, by induction on $|\alpha_1|$.
Induction basis. First consider $|\alpha_1| = 0$. If $\tau_1 = \tau_2$ and $|\alpha_2| = 0$, then $\Sigma \vdash_{\mathcal{I}_{inv}} \varphi$ by the Inv1 rule in $\mathcal{I}_{inv}$, which contradicts our assumption. If $|\alpha_2| = 0$ but $\tau_2 \neq \tau_1$, then $type(\tau_2.\alpha_2) = \tau_2 \neq \tau_1$, which contradicts our assumption. If $|\alpha_2| \neq 0$, then we can modify the children or attribute definition of the nodes in the path α_2 from y such that $x \in nodes(y.\alpha_2)$ but $x \neq y$ (i.e., $y \notin nodes(x.\alpha_1)$), without affecting the satisfaction of Σ. This can be verified by a straightforward induction on $|\alpha_2|$. Let G be G_1 with this modification. Then $G \models \Sigma$ but $G \not\models \varphi$. Thus the claim holds for $|\alpha_1| = 0$.
Inductive step. Assume the claim for $|\alpha| = k$. We show that the claim also holds for $\alpha_1 = \alpha_1'.\eta_1$, where $|\alpha_1'| = k$ and η_1 is either an element type or an attribute. Assume $type(\tau_1.\alpha_1') = \tau$. If $\alpha_2 = \epsilon$, then the argument for $|\alpha_1| = 0$ can show that there is a finite tree G such that $G \models \Sigma$ but $G \not\models \varphi$. Now assume $\alpha_2 = \eta_2.\alpha_2'$, where η_2 is either an element type or an attribute. Assume $type(\tau_2.\eta_2) = \tau'$. We consider the following cases. (1) If $\tau \neq \tau'$, then we can modify the children or attribute definition of y in G_1 such that $nodes(y.\eta_2) = \{z\}$ but $x \notin nodes(z.\alpha_2')$ without affecting the satisfaction of Σ, where z is a τ' element. Again, this can be verified by induction on $|\alpha_2'|$ with slight modification to the construction given in the proof of Theorem 3.1. Let G be G_1 with this modification. Then $G \models \Sigma$ but $G \not\models \varphi$. (2) Assume $\tau = \tau'$. By assumption, $\Sigma \not\vdash_{\mathcal{I}_{inv}} \varphi$. Thus either $\Sigma \not\vdash_{\mathcal{I}_{inv}} \tau_1.\alpha_1' \rightleftharpoons \tau.\alpha_2'$ or $\Sigma \not\vdash_{\mathcal{I}_{id}} \tau.\eta_1 \rightleftharpoons \tau_2.\eta_2$, since otherwise the Inv2 rule in $\mathcal{I}_{inv}$ shows $\Sigma \vdash_{\mathcal{I}_{inv}} \varphi$. First assume $\Sigma \not\vdash_{\mathcal{I}_{inv}} \tau_1.\alpha_1' \rightleftharpoons \tau.\alpha_2'$. By the induction hypothesis, there is a finite data tree G_2 such that $G_2 \models \Sigma$ but $G_2 \not\models \tau_1.\alpha_1' \rightleftharpoons \tau.\alpha_2'$. Thus there exist τ_1 element x and τ element z such that either $x \in nodes(z.\alpha_2')$ but $z \notin nodes(x.\alpha_1')$, or $z \in nodes(x.\alpha_1')$ but $x \notin nodes(z.\alpha_2')$. We modify G_2 such that $nodes(z.\eta_1) = \{y\}$ and $nodes(y.\eta_2) = \{z\}$ for some τ_2 element y without affecting the satisfaction of Σ. This is possible because there are more than one τ (resp. τ_2) elements by the construction given in the proof of Theorem 3.1, and thus we can switch the assignment of children or attribute of z (resp. y) with other elements. Let G be G_2 with this modification. Then $G \models \Sigma$ but $G \not\models \varphi$. Next assume $\Sigma \vdash_{\mathcal{I}_{inv}} \tau_1.\alpha_1' \rightleftharpoons \tau.\alpha_2'$ but $\Sigma \not\vdash_{\mathcal{I}_{id}} \tau.\eta_1 \rightleftharpoons \tau_2.\eta_2$. Consider a node $z \in nodes(x.\alpha_1')$ in G_1, which must exist by our construction of G_1. We can find another τ element z' such that $nodes(z.\eta_1) = \{y\}$ and $nodes(y.\eta_2) = \{z'\}$, where y is the τ_2 element in $nodes(x.\alpha_1)$ as given in the construction of G_1 above. By the same argument given above, this is possible without affecting the satisfaction of Σ. Let G be G_1 with this modification. Then $G \models \Sigma$ but $G \not\models \varphi$. Thus the claim holds for $|\alpha_1| = k+1$. This completes the proof of Theorem 4.3. ■

5. CONCLUSION

We have proposed an extension of XML DTDs that specifies both syntactic structure and integrity constraints for XML data. The semantics of XML documents is captured with simple key, foreign key and inverse constraints. We have introduced several classes of constraints useful either for specifying native XML documents or for preserving the semantics of data originating in structured databases. In addition, these constraints improve the XML reference mechanism with typing and scoping. We have investigated the implication and finite implication problems for these basic XML constraints, and established a number of complexity and axiomatizability results. These results not only are useful for XML applications, but also extend relational dependency theory, notably, on the interaction between (primary) keys and foreign keys. We have also studied path functional, inclusion and inverse constraints and their implication by basic XML constraints.

It should be mentioned that we only investigated constraint implication that generally holds independently of DTDs. This allows us to study XML documents that do not come with a DTD, and simplifies our proofs. As indicated in [12], integrity constraints may interact with schema (DTDs) and the interaction may not be simple. As a result, constraint implication may have widely different complexities in the presence and absence of a schema, and our proof techniques no longer apply in the typed context. Recently, (finite) implication of XML constraints has been studied [24] in the presence of DTDs.

On the theoretical side, a number of questions are still open. First, it can be shown that (finite) implication of multi-attribute primary keys and foreign keys is in PSPACE. Can this be tested more efficiently? Second, we only investigated implication of path constraints by basic constraints. Implication of path constraints by path constraints has not been settled. Third, more expressive key and foreign key constraints have been proposed for XML in, e.g., XML Schema [35]. The implication and finite implication problems associated with these constraints are still open, especially when they are considered in the presence of DTDs or types for XML data. On the practical side, we believe that the approach proposed here is promising. The basic constraints are simple and important enough to assume that they could be adopted by the XML designer and maintained by the system. One topic for future work is to study constraints in data integration [18], which is an important application of XML. In this context, natural questions are how constraints propagate through integration programs, and how they can help in verifying the correctness of the programs. Furthermore, we only proposed three independent languages, while data integration would require a framework that encompass all of these constraints.

ACKNOWLEDGMENT

We thank Peter Buneman, Richard Hull, Leonid Libkin and Frank Neven for helpful discussions. We would also like to thank the referees for valuable comments. Wenfei Fan is currently on leave from Temple University and is supported in part by NSF grant IIS 00-93168.

REFERENCES

1. S. Abiteboul, S. Cluet, T. Milo, P. Mogilevsky, J. Siméon, and S. Zohar. Tools for data translation and integration. *IEEE Data Engineering Bulletin*, 22(1):3–8, 1999.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. S. Abiteboul and P. C. Kanellakis. Object identity as a query language primitive. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 159–173, Portland, Oregon, June 1989.
4. S. Abiteboul and V. Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences (JCSS)*, 58(4):429–452, 1999.
5. J. Barwise. On moschovakis closure ordinals. *Journal of Symbolic Logic*, 42:292–296, 1977.
6. 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.

7. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.
8. A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.
9. 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/`.
10. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. In *Proceedings of International World Wide Web Conference (WWW)*, pages 201–210, 2001.
11. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about keys for XML. In *Proceedings of International Workshop on Database Programming Languages (DBPL)*, 2001.
12. 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.
13. P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences (JCSS)*, 61(2):146–193, 2000.
14. 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.
15. R. G. Cattell. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.
16. J. Clark and S. DeRose. XML Path Language (XPath). W3C Working Draft, Nov. 1999. `http://www.w3.org/TR/xpath`.
17. J. Clarke. XSL transformations (XSLT). W3C Recommendation, Nov. 1999. `http://www.w3.org/TR/xslt`.
18. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 177–188, Seattle, Washington, June 1998.
19. S. Cluet and J. Siméon. YATL: a functional and declarative language for XML. Draft manuscript, Mar. 2000.
20. S. S. Cosmadakis, P. C. Kanellakis, and M. Y. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *Journal of ACM*, 37(1):15–46, Jan. 1990.
21. A. Davidson, M. Fuchs, M. Hedin, M. Jain, J. Koistinen, C. Lloyd, M. Maloney, and K.Schwarzhof. Schema for Object-Oriented XML 2.0. W3C Note, July 1999. `http://www.w3.org/TR/NOTE-SOX`.
22. A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. XML-QL: A query language for XML. W3C Note, Aug. 1998. `http://www.w3.org/TR/NOTE-xml-ql`.
23. A. Deutsch, L. Popa, and V. Tannen. Physical data independence, constraints, and optimization with universal plans. In *Proceedings of International Conference on Very Large Databases (VLDB)*, pages 459–470, Edinburgh, Scotland, Sept. 1999.
24. W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 114–125, Santa Barbara, California, May 2001.
25. M. F. Fernandez, J. Siméon, and P. Wadler. A semi-monad for semi-structured data. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 263–300, London, UK, Jan. 2001.
26. D. Florescu, L. Raschid, and P. Valduriez. A methodology for query reformulation in CIS using semantic knowledge. *International Journal of Cooperative Information Systems (IJCIS)*, 5(4):431–468, 1996.
27. E. Grädel, P. G. Kolaitis, and M. Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, Mar. 1997.
28. C. Hara and S. Davidson. Reasoning about nested functional dependencies. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 91–100, Philadephia, Pennsylvania, May 1999.
29. M. Ito and G. E. Weddell. Implication problems for functional constraints on databases supporting complex objects. *Journal of Computer and System Sciences (JCSS)*, 50(1):165–187, 1995.
30. O. Lassila and R. R. Swick. Resource Description Framework (RDF) model and syntax specification. W3C Recommendation, Feb. 1999. `http://www.w3.org/TR/REC-rdf-syntax/`.
31. 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`.
32. F. Neven. Extensions of attribute grammars for structured document queries. In *Proceedings of International Workshop on Database Programming Languages (DBPL)*, pages 99–116, 1999.
33. L. Popa. *Object/Relational Query Optimization with Chase and Backchase.* PhD thesis, University of Pennsylvania, 2000.
34. J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). Workshop on XML Query Languages, 1998.
35. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema. W3C Working Draft, May 2001. `http://www.w3.org/XML/Schema`.
36. J. D. Ullman. *Database and Knowledge Base Systems*. Computer Science Press, 1988.