Reasoning about Keys for XML

Peter Buneman
University of Pennsylvania
peter@cis.upenn.edu

Susan Davidson
University of Pennsylvania
susan@cis.upenn.edu

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

Carmem Hara
Universidade Federal do Parana, Brazil
carmem@inf.ufpr.br

Wang-Chiew Tan
University of Pennsylvania
wctan@saul.cis.upenn.edu

Abstract

We study two classes of XML keys introduced in [6], and investigate their associated (finite) implication problems. In contrast to other proposals of keys for XML, these two classes of keys can be reasoned about efficiently. In particular, we show that their (finite) implication problems are finitely axiomatizable and are decidable in square time and cubic time, respectively.

Keywords: XML, Keys, Constraints, (Finite) implication, Axiomatization.

1 Introduction

Keys are a fundamental concept within databases. They provide a means of locating a specific object within the database and of referencing an object from another object (e.g. relationships); they are also an important class of constraints on the validity of data. In particular, value-based keys (as used in relational databases) provide an invariant connection from an object in the real world to its representation in the database. This connection is crucial for modifying the database as the world that it models changes.

As XML is increasingly used in the context of databases, the importance of a value-based method of locating an element in an XML document is being recognized. Key specifications for XML have been proposed in the XML standard [5], XML Data [21], and XML Schema [24]. More recently, in [6] we give a proposal for keys which has the following benefits:[1]

1. Keys can be scoped within a class of elements.
2. The specification of keys is orthogonal to the typing specification for the document (e.g. DTD or XML Schema).

[1]A detailed discussion of the differences between the proposal in [6] and those of the XML standard [5], XML Data [21], and XML Schema [24] can be found in [6].

3. Keys are defined in terms of one or more path expressions, i.e. they may involve one or more attributes, subelements or more general structures.

As an example, it is possible in our key language to express the following: 1) SSN is a key for a Person element, no matter where the SSN element appears in a subtree rooted at Person; 2) The FirstName and LastName subelements of Person also form a key; 3) The DateOfBirth subelement of Person is unique, i.e. the label itself forms a key for that element. However, there are several unanswered questions in that paper. First, what is the effect of the path expression language that is chosen to define keys? Second, how can these keys be reasoned about and can it be done efficiently?

One of the most interesting questions involving keys is that of logical *implication*, i.e. deciding if a new key holds given a set of existing keys. This is important for minimizing the expense of checking that a document satisfies a set of key constraints, and may also provide the basis for reasoning about how constraints can be propagated through view definitions. Another interesting question is whether a set of keys is "reasonable" in the sense that there exists some (finite) document that satisfies the key specification (*finite satisfiability*). We therefore focus on these two problems in this paper in context of two path expression languages proposed in [6], and show that two key specification languages defined with these path languages can be reasoned about efficiently. One key specification language, referred to as L_w, is defined in terms of paths with wild card that matches any tag, and the other, denoted by L, allows one to specify keys for elements at arbitrary depths of XML document trees by supporting a combination of wild card and the Kleene star.

Note that in relational databases, the (finite) implication problems for keys and more general functional dependencies have been well studied (see, e.g., [2, 22]). These problems have also been investigated for XML [16, 15, 14]. [16] studies the (finite) implication problems associated with a class of simple keys (and foreign keys) in the absence of DTDs, and [15, 14] investigates the interaction between XML DTDs and these constraints. The key constraints considered in these papers are defined in terms of XML attributes and are therefore not as expressive as the keys studied in this paper. Constraints defined in terms of navigation paths have been studied for semistructured [1] and XML data in [3, 8, 9, 10, 11]. These constraints are generalizations of inclusion dependencies commonly found in relational databases, and are not capable of expressing keys. Generalizations of functional dependencies have also been studied [17, 20]. However these generalizations were investigated in database settings, which are quite different from the tree model for XML data considered in this paper.

Contributions. The main contributions of the paper are the following.

- We investigate the containment problem for two classes of regular path expressions, which are star-free languages. While the containment problem for star-free languages is coNP-complete in general [19], we show that the problems for these two classes are finitely axiomatizable and moreover, decidable in linear time and square time, respectively. These results are not only interesting in their own right, but also important for the analysis of key implication.
- We show that keys defined in our specification are always finitely satisfiable. We also establish complexity and axiomatizability results for the (finite) implication problems associated with

the two key languages L_w and L. More specifically, we provide sound and complete sets of inference rules, and algorithms for determining (finite) implication of keys expressed in these two languages in square time and cubic time, respectively. The low complexities allow one to use and reason about keys in our specification efficiently in practice.

It should be noted that we do not consider foreign keys and DTDs in this paper. Furthermore, although this paper follows [6] in the definition of (absolute) keys, relative key implication is the topic of another paper.

Organization. The rest of the paper is organized as follows. Section 2 defines XML trees, value equality, and two key specification languages for XML: L_w and L. Section 3 investigates containment of regular path expressions used in keys of L_w and L. Section 4 establishes the complexity and axiomatizability results for reasoning about keys of the two languages. Finally, Section 5 outlines directions for further research. Proofs of the results are given in an appendix.

2 Key constraint languages

In this section, we first present a tree model for XML data, and then define a notion of value equality for XML trees. Value equality is central to the definition of keys for XML data. As important is the path language used to refer to nodes or describe a collection of nodes in an XML tree. We therefore introduce three languages for path expressions. Using these path languages, we define two key constraint languages for XML and describe their associated (finite) satisfiability and (finite) implication problems.

2.1 A Tree Model and Value Equality

XML documents are typically modeled as trees, e.g., in DOM [4], XSL [12, 25], XQL [23] and XML Schema [24]. We formally define XML trees as follows.

Assume a countably infinite set $\mathbf{E}$ of element labels (tags), a countably infinite set $\mathbf{A}$ of attribute names, and a symbol $\mathtt{S}$ indicating text (e.g., PCDATA in XML [5]).

Definition 2.1: An *XML tree* is defined to be $T = (V, lab, ele, att, val, r)$, where

- V is a set of *vertices* (*nodes*);
- lab is a function from V to $\mathbf{E} \cup \mathbf{A} \cup \{\mathtt{S}\}$;
- ele is a partial function from V to sequences of V vertices such that for any $v \in V$, if $ele(v)$ is defined then $lab(v) \in \mathbf{E}$;
- att is a partial function from $V \times \mathbf{A}$ to V such that for any $v \in V$ and $l \in \mathbf{A}$, if $att(v, l) = v'$ then $lab(v) \in \mathbf{E}$ and $lab(v') = l$;

```
⟨db⟩
  ⟨driver⟩
    ⟨name⟩ Schumacher ⟨/name⟩
    ⟨formula1  year=2000⟩
      ⟨team⟩ Ferrari ⟨/team⟩
    ⟨/formula1⟩
  ⟨/driver⟩
  ⟨driver⟩
    ⟨name⟩ Barrichello ⟨/name⟩
    ⟨born⟩ 1972 ⟨/born⟩
    ⟨formula1  year=1999⟩
      ⟨team⟩ Stewart ⟨/team⟩
      ⟨position⟩ 7 ⟨/position⟩
    ⟨/formula1⟩
    ⟨formula1  year=2000⟩
      ⟨team⟩ Ferrari ⟨/team⟩
    ⟨/formula1⟩
  ⟨/driver⟩
⟨/db⟩
```

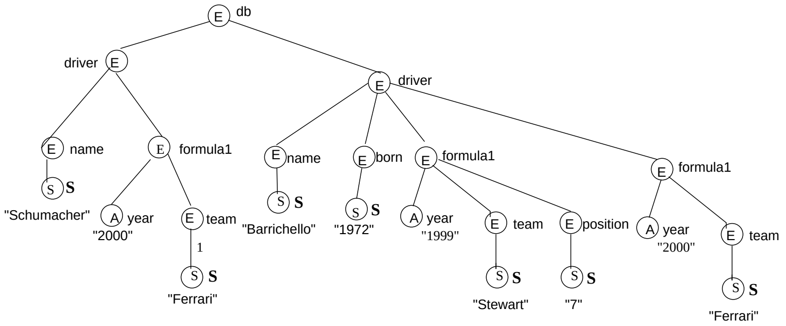

Figure 1: Example of some XML data and its representation as a tree

- *val* is a partial function from V to string values such that for any node $v \in V$, $val(v)$ is a string iff either $lab(v) = \mathtt{S}$ or $lab(v) \in \mathbf{A}$;
- r is a distinguished vertex in V and is called *the root of* T; without loss of generality, assume $lab(r) = r$. We assume that there is a unique node in T labeled r.

For any $v \in V$, if $ele(v)$ is defined then nodes in $ele(v)$ are called *subelements* of v. For any $l \in \mathbf{A}$, if $att(v, l) = v'$ then v' is called *an attribute* of v. In either case we say that there is a *parent-child edge* from v to v'. Subelements and attributes of v are called *children* of v.

An XML tree has a tree structure, i.e., for each $v \in V$, there is a unique path of parent-child edges from root r to v.

An XML tree T is said to be *finite* if V is finite. ■

Intuitively, V is the set of nodes of the tree T. These nodes can be classified into three types: element nodes, attribute nodes, and text nodes. As illustrated in Fig. 1 text nodes (S) have no name but carry text, attribute nodes (A) both have a name and carry text, and element nodes (E) have a name. More specifically, if a node v is labeled τ in $\mathbf{E}$, then functions *ele* and *att* define the children of v, which are partitioned into *subelements* and *attributes*. Subelements of node v are ordered, whereas attributes of node v are unordered and are identified by their labels (names). The function *val* assigns string values to attribute nodes and text nodes. Because T has a tree structure, sharing of nodes is not allowed in T. Observe that there is an one-to-one mapping between XML trees and XML documents. Next we define our notion of value equality on XML trees.

Let $T = (V, lab, ele, att, val, r)$ be an XML tree, and v, v' be two nodes in V. Informally, v, v' are value equal if they have the same tag (label) and in addition, either they have the same (string) value (when they are $\mathtt{S}$ or A nodes) or their children are pairwise value equal (when they are $\mathtt{E}$

nodes). More formally,

Definition 2.2: n_1 and n_2 are *value equal*, denoted by $n_1 =_v n_2$, iff the following three conditions are satisfied: (1) $lab(v) = lab(v')$. (2) if $lab(v) = \texttt{S}$ or $lab(v) \in \mathbf{A}$ then $val(v) = val(v')$. (3) if $lab(v) \in \mathbf{E}$ then

- for any $l \in \mathbf{A}$, $att(v, l)$ is defined iff $att(v', l)$ is defined, and $val(att(v, l)) = val(att(v', l))$;
- if $ele(v) = [v_1, \ldots, v_k]$ then $ele(v') = [v'_1, \ldots, v'_k]$ and for all $i \in [1, k]$, $v_i =_v v'_i$.

■

As an example, in Fig. 1, the single `formula1` element of the first `driver` and the second `formula1` element of the second `driver` are value equal.

2.2 Path Languages

Three path languages, PL_s, PL_w and PL, are shown in the table below.

Path Language	Syntax
PL_s	$\rho \ ::= \ \epsilon \mid l.\rho$
PL_w	$p \ ::= \ \epsilon \mid l \mid p.p \mid \_$
PL	$q \ ::= \ \epsilon \mid l \mid q.q \mid \_*$

In PL_s, a path is a (possibly empty) sequence of node labels. Here ϵ represents the empty path, node label $l \in \mathbf{E} \cup \mathbf{A} \cup \{\texttt{S}\}$, and "." is a binary operator that concatenates two path expressions. Intuitively, a path in PL_s corresponds to the sequence of tags (labels) of nodes in a parent-child path. The language PL_w is a mild generalization of PL_s by including the wild card symbol "_", which can match any node label. Another generalization of PL_s, PL, allows the symbol "_*", a combination of wild card and Kleene closure. This symbol represents any (possibly empty) sequence of node labels. It should be noted that for any path expression P in any of the path languages, the following equality holds: $P.\epsilon = \epsilon.P = P$. Also, observe that these path languages are subclasses of regular path expressions [18]. Although the set of paths described by PL_s is contained in both PL_w and PL, neither PL_w nor PL subsumes each other.

For example, $a.b.c$ is a path expression in PL_s, PL_w and PL while $a.\_.c$ is a path expression in PL_w exclusively and $a.\_*.c$ is a path expression in PL exclusively.

As mentioned earlier, a path intends to represent a parent-child path in an XML tree. Observe that an attribute node must be a leaf in an XML tree and it cannot have outgoing edges. Therefore, we assume in the rest of the paper that for any path expression P, if P contains an attribute, then P is of the form $P'.l$, where P' does not contain any attribute. In other words, an attribute can only be the last symbol of a path expression. Next we describe some notation in connection with path expressions.

Length. The *length* of a path expression P, denoted as $|P|$ is the number of labels in the path sequence. The empty path has length 0, "_" and "_*" are each counted as labels with length 1.

For example, $a.b.c$, $a.\_.b$ and $a.\_*.c$ are each of length 3.

Existence of a path. Let T be an XML tree, ρ be a path in PL_s, and n_1, n_2 be nodes in T. We say *there is path ρ from n_1 to n_2*, denoted by $T \models \rho(n_1, n_2)$, if in T there is a path of parent-child edges from n_1 to n_2 and the sequence of nodes labels in the path is ρ. Let P be a path expression in PL_w or PL. We say n_2 is *reachable* from n_1 by following P, denoted by $T \models P(n_1, n_2)$, if there is a path $\rho \in P$ such that $T \models \rho(n_1, n_2)$.

For example, if T is the XML tree in Fig. 1 and n is the name subelement of the first driver then $T \models \text{driver.name}(r, n)$. Also, $T \models \_^*(r, n)$.

Node set. Let T be an XML tree, n be a node in T and P be a path expression in one of the path languages. Then $n[\![P]\!]$ denotes the set of nodes in T that can be reached by following the path P from node n. That is, $n[\![P]\!] = \{n' \mid T \models P(n, n')\}$. We shall use $[\![P]\!]$ as an abbreviation for $r[\![P]\!]$, where r is the root node of T.

Value Intersection. Let n_1 and n_2 be two nodes in an XML tree T and P be a path expression. The *value intersection* of $n_1[\![P]\!]$ and $n_2[\![P]\!]$, denoted as $n_1[\![P]\!] \cap_v n_2[\![P]\!]$, is defined as follows:

$$n_1[\![P]\!] \cap_v n_2[\![P]\!] = \{(z, z') \mid \exists\, \rho \in P, z \in n_1[\![\rho]\!], z' \in n_2[\![\rho]\!], z =_v z'\}$$

Intuitively, $n_1[\![P]\!] \cap_v n_2[\![P]\!]$ consists of pairs of nodes that are value equal and are reachable by following the same simple path in the language defined by P starting from n_1 and n_2, respectively. This notion is central, and will be used in the definition of keys for XML.

For example, let n_1 and n_2 be the first and second driver elements in Fig. 1. Then $n_1[\![\_]\!] \cap_v n_2[\![\_]\!]$ is a set consisting of a pair of nodes corresponding to the `formula1` subelement of the first driver and the second `formula1` subelement of the second driver.

Containment. Let P and Q be two path expressions in PL_s, PL_w or PL. We use $P \subseteq Q$ to denote that the language defined by P is a subset of the language defined by Q.

For example, $a.b.c \subseteq a.\_.c \subseteq \_.\_.c$ and $a.b.c \subseteq a.\_*.c \subseteq a.\_*$. However, $a.\_* \not\subseteq a.\_*.c$.

The *containment problem* for a path language is to determine, given any path expressions P and Q in the language, whether $P \subseteq Q$. Observe that PL_w and PL are subclasses of regular expressions. It is known that containment of general regular expressions is not finitely axiomatizable, i.e., there is no finite set of inference rules that is sound and complete for containment of regular expressions. In contrast, in Section 3 we shall show that for PL_w and PL, the containment problems are finitely axiomatizable.

2.3 Key constraint languages for XML

We next define keys for XML and what it means for an XML document to satisfy a key.

Definition 2.3: A *key constraint* φ for XML is an expression of the form $(Q, \{P_1, \ldots, P_k\})$, where Q and P_i are path expressions. Q is called the *target path* of φ, and P_1, ..., P_k are called the *key paths* of φ. Two classes of key constraints are defined as follows:

- L_w: the set of key constraints whose target path and key paths are in PL_w.
- L: the set of key constraints whose target path and key paths are in PL.

■

A key specifies two parts: the target path identifies a set of nodes, referred to as the *target set*, on which the key is defined; and the set of key paths. The values of the key paths uniquely identify an element in the target set. The target set is analogous to a set of tuples in a relation and the key paths to the attributes of the relation designated as a key.

For example, $(a, \{b.c\})$ is a key in both L_w and L, $(\_.a, \{b.\_\})$ is a key in L_w, and $(\_*.a, \{b\})$ is a key in L.

Observe that for any key φ whose target and key paths are in PL_s, φ is in both L_w and L. However, neither L_w nor L subsumes the other. Also, to simplify discussion, we assume that in any key $(Q, \{P_1, \ldots, P_n\})$, the target path Q does not contain any attribute. This is because in an XML tree, an attribute node cannot have any outgoing edge.

XPath. As an aside, we observe that there is an easy translation from any of our path languages used in a key constraint to XPath-like syntax. See [6]. Informally, "/" is used as the concatenation operator instead of ".". A path starting from the root is prefixed with "/". Wild card "_" is replaced with "*", "_*" is replaced with "//" and "." is an XPath equivalent of ϵ.

However, for discussions in this paper, we will use the conventional regular language syntax of "_", "_*" and "." for wild card, the combination of wild card and kleene star, and path concatenator.

Definition 2.4: Let $\varphi = (Q, \{P_1, \ldots, P_k\})$ be a key. An XML tree T *satisfies* φ, denoted as $T \models \varphi$, iff for any n_1, n_2 in $[\![Q]\!]$, if for all $i \in [1, k]$ the value intersection of $n_1[\![P_i]\!]$ and $n_2[\![P_i]\!]$ is non-empty then $n_1 = n_2$. That is,

$$\forall n_1\, n_2 \in [\![Q]\!] \;\; ((\bigwedge_{1 \leq i \leq k} n_1[\![P_i]\!] \cap_v n_2[\![P_i]\!] \neq \{\}) \rightarrow n_1 = n_2).$$

Note that the target path Q starts at the root of T. ■

Intuitively, the key requires that if two nodes in $[\![Q]\!]$ are distinct, then the two sets of nodes reached on some P_i must be disjoint up to value equality. More specifically, for any distinct nodes n_1, n_2 in $[\![Q]\!]$, there must exist some P_i, $1 \leq i \leq k$, such that for all paths $\rho \in P_i$ and for all nodes x in $n_1[\![\rho]\!]$ and y in $n_2[\![\rho]\!]$, $x \neq_v y$. The key has no impact on those nodes at which some key path is *missing*, i.e. nodes n such that $n[\![P_i]\!]$ is empty for some P_i. For any n_1, n_2 in $[\![Q]\!]$, if P_i is missing at either n_1 or n_2 then $n_1[\![P_i]\!]$ and $n_2[\![P_i]\!]$ are by definition disjoint. This is similar to *unique constraints* of XML Schema. In contrast to unique constraints, however, our notion of key specification is capable of comparing nodes at which a key path may lead to multiple nodes.

For example, $\varphi_1 = (db.driver, \{name, formula1\})$ and $\varphi_2 = (db.driver, \{formula1\})$ are two keys in both L_w and L. The XML document depicted in Fig. 1 satisfies φ_1 because different drivers have different `name` values. However, it does not satisfy φ_2 because the `formula1` subelement of the first driver and the second `formula1` subelement of the second driver are value equal. Observe that drivers may have multiple $formula1$ subelements. Unique constraints of XML Schema cannot be specified for such drivers.

It should be noted that two notions of equality are used to define keys: value equality ($=_v$) when comparing nodes reached by following key paths, and node identity ($=$) when comparing two nodes in the target set. This is a departure from keys in relational databases, in which only value equality is considered.

Let Σ be a finite set of L_w keys and T be an XML tree. We use $T \models \Sigma$ to denote that T *satisfies* Σ. That is, for any $\psi \in \Sigma$, $T \models \psi$. We can define satisfaction of a finite set of L keys similarly.

2.4 Decision problems for keys

In relational databases, we are often interested in knowing if a given set of dependencies can be satisfied. In addition, if an instance satisfies a set of dependencies, it is useful to know what other dependencies are necessarily satisfied by that instance (logical implication). These problems can also be defined in the context of XML keys.

Satisfiability. The *(finite) satisfiability problem* for a key constraint language K is to determine, given any finite set Σ of keys in K, whether there exists an (finite) XML tree satisfying Σ.

In relational databases, given any relational schema and a finite set of keys over the schema, one can always construct a non-empty instance of the schema that satisfies the keys by creating a tuple for each relation. Thus the (finite) satisfiability problem for relational keys is trivial. The (finite) satisfiability problem for keys in L_w or L is also trivial. Observe that any set of keys in L_w or L can always be satisfied by the single node tree. Therefore, we have the following observation.

Observation. For any finite set Σ of keys in L_w or L, one can always find a finite XML tree that satisfies Σ.

Logical Implication. Let $\Sigma \cup \{\varphi\}$ be a finite set of keys. We use $\Sigma \models \varphi$ to denote Σ *implies* φ, that is, for any XML tree T, if $T \models \Sigma$, then $T \models \varphi$. The *implication problem* for a key language K is to determine, given any finite set of keys $\Sigma \cup \{\varphi\}$ in K, whether $\Sigma \models \varphi$. The *finite implication problem* for K is to determine whether Σ *finitely implies* φ, that is, whether it is the case that for any finite XML tree T, if $T \models \Sigma$, then $T \models \varphi$.

For example, $\{(a, \{b\}), (a.b, \{c\})\} \models (a, \{b.c\})$. In fact we also have $\{(a.b, \{c\})\} \models (a, \{b.c\})$. To see this, observe that by the definition of keys, if an XML tree T satisfies $(a.b, \{c\})$, then the set of c elements under any two distinct $a.b$ nodes are pairwise disjoint up to value equality. Therefore, if there exists any $b.c$ nodes under a that are value equal, they must be under the same a node. Hence T also satisfies $(a, \{b.c\})$.

Observe that given any finite set $\Sigma \cup \{\varphi\}$ of L_w (L) constraints, if there is an XML tree T such that $T \models \bigwedge \Sigma \wedge \neg\varphi$, then there must be a finite XML tree T' such that $T' \models \bigwedge \Sigma \wedge \neg\varphi$. More specifically, let $\varphi = (Q, \{P_1, \ldots, P_k\})$. Since $T \not\models \varphi$, there are nodes $n_1, n_2 \in [\![Q]\!]$, $x_i \in n_1[\![P_i]\!]$ and $y_i \in n_2[\![P_i]\!]$ for $i \in [1, k]$ such that $x_i =_v y_i$ but $n_1 \neq n_2$. Let T' be the finite subtree of T that

$$\frac{l \text{ is a label}}{P.l.Q \subseteq P.\_.Q} \quad \text{(Containment)}$$

$$\frac{}{P \subseteq P} \quad \text{(Reflexivity)}$$

$$\frac{}{\epsilon.P \subseteq P \quad P \subseteq \epsilon.P \quad P.\epsilon \subseteq P \quad P \subseteq P.\epsilon} \quad \text{(Empty-path)}$$

$$\frac{P \subseteq Q \quad Q \subseteq R}{P \subseteq R} \quad \text{(Transitivity)}$$

Table 1: $\mathcal{I}_w^p$: **Inference rules for inclusion of PL_w expressions**

consists of all and only the nodes in the paths from root to x_i, y_i for all $i \in [1, k]$. It is easy to verify that $T' \models \Sigma$ but $T' \models \neg\varphi$. Therefore, key constraint implication has the finite model property:

Proposition 2.1: For each of L_w and L, the implication and finite implication problems coincide. ■

In light of Proposition 2.1, we can also use $\Sigma \models \varphi$ to denote that Σ finitely implies φ. We investigate the finite implication problems for L_w and L in Section 4.

3 Inclusion of path expressions

In this section, we study containment of path expressions in our path languages PL_w and PL defined in the last section. The results of this section are not only interesting in their own right, but also important in the analysis of key constraint implication to be studied in the next section.

We first give a set of inference rules for PL_w expression inclusion, denoted by $\mathcal{I}_w^p$, in Table 1.

Proposition 3.1: The set $\mathcal{I}_w^p$ is sound and complete for inclusion of path expressions in PL_w. In addition, inclusion of PL_w expressions can be determined in linear time. ■

This can be verified by a straightforward induction on the number of occurrences of "_" in path expressions. See Appendix for a detailed proof.

In light of $\mathcal{I}_w^p$, a linear time (recursive) function for testing inclusion of PL_w expressions can be constructed as follows. The function, $Incl_w(P, Q)$, returns true iff $P \subseteq Q$, where P, Q are path expressions in PL_w. Without loss of generality by the Empty-path rule, we assume that P (Q) does not contain ϵ unless $P = \epsilon$ ($Q = \epsilon$).

Algorithm 3.1: $Incl_w(P, Q)$

1. if $P = Q = \epsilon$
   then return true;
2. if ($P = l.P'$ and $Q = l.Q'$) or ($P = l.P'$ and $Q = \_.Q'$) or ($P = \_.P'$ and $Q = \_.Q'$)

then return $Incl_w(P', Q')$;
else return false;

The inference rules for inclusion of PL expressions, denoted by $\mathcal{I}^p$, are the same as those in $\mathcal{I}_w$ except the following:

$$\frac{}{P.R.Q \subseteq P.\_*.Q} \qquad \text{(Containment)}$$

It should be mentioned that PL is a star-free language. Recall that in general, the containment problem for star-free languages is co-NP complete [19]. In contrast, the containment problem for PL has low complexity.

Theorem 3.2: The set $\mathcal{I}^p$ is sound and complete for inclusion of path expressions in PL. In addition, inclusion of PL expressions can be determined in square time. ∎

The soundness of $\mathcal{I}^p$ can be verified by induction on the lengths of $\mathcal{I}^p$-proofs. The proof of completeness is more involved, and uses an idea of simulation. To give the proof, we first introduce some notation.

An expression P in PL is in *normal form* iff it does not contain consecutive $\_*$'s, i.e., P does not contain $\_*.\_*$ and P does not contain ϵ unless $P = \epsilon$. This is easily done using the Containment rule. By the Empty-path rule, we can also assume that P does not contain ϵ unless $P = \epsilon$. It takes linear time to rewrite P to an equivalent normal form expression. We assume from here onwards that a path expression $P \in PL$ is in normal form.

Let P, Q be path expressions in PL. To determine whether $P \subseteq Q$, we consider their nondeterministic finite state automata (NFAs) [18]. We use $M(P)$ to denote a NFA for P. Observe that $M(P)$ has a "linear" structure as shown in Fig. 2. The number of states in $M(P)$ is linear in the size of P. Thus $M(P)$ and $M(Q)$ can be constructed in $O(|P|)$ and $O(|Q|)$ time, respectively. Let

$$M(P) = (N_1, T \cup \{\_\}, \delta_1, S_1, F_1), \qquad M(Q) = (N_2, T \cup \{\_\}, \delta_2, S_2, F_2),$$

where N_1, N_2 are the sets of states, T is the alphabet, δ_1, δ_2 are transition functions, S_1, S_2 are start states, and F_1, F_2 are final states of $M(P)$ and $M(Q)$, respectively. Note that we extend the definition of NFAs by treating the wild card symbol $\_$ as a "letter", which matches any letter in T. Observe that $M(P)$ has the following properties ($M(Q)$ has similar properties): 1) It has a single final state F_1. In addition, $\delta_1(F_1, a) = \emptyset$ for any $a \in T$, but it is possible that $\delta_1(F_1, \_) \neq \emptyset$. 2) For any $n \in N_1$, if $n \neq F_1$, then there must be $a \in T$ and $n' \in N_1$ such that $\delta_1(n, a) = \{n'\}$ and $n \neq n'$. We write $\delta_1(n, a) = n'$ if n' is the only element of $\delta_1(n, a)$. 3) For any $n \in N_1$, either $\delta_1(n, \_) = n$ or $\delta_1(n, \_) = \emptyset$.

We now define a *simulation relation*, $\lhd$, on $N_1 \times N_2$. For any $n_1 \in N_1$ and $n_2 \in N_2$, $n_1 \lhd n_2$ iff one of the following conditions is satisfied:

- If $n_1 = F_1$ then $n_2 = F_2$ and either $\delta_1(F_1, \_) = \emptyset$ or $\delta_2(F_2, \_) = F_2$.

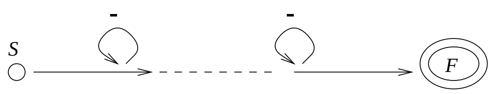

Figure 2: A finite state automata for a path expression of PL

- For $n_1 \neq F_1$, if $\delta_1(n_1, \_) = n_1$ then $\delta_2(n_2, \_) = n_2$. Moreover, for any $a \in T$ if $\delta_1(n_1, a) = n_1'$ for some $n_1' \in N_1$, then either $\delta_2(n_2, \_) = n_2$ and $n_1' \lhd n_2$ or there exists $n_2' \in N_2$ such that $\delta_2(n_2, a) = n_2'$ and $n_1' \lhd n_2'$.

To prove the completeness of $\mathcal{I}^p$, it suffices to show the following (see Appendix for proofs):

(1) $P \subseteq Q$ iff $S_1 \lhd S_2$.

(2) If $S_1 \lhd S_2$, then $P \subseteq Q$ can be proved using the inference rules $\mathcal{I}^p$.

Given $\mathcal{I}^p$ and the claims, we provide a function $Incl(n_1, n_2)$ for testing inclusion of PL expressions. The function assumes the existence of $M(P), M(Q)$ as described above. In addition, we assume that P and Q are in normal form and do not contain ϵ (unless they are ϵ). The function $Incl(n_1, n_2)$ returns true iff $n_1 \lhd n_2$, where n_1 and n_2 are states from N_1 and N_2 respectively. Since $P \subseteq Q$ iff $S_1 \lhd S_2$, $P \subseteq Q$ iff $Incl(S_1, S_2)$. Initially, $visited(n_1, n_2)$ is false for all $n_1 \in N_1$, $n_2 \in N_2$.

Algorithm 3.2: $Incl(n_1, n_2)$

1. if $visited(n_1, n_2)$
   then return false
   else mark $visited(n_1, n_2)$ as true;
2. process n_1, n_2 as follows:
   Case 1: if $n_1 = F_1$
   then if $n_2 = F_2$ and $(\delta_1(F_1, \_) = \emptyset$ or $\delta_2(F_2, \_) = F_2)$
   then return true;
   else return false;
   Case 2: if $\delta_1(n_1, a) = n_1'$ and $\delta_2(n_2, a) = n_2'$ and $\delta_1(n_1, \_) = \emptyset$ and $\delta_2(n_2, \_) = \emptyset$
   then return $Incl(n_1', n_2')$;
   Case 3: if $\delta_1(n_1, a) = n_1'$ and $\delta_2(n_2, \_) = n_2$ and $\delta_2(n_2, a) = n_2'$
   then return $(Incl(n_1', n_2)$ or $Incl(n_1', n_2'))$
   else if $\delta_1(n_1, a) = n_1'$ and $\delta_2(n_2, \_) = n_2$ and $\delta_2(n_2, a) = \emptyset$
   then return $Incl(n_1', n_2)$;
3. return false

The correctness of the algorithm follows from the claims given above. The construction of $M(P), M(Q)$, as well as transforming P, Q to normal form, can be done in $O(|P|)$ and $O(|Q|)$ time, respectively. The first statement takes $O(|P| \cdot |Q|)$ time. Since any pair of states (n_1, n_2) from $N_1 * N_2$ is never processed twice, it is easy to see that the second statement and thus $Incl(S_1, S_2)$ run in $O(|P| \cdot |Q|)$ time.

4 Key constraint implication

We now turn to finite implication problems for L_w and L. For each of these languages, we provide a finite axiomatization and an algorithm for determining finite implication. Recall that by Proposition 2.1, all the results established on finite implication also hold for implication.

4.1 Axiomatization for L_w

The inference rules for L_w key implication, denoted by $\mathcal{I}_w$, are shown in Table 2. The Superkey rule states that if a set S of key paths uniquely identifies a node in the target set $[\![Q]\!]$, then so does any superset of S. This rule is also sound in the context of relational databases. In contrast, other rules in $\mathcal{I}_w$ do not have relational counterparts. A brief discussion of the rules follows.

- Subnodes: observe that any node v in $[\![Q.Q']\!]$ must be in the subtree of some node v' in $[\![Q]\!]$. Because XML trees do not allow sharing of nodes, v uniquely identifies v' in $[\![Q]\!]$. Thus if a key path P uniquely identifies nodes in $[\![Q.Q']\!]$, then $Q'.P$ uniquely identifies nodes in $[\![Q]\!]$.
- Path-containment: if a set $S \cup \{P_i, P_j\}$ of key paths uniquely identifies nodes in $[\![Q]\!]$ and $P_i \subseteq P_j$, then we can leave out P_j from the set of key paths for $[\![Q]\!]$. This is because for any nodes $n_1, n_2 \in [\![Q]\!]$, if $n_1[\![P_i]\!] \cap_v n_2[\![P_i]\!] \neq \emptyset$, then we must have $n_1[\![P_j]\!] \cap_v n_2[\![P_j]\!] \neq \emptyset$ given $P_i \subseteq P_j$. Thus by the definition of keys, $S \cup \{P_i\}$ is a key for $[\![Q]\!]$.
- Target-containment: any key for the set $[\![Q]\!]$ is also a key for any subset of $[\![Q]\!]$. Observe that $[\![Q']\!] \subseteq [\![Q]\!]$ if $Q' \subseteq Q$.
- Key-containment: for any nodes $n_1, n_2 \in [\![Q]\!]$, if $n_1[\![P_i']\!] \cap_v n_2[\![P_i']\!] \neq \emptyset$, then we must have $n_1[\![P_i]\!] \cap_v n_2[\![P_i]\!] \neq \emptyset$ given $P_i' \subseteq P_i$. Thus if $S \cup \{P_i\}$ is a key for $[\![Q]\!]$ then so is $S \cup \{P_i'\}$.
- Epsilon: this rule is sound because any XML tree has a unique root. In other words, in any XML tree T, $[\![\epsilon]\!] = \{r\}$ where r is the root of T.

Given a finite set $\Sigma \cup \{\varphi\}$ of L_w constraints, we use $\Sigma \vdash_{\mathcal{I}_w} \varphi$ to denote that φ is provable from Σ using $\mathcal{I}_w$. That is, there is an $\mathcal{I}_w$-proof of φ from Σ.

To simplify the discussion, we assume that keys are in key normal form. A key constraint $\phi = (Q, S)$ in L_w is in the *key normal form* if for every pair of paths P_i and P_j in S, $P_i \not\subseteq P_j$. By the Path-containment and Superkey rules, one can assume without loss of generality that keys are always in the key normal form.

Theorem 4.1: The set $\mathcal{I}_w$ is sound and complete for finite implication of L_w constraints. ■

Soundness of $\mathcal{I}_w$ can be verified by induction on the lengths of $\mathcal{I}_w$-proofs. For the proof of completeness, given any finite set $\Sigma \cup \{\varphi\}$ of keys in L_w, it suffices to show that either $\Sigma \vdash_{\mathcal{I}_w} \varphi$, or there is a finite XML tree G such that $G \models \Sigma$ and $G \models \neg\varphi$, i.e., $\Sigma \not\models \varphi$. In other words, if $\Sigma \models \varphi$ then $\Sigma \vdash_{\mathcal{I}_w} \varphi$.

$$\frac{(Q,\, S) \quad P \text{ is any path expression}}{(Q,\, S \cup \{P\})} \qquad \text{(Superkey)}$$

$$\frac{(Q.Q',\, \{P\})}{(Q,\, \{Q'.P\})} \qquad \text{(Subnodes)}$$

$$\frac{(Q,\, S \cup \{P_i, P_j\}) \quad P_i \subseteq P_j}{(Q,\, S \cup \{P_i\})} \qquad \text{(Path-containment)}$$

$$\frac{(Q,\, S) \quad Q' \subseteq Q}{(Q',\, S)} \qquad \text{(Target-containment)}$$

$$\frac{(Q,\, S \cup \{P_i\}) \quad P_i' \subseteq P_i}{(Q,\, S \cup \{P_i'\})} \qquad \text{(Key-containment)}$$

$$\frac{\text{for any set of path expressions } S}{(\epsilon,\, S)} \qquad \text{(Epsilon)}$$

Table 2: $\mathcal{I}_w$: **Inference rules for L_w constraint implication**

To do so, we introduce some notation. An *abstract tree with "_"* extends an XML tree by allowing "_" as a node label. Let T be an abstract tree with "_", R_1 be the labels in a parent-child path in T, and a, b be nodes in T. We say $T \models R_1(a, b)$ if there is a parent-child path from a to b such that the sequence of labels in the path is R_1. Note that R_1 is a path expression of PL_w and possibly contains occurrences of "_". Let R_2 be any path expression in PL_w. We say $T \models R_2(a,\, b)$ if $R_1 \subseteq R_2$. Given this, the definitions of node sets and satisfaction of key constraints in L_w can be easily generalized for abstract trees. Abstract trees with "_" have the following property (proof of the lemma can be found in the Appendix):

Lemma 4.2: Let $\Sigma \cup \{\varphi\}$ be a finite set of L_w keys. If there is a finite abstract tree T with "_" such that $T \models \Sigma$ and $T \models \neg\varphi$, then there is a finite XML tree G such that $G \models \Sigma$ and $G \models \neg\varphi$. ■

Given these, we prove the completeness of $\mathcal{I}_w$ in two steps. Let $\Sigma \cup \{\varphi\}$ be a finite set of keys in L_w, where $\varphi = (Q, \{P_1, ..., P_k\})$. Assume $Q \neq \epsilon$, since otherwise we have $\Sigma \vdash_{\mathcal{I}_w} \varphi$ by the rule Epsilon in $\mathcal{I}_w$. We start with a finite abstract tree T that does not satisfy φ. The tree T consists of two distinct branches T_1 and T_2 from its root r. Each branch has a Q path that leads to paths $P_1, \ldots, P_k$, as depicted in Fig. 3 (a). Let n_1 be the (single) node in T_1 and $[\![Q]\!]$, and n_2 be the node in T_2 and $[\![Q]\!]$. Moreover, for each $i \in [1, k]$, let x_i be the node in T_1 and $[\![Q.P_i]\!]$, and y_i be the node in $[\![Q.P_i]\!]$ and T_2. Assume that for each $i \in [1, k]$, $x_i =_v y_i$, but for any other pair x, y in T, $x \neq_v y$. This can be achieved as follows: in each element in T we add a new text subelement E (E does not appear anywhere in the constraints) at the end of the sequence of its subelements, followed by a text ($\mathtt{S}$) subelement, and let $x_i.E.S =_v y_i.E.S$ for each $i \in [1, k]$, but for any other pair x, y in T, let $x.E.S \neq_v y.E.S$. Given T, we examine each ϕ in Σ. If the tree does not satisfy ϕ, then we merge nodes in the tree such that the new tree satisfies ϕ, as shown in Fig. 3 (b) and (c). Let T' be the tree obtained after all keys in Σ have been processed. Obviously, $T' \models \Sigma$. If $T' \models \varphi$, then we show that it is indeed the case that $\Sigma \vdash_{\mathcal{I}_w} \varphi$. Otherwise by Lemma 4.2, there is a finite XML tree G such that $G \models \Sigma$ but $G \not\models \varphi$. That is, $\Sigma \not\models \varphi$. The details of the proof are given in Appendix.

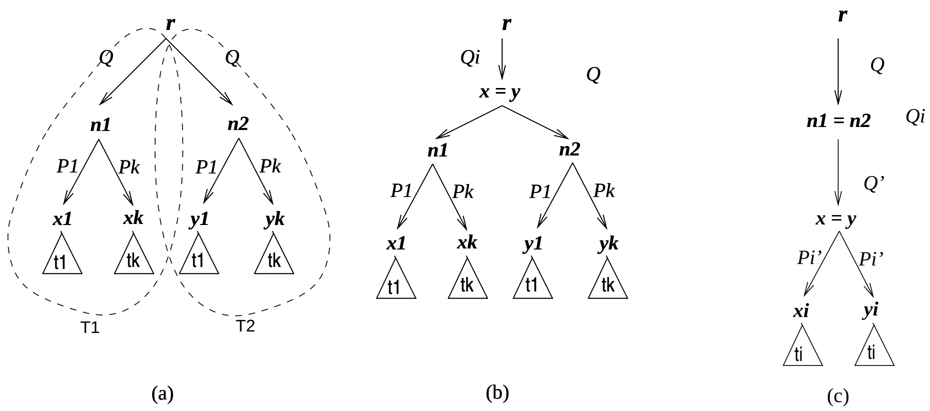

Figure 3: Abstract trees constructed in the proof of Theorem 4.1

Using Theorem 4.1 we can show the following:

Theorem 4.3: The finite implication problem for L_w is decidable in square time. ■

A square time algorithm for determining L_w constraint implication is given below:

Algorithm 4.1: Finite implication of L_w constraints

Input: a finite subset $\Sigma \cup \{\varphi\}$ of L_w constraints, where $\varphi = (Q, \{P_1, ..., P_k\})$
Output: true iff $\Sigma \models \varphi$

1. for each $(Q_i, S_i) \in \Sigma \cup \{\varphi\}$ do
   repeat until no further change
   if $S_i = S \cup \{P', P''\}$ such that $P' \subseteq P''$
   then $S_i := S_i \setminus \{P''\}$
2. for each $\phi \in \Sigma$ do
   (1) if $\phi = (Q', \{P'_1, ..., P'_m\})$, $Q \subseteq Q'$ and
   for all $i \in [1..m]$ there exists $j \in [1..k]$ such that $P_j \subseteq P'_i$
   then output true and terminate
   (2) if $\phi = (Q'.Q'', \{P\})$, $Q \subseteq Q'$ and $P_i \subseteq Q''.P$ for some $i \in [1..k]$ or
   $\phi = (Q', \emptyset)$ and $Q.P_i \subseteq Q'$ for some $i \in [1..k]$
   then output true and terminate
3. output false

The correctness of the algorithm follows from Theorem 4.1 and its proof (see Appendix). We next show that the algorithm is in square time. Step 1 of the algorithm transforms keys in $\Sigma \cup \{\varphi\}$ to key normal form, i.e., it ensures that key paths in each key are pairwise non-containing. By Proposition 3.1, this can be done in square time. In step 2 of the algorithm, each key constraint ϕ in Σ is processed at most once in either Case 1 or Case 2. Again by Proposition 3.1, it is easy to verify that step 2, and thus the algorithm, are in square time.

4.2 Axiomatization for L

Finally, we investigate finite implication of keys defined in L. The inference rules for L constraint implication are the same as those given Table 2, except here path expressions are in PL. Let us denote the rules with this modification as $\mathcal{I}$. Given a finite set $\Sigma \cup \{\varphi\}$ of L constraints, we use $\Sigma \vdash_{\mathcal{I}} \varphi$ to denote that φ is provable from Σ using $\mathcal{I}$.

As for L_w constraints, we define the key normal form for L constraints as follows. A key $\phi = (Q, S)$ in L is in the *key normal form* if for every pair of paths P_i and P_j in S, $P_i \not\subseteq P_j$, and moreover, every path expression in ϕ is in the normal form as defined in Section 3. By Path-containment and Superkey rules in $\mathcal{I}$, one can show that for every key ϕ in L, there is a key ϕ' of L in the key normal form such that for any XML tree T, $T \models \phi$ iff $T \models \phi'$. Thus without loss of generality, in the sequel we assume that keys of L are in the key normal form.

Theorem 4.4: The set $\mathcal{I}$ is sound and complete for finite implication of L constraints. ∎

The proof of the theorem is similar to that of Theorem 4.1. Soundness of $\mathcal{I}$ can be verified by induction on the lengths of $\mathcal{I}$-proofs. To prove the completeness, we show that given any finite set $\Sigma \cup \{\varphi\}$ of keys in L, either $\Sigma \vdash_{\mathcal{I}} \varphi$, or there is a finite XML tree G such that $G \models \Sigma$ and $G \models \neg\varphi$. To do so, we define an *abstract tree with "_$*$"* to be an extension of XML tree by allowing "_$*$" as node label. Let T be an abstract tree with "_$*$". A path in T is a parent-child path that may contain occurrences of "_$*$". Let R_1 be the sequence of labels in a path from node a to b in T, denoted by $T \models R_1(a, b)$. Observe that R_1 is a path expression of PL. For any path expression R_2 in PL, we say $T \models R_2(a, b)$ if $R_1 \subseteq R_2$. Given this, we can define node sets and satisfaction of key constraints in L for abstract trees with "_$*$". Analogous to Lemma 4.2, about abstract trees with "_$*$" we have the following (see appendix for a proof):

Lemma 4.5: For any finite set $\Sigma \cup \{\varphi\}$ of keys in L, if there is a finite abstract tree T with "_$*$" such that $T \models \Sigma$ and $T \models \neg\varphi$, then there is a finite XML tree G such that $G \models \Sigma$ and $G \models \neg\varphi$. ∎

Along the same lines of the proof of Theorem 4.1, we verify the completeness of $\mathcal{I}$ as follows. Let $\Sigma \cup \{\varphi\}$ be a finite set of keys in L, where $\varphi = (Q, \{P_1, ..., P_k\})$. If $Q = \epsilon$, then we have $\Sigma \vdash_{\mathcal{I}} \varphi$ by the rule Epsilon in $\mathcal{I}$. If $Q \neq \epsilon$, we construct a finite abstract tree T with "_$*$" such that $T \not\models \varphi$ in the same way as in the proof of Theorem 4.1. The tree T has the form shown in Fig. 4. We then modify T by "applying" keys in Σ. More specifically, for each ϕ in Σ, if the tree does not satisfy ϕ, then we merge nodes in the tree such that the modified tree satisfies ϕ, again in the same way as in the proof of Theorem 4.1. Finally, we obtain an abstract tree T' with "_$*$" such that $T' \models \Sigma$. If $T' \not\models \varphi$, then by Lemma 4.5, there is a finite XML tree G such that $G \models \Sigma$ but $G \not\models \varphi$. Thus $\Sigma \not\models \varphi$. Otherwise we can show $\Sigma \vdash_{\mathcal{I}} \varphi$. The rest of the proof is the same as that of Theorem 4.1.

Theorem 4.6: The finite implication problem for L is decidable in cubic time. ∎

Algorithm 4.1 can also be used to determine finite implication of keys expressed in L. It should be mentioned that checking containment of PL expressions is different from that for PL_w expressions. Below we give a brief analysis of the complexity of the algorithm when it is applied to L constraints.

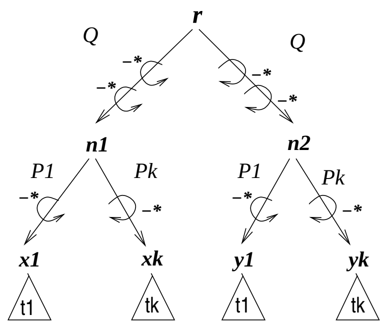

Figure 4: The abstract tree constructed in the proof of Theorem 4.4

Let $\Sigma \cup \{\varphi\}$ be a finite set of keys in L. Without loss of generality, assume that all path expressions in the set are in the normal form. As shown in Section 3, it takes linear time to transform a PL expression to an equivalent PL expression in the normal form. Step 1 of the algorithm transforms keys in $\Sigma \cup \{\varphi\}$ to the key normal form. By Theorem 3.2, this can be done in square time. To see this, consider a key $\phi = (Q_i, S_i)$ in $\Sigma \cup \{\varphi\}$, where $S_i = \{P_{i,1}, ..., P_{i,m}\}$. It takes at most $(|P_{i,1}| + ... + |P_{i,m}|)(|P_{i,1}| + ... + |P_{i,m}|)$ units of time to check containment of path expressions in S_i. Therefore, transforming ϕ to the key normal form takes $O(|\phi|^2)$ time, where $|\phi|$ is the size of ϕ. As a result, step 1 can be done in square time. In step 2, each constraint ϕ in Σ is examined at most once. Again by Theorem 3.2, processing ϕ takes $O(|\phi| \cdot |\varphi|)$ time in Case 1 and $O(|Q_i| \cdot |\phi| \cdot |\varphi|)$ time in Case 2 for testing possible partitions of Q_i, where $\phi = (Q_i, S_i)$. Thus the step 2 of the algorithm takes at most $O(|\Sigma|^2 \cdot |\varphi|)$ time in total, where $|\Sigma|$ and $|\varphi|$ are the sizes of Σ and φ, respectively. Therefore, the algorithm is in cubic time for determining L constraint implication.

5 Discussions

We have investigated two classes of key constraints for XML data introduced in [6] and studied their associated (finite) satisfiability and (finite) implication problems. These keys are capable of expressing many important properties of XML data [6]; moreover, in contrast to other proposals, keys defined in these two languages can be reasoned about efficiently. More specifically, these keys are always finitely satisfiable. In addition, inference rules and algorithms were provided for determining (finite) implication of key constraints of these two languages in square time and cubic time, respectively. We believe that these key constraints are simple yet expressive enough to be adopted by XML designers and maintained by systems for XML applications.

For further research, a number of issues deserve investigation. First, despite their simple syntax, there is an interaction between DTDs and our key constraints. To illustrate this, let us consider a simple key $\varphi = (X, \{\ \})$ and a simple DTD D:

```
<!ELEMENT  foo  (X, X)>
```

Obviously, there exists a finite XML tree that conforms to the DTD D (see, e.g., Fig. 5 (a)), and

(a) (b)

Figure 5: An XML tree conforming to D, and an XML tree satisfying φ

there is a finite XML tree that satisfies the key φ (e.g., Fig. 5 (b)). However, there is no XML data tree that both conforms to D and satisfies φ. This is because D requires an XML tree to have two distinct X elements, whereas φ requires that there is at most one X node immediately under the root. This shows that in the presence of DTDs, the analysis of key satisfiability and implication can be wildly different. It should be mentioned that keys defined in other proposals for XML, such as those introduced in XML Schema [24], also interact with DTDs or other type systems for XML. This issue was recently investigated in [15] for a class of keys defined in terms of XML attributes.

A second question is about foreign keys. As shown by [16, 14], the implication and finite implication problems for a class of keys and foreign keys defined in terms of XML attributes are undecidable, in the presence or absence of DTDs. However, under certain practical restrictions, these problems are decidable in PTIME. Whether these decidability results still hold for more complex keys and foreign keys needs further investigation.

Third, as shown in [6], relative keys are important for hierarchically structured data, including but not limited to XML data. We defer a full treatment of relative keys and their associated decision problems to another publication [7].

Finally, one might be interested in using different path languages to express keys, e.g., XPath [13] expressions. Questions in connection with containment and equivalence of these powerful path expressions, as well as (finite) satisfiability and (finite) implication of keys defined in terms of these complex path expressions are, to the best of our knowledge, still open.

Acknowledgements. We thank Leonid Libkin and Micheal Benedikt for helpful discussions.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufman, 2000.

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

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

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

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

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

[7] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Reasoning about relative keys for XML. Draft manuscript, 2000.

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

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

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

[11] P. Buneman, W. Fan, and S. Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences (JCSS)*, in press.

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

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

[14] W. Fan and L. Libkin. Finite implication of key and foreign key constraints for XML data. Technical Report TUCIS-TR-2000-003, Department of Computer and Information Sciences, Temple University, 2000.

[15] W. Fan and L. Libkin. Finite satisfiability of key and foreign key constraints for XML data. Technical Report TUCIS-TR-2000-002, Department of Computer and Information Sciences, Temple University, 2000.

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

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

[18] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addision Wesley, 1979.

[19] H. Hunt, D. Rosenkrantz, and T. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences (JCSS)*, 12:222–268, 1976.

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

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

[22] R. Ramakrishnan and J. Gehrke. *Database Management Systems.* McGraw-Hill Higher Education, 2000.

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

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

[25] P. Wadler. A Formal Semantics for Patterns in XSL. Technical report, Computing Sciences Research Center, Bell Labs, Lucent Technologies, 2000.
`http://www.cs.bell-labs.com/~wadler/topics/xml#xsl-semantics`.

Appendix

Proof of Theorem 3.1: The soundness of $\mathcal{I}_w^p$ can be verified by induction on the lengths of $\mathcal{I}_w^p$-proofs. For the completeness, we show that if $P \subseteq Q$ then it has an $\mathcal{I}_w^p$ proof. We prove this by induction on the number of occurrences of "_" in the path expression Q. First, observe that if $P \subseteq Q$ then they must be of the same length. Let $P = p_1.....p_n$ and $Q = q_1.....q_n$ where p_i or q_i represents a label or "_". When there are no wild cards in Q, $P \subseteq Q$ only when $P = Q$. In the proof system $\mathcal{I}_w^p$, this can be shown using the Reflexivity rule. Now, assume that for any Q with less than k occurrences of "_", if $P \subseteq Q$ then it has an $\mathcal{I}_w^p$ proof. We show the statement also holds for Q with k occurrences of "_". If P and Q are the same path expression then $P \subseteq Q$ can be proven using Reflexivity rule as before. Otherwise, let j be the largest index at which $p_1....p_j = q_1.....q_j$ ($j < n$). We denote the sequence of labels before and after p_{j+1} (q_{j+1}) in P (Q) as P_l (Q_l) and P_r (Q_r) respectively. As assumed, $P_l = Q_l$ and $p_{j+1} \neq q_{j+1}$. Since $P \subseteq Q$, it follows that p_{j+1} must be some label l and q_{j+1} is "_". Now, $P_l.l.Q_r \subseteq Q_l.\_.Q_r$ from Containment rule and $P_l.l.P_r \subseteq P_l.l.Q_r$ by induction hypothesis ($P_l.l.Q_r$ has one less wild card). Therefore, by Transitivity rule, $P \subseteq Q$. ■

Proof of Theorem 3.2: Given PL expressions P and Q, let $M(P)$ and $M(Q)$ be their NFAs, as defined in Section 3. Suppose $P \subseteq Q$. To show that this can be proved by using $\mathcal{I}^p$, it suffices to show the following claims:

Claim 1: $P \subseteq Q$ iff $S_1 \lhd S_2$, where $\lhd$ is a simulation relation as defined in Section 3.

Claim 2: If $S_1 \lhd S_2$, then $P \subseteq Q$ can be proved using inference rules in $\mathcal{I}^p$.

Before we show the claims, first observe that given a simulation relation $\lhd$ such that $S_1 \lhd S_2$, one can define a total function $\theta : N_1 \to N_2$ as follows: 1) $\theta(S_1) = S_2$. 2) Suppose $\theta(n_1) = n_2$. If $n_1 = F_1$ then by the definition of $\lhd$ and the properties of $M(P)$, we have $F_1 \lhd F_2$. In this case we define $\theta(F_1) = F_2$. If $n_1 \neq F_1$, then by the properties of $M(P)$, there exist $a \in T$ and $n_1' \in N_1$ such that $\delta_1(n_1, a) = n_1'$ and $n_1 \neq n_1'$. By the definition of $\lhd$, either $\delta_2(n_2, \_) = n_2$ and $n_1' \lhd n_2$, or $\delta_2(n_2, a) = n_2'$ for some $n_2' \in N_2$ and $n_1' \lhd n_2'$. Choose one such state n_2' and let $\theta(n_1') = n_2'$. Note that it is possible that $n_2' = n_2$. It is easy to verify that θ is a function with the following properties: $\theta(S_1) = S_2$, $\theta(F_1) = F_2$ and for any $n_1 \in N_1$, $n_1 \lhd \theta(n_1)$. In fact, θ is a simulation relation on $N_1 \times N_2$. Thus without loss of generality, we assume that simulation relation $\lhd$ is a function with these properties. As a result, there is no $n_2 \in N_2$ such that $F_1 \lhd n_2$ and $n_2 \neq F_2$.

To show Claim 1, recall [18] that the closure function of a transition function δ is defined to be $\hat{\delta} : N \times (T \cup \{\_\})^* \to \mathcal{P}(N)$ such that:

$$\begin{aligned} \hat{\delta}(n, \epsilon) &= \{n\} \\ \hat{\delta}(n, w.a) &= \{p \mid \exists x \in \hat{\delta}(n, w),\, p \in \delta(x, a)\} \end{aligned}$$

where $\mathcal{P}(N)$ denotes the powerset of N. Let $\hat{\delta}_1, \hat{\delta}_2$ be the closure functions of δ_1 and δ_2, respectively. Observe that $P \subseteq Q$ iff for any $\rho \in P$, if $F_1 \in \hat{\delta}_1(S_1, \rho)$ then $F_2 \in \hat{\delta}_2(S_2, \rho)$. Using this notion we show Claim 1 as follows. Assume $S_1 \lhd S_2$. By induction on $|\rho|$, where ρ is a path, one can show that if $n_1 \in \hat{\delta}_1(S_1, \rho)$ then there exists $n_2 \in \hat{\delta}_2(S_2, \rho)$ such that $n_1 \lhd n_2$. If $\rho = \epsilon$ then by the definition of $\hat{\delta}$, $\hat{\delta}_1(S_1, \epsilon) = \{S_1\}$, and $\hat{\delta}_2(S_2, \epsilon) = \{S_2\}$, and $S_1 \lhd S_2$ by assumption. Assume the statement for $|\rho| < k$. We next show that the statement holds for $|\rho| = k$. For $|\rho| > 0$, let $\rho = \rho'.l$, where $l \in T \cup \{\_\}$, and $\hat{\delta}_1(S_1, \rho) = n_1$. If $n_1 = F_1$, then $n_2 = F_2$ and $F_1 \lhd F_2$ by definition of $\lhd$. If $n_1 \neq F_1$, then let $\hat{\delta}_1(S_1, \rho') = n_1'$, and $\hat{\delta}_2(S_2, \rho') = n_2'$. By definition of $\lhd$, if $l = \_$ then $\delta_1(n_1', \_) = n_1'$, $\delta_2(n_2', \_) = n_2'$, and by the induction hypothesis $n_1' \lhd n_2'$. Since $n_1 = n_1'$ and $n_2 = n_2'$, we have $n_1 \lhd n_2$. If $l = a$, $a \in T$, then by the definition of $\lhd$, either $\delta_2(n_2', \_) = n_2'$ and $n_1 \lhd n_2'$, or there exists $n_2 \in N_2$ such that $\delta_2(n_2', a) = n_2$ and $n_1 \lhd n_2$. By the induction hypothesis, $n_1' \lhd n_2'$, and therefore $n_1 \lhd n_2$. Thus if $F_1 \in \hat{\delta}_1(S_1, \rho)$ then by the definition of $\lhd$, we must have $F_2 \in \hat{\delta}_2(S_2, \rho)$. That is, $P \subseteq Q$. For the other direction, assume $P \subseteq Q$. We can show that for any path ρ, if $n_1 \in \hat{\delta}_1(S_1, \rho)$ then there exists $n_2 \in \hat{\delta}_2(S_2, \rho)$ such that $n_1 \lhd n_2$. To see this, note that for any $\rho \in P$, we have $F_1 \in \hat{\delta}_1(S_1, \rho)$, and since $P \subseteq Q$, $F_2 \in \hat{\delta}_2(S_2, \rho)$. Thus we can define $F_1 \lhd F_2$. In addition, for any path ρ, if $\hat{\delta}_1(S_1, \rho) \subseteq N_1$, then there is path ρ' such that $F_1 \in \hat{\delta}_1(S_1, \rho.\rho')$. Thus the statement can be easily verified by contradiction. Observe that $\hat{\delta}_1(S_1, \epsilon) = \{S_1\}$ and $\hat{\delta}_2(S_2, \epsilon) = \{S_2\}$. Thus $S_1 \lhd S_2$. Hence Claim 1 holds.

We next prove Claim 2. Assume there is a simulation relation $\lhd$ such that $S_1 \lhd S_2$. By the definition of $\lhd$ and the properties of $M(P)$ given above, there is a total mapping $\theta : N_1 \to N_2$ such that $\theta(S_1) = S_2$, $\theta(F_1) = F_2$, and for any $n_1 \in N_1$, $n_1 \lhd \theta(n_1)$. Let the sequence of states in $M(P)$ be $\vec{v_1} = p_1, \ldots, p_k$, where $p_1 = S_1$ and $p_k = F_1$, and similarly, let the sequence of states in $M(Q)$ be $\vec{v_2} = q_1, \ldots, q_l$, where $q_1 = S_2$ and $q_l = F_2$. It is easy to verify that for any $i, j \in [1, k]$, if $i < j$, $\theta(p_i) = q_{i'}$ and $\theta(p_j) = q_{j'}$, then $i' \leq j'$. We define an equivalence relation $\sim$ on N_1 as follows:

$$p_i \sim p_j \quad \text{iff} \quad \theta(p_i) = \theta(p_j).$$

Let $[p]_\sim$ denote the equivalence classes of p with respect to $\sim$. An equivalence class is *non-trivial* if it contains more than one state. For any equivalence class $[p]$, let p_i and p_j be the smallest and largest states in $[p]$ respectively. That is, for any $p_s \in [p]$, $i \leq s \leq j$. By treating p_i as the start state and p_j as the final state, we have a NFA that recognizes a regular expression, denoted by $P_{i,j}$. Similarly, we can define $P_{1,i}$ and $P_{j,k}$ such that $P = P_{1,i}.P_{i,j}.P_{j,k}$. It is easy to verify that if $[p]$ is a non-trivial equivalence class, then there must be $\delta_2(\theta(p_i), \_) = \theta(p_i)$. In other words, $\theta(p_i)$ indicates an occurrence of "$\_*$" in Q. Observe that $P_{1,i}.P_{i,j}.P_{j,k} \subseteq P_{1,i}.\_*.P_{j,k}$. This is an application of the Containment rule in $\mathcal{I}^p$. By an induction on the number of non-trivial equivalence classes, one can show that $P \subseteq Q$ can always be proved using the Containment, Transitivity and Reflexivity rules in $\mathcal{I}^p$ as illustrated above. Thus $\mathcal{I}^p$ is complete for inclusion of PL expressions. ∎

Proof of Lemma 4.2: Let $\Sigma \cup \{\varphi\}$ be a finite set of L_w keys, and T be a finite abstract tree with "$\_$" such that $T \models \Sigma$ and $T \models \neg\varphi$. We define a finite XML tree G as follows. Let η be a label that does not occur in any key of $\Sigma \cup \{\varphi\}$. We replace every occurrence of "$\_$" in T by η. Let G be T with this modification. Observe that G and T have the same set of nodes. In addition, for any nodes a, b in G, if there is a path ρ such that $G \models \rho(a, b)$, then there is a parent-child path R in T such that $T \models R(a, b)$. Observe that R is a path expression in PL_w. In addition, R and ρ are the same except for each occurrence of "$\_$" in R, η appears at the corresponding position in ρ. Let us refer to R as the *path expression w.r.t.* ρ and conversely, ρ as the *path w.r.t.* R. We show $G \models \Sigma$ and $G \models \neg\varphi$. To do so, it suffices to show the following:

Claim: Let P be a path expression in PL_w, and a, b be nodes in G. Then there is ρ in P such that $G \models \rho(a, b)$ iff $T \models P(a, b)$, i.e., $T \models R(a, b)$ and $R \subseteq P$, where R is the path expression w.r.t. ρ.

From the claim follows immediately that for any path expression P in PL_w, $[\![P]\!]$ consists of the same nodes in T and G. For if node a is in $[\![P]\!]$ in T, then $T \models P(r, a)$, where r is the root of T. Thus there is a path expression R in T such that $T \models R(r, a)$ and $R \subseteq P$. By the claim, we have $G \models \rho(r, a)$, where ρ is the path w.r.t. R and $\rho \in P$. That is, a is in $[\![P]\!]$ in G. Conversely, if a is in $[\![P]\!]$ in G, then there is a path $\rho \in P$ such that $G \models \rho(r, a)$. Again by the claim, $T \models R(r, a)$ and $R \subseteq P$, where R is the path expression w.r.t. ρ. Thus a is in $[\![P]\!]$ in T.

Given these, we show $G \models \Sigma$. Suppose, by contradiction, there is a key $\phi = (Q, \{P_1, ..., P_k\})$ in Σ such that $G \models \neg\phi$. Then there must be two distinct nodes $n_1, n_2 \in [\![Q]\!]$ and moreover, for all $i \in [1, k]$, there are path $\rho_i \in P_i$ and nodes $x_i \in n_1[\![\rho_i]\!]$, $y_i \in n_2[\![\rho_i]\!]$ such that $x_i =_v y_i$. By the claim, $T \models P_i(n_1, x_i) \wedge P_i(n_2, y_i)$ for all $i \in [1, k]$. Thus $T \not\models \phi$, which contradicts our assumption. Similarly, we show $G \models \neg\varphi$. Let $\varphi = (Q, \{P_1, ..., P_k\})$. By $T \models \neg\varphi$, there exist two distinct nodes $n_1, n_2 \in [\![Q]\!]$ and for all $i \in [1, k]$, there exist nodes x_i, y_i such that $x_i =_v y_i$ and $T \models P_i(n_1, x_i) \wedge P_i(n_2, y_i)$. That is, there exists a path (expression) R_i in T such that $T \models R_i(n_1, x_i) \wedge R_i(n_2, y_i)$, where $R_i \subseteq P_i$. Thus by the claim, there is path $\rho_i \in P_i$ such that $x_i \in n_1[\![\rho_i]\!]$, $y_i \in n_2[\![\rho_i]\!]$. Hence $G \models \neg\varphi$.

Next, we show the claim.

(1) Assume that $T \models P(a, b)$, i.e., there is a parent-child path from a to b in T such that the sequence of labels of the path is R and $R \subseteq P$. By the definition of G, we have $G \models \rho(a, b)$, where ρ is the path w.r.t. R. Recall that ρ is obtained by replacing occurrences of "_" with η. Since $R \subseteq P$, obviously $\rho \in P$.

(2) Conversely, assume that there exists a path $\rho \in P$ such that $G \models \rho(a, b)$. By the definition of G, we have $T \models R(a, b)$, where R is the path expression w.r.t. ρ. We next show $R \subseteq P$ by induction on the number of occurrences of "_" in R, denoted by w_R. When $w_R = 0$, we have $R = \rho$ and $R \subseteq P$ since $\rho \in P$. Assume the statement for $w_R < k$. We next show the statement also holds for $w_R = k$. Let $R = R_1.\_.R_2$, where R_2 does not contain any "_", and R_1 contains less than k occurrences of "_". Observe that ρ must be $\rho_1.\eta.\rho_2$, where $\rho_2 = R_2$ and ρ_1 is the path w.r.t. R_1. Thus $|\rho_1| = |R_1|$. By $\rho \in P$ and the definition of PL_w expressions, we have $|\rho| = |P|$. Therefore, we can write P as $P = P_1.l.P_2$ such that $|P_1| = |\rho_1| = |R_1|$ and $|P_2| = |\rho_2| = |R_2|$. By the induction hypothesis, $R_1 \subseteq P_1$ and $R_2 \subseteq P_2$. Moreover, by $\eta \subseteq l$ and the choice of η, which is not in P, l must be the wild card "_". Thus $R_1.\_.R_2 \subseteq P_1.\_.P_2$ by Transitivity in $\mathcal{I}_w^p$. Hence the statement also holds for $w_R = k$. This completes the proof of Lemma 4.2. ■

Proof of Theorem 4.1: We show that $\mathcal{I}_w$ is complete for finite implication of L_w constraints. Let $\Sigma \cup \{\varphi\}$ be a finite set of keys in L_w, where $\Sigma = \{\phi_1, \ldots, \phi_n\}$, $\phi_i = (Q_i, \{P_{i1}, ..., P_{im_i}\})$ and $\varphi = (Q, \{P_1, ..., P_k\})$. We show that if $\Sigma \models \varphi$ then $\Sigma \vdash_{\mathcal{I}_w} \varphi$. Let T be the finite abstract tree given in Fig. 3 (a). We execute the following algorithm on T:

for each $i \in [1, n]$ do /* Recall $\phi_i = (Q_i, \{P_{i_1}, ..., P_{i_{m_i}}\})$ */
    if there are nodes $x, x'_1, \ldots, x'_{m_i}$ in T_1 and $y, y'_1, \ldots, y'_{m_i}$ in T_2 such that

$T \models Q_i(r, x) \wedge Q_i(r, y) \wedge$
$P_{i1}(x, x'_1) \wedge \ldots \wedge P_{im_i}(x, x'_{m_i}) \wedge P_{i1}(y, y'_1) \wedge \ldots \wedge P_{im_i}(y, y'_{m_i}) \wedge$
$x'_1 =_v y'_1 \wedge \ldots \wedge x'_{m_i} =_v y'_{m_i}$
then merge x and y as follows:
Case 1: if there is Q' such that $Q' \preceq Q$ and $Q' \subseteq Q_i$ (a prefix of Q is contained in Q_i)
then merge x, y and their ancestors as shown in Fig. 3 (b)
Case 2: if there is Q' such that $Q' \prec Q_i$ and $Q \subseteq Q'$ (Q is contained in a proper prefix of Q_i)
then merge x, y and their ancestors as shown in Fig. 3 (c)

In both cases in the above algorithm, we merge T_1's and T_2's nodes in path Q_i. In Case 1, the subtree under x and the subtree under y will both be under the same node $x = y$. In Case 2, since Q is contained in a proper prefix of Q_i and by the definition of T, we must have $m_i = 1$. That is, $\phi_i = (Q_i, \{P'_i\})$. The subtree P'_i in T_1 and T_2 will both be rooted at the same node $x = y$, as illustrated in Fig. 3 (c). Since φ is satisfied at this point, that is, we will show that φ is provable, we can therefore discard the rest of the key paths in $\{P_1, ..., P_k\}$.

It is clear that this algorithm terminates. Let T' be the abstract tree with "_" obtained by executing the algorithm. It is easy to verify $T' \models \Sigma$. Moreover, if $T' \not\models \varphi$, then by Lemma 4.2, there is a finite XML tree G such that $G \models \Sigma$ and $G \not\models \varphi$. Thus $\Sigma \not\models \varphi$. If $T' \models \varphi$, we show $\Sigma \vdash_{\mathcal{I}_w} \varphi$. Observe that by the definitions of T and $T \models R(x, y)$ for path expression R in PL_w, if $x_l =_v y_l$ for all $l \in [1, m_i]$ then in Case 1 we have $Q = Q'.Q''$, $Q' \subseteq Q_i$ and for every $l \in [1, m_i]$, there exists a $j \in [1, k]$ such that $Q''.P_j \subseteq P_{il}$. In Case 2, we have $Q_i = Q'.Q''$, $Q \subseteq Q'$ and for some $j \in [1, k]$, either $m_i = 1$ and $P_j \subseteq Q''.P_{i1}$, or $m_i = 0$ and $P_j \subseteq Q''$. In addition, n_1 and n_2 are merged only in one of the following cases:

(a) There exists $\phi_i = (Q_i, \{P_{i1}, \ldots, P_{im_i}\})$ in Σ such that $Q \subseteq Q_i$ and for every $l \in [1, m_i]$, there is $j \in [1, k]$ such that $P_j \subseteq P_{il}$. This makes Case 1 of the algorithm applicable. Merging n_1 and n_2 due to this constraint corresponds to applications of the Target-containment, Key-containment, and Superkey rules. Thus $\Sigma \vdash_{\mathcal{I}_w} \varphi$.

(b) For some $\phi_i \in \Sigma$, $\phi_i = (Q_i, \{P_{i1}\})$ such that $Q_i = Q'.Q''$, $Q \subseteq Q'$, $|Q''| > 0$ and for some $j \in [1, k]$, $P_j \subseteq Q''.P_{i1}$. This makes Case 2 of the algorithm applicable. Merging n_1 and n_2 due to this constraint corresponds to applications of Subnodes, Target-containment, Key-containment rules, and Superkey rule (when $k > 1$). Thus again $\Sigma \vdash_{\mathcal{I}_w} \varphi$.

(c) For some $\phi_i \in \Sigma$, $\phi_i = (Q_i, \emptyset)$ such that $Q_i = Q'.Q''$, $Q \subseteq Q'$, and for some $j \in [1, k]$, $P_j \subseteq Q''$. This again makes Case 2 of the algorithm applicable. Identifying n_1 and n_2 by this constraint corresponds to applications of Superkey (i.e., if $(Q_i, \emptyset)$ then $(Q_i, \{\epsilon\})$) Subnodes (i.e., if $(Q'.Q'', \{\epsilon\})$ then $(Q', \{Q''\})$), Target-containment, Key-containment rules, and Superkey rule (when $k > 1$). Hence $\Sigma \vdash_{\mathcal{I}_w} \varphi$.

Therefore, if $\Sigma \models \varphi$, then $\Sigma \vdash_{\mathcal{I}_w} \varphi$. That is, $\mathcal{I}_w$ is complete for L_w constraint implication. ■

Proof of Lemma 4.5: The proof is similar to that of Lemma 4.2, except the proof of a claim. Let $\Sigma \cup \{\varphi\}$ be a finite set of keys in L, and T be a finite abstract tree with "_*" such that $T \models \Sigma$ and $T \not\models \varphi$. We define a finite XML tree G as follows. Let η be a label that does not occur in any key of $\Sigma \cup \{\varphi\}$. We substitute η for every occurrence of "$\_*$" in T. Let G be T with this modification. Observe that G and T have the same set of nodes. In addition, for any nodes a, b in G, if there is a path ρ such that $G \models \rho(a,b)$, then there is a parent-child path R in T such that $T \models R(a,b)$. Observe that R is a path expression in PL and may contain "$\_*$". In addition, R and ρ are the same except for each occurrence of "$\_*$" in R, the label η appears at the corresponding position in ρ. Let us refer to R as the *path expression w.r.t.* ρ and conversely, ρ as the *path w.r.t.* R. We show $G \models \Sigma$ and $G \models \neg\varphi$. To do so, it suffices to show the following claim. For if the claim holds, then we can show $G \models \Sigma$ and $G \models \neg\varphi$ as in the proof of Lemma 4.2.

Claim 1: Let P be a path expression in PL, and a, b be nodes in G. Then exists ρ in P such that $G \models \rho(a,b)$ iff $T \models P(a,b)$, i.e., $T \models R(a,b)$ and $R \subseteq P$, where R is the path expression w.r.t. ρ.

(1) Assume that $T \models P(a,b)$, i.e., there is a parent-child path R from a to b in T such that $R \subseteq P$. By the definition of G, we must have $G \models \rho(a,b)$, where ρ is the path w.r.t. R. Recall that ρ is obtained by substituting η for occurrences of "$\_*$". Since $R \subseteq P$, we have $\rho \in P$.

(2) Conversely, assume that there exists a path $\rho \in P$ such that $G \models \rho(a,b)$. By the definition of G, we have $T \models R(a,b)$, where R is the path expression w.r.t. ρ. Thus it suffices to show $R \subseteq P$. To do so, we consider the NFAs of R, P and ρ as defined in Section 3:

$$\begin{aligned} M(R) &= (N_R,\, A \cup \{\_\},\, \delta_R, S_R, F_R), \\ M(P) &= (N_P,\, A \cup \{\_\},\, \delta_P, S_P, F_P), \\ M(\rho) &= (N_\rho,\, A \cup \{\eta\},\, \delta_\rho, S_\rho, F_\rho), \end{aligned}$$

where A is an alphabet that contains neither "$\_$" nor η. Recall that NFAs for PL expressions have a "linear" structure as shown in Fig. 2. In particular, since ρ does not contain "$\_*$", it has a strict linear structure. Let the sequence of states in N_ρ be $s_1, \ldots, s_m$, where $s_1 = S_\rho$ and $s_m = F_\rho$. Then for any $i \in [1, m-1]$, there is exactly one $l \in A \cup \{\eta\}$ such that $\delta_\rho(s_i, l) \neq \emptyset$. More precisely, $\delta_\rho(s_i, l) = s_{i+1}$. For any $l \in A \cup \{\eta\}$, $\delta_\rho(F_\rho, l) = \emptyset$. By the definition of G, there is a function f from N_ρ to N_R. More specifically, let the sequence of states in N_R be $n_1, \ldots, n_k$, where $n_1 = S_R$ and $n_k = F_R$. Then we have the following:

- $f(S_\rho) = S_R$ and $f(F_\rho) = F_R$.
- For any $i, j \in [1, m]$, if $f(s_i) = n_{i'}$, $f(s_j) = n_{j'}$ and $i < j$, then $i' \leq j'$.
- For any $i \in [1, m]$ and $l \in A$, $\delta_\rho(s_i, l) = s_{i+1}$ iff $\delta_R(f(s_i), l) = f(s_{i+1})$ and $f(s_i) \neq f(s_{i+1})$.
- For any $i \in [1, m]$, $\delta_\rho(s_i, \eta) = s_{i+1}$ iff $\delta_R(f(s_i), \_) = f(s_{i+1})$ and $f(s_i) = f(s_{i+1})$. In particular, if $\delta_R(F_R, \_) = F_R$ then $\delta_\rho(s_{m-1}, \eta) = F_\rho$ and $f(s_{m-1}) = f(F_\rho) = F_R$.

We define an equivalence relation $\sim$ on N_ρ such that

$$s \sim s' \quad \text{iff} \quad f(s) = f(s').$$

Let us use $[s]$ to denote the equivalence class of s w.r.t. $\sim$. We assume without loss of generality that R is in the normal form. Then observe that $[s]$ consists of at most two states; if $[s] = \{s\}$, then there is $l \in A$ such that $\delta_\rho(s, l) = s'$, and if $[s] = \{s, s'\}$ then there is some $i \in [1, m-1]$ such that $s = s_i$, $s' = s_{i+1}$, $\delta_\rho(s, \eta) = s'$ and $f(s) = f(s')$. Given these, we define a function g from N_R to the equivalence classes such that for all $n \in N_R$,

$$g(n) = [s] \quad \text{iff} \quad f(s) = n.$$

Recall in the proof of Theorem 3.2, we have shown the following: for any PL expressions Q and Q', let $M(Q), M(Q')$ their NFAs, $N_Q, N_{Q'}$ the sets of states in $M(Q), M(Q')$, $S_Q, S_{Q'}$ the start states of $M(Q), M(Q')$, and $F_Q, F_{Q'}$ the final states of $M(Q), M(Q')$, respectively, then 1) $Q \subseteq Q'$ iff $S_Q \lhd S_{Q'}$, where $\lhd$ is a simulation relation as defined in Section 3; 2) there is a function θ from N_Q to $N_{Q'}$ such that $\theta(S_Q) = S_{Q'}$, $\theta(F_Q) = F_{Q'}$, and for any $s \in N_Q$, $s \lhd \theta(s)$. By $\rho \in P$, we have that the language defined by ρ (which consists of a single string ρ) is contained in the language defined by P, i.e., $\rho \subseteq P$. Thus there exists such a function θ from N_ρ to N_P and a simulation relation $\lhd$ such that $\theta(S_\rho) = S_P$, $\theta(F_\rho) = F_P$, and for any $s \in N_\rho$, $s \lhd \theta(s)$. It is easy to verify:

Claim 2: for all $s, s' \in [s]$, $\theta(s) = \theta(s')$.

Indeed, as observed earlier, if $s, s' \in [s]$, then there is some $i \in [1, m-1]$ such that $s = s_i$, $s' = s_{i+1}$ and $\delta_\rho(s, \eta) = s'$. Since η does not appear in P, if $\theta(s) = n'$ and $\theta(s') = n''$, then there must be $\delta_P(n', \_) = n''$ and $n' = n''$, by the definition of simulation relations. As a result, we can define $\theta([s])$ to be $\theta(s)$. Given these, to show $R \subseteq P$, it suffices to show that for any $n \in N_R$,

$$n \;\lhd\; \theta(g(n)).$$

For if it holds, then $S_R \lhd \theta(g(S_R)) = \theta(S_\rho) = S_P$. We next show that this holds. Assume, by contradiction, there is $n \in N_R$ such that it is not the case that $n \lhd \theta(g(n))$. Let n be such a state with the largest index in the sequence of states in N_R starting from S_R. Then by the definition of simulation relations given in Section 3, we must have one of the following cases.

(i) $n = F_R$ and either 1) $\theta(g(F_R)) \neq F_P$, or 2) $\theta(g(F_R)) = F_P$ but $\delta_R(F_R, \_) = F_R$, $\delta_P(F_P, \_) = \emptyset$. The first case contradicts the assumption that $g(F_R) = [F_\rho]$ and $\theta([F_\rho]) = \theta(F_\rho) = F_P$. In the second case, by $\delta_R(F_R, \_) = F_R$, we have $g(F_R) = \{F_\rho, s_{m-1}\}$ and $\delta_\rho(s_{m-1}, \eta) = F_\rho$. By Claim 2, there must be $\theta(s_{m-1}) = \theta(F_\rho) = F_P$ and $\delta_P(F_P, \_) = F_P$. Again this contradicts the assumption.

(ii) $n \neq F_R$ and either 1) $\delta_R(n, \_) = n$ but $\delta_P(\theta(g(n)), \_) \neq \theta(g(n))$, or 2) there is some $l \in A$ such that $\delta_R(n, l) = n'$ but neither $\delta_P(\theta(g(n)), l) \neq \theta(g(n'))$ nor $\delta_P(\theta(g(n)), \_) = \theta(g(n))$. In the first case, we must have $g(n) = \{s_i, s_{i+1}\}$ and $\delta_\rho(s_i, \eta) = s_{i+1}$. By Claim 2, there must be $\theta(s_i) = \theta(s_{i+1})$, $\delta_P(\theta(s_i), \_) = \theta(s_i)$ and $\theta(g(n)) = \theta(s_i)$. Thus $\delta_P(\theta(g(n)), \_) = \theta(g(n))$, which contradicts the assumption. In the second case, given $\delta_R(n, l) = n'$, there must be either $\delta_P(\theta(g(n)), l) = \theta(g(n'))$ or $\delta_P(\theta(g(n)), \_) = \theta(g(n))$, by the definition of simulation relations and $g(n) \lhd \theta(g(n))$. Again this contradicts the assumption. Thus we have $n \lhd \theta(g(n))$ for all $n \in N_R$. This shows that Claim 1 holds and completes the proof of Lemma 4.5. ∎