Integrity Constraints for XML

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

Jérôme Siméon
Bell Laboratories
simeon@research.bell-labs.com

Abstract

Integrity constraints are useful for semantic specification, query optimization and data integration. The ID/IDREF mechanism provided by XML DTDs relies on a simple form of constraint to describe references. Yet, this mechanism is not sufficient to express semantic constraints, such as keys or inverse relationships, or stronger, object-style references. In this paper, we investigate integrity constraints for XML, both for semantic purposes and to improve its current reference mechanism. We extend DTDs with several families of constraints, including key, foreign key, inverse constraints and constraints specifying the semantics of object identities. These constraints are useful both for native XML documents and to preserve the semantics of data originating in relational or object databases. Complexity and axiomatization results are established for the (finite) implication problems associated with these constraints. These results also extend relational dependency theory on the interaction between (primary) keys and foreign keys. In addition, we investigate implication of more general constraints, such as functional, inclusion and inverse constraints defined in terms of navigation paths.

1 Introduction

XML [9] is designed to simplify information exchange between Web applications. It relies on a concrete syntax for annotated trees, which is very convenient to represent data from any source, but provides only limited semantic information. A number of recent proposals aim at recovering semantics in XML, following various approaches: type systems [6, 26, 17, 24], description logics [13], meta-data descriptions [23], etc. As some of these proposals [26, 24] point out, integrity constraints are important for specifying semantics. In addition, they are useful for query optimization [18], update anomaly prevention [2], and information preservation in data integration [1, 15]. In relational databases, an important application of integrity constraints is to model references, through keys and foreign keys.

The standard XML schema language (i.e. Document Type Definition or DTD) also supports a reference mechanism. The so-called ID (IDREF) attributes provide a means to uniquely identify (refer to) a given element in an XML document. The way ID/IDREF attributes operate resembles the key and foreign key mechanism used in relational databases. Yet, it is neither sufficient to specify semantic constraints such as keys or inverse relationships, nor powerful enough to model object-style references. In response to these problems, this paper investigates the use of integrity constraints in XML. More specifically, we make the following contributions:

- We introduce a model for XML data with schema and integrity constraints. We define L, L_{id} and L_u, three basic constraint languages that support both a reference mechanism and better semantics. Language L_u is a simple extension of the original ID/IDREF mechanism, sufficient for native XML documents. L_{id} and L can be used to capture semantic constraints when data originates in object-oriented and relational databases, respectively.
- We study 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.

This work is motivated by the need for integrity constraints arising from practical XML applications. So before we dive into a more formal presentation, we first illustrate these various application contexts and the deficiencies of the current ID/IDREF mechanism.

The ID/IDREF mechanism, constraints and references

Let us consider the following XML document in its now familiar syntax.

```
<?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>
      <text>...</text>
        <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>
```

This document contains information about books. For each book, it gives the isbn, title and publisher in an entry element, then the list of authors, the book content and a set of bibliographical references. This could correspond to the following DTD:

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

In a DTD, each element has an element type and an attribute type description. 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. This property must hold on the whole document for all ID attributes. An `IDREF(S)` annotation indicates a reference, i.e., it should contain a (set of) value(s) of the ID attribute(s) present in the document.

Observe that the ID/IDREF mechanism has similarities to both the identity-based notion of references from object-oriented databases [3] and keys/foreign keys from relational databases. Like object identifiers, ID attributes uniquely identify elements within the whole document. Because XML has a textual format, the reference semantics is obtained by implicit constraints that must hold on attribute values, in the spirit of keys and foreign keys. Yet, it captures neither the complete semantics of 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 across all the ID attributes in the document. This is a much stronger assumption, preventing other elements, e.g., `book` elements 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` element. 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` (`sid`) attribute as a key for `entry` (`section`) elements, (ii) add an inclusion constraint, corresponding to a foreign key, asserting that `ref.to` is a subset of `entry.isbn`. These can be expressed in our language L_u.

Capturing the semantics of legacy data

A large amount of XML data originates in legacy sources, notably relational and object databases. In these databases, keys, foreign keys and inverse relationships are common [2, 14]. These constraints convey a fundamental part of the original information that we do not want to lose. Consider, for instance, the following object-oriented schema (in ODL syntax [14]):

```
class Person
      { attribute String name;
        attribute String address;
        relationship set<Dept> in_dept
             inverse Dept::has_staff; }
class Dept
      { 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: `name` and `dname` are keys for the `Person` and `Dept` classes respectively, and there is an inverse relationship between `Person.in_dept` and `Dept.has_staff`.

When exporting this object database to XML, the following DTD could be generated, trying to preserve most 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] (see the `oid` attribute). However, references through IDREF are weaker: because IDREF attributes are "untyped", we no longer know that the `person.in_dept` attribute should reference departments. In addition, as in the previous example, keys are not precisely captured (here we cannot even use ID for `name` and `dname` as XML only allows one single ID attribute). Last, we have no way to express the inverse relationships. We want to overcome these limitations while preserving the semantics of the original notion of object identities. To do so, we can add the following constraints to the original DTD's semantics: (i) an inclusion constraint to specify that `person.in_dept` refers to departments only; (ii) `name` and `dname` as keys for persons and departments in addition to `oid`; (iii) an inverse constraint between `person.in_dept` and `dept.has_staff`. These can be expressed in our language L_{id}.

Last, assume that the following DTD is translated from a relational schema:

```
<!ELEMENT publishers (publisher*)>
<!ELEMENT publisher (pname,country,address)>
<!ELEMENT editors (editor*)>
<!ELEMENT editor (name,pname,country)>
```

We would also like to capture the semantic constraints from the relational database: (`pname`, `country`) is a key for relation publishers, `name` is a key for relation editors, and (`pname`, `country`) is a foreign key in editors referencing publishers. To do so, we need to be able to express constraints on sub-elements (and not only attributes), as well as the typical relational keys and foreign keys by means of multi-attributes. These are captured by our language L.

Implication problems for XML constraints and related work

There is a large body of work on integrity constraints in the relational context [2, 27] that we can try to exploit. An important remark is that ID and IDREF attributes are unary. Thus the work on unary inclusion and functional dependencies by Cosmadakis, Kanellakis and Vardi [16] is particularly relevant. However, because of the semantics of ID attributes, results from [16] are not directly applicable in the XML context. 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, especially those on L_{id} constraint implication, address these issues. As language L is designed to capture semantic constraints from relational databases, it fits more directly into the relational setting. However, to the best of our knowledge, implication problems for general/primary keys and foreign keys have never been addressed before.

XML documents can have arbitrarily nested structures (note that the `section` elements from 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.

Path constraints have been studied formally in [4, 10, 11, 12, 21, 22, 28]. The path constraint languages introduced in [4, 10, 11, 12] specify inclusions among certain sets of objects, and are studied for semistructured data and XML. They are generalizations of (unary) inclusion dependencies. Inverse constrains are also expressible in the languages of [10, 11, 12]. However, these languages cannot express key constraints. [21] studies extended functional dependencies for nested relations. Another generalization of functional dependencies, called path functional dependencies, has been investigated for a restricted object-oriented data model in [22, 28]. These generalizations of functional dependencies are capable of expressing neither foreign keys nor inverse constraints. Furthermore, they are studied in the context of data constrained by type systems. As shown by [11], the interaction between path constraints and type constraints is not simple. More precisely, path constraint implication has wildly different complexities in the presence and absence of type systems.

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 [20]. It should be mentioned that many constraints considered here are not expressible in FO^2, including foreign key constraints of L, inverse constraints of L_{id} and L_u, and (unary) key constraints of all three languages. These can be verified by using the 2-pebble Ehrenfeucht-Fraïssé (EF) style game [5]. As an example, consider the structures G and G' given in Figure 1. Using 2-pebble EF game, one can show that G and G' are equivalent in FO^2. However, they are distinguished by the unary key constraint $\varphi = \tau.l \rightarrow \tau$, i.e.,

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

This constraint asserts that for any τ-elements x, y, if x and y have the same l-attribute value, then they are equal. Observe that $G \models \varphi$ but $G' \not\models \varphi$. This shows that φ is not expressible in FO^2.

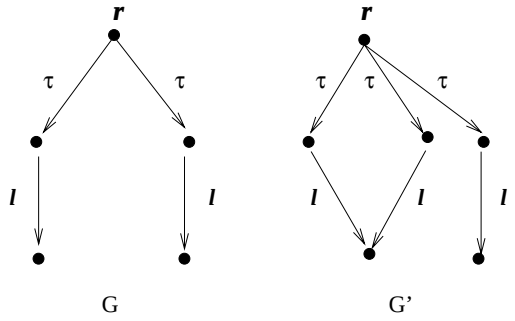

Figure 1: Structures distinguishable by key constraint

As shown by Borgida [8], FO^2 has equivalent expressive power as $DL \setminus \{trans, compose, at\_least, at\_most\}$, i.e., description logic omitting the transitive closure and composition constructors as well as counting quantifiers. As an immediate result, many XML constraints considered here 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 satisfiability and finite satisfiability 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 constraint languages L, L_{id} and L_u, and shows how they capture the examples above. Section 3 investigates implication, finite implication and axiomatization problems associated with these constraint languages. Section 4 studies implication of path constraints by basic constraints of L_{id}. Section 5 concludes the paper.

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 structure and semantics of the data. We formalize a DTD as a structural

specification and a set of integrity constraints. We will use various families of constraints to specify various extensions over DTD structures. The clear separation between structural and integrity constraints yields a robust framework. In particular, most of our results on integrity constraints are easily extensible to other XML type systems [6, 26].

2.1 Documents

We begin with the data model for representing XML documents. In the following, we assume the existence of a set **E** of element names, a set **A** of attribute names, and a set **S** of string values. We assume, without loss of generality, that all atomic values are of the same type, denoted by S. 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, i.e., a subset of **V**;
- *elem* is a function mapping vertices to their labels and children, i.e., from V to $\mathbf{E} \times F(\mathbf{S} \cup V)$;
- *att* is a partial function from vertex and attribute name pairs to a set of atomic values, i.e., from $V \times \mathbf{A}$ to $P(\mathbf{S})$;
- *root* is a distinguished element of V called the root of the tree.

■

Intuitively, V is the set of (internal) vertices of the tree. The function *elem* indicates for each node its label (an element name) and its list of children (either string values or sub-trees). The function *att* defines the attributes of each node. In XML, the attributes of an element are unordered and each contains a set of atomic values. We will use DTDs to specify the structure and semantics of data trees. Figure 2 shows a data tree depicting our book document given in Section 1. Note that the indications about the ID/IDREF semantics of attributes assume the corresponding DTD is available.

We will 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 $\{x.l \mid x \in ext(\tau)\}$. Furthermore, let X be a sequence of attributes $(l_1, ..., l_n)$. We use $x[X]$ to denote $(x.l_1, ..., x.l_n)$.

2.2 Document Type Definitions

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

Document Structure

In the literature [6, 13, 25], DTDs are often modeled as *Extended Context Free Grammars* (*ECFGs*), with elements as non-terminals, basic XML types as terminals and element definitions (with regular expressions) as productions. While ECFGs can specify the syntactic structure of elements, they fail to describe attributes, notably the ID/IDREF mechanism. Here we start from ECFGs [13] and extend them to capture 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*: $P(\tau) = \alpha$, where α is a regular expression, defined as follows:

  $$\alpha \quad ::= \quad \mathtt{S} \mid e \mid \epsilon \mid \alpha + \alpha \mid \alpha\,,\alpha \mid \alpha^*$$

  where S is the type of atomic values given above, $e \in E$, ϵ denotes the empty element, "+" stands for union, "," for the 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 or $\mathtt{S}^*$.

  We use $Att(\tau)$ to denote the set of attributes of τ, i.e., $\{l \in \mathbf{A} \mid R(\tau, l) \text{ is defined}\}$. An attribute l is called a *set-valued attribute of* τ if $R(\tau, l) = \mathtt{S}^*$, and *singled-valued* otherwise.
- *kind* is a partial function identifying the ID and IDREF attributes, from $E \times \mathbf{A}$ to $\{ID, IDREF\}$.

  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.

■

Document Constraints

Next, we introduce the three constraint languages that we will use in the remainder of the paper.

Language L**.** The first language, L, will be used to capture integrity constraints from relational databases. Therefore, it defines the classical key and foreign key constraints [2]. For a *DTD* Structure $S = (E, P, R, kind, r)$, a constraint of L has one of the following forms:

- *Key constraint:* $\tau[X] \rightarrow \tau$, where $\tau \in E$ and X is a *set* of single-valued attributes in $Att(\tau)$. It asserts

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

  i.e., X is a key for τ.
- *Foreign key constraint:* $\tau[X] \subseteq \tau'[Y]$, where $\tau, \tau' \in E$, X, Y are *sequences* of single-valued attributes in $Att(\tau)$ and $Att(\tau')$, respectively, and X and Y have the same length, which is not zero. In addition, Y is the key of τ', i.e., $\tau'[Y] \rightarrow \tau'$. It asserts that

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

  i.e., X is a foreign key of τ referring to τ'.

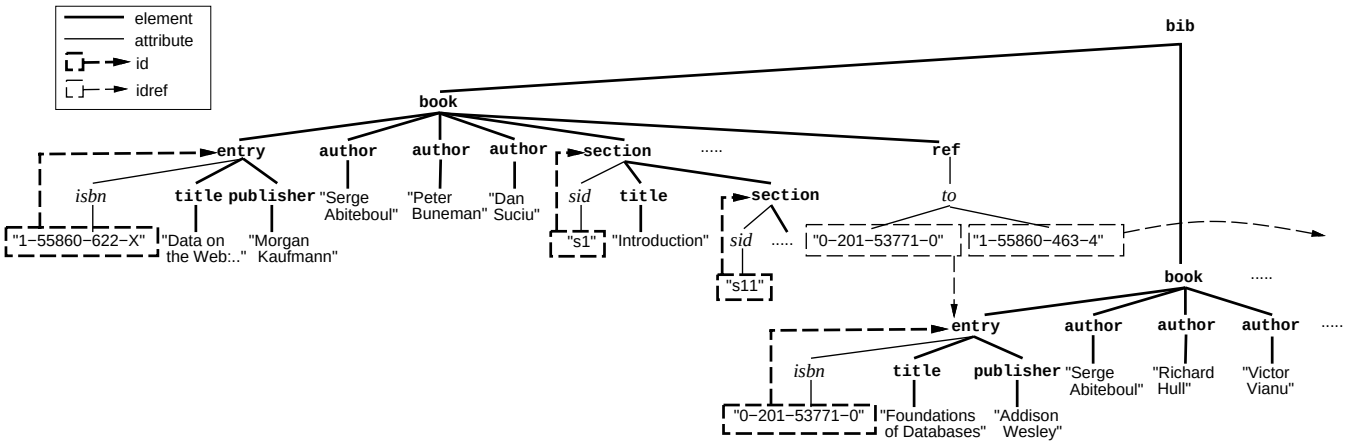

Figure 2: Graph representation of an XML document

Language L_u. The purpose of language L_u is to provide a minimal extension of DTDs that captures keys, references and inverse constraints. 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 and have an $IDREF$ kind. To be as close to the original XML as possible, we consider L constraints in which the sequences 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 and inverse constraints. Based on the considerations about the book document from the introduction, we then assume that the ID value of an element is unique among elements of the same type, rather than within the entire document.

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

- *Unary key constraint of L:* $\tau.l \rightarrow \tau$.
- *Unary foreign key constraint of L:* $\tau.l \subseteq \tau'.l'$.
- *Set-valued foreign key constraint:* $\tau.l \subseteq_S \tau'.l'$. Here $\tau, \tau' \in E$, l is a set-valued attribute of τ and l' is a single-valued attribute of τ' such that $\tau'.l' \rightarrow \tau'$. It asserts
$$\forall\, x \in ext(\tau)\ (x.l \subseteq ext(\tau').l').$$
- *Inverse constraint:* $\tau(l_k).l \rightleftharpoons \tau'(l'_k).l'$. Here we have $\tau, \tau' \in E$, l, l' are set-valued attributes of τ, τ', respectively, and l_k, l'_k are single-valued attributes of τ, τ' such that $\tau.l_k \rightarrow \tau$ and $\tau'.l'_k \rightarrow \tau'$. It asserts that $\tau.l \subseteq_S \tau'.l'_k$, $\tau'.l' \subseteq_S \tau_k.l_k$ and in addition,
$$\forall\, x \in ext(\tau)\, \forall\, y \in ext(\tau')\ \ (x.l_k \in y.l' \rightarrow y.l'_k \in x.l),$$
$$\forall\, x \in ext(\tau')\, \forall\, y \in ext(\tau)\ \ (x.l'_k \in y.l \rightarrow y.l_k \in x.l').$$
It should be noted we need to specify explicitly which keys are involved in an inverse constraint.

Constraints of L_u provide a simple reference mechanism that can be viewed as an extension of the one used in relational databases to the XML context.

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 identifiers from object databases. To do so, we keep the original semantics of ID attributes, whose value is unique within the whole document. Yet, we want to extend it with 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:* $\tau.l \rightarrow \tau$.
- *ID constraint:* $\tau.id \rightarrow_{id} \tau$, where $\tau \in E$ and there is $l \in Att(\tau)$ such that $kind(\tau, l) = ID$. It asserts
$$\forall\, x \in ext(\tau)\ \exists\, s \in \mathbf{S}\ (x.id = s \wedge \forall\, y\, (y.id = s \rightarrow x = y)).$$
- *Foreign key constraint:* $\tau.l \subseteq \tau'.id$. Here we have $\tau, \tau' \in E$, $l \in \mathbf{A}$, $kind(\tau, l) = IDREF$, l is a single-valued attribute of τ, and moreover, $\tau'.id \rightarrow_{id} \tau'$. It asserts
$$\forall\, x \in ext(\tau)\ (x.l \in ext(\tau').id).$$
- *Set-valued foreign key constraint:* $\tau.l \subseteq_S \tau'.id$. Here $\tau, \tau' \in E$, $l \in \mathbf{A}$, $kind(\tau, l) = IDREF$, and l is a set-valued attribute of τ. Moreover, $\tau'.id \rightarrow_{id} \tau'$. It asserts
$$\forall\, x \in ext(\tau)\ (x.l \subseteq ext(\tau').id).$$
- *Inverse constraint:* $\tau.l \rightleftharpoons \tau'.l'$. Here we have $\tau, \tau' \in E$, $l, l' \in \mathbf{A}$, $kind(\tau, l) = kind(\tau', l') = IDREF$, l and l' are both set-valued attributes, and moreover, τ and τ' have ID attributes, i.e., $\tau.id \rightarrow_{id} \tau$ and $\tau'.id \rightarrow_{id} \tau'$. It asserts that there is an inverse relationship between l and l'. Formally, that is: $\tau.l \subseteq_S \tau'.id$, $\tau'.l' \subseteq_S \tau.id$ and in addition,
$$\forall\, x \in ext(\tau)\, \forall\, y \in ext(\tau')\ \ (x.id \in y.l' \rightarrow y.id \in x.l)$$
$$\forall\, x \in ext(\tau')\, \forall\, y \in ext(\tau)\ \ (x.id \in y.l \rightarrow y.id \in x.l')$$

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 key constraints.

We will refer to the constraints of these languages as the *basic XML constraints*.

Finally, we define DTDs with constraints as follow.

Definition 2.3: A *Document Type Definition with constraints* (or DTD^C) is denoted by $D = (S, \Sigma)$, where:

- S is a DTD Structure,
- Σ is a set of basic XML constraints expressed in one of the constraint languages defined above.

■

2.3 Valid documents

Given a DTD^C, we can define the notion of valid documents, i.e., documents that conform to it.

Definition 2.4: Let $D = ((E, P, R, kind, r), \Sigma)$ be a DTD^C and $G = (V, elem, att, root)$ be a data tree. We say that G is *valid with respect to* D if and only if there is a mapping $\mu : V \cup \mathbf{S} \rightarrow E \cup \{\mathtt{S}\}$, such that:

- $\mu(root) = r$,
- for any s in $\mathbf{S}$, $\mu(s) = \mathtt{S}$,
- for any $v \in V$ such that $elem(v) = (e, [v_1, \ldots, v_n])$, with $P(e) = \alpha$, then $e = \mu(v)$ and $[\mu(v_1), \ldots, \mu(v_n)]$ belongs to the regular language defined by α,
- 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,
- $G \models \Sigma$.

■

We borrow the standard notion of models from logic. Let φ be a constraint and G a data tree. We use $G \models \varphi$ to denote that G *satisfies* φ, i.e., G is a model of φ. Let Σ be a set of constraints. We use $G \models \Sigma$ to indicate that G satisfies all the constraints in Σ. Note that the function *kind* does not appear in the definition of validity but is implicitly used in constraint satisfaction (only for language L_{id} though). Indeed, by requiring $G \models \Sigma$, we can view attributes as references to other elements.

2.4 Examples

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

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

E = { book, entry , section , ref }
P(book) = (entry, author*, section*, ref)
P(entry) = (title, publisher)
P(section) = (title, text + section)*
P(ref) = ϵ
R(entry, isbn) = S
R(section, sid) = S
R(ref, to) = S*
r = book
Σ = { entry.isbn $\rightarrow$ entry,
section.sid $\rightarrow$ section,
ref.to $\subseteq_S$ entry.isbn }

Note that 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 constraints in L_{id} to describe the structure of our person/dept object-oriented database: $D_o = ((E_o, P_o, R_o, kind_o, r_o), \Sigma_o)$, with:

E_o = { db, person , dept , name , address , dname }
P_o(db) = (person*, dept*)
P_o(person) = (name, address)
P_o(dept) = dname
R_o(person, oid) = S
R_o(person, in_dept) = S*
R_o(dept, oid) = S
R_o(dept, manager) = S
R_o(dept, has_staff) = S*
$kind_o$(person, oid) = ID
$kind_o$(person, in_dept) = $IDREF$
$kind_o$(dept, oid) = ID
$kind_o$(dept, manager) = $IDREF$
$kind_o$(dept, has_staff) = $IDREF$
r_o = db
Σ_o = { person.oid $\rightarrow_{id}$ person,
dept.oid $\rightarrow_{id}$ dept,
person.name $\rightarrow$ person,
dept.dname $\rightarrow$ dept,
person.in_dept $\subseteq_S$ dept.oid,
dept.manager $\subseteq$ person.oid,
dept.has_staff $\subseteq_S$ person.oid
dept.has_staff $\rightleftharpoons$ person.in_dept }

In Section 3.4, we will show how to extend L_{id} to specify constraints in terms of sub-elements. Thus we do not have to redefine name, dname as attributes. Note here we specify key constraints in addition to object identities.

Finally, consider the publisher DTD given in Section 1. We use the following constraints in language L to specify that (pname, country) is a key of publisher and a foreign key of editor referring to publisher.

$publisher[pname, country] \rightarrow publisher$
$editor[pname, country] \subseteq publisher[pname, country]$

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 defined in the last section. 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. At the end of the section, we extend our constraint languages to incorporate sub-elements.

We first give a formal description of the implication problems for XML constraints. Let C be either L_{id}, L_u or L, and $\Sigma \cup \varphi$ be a finite subset of C. Let D be a DTD^C, as described in Definition 2.3, such that Σ is the set of constraints in D. We use $\Sigma \models \varphi$ (resp. $\Sigma \models_f \varphi$) to denote that for any (resp. finite) data tree G of D, if $G \models \Sigma$ then $G \models \varphi$.

The *(finite) implication problem for C* is to determine, for any finite subset $\Sigma \cup \varphi$ of C, and for any DTD^C D in which Σ is the set of constraints, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$).

To simplify the discussion, in the sequel we assume that the element type definition of the root r of a DTD^C has the form $(\tau_1^*, ..., \tau_n^*)$, where for $i \in [1, n]$, τ_i is an element type. This ensures that any element type τ may have at least two elements. In practice, DTD^C's of this form are common.

We omit detailed proofs of the results in Sections 3 and 4 due to the lack of space, but we suggest the interested reader consult [19].

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 $\tau_1.id \subseteq \tau_2.id$ nor $\tau_1.id \subseteq_S \tau_2.id$; and moreover, for any $l \in Att(\tau_1)$, 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}. To distinguish ID and key constraints, we assume that DTD^C has at least two element types having ID attributes.

A finite axiomatization $\mathcal{I}_{id}$ for L_{id} is given below:

- FK-ID: $\dfrac{\tau.l \subseteq \tau'.id}{\tau'.id \rightarrow_{id} \tau'}$
- SFK-ID: $\dfrac{\tau.l \subseteq_S \tau'.id}{\tau'.id \rightarrow_{id} \tau'}$
- Inv-SFK-ID: $\dfrac{\tau.l \rightleftharpoons \tau'.l'}{\tau.l \subseteq_S \tau'.id \quad \tau'.l' \subseteq_S \tau.id}$
- Inv-commu: $\dfrac{\tau.l \rightleftharpoons \tau'.l'}{\tau'.l' \rightleftharpoons \tau.l}$

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} are decidable in linear time. ∎

Proof sketch: Let $\Sigma \cup \{\varphi\}$ be a finite subset of L_{id}.

(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 such that $G \models \Sigma$ and in addition, if $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$, then $G \not\models \varphi$.

To build G, we first define the *inference graph* G_I for $\Sigma \cup \{\varphi\}$. The vertices of G_I are element types and their attributes mentioned in $\Sigma \cup \{\varphi\}$. The edges are defined as follows: (1) there is an edge from each element type to each of its attributes; (2) there is an edge labeled "$\rightarrow_{\mathcal{I}_{id}}$" from the ID attribute of τ, $\tau.id$, to τ if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.id \rightarrow_{id} \tau$; (3) there is an edge labeled "$\rightarrow$" from the l attribute of τ, $\tau.l$, to τ if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \rightarrow \tau$; (4) there is an edge labeled with "$\supseteq$" from $\tau.id$ to $\tau'.l'$ if $\Sigma \vdash_{\mathcal{I}_{id}} \tau'.l' \subseteq \tau.id$; (5) there is an edge labeled "$\supseteq_S$" from $\tau.id$ to $\tau'.l'$ if $\Sigma \vdash_{\mathcal{I}_{id}} \tau'.l' \subseteq_S \tau.id$; (6) there is an undirected edge labeled "$\rightleftharpoons$" between $\tau.l$ and $\tau'.l'$ if $\Sigma \vdash_{\mathcal{I}_{id}} \tau.l \rightleftharpoons \tau'.l'$. The inference graph G_I has a simple structure because of the absence of transitivity rules. It depicts, among others, a topological order on element types and attributes. Based on the topological order, an algorithm can be developed to populate $ext(\tau)$ in G for each τ in $\Sigma \cup \{\varphi\}$, such that $G \models \Sigma$. Moreover, if $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$, the algorithm ensures that G contains certain elements according to different forms of φ, such that $G \not\models \varphi$.

(2) To show that L_{id} constraint implication is in linear time, we define another graph G_D for $\Sigma \cup \{\varphi\}$, called the *dependency graph*. As in G_I, the vertices of G_D represent element types and their attributes in $\Sigma \cup \{\varphi\}$. The edges capture dependencies between attributes as well as dependencies between element types and attributes. It can be shown that $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$ iff in G_D, there exists certain edge between the vertices representing element types and attributes in φ. Thus whether $\Sigma \vdash_{\mathcal{I}_{id}} \varphi$ can be determined by inspecting G_D. It takes $O(|\Sigma| + |\varphi|)$ time to construct and check G_D. Therefore, by (1), whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) can be determined in linear time. ∎

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 referred to by means of any one of its keys.

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

- UK-FK: $\dfrac{\tau.l \rightarrow \tau}{\tau.l \subseteq \tau.l}$
- UFK-K: $\dfrac{\tau.l \subseteq \tau'.l'}{\tau'.l' \rightarrow \tau'}$
- SFK-K: $\dfrac{\tau.l \subseteq_S \tau'.l'}{\tau'.l' \rightarrow \tau'}$
- UFK-trans: $\dfrac{\tau_1.l_1 \subseteq \tau_2.l_2 \quad \tau_2.l_2 \subseteq \tau_3.l_3}{\tau_1.l_1 \subseteq \tau_3.l_3}$
- USFK-trans: $\dfrac{\tau_1.l_1 \subseteq_S \tau_2.l_2 \quad \tau_2.l_2 \subseteq \tau_3.l_3}{\tau_1.l_1 \subseteq_S \tau_3.l_3}$
- Inv-SFK: $\dfrac{\tau(l_k).l \rightleftharpoons \tau'(l'_k).l'}{\tau.l \subseteq_S \tau'.l'_k \quad \tau'.l' \subseteq_S \tau.l_k}$
- Inv-commu: $\dfrac{\tau.l \rightleftharpoons \tau'.l'}{\tau'.l' \rightleftharpoons \tau.l}$

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 [16] that in relational databases, implication and finite implication of unary inclusion and functional dependencies are different problems. In other words, 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. More specifically, for finite implication of L_u, we also have cycle rules: for each positive integer k, there is a cycle rule Ck:

$$\frac{\tau_1.l_1 \rightarrow \tau_1 \quad \tau_2.l_2 \subseteq \tau_1.l'_1 \quad ... \quad \tau_k.l_k \rightarrow \tau_k \quad \tau_1.l_1 \subseteq \tau_k.l'_k}{\tau_1.l'_1 \rightarrow \tau_1 \quad \tau_1.l'_1 \subseteq \tau_2.l_2 \quad ... \quad \tau_k.l'_k \rightarrow \tau_k \quad \tau_k.l'_k \subseteq \tau_1.l_1}$$

Let $\mathcal{I}_u^f$ consist of $\mathcal{I}_u$ rules and Ck for each positive integer k. We can verify the following.

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

Proof sketch: We consider (2) only. The proof of (1) is similar and simpler.

Along the same lines of the proof of Theorem 3.1, we can verify that $\mathcal{I}_u^f$ is sound and complete for finite implication of L_u constraints. For any finite subset $\Sigma \cup \{\varphi\}$ of L_u, a similar inference graph G_I is also used here to construct a finite data tree G such that $G \models \Sigma$ and moreover, if $\Sigma \not\vdash_{\mathcal{I}_u^f} \varphi$, then $G \not\models \varphi$. However, because of the transitivity and cycle rules, the structure of G_I is more complex here than in the case of L_{id}. Here we need to define equivalence relations $\sim_{\rightarrow}$ and $\sim_{\subseteq}$ on attributes of element types:

$$\tau.l \sim_{\rightarrow} \tau.l' \quad \text{iff} \quad \Sigma \vdash_{\mathcal{I}_u^f} \tau.l \rightarrow \tau \text{ and } \Sigma \vdash_{\mathcal{I}_u^f} \tau.l' \rightarrow \tau$$

$$\tau.l \sim_{\subseteq} \tau'.l' \quad \text{iff} \quad \Sigma \vdash_{\mathcal{I}_u^f} \tau.l \subseteq \tau'.l' \text{ and } \Sigma \vdash_{\mathcal{I}_u^f} \tau'.l' \subseteq \tau.l$$

We topologically sort the equivalence classes with respect to $\sim_{\rightarrow}$ and $\sim_{\subseteq}$. Based on the topological order, we generate $ext(\tau)$ in G for each element type τ. To do so, we borrow the algorithm developed in the proof of the result of [16] mentioned above. The algorithm generates a relation that satisfies $\Sigma' \cup \{\neg\varphi'\}$, where $\Sigma' \cup \{\varphi'\}$ is a finite set of unary inclusion and functional dependencies such that φ' is not provable from Σ' by using certain inference rules. With modifications to deal with set-valued foreign key and inverse constraints, the algorithm can be used to generate elements of the date tree G described above. ∎

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.

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

Proof sketch: Again, we consider finite implication of L_u constraints. The analysis of implication of L_u constraints is similar. Let $\Sigma \cup \{\varphi\}$ be a finite subset of L_u. As in the proof of Theorem 3.1, to test whether $\Sigma \models_f \varphi$, we define a dependency graph G_D. It can be shown that $\Sigma \not\vdash_{\mathcal{I}_{id}} \varphi$ iff in G_D, there are certain paths between the vertices representing element types and attributes in φ. Observe that in contrast to the case of L_{id}, for L_u we need to check the existence of certain paths instead of edges. It takes $O(|\Sigma| + |\varphi|)$ time to construct and inspect G_D. Therefore, from Theorem 3.2 it follows that whether $\Sigma \models_f \varphi$ can be determined in linear time. The cycle rules show that L_u constraint implication does not have the finite model property, and thus is different from finite implication of L_u constraints. ∎

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 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 $l \neq l'$.

Interestingly, 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, which is a departure from [16].

Theorem 3.4: Under the primary key restriction, $\mathcal{I}_u$ is sound and complete for both implication and finite implication of L_u constraints. ∎

Proof sketch: The proof is similar to that of Theorem 3.2. Under the primary key restriction, the structure of the inference graph G_I becomes simpler. To see this, for an element type τ, let us consider the subgraph G_τ of G_I consisting of τ and its attributes. The graph G_τ has edges emanating from it only if τ has an attribute l such that $\Sigma \vdash_{\mathcal{I}_u} \tau.l \rightarrow \tau$. In addition, all the outgoing edges of G_τ are from the node $\tau.l$. Because of this, the cycle rules are not needed here. ∎

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

Corollary 3.5: In relational databases, the implication and finite implication problems for primary unary key and foreign key constraints coincide and are decidable in linear time. ∎

3.3 Implication of L constraints

Next, we investigate constraint language L. In L, one can express multi-attribute keys and foreign keys that are common in relational databases. As observed by [26], these constraints are also of practical interest for native XML data.

The analysis of L constraint implication, however, is not a trivial problem.

Theorem 3.6: The implication and finite implication problems for L are undecidable. ∎

This result also holds for relational databases.

Theorem 3.7: 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 corresponding proof). Below we give a proof sketch of Theorem 3.7.

Proof sketch: For a relation schema R, we use $Att(R)$ to denote the set of all attributes of R, and $Inst(R)$ to denote the set of instances of R. We use $X \rightarrow R$ to denote that X is a key for R, where X is a subset of $Att(R)$. Functional and inclusion dependencies are expressed as $R : X \rightarrow Y$ and $R_1[X] \subseteq R_2[Y]$, respectively, where $X \subseteq Att(R)$, R_1, R_2 are relation schemas, and $R_1[X], R_2[Y]$ denote sequences of attributes of R_1, R_2.

Let θ be either a functional or an inclusion dependency. We encode it in terms of key and foreign key constraints.

(1) $\theta = R : X \rightarrow Y$. Given θ, we define another relation schema R' such that $Att(R') = XY$. Observe that XY is a key for R'. We encode θ as:

$$\begin{aligned} \phi_1 &= X \rightarrow R' \\ \phi_2 &= R[XY] \subseteq R'[XY] \end{aligned}$$

It is easy to verify that there exists (finite) $I \in Inst(R)$ such that $I \models \theta$ iff there exist (finite) $I' \in Inst(R')$ and $I \in Inst(R)$ such that $(I, I') \models \phi_1 \wedge \phi_2$.

(2) $\theta = R_1[X] \subseteq R_2[Y]$. Let Z be a key for R_2, i.e., $Z \rightarrow R_2$. We define a relation schema R such that $Att(R) = YZ$.

Observe that YZ is a key for R_2. We encode θ as:

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

We can verify that there exist (finite) $I_1 \in Inst(R_1)$ and $I_2 \in Inst(R_2)$ such that $(I_1, I_2) \models \theta$ iff there exist (finite) $I \in Inst(R)$, $I_1 \in Inst(R_1)$ and $I_2 \in Inst(R_2)$ such that $(I, I_1, I_2) \models \phi_1 \wedge \phi_2 \wedge \phi_3$.

In general, given any finite set $\Sigma \cup \{\theta\}$ of functional and inclusion dependencies over schema $\mathbf{R}$, we can encode it as a finite set $\Sigma' \cup \Phi$ of key and foreign key constraints over schema $\mathbf{R'}$ in the way described above. In addition, there is $\mathbf{I} \in Inst(\mathbf{R})$ such that $\mathbf{I} \models \bigwedge \Sigma \wedge \neg\theta$ iff there is $\mathbf{I'} \in Inst(\mathbf{R'})$ such that $\mathbf{I'} \models \bigwedge \Sigma \wedge \neg \bigwedge \Phi$. Furthermore, $\mathbf{I}$ is finite iff $\mathbf{I'}$ is finite. Therefore, there is a reduction from the implication and finite implication problems for functional and inclusion dependencies. ∎

The undecidability result suggests that we examine L constraint implication under the primary key restriction. That is, we assume that for any element type $\tau \in E$, there is at most one subset X of single-valued attributes in $Att(\tau)$ such that $\tau[X] \rightarrow \tau$, and moreover, for any proper subset Y of X, $\tau[Y] \not\rightarrow \tau$. Furthermore, we cannot have both $\tau_1[X_1] \subseteq \tau[Y]$ and $\tau_2[X_2] \subseteq \tau[Y']$ if $Y \neq Y'$. Similarly, we cannot have both $\tau'[X] \subseteq \tau[Y]$ and $\tau[Y'] \rightarrow \tau$ if $Y \neq Y'$. When the primary key restriction is placed, we refer to L constraints as primary key and foreign key constraints.

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: $\dfrac{\tau[X] \rightarrow \tau}{\tau[X] \subseteq \tau[X]}$
- PFK-K: $\dfrac{\tau[X] \subseteq \tau'[Y]}{\tau'[Y] \rightarrow \tau'}$
- PFK-perm: for each sequence $i_1, i_2, ..., i_n$ of distinct integers in $[1, ..., n]$,

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

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

Theorem 3.8: Under the primary key restriction, $\mathcal{I}_p$ is sound and complete for both implication and finite implication of L constraints. ∎

Observe that under the primary key restriction, the implication and finite implication problems for L coincide.

Proof sketch: Soundness of $\mathcal{I}_p$ can be verified by induction on the lengths of $\mathcal{I}_p$-proofs. For the completeness of $\mathcal{I}_p$, it suffices to construct a finite data tree G for a finite subset $\Sigma \cup \{\varphi\}$ of primary key and foreign key constraints, such that $G \models \Sigma$ and moreover, if $\Sigma \not\vdash_{\mathcal{I}_p} \varphi$, then $G \not\models \varphi$. The construction of G takes advantage of the primary key restriction: for any $\tau_1[X_1] \subseteq \tau_2[X_2]$, X_2 is the only key for τ_2. This makes it easy to ensure that primary key constraints in Σ are satisfied while processing foreign key constraints of Σ. Based on this observation, a simple algorithm can be developed to populate $ext(\tau)$ in G for each element type τ. ∎

This result is also applicable to relational databases.

Corollary 3.9: In relational databases, the implication and finite implication problems for primary keys and foreign keys coincide and are decidable. ∎

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

3.4 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)>
```

It is perfectly reasonable to assume that `name` is a key for `person`. This was easily captured in our corresponding DTD specification (see D_o in Section 2.4), by including

$$person.name \rightarrow person$$

in the constraint set (Σ_o). This suggests that we extend the definition of key constraints in L_{id}. Let τ be specified by an element type definition $P(\tau) = \alpha$ and S be a sub-element of τ. We may specify constraints of the form

$$\tau.S \rightarrow \tau$$

if S is *a unique sub-element of* τ, i.e., for any $w \in L(\alpha)$, S occurs exactly once in w, where $L(\alpha)$ is the regular language defined by α. This is a syntactic restriction that can be checked by examining the DTD. It is easy to verify that the results established in Section 3.1 still hold for this extension.

Along the same lines, we extend the definitions of key and foreign key constraints in L_u and L to incorporate sub-elements. It can be verified that the results of Sections 3.2 and 3.3 also apply to the corresponding extensions. In the rest of the paper, we will allow key constraints to be specified in terms of sub-elements.

4 Implication of path constraints

Navigation paths are commonly used in XML query languages. Constraints defined in terms of paths are useful for, among others, 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 given in a DTD^C. 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 sequence of node labels. More specifically, let D be a DTD^C $((E, P, R, kind, r), \Sigma)$. In a data tree G of D, a path is a sequence of symbols in $E \cup \mathbf{A}$ that are labels of a sequence of nodes. For example, paths in Figure 2 include $book.entry$, $book.author$, $book.ref.to.author$. 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. For any element type $\tau \in E$, we define *the set of paths of* τ, denoted by $paths(\tau)$, and for any $\rho \in paths(\tau)$, we define *the type of* ρ, denoted by $type(\tau.\rho)$, as follows.

- $\epsilon \in paths(\tau)$ and $type(\epsilon) = \tau$.
- Assume $\rho \in paths(\tau)$ with $type(\tau.\rho) = \tau_1$.
  - For any $l \in Att(\tau_1)$, $\rho.l$ is in $paths(\tau)$. If there exists $\tau_2 \in E$ such that either $\Sigma \models \tau_1.l \subseteq \tau_2.id$ or $\Sigma \models \tau_1.l \subseteq_S \tau_2.id$, then $type(\tau.\rho.l) = \tau_2$. Otherwise $type(\tau.\rho.l) = \mathtt{S}$.
  - Suppose that the element type definition of τ_1 is $P(\tau_1) = \alpha$. For any $\tau_2 \in E \cup \{\mathtt{S}\}$ that occurs in α, $\rho.\tau_2$ is in $paths(\tau)$ and $type(\tau.\rho.\tau_2) = \tau_2$.

Here $Att(\tau)$ is the set of attributes of τ as defined in Definition 2.2. By the definition of L_{id}, one can easily verify that for any $\tau \in E$ and $\rho \in paths(\tau)$, there is a unique τ' such that $type(\tau.\rho) = \tau'$. That is, $type(\tau.\rho)$ is well-defined. To show this, observe that for any $\tau_1 \in E$ and $l \in Att(\tau_1)$, there is at most one $\tau_2 \in E$ 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 DTD^C D_o for our `person/dept` database given in Section 2.4, we have

$$db\ .\ person\ .\ in\_dept \in paths(db)$$
$$person\ .\ in\_dept\ .\ has\_staff \in paths(person)$$
$$type(db\ .\ person\ .\ in\_dept) = dept$$
$$type(person\ .\ in\_dept\ .\ has\_staff) = person$$

For any data tree G of D, any $x \in ext(\tau)$ in G and $\rho \in paths(\tau)$, we define *the set of vertices reachable from* x *via* ρ, denoted by $nodes(x.\rho)$, as follows.

- If $\rho = \epsilon$, then $nodes(x.\rho) = \{x\}$.
- If $\rho = \varrho.\tau_1$ and $\tau_1 \in E$, then for any $y \in nodes(x.\varrho)$, children of y labeled with τ_1 are in $nodes(x.\rho)$.
- For $\rho = \varrho.l$ and $l \in \mathbf{A}$, if $type(\tau.\varrho) = \mathsf{S}$, then for any $y \in nodes(x.\varrho)$ and $z \in y.l$, z is in $nodes(x.\rho)$. Otherwise we have $type(\tau.\varrho) = \tau_1$ and $\Sigma \models \tau_1.l \subseteq \tau_2.id$ (resp. $\Sigma \models \tau_1.l \subseteq_S \tau_2.id$) for some $\tau_2 \in E$. For any $y \in nodes(x.\varrho)$ and any vertex z labeled τ_2 such that $z.id = y.l$ (resp. $z.id \in y.l$), z is in $nodes(x.\rho)$.

We use $ext(\tau.\rho)$ to denote the set of nodes reachable from τ elements by following ρ, i.e,

$$ext(\tau.\rho) = \bigcup_{x \in ext(\tau)} nodes(x.\rho).$$

4.2 Path constraints

Next, we define path constraints and investigate their implication by L_{id} constraints.

Path functional constraints. Let D be a DTD^C as described in Definition 2.3: $((E, P, R, kind, r), \Sigma)$. A *path functional constraint* φ of D is an expression of the form

$$\tau.\rho \rightarrow \tau.\varrho,$$

where $\tau \in E$ and $\rho, \varrho \in paths(\tau)$. For any data tree G of D, we use $G \models \varphi$ to denote that in G,

$$\forall\, x, y \in ext(\tau)\ (nodes(x.\rho) = nodes(y.\rho) \rightarrow nodes(x.\varrho) = nodes(y.\varrho))$$

As an example, let us consider the `book` document given in Section 1. The constraint φ_0 below is an example of path functional constraints of the `book` DTD^C.

$$\varphi = book\ .\ entry\ .\ isbn \rightarrow book\ .\ author.$$

Constraint φ states that the isbn of the entry of a book determines the authors of the book.

Suppose a set Σ_0 of L_{id} constraints is specified in the `book` DTD^C, including

$$entry\ .\ isbn \rightarrow_{id} entry$$

Given that Σ_0 holds, one may ask whether φ_0 also holds. In general, given a DTD^C D with a set Σ of L_{id} constraints, we want to know whether some path functional constraint φ is implied by Σ. That is, whether for any data tree G of D, if $G \models \Sigma$ then $G \models \varphi$. We refer to this question as implication of path functional constraints by basic XML constraints.

About implication of path functional constraints by L_{id} constraints we have the following result.

Proposition 4.1: For any D, a DTD^C with a set of constraints Σ, and any path functional constraint φ, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) is decidable in $O(|\varphi|\,(|\Sigma| + |P|))$ time, where P is the set of element type definitions in D, and $|\Sigma|$, $|P|$, $|\varphi|$ are the lengths of Σ, P, φ, respectively. ■

Proof sketch: We first define the notion of path prefix. A path ρ_1 is called a *prefix* of path ρ if there exists ρ_2 such that $\rho = \rho_1.\rho_2$. The *longest common prefix* of paths ρ and ϱ is the prefix η of ρ and ϱ such that there is no ζ, $\zeta \neq \epsilon$ and $\eta.\zeta$ is a prefix of both ρ and ϱ.

Let $\varphi = \tau.\rho \rightarrow \tau.\varrho$, $\rho = \eta.\zeta$, η be the longest common prefix of ρ and ϱ, and $type(\tau.\eta) = \tau'$. To prove Proposition 4.1, it suffices to show the following claim.

Claim: $\Sigma \models \varphi$ iff ζ is a *key path* of τ', defined as follows: (1) ϵ is a key path of τ'; (2) suppose ρ' is a key path of τ' with $type(\tau'.\rho') = \tau_1$, then for any sub-element τ_2 of τ_1, $\rho'.\tau_2$ is a key path of τ' if $\Sigma \models \tau_1.\tau_2 \rightarrow \tau_1$; and moreover, if for some $l \in Att(\tau_1)$, either $\Sigma \models \tau_1.l \rightarrow \tau_1$, or $kind(\tau_1, l) = ID$ and $\Sigma \models \tau_1.id \rightarrow_{id} \tau_1$, then $\rho'.l$ is a key path of τ'.

If it holds, then one can determine whether $\Sigma \models \varphi$ by first finding the longest common prefix η of paths ρ and ϱ and then testing whether ζ is a key path of $type(\tau.\eta)$, where $\rho = \eta.\zeta$. By Theorem 3.1, these take $O(|\varphi|\,(|\Sigma| + |P|))$ time. The claim can be verified by induction on the length of ζ. ■

Path inclusion constraints. Along the same lines, given a DTD^C $D = ((E, P, R, kind, r), \Sigma)$, we define a *path inclusion constraint* φ of D to be an expression of the form

$$\tau_1.\rho_1 \subseteq \tau_2.\rho_2,$$

where $\tau_1, \tau_2 \in E$, $\rho_1 \in paths(\tau_1)$, and $\rho_2 \in paths(\tau_2)$. For any data tree G of D, $G \models \varphi$ means that in G,

$$ext(\tau_1.\rho_1) \subseteq ext(\tau_2.\rho_2).$$

For example, for our book DTD^C, path inclusion constraints include:

$$\begin{aligned} book\ .\ ref\ .\ to &\subseteq entry \\ book\ .\ ref\ .\ to\ .\ title &\subseteq entry\ .\ title \end{aligned}$$

In particular, observe that when $\rho_2 = \epsilon$, path inclusion constraints have the form $\tau_1.\rho_1 \subseteq \tau_2$. Constraints of this form describe typing information.

Implication of path inclusion constraints by L_{id} constraints is also decidable.

Proposition 4.2: For any D, a DTD^C with a set of constraints Σ, and any path inclusion constraint φ, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) is decidable in $O(|\varphi|\,(|\Sigma| + |P|))$ time, where P is the set of element type definitions in D, and $|\Sigma|$, $|P|$, $|\varphi|$ are the lengths of Σ, P, φ, respectively. ∎

Proof sketch: We first define some notions. An element type τ_2 is called the *unique parent* of τ_1 if τ_1 is in $P(\tau_2)$ but for any τ_3, τ_1 is not in $P(\tau_3)$ if $\tau_2 \neq \tau_3$. A path ρ is called the *unique ancestor path* from τ_2 to τ_1 if either (1) τ_2 is the unique parent of τ_1 and $\rho = \tau_1$, or (2) $\rho = \varrho.\tau_1$, $type(\tau_2.\varrho) = \tau'$, ϱ is the unique ancestor path from τ_2 to τ' and τ' is the unique parent of τ_1. We call τ_2 a *unique ancestor* of τ_1 if there is a unique ancestor path from τ_2 to τ_1. Observe that if there is a unique ancestor path ρ from τ_2 to τ_1 then $\tau_1 \subseteq \tau_2.\rho$ and $\tau_2.\rho \subseteq \tau_1$. We can find unique ancestors and unique ancestor paths of element types by constructing and checking the *element definition graph* G_P of DTD^C D. The vertices of G_P are element types of D. Element type definitions are depicted by green edges in G_P, and the unique parent relation is represented by red edges. The construction of G_P takes $O(P)$ time.

To determine whether $\Sigma \models \tau_1.\rho_1 \subseteq \tau_2.\rho_2$ we develop an algorithm based on the following rules:

- If ρ is the unique ancestor path from τ' to τ then $\tau \subseteq \tau'.\rho$ and $\tau'.\rho \subseteq \tau$.
- If $\Sigma \models \tau.l \subseteq \tau'.id$ or $\Sigma \models \tau.l \subseteq_S \tau'.id$, then $\tau.l \subseteq \tau'$.
- If $\tau.\rho \subseteq \tau'.\rho'$ and $\tau'.\rho' \subseteq \tau''.\rho''$, then $\tau.\rho \subseteq \tau''.\rho''$.
- If $\tau.\rho \subseteq \tau'.\rho'$ then $\tau.\rho.\varrho \subseteq \tau'.\rho'.\varrho$ for any path ϱ.

The algorithm traverses paths ρ_1 and ρ_2 in G_P starting from nodes representing τ_1 and τ_2. By Theorem 3.1, this takes at most $O(|\varphi|\,(|\Sigma| + |P|))$ time. ∎

Path inverse constraints. A more general form of inverse constraints is defined as follows. A *path inverse constraint* φ of a DTD^C $D = ((E, P, R, kind, r), \Sigma)$ is an expression of the form

$$\tau_1.\rho_1 \rightleftharpoons \tau_2.\rho_2,$$

where $\tau_1, \tau_2 \in E$, $\rho_1 \in paths(\tau_1)$ and $\rho_2 \in paths(\tau_2)$. It states an inverse relationship between paths ρ_1 and ρ_2. That is, for any data tree G of D, $G \models \varphi$ iff in G,

$$\begin{aligned} &\forall\, x \in ext(\tau_1)\, \forall\, y \in ext(\tau_2)\ (y \in nodes(x.\rho_1) \rightarrow \\ &\qquad\qquad x \in nodes(y.\rho_2)) \\ &\forall\, x \in ext(\tau_2)\, \forall\, y \in ext(\tau_1)\ (y \in nodes(x.\rho_2) \rightarrow \\ &\qquad\qquad x \in nodes(y.\rho_1)) \end{aligned}$$

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:

$$\begin{aligned} student\ .\ taking\ .\ taught\_by \quad &\rightleftharpoons \\ teacher\ .\ teaching\ .\ taken\_by& \end{aligned}$$

Assume the following basic inverse constraints:

$$\begin{aligned} student\ .\ taking \quad &\rightleftharpoons \quad course\ .\ taken\_by \\ teacher\ .\ teaching \quad &\rightleftharpoons \quad course\ .\ taught\_by \end{aligned}$$

Then these imply the path inclusion constraint φ.

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

Proposition 4.3: For any DTD^C D with a set Σ of L_{id} constraints and for any path inverse constraint φ, whether $\Sigma \models \varphi$ ($\Sigma \models_f \varphi$) is decidable in $O(|\varphi|\,(|\Sigma| + |P|))$ time, where P is the set of element type definitions in D, and $|\Sigma|$, $|P|$, $|\varphi|$ are the lengths of Σ, P and φ, respectively. ∎

Proof sketch: We determine whether $\Sigma \models \tau_1.\rho_1 \rightleftharpoons \tau_2.\rho_2$ by using the following rules:

- If $\Sigma \models \tau_2.\rho \subseteq \tau_1$ and $\Sigma \models \tau_1 \subseteq \tau_2.\rho$ then $\tau_1 \rightleftharpoons \tau_2.\rho$.
- If $\Sigma \models \tau_1.\rho_1 \rightleftharpoons \tau_2.\rho_2$ and $\Sigma \models \tau_2.\rho_2' \rightleftharpoons \tau_3.\rho_3$ then $\tau_1.\rho_1.\rho_2' \rightleftharpoons \tau_3.\rho_3.\rho_2$.

Using element definition graph described in the proof of Proposition 4.2, an algorithm can be developed for testing implication of inverse constraints by L_{id} constraints. The algorithm uses heavily a special case of the second rule: if $\Sigma \models \tau_1.l_1 \rightleftharpoons \tau_2.l_2$ and $\Sigma \models \tau_2.l_2' \rightleftharpoons \tau_3.l_3$ then $\tau_1.l_1.l_2' \rightleftharpoons \tau_3.l_3.l_2$. ∎

5 Conclusions

We have proposed a formalization for XML DTDs that specifies both the syntactic structure and integrity constraints. The semantics of XML documents is captured with simple key, foreign key and inverse constraints. We have introduced several families of constraints useful either for native 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 are not only useful for XML query optimization, 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.

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. On the practical side, the basic constraints provide a good compromise between expressiveness and complexity. An important application of XML is data integration [15]. In this context, important questions are how constraints propagate through integration programs, and how they can help in verifying the correctness of the programs?

Acknowledgements. We thank Peter Buneman, Richard Hull, Val Tannen, Scott Weinstein and Limsoon Wong for valuable comments and suggestions.

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*, volume 18, pages 159–173, Portland, Oregon, June 1989.

[4] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 122–133, Tucson, Arizona, May 1997.

[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)*, Lecture Notes in Computer Science, 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, W. Fan, and S. Weinstein. Path constraints on semistructured and structured data. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 129–138, Seattle, Washington, June 1998.

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

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

[13] 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.

[14] R. G. Cattell. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997.

[15] 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.

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

[17] 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`.

[18] 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.

[19] W. Fan and J. Siméon. Integrity constraints for XML. Technical Report TUCIS-TR-2000-001, Department of Computer and Information Sciences, Temple University, Feb., 2000.

[20] 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.

[21] C. S. Hara and S. B. Davidson. Reasoning about nested functional dependencies. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 91–100, Philadephia, Pennsylvania, May 1999.

[22] M. Ito and G. E. Weddell. Implication problems for functional constraints on databases supporting complex objects. *Journal of Computer and System Sciences*, 50(1):165–187, 1995.

[23] 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/`.

[24] 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`.

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

[26] H. . S. . Thompson, D. . Beech, M. . Maloney, and N. . Mendelsohn. XML schema part 1: Structures. W3C Working Draft, Sept. 1999.

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

[28] M. F. van Bommel and G. E. Weddell. Reasoning about equations and functional dependencies on complex objects. *IEEE Transactions on Data and Knowledge Engineering*, 6(3):455–469, 1994.