Structural Constraints for XML

April Kwong and Michael Gertz
Department of Computer Science
University of California at Davis
Davis, CA 95616-8562, U.S.A.
{kwonga|gertz}@cs.ucdavis.edu

Abstract

Recently, there have been several proposals for modeling traditional integrity constraints, such as keys and foreign keys, in the context of grammar-based schema formalisms for XML. Purely pattern-based schema formalisms, however, have not been investigated as alternative or add-on for modeling schema constraints for XML.

In this paper, we propose the concept of structural constraints (XSCs) for XML as a pattern-based schema formalism. XSCs provide effective means to specify important patterns in form of path implication, absence and co-occurrence conditions XML documents have to satisfy. XSCs can be used as stand-alone schema and in conjunction with a grammar-based schema. As stand-alone formalisms, we show that the implication and consistency problems are efficiently decidable, and that there exists a sound and complete axiomatization for XSCs. If used in conjunction with a grammar-based schema formalism (here DTD), we show how specialized DTDs can be derived, thus providing a means to map a combination of grammar- and pattern-based schema specification to a semantic rich grammar-based specification.

1 Introduction

Although XML documents are supposed to be self-describing and thus do not necessarily require any schema or type system, such systems are known to be useful in order to more efficiently process, query, and manage XML documents. Several schema formalisms have been proposed, including Document Type Definition (DTD) [7] and XML Schema [23], both of which are standardized by the W3C, and more formal language-based schema systems, such as RELAX NG [11] or XDuce [16].

The above grammar-based schema formalisms, however, fall short when it comes to the specification of complex structural constraints that go beyond the specification of admissible parent-child and sibling relationships among elements in XML documents. Consider, for example, a scenario where in a document fragment of a specific type (element) the existence of a path emanating from that type requires (or precludes) the existence of one or more other paths in the same fragment. Using existing schema formalisms, such relationships can only be modeled through extensive type specializations, often resulting in complex (in terms of content models) and large (in terms of elements) schema specifications, which are not necessarily desirable from a design and application point of view.

Pattern-based schema formalisms, in contrast, offer expressive models (typically based on XPath [10]) to specify patterns and relationships among patterns, ranging from simple path to complex tree patterns. Interestingly enough, there are only very few pattern-based schema formalisms, most notably the not-yet-formalized Schematron [17]. Pattern-based schema languages, however, do not provide rich typing mechanisms or data types.

From a schema design point of view, it thus seems desirable to combine grammar- and pattern-based schema formalisms. In this paper, we address this issue by describing a schema specification language that allows users to specify path-based, structural constraints (XSCs) XML documents have to satisfy. In particular, we study the implication and consistency problem of respective schema specifications, both as a stand-alone schema formalisms and in combination with a grammar-based schema (DTD). Before we present the specific objectives and results of the approach presented in this paper, we first detail the principles behind XSCs using an example.

Motivating Example. To better illustrate the basic ideas behind structural constraints for XML documents and their interaction with grammar-based schema formalisms, throughout this paper we will use a sample scenario where information about auctions, auction items, sellers, and buyers is represented as XML documents. For this, we assume the following self-explanatory DTD (types not further specified are assumed to be of type `#PCDATA`):

```
<!ELEMENT auctions      (auction)*>
<!ELEMENT auction       (item, seller, buyer, price, payment)>
<!ELEMENT buyer         (contact)>
<!ELEMENT seller        (contact, type)>
<!ELEMENT contact       (name, email?, phone?)>
<!ELEMENT type          (personal | store)>
<!ELEMENT store         (name, UBI)>
<!ELEMENT item          (itemID, descr)>
<!ELEMENT price         (closingBid, tax?)>
<!ELEMENT payment       (creditCard | moneyorder | paypal)>
<!ELEMENT creditCard    (cardNum, expDate)>
```

Besides the vocabulary (element names), this DTD specifies the admissible nesting of elements as parent-child and sibling constraints using content models. Figure 1 shows an XML document tree that conforms to the above DTD.

Assume one wants to model the aspect that for an auction, if a seller is of type store, then the price must contain tax information, and no tax information otherwise. This condition, which naturally restricts admissible paths in document fragments of type `auction`, cannot be expressed in a DTD using the above vocabulary but requires a specialization of the DTD and thus the introduction of specialized types. Now assume another condition requiring that if the payment type is `paypal`, the buyer must provide email information. This condition can be modeled in a way similar to the first condition. Obviously, the more such structural constraints are added, the more specialized types need to be introduced.

The above two constraints can be specified in form of *structural constraints* that specify relationships among paths and that XML documents have to satisfy. Informally, the first condition above, for example, requires that in the context `auction`, the existence of the path `seller.type.store` requires the existence of the path `price.tax`. Similarly, in the same context, the existence of the path `seller.type.personal` precludes the existence of the path `price.tax`. Given the above

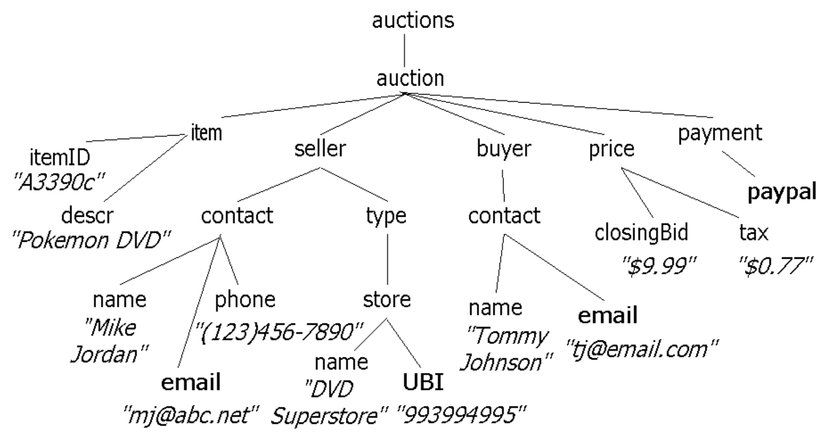

Figure 1: Example XML document tree

notion of structural constraints, which will be formalized in the next section, two classes of questions arise that address properties of structural constraints as a stand-alone schema formalism and in combination with a grammar-based schema formalism.

First, given a set Σ of structural constraints specifying admissible relationships among paths, what other structural constraints are implied by Σ? The same question occurs in the context of a DTD (as an example of a grammar-based schema formalism) since a DTD implies some trivial parent-child and sibling constraints, which can be expressed as structural constraints. The answer to these questions obviously plays an important role in, e.g., optimizing queries that are based on path-patterns, such as XPath or XQuery [8]. For example, if it is known that in every document, the existence of one path implies the existence of another path, this information can be used to rewrite an XPath expression such that it contains no redundant (trivially satisfied) conditions to check.

Second, given a set Σ of structural constraints, is there an XML document that conforms to Σ? The answer to this question is of particular interest if structural constraints are used as stand-alone schema formalism and a schema is developed incrementally. Again, the same question can be asked if Σ is considered in the context of a DTD. Both questions address the consistency (satisfiability) of the structural aspects of a (combined) schema specification and are essential in designing a schema for which XML documents that conform to the schema exist.

Contributions. In this paper, we study in detail the role and usage of XML structural constraints (XSCs) as a stand-alone schema formalisms and in combination with a grammar-based schema formalism. The main contributions of this paper are as follows:

1. We present the syntax and semantics of XSCs based on path expressions. Such constraints allow the specification of path implication, path absence, and path co-occurrence as basic patterns XML documents have to exhibit.

2. We show that the implication and consistency problems for XSCs, if used as stand-alone schema formalism, are efficiently decidable. For both problems, the notion of constraint graph is introduced, which is used to (1) reason about implication and consistency and (2)

to derive templates of XML documents satisfying the XSCs.

3. We provide a sound and complete axiomatization of XSCs in form of inference rules.
4. Using DTD as a representative of a grammar-based schema formalisms, we show how XSCs can be derived from a DTD.
5. Given a DTD and a set of XSCs, we show how a specialized DTD can be computed from a combined schema specification. This method also allows to check the consistency of a DTD in the presence of XSCs.

It should be noted that a major motivation of our work is to provide users with means to add more semantics to schema specifications while keeping specifications simple and easy to maintain. The latter aspect is in particular important if storage schemes for XML documents utilize a mapping from a schema specification to, e.g., (object-)relational databases.

Related Work. Abiteboul's and Vianu's work on path constraints [3] has inspired numerous proposals on extending schema formalisms for semistructured data and XML by mechanisms that resemble sophisticated forms of key, foreign key, and inverse constraints known from relational and object-oriented databases [5, 4, 14, 12]. The interaction, i.e., implication and consistency, between path constraints and schema types has been studied in [6, 13, 1]. The aim of these works can be considered as orthogonal to ours. Whereas our approach is concerned with (path-based) structural constraints as stand-alone or add-on schema formalism, the above works focus almost exclusively on key and foreign key models.

The general role of pattern-based languages for tree-structured (XML) data is surveyed in [20], primarily focusing on the features and expressiveness of respective query languages. There are only a few works that deal with rich typing mechanisms (in addition to a given schema), which can be considered as structural constraints. [9] introduce the concept of extended local constraints to specify required edges in semistructured data. Type subsumption has been studied in detail in [16, 18]. In [21, 22], the concept of specialized DTDs is introduced to represent richer type structures in addition to "normal" DTDs. In these works specialized DTDs are determined in the context of queries against XML documents. Although there is no explicit mentioning that specialized DTDs can be used to represent certain types of structural constraints, in Section 4 we will show how a specialized DTD can be obtained from a set of structural constraints in the presence of a DTD. To the best of our knowledge, Schematron [17] is the only framework that exclusively deals with specifying schemas for XML documents through path and tree patterns. Schematron, however, has not been formalized yet. Besides providing a first step towards a formalization of Schematron, our approach furthermore suggests important tools for checking of and reasoning about a pattern-based schema formalism for XML.

Organization. In Section 2, we introduce the syntax and semantics of XSCs. The implication and consistency problem for XSCs as stand-alone schema formalism is studied in Section 3. In Section 4, we show how XSCs can be derived from a DTD and implied XSCs can be determined. In this section, we also outline a method for deriving a specialized DTD from a set of XSCs and a DTD. We conclude the paper in Section 5 with a summary and outline of future extensions to the proposed work.

2 Structural Constraints – Syntax and Semantics

We adopt the standard view in which XML documents are modeled as ordered, node labeled trees. The labels for the nodes $\mathbf{V}_d$ in a document d are taken from a finite alphabet $\mathbf{E}$ of element names. A *path* p is simply a sequence of element names $e_1.e_2.\cdots.e_n, e_i \in \mathbf{E}$. The empty path is denoted ϵ. The number of elements in a path p is called the path's length, denoted $length(p)$. A path is a *document path* if it describes a path occurring in a given XML document d. A *rooted document path* has the document's root element as first element. Given an XML document d, the same (rooted) document path $p = e_1.e_2.\cdots.e_n$ obviously can specify different nodes in $\mathbf{V}_d$ that are labeled e_n. The path leading to a node $v \in \mathbf{V}_d$ is also called the path of v.

The concept underlying structural constraints for XML documents is based on how sets of nodes specified by two rooted document paths are related. For this, the *path extent* of a rooted document path p in a document d, denoted $extent_d(p)$, is defined as the set of all nodes in d having p as path. The following relationship is established among nodes that are members of such extents.

Definition 2.1 (Descendant Extents) Let d be an XML document and p_1 and p_2 be rooted document paths. The *descendant extent* with respect to p_1 and p_2, denoted $dext_d(p_1, p_2)$, is the subset $\mathbf{V}' \subseteq \mathbf{V}_d$ such that every node $v \in \mathbf{V}'$ (1) is a member of $extent_d(p_1)$ and (2) has a descendant node $v' \in \mathbf{V}_d$ with v' being a member of $extent_d(p_2)$. If $p_1 = p_2$, then $dext_d(p_1, p_2) := extend_d(p_1)$. □

This definition captures the aspect that with nodes specified by a path p_1, a set of descendant nodes specified by a path p_2 having p_1 as prefix co-occurs in document d (see Figure 2 for an illustration).

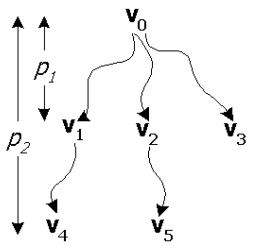

Figure 2: Let $v_1, v_2, v_3 \in extent_d(p_1)$ and $v_4, v_5 \in extent_d(p_2)$ for a document d. Then, $dext_d(p_1, p_2) = \{v_1, v_2\}$. v_3 is not included since v_3 does not have a descendant node specified by path p_2.

In this paper, we examine two important types of structural constraints for XML, *path implication* and *path absence*. Conceptually, a constraint specification consists of two components, a *context*, which specifies the (element) type of document fragment in which a relationship between two paths must hold, and a *path relationship* between two paths emanating from that context. Since both parts comprise paths, such constraints specify path patterns that must (not) occur in a document. Order among sibling elements is not considered by XSCs. The syntax and semantics of path implication constraints is formally defined as follows. For this, let p_1 and p_2 be two paths. The *longest common prefix* of p_1 and p_2 is denoted $lcp(p_1, p_2)$.

Definition 2.2 (Path Implication) Let p_1 and p_2 be two paths such that $lcp(p_1, p_2) = p \neq \epsilon$. A *path implication constraint* has the pattern $p_1 \rightarrow p_2$ and is said to be *satisfied* by a document d, denoted $d \models p_1 \rightarrow p_2$, if and only if $dext_d(p, p_1) \subseteq dext_d(p, p_2)$. The longest common prefix p among p_1 and p_2 is referred to as the *context* of the constraint. □

In other words, a path implication constraint requires that in all document fragments rooted at a node whose path matches p $(= lcp(p_1, p_2))$, the existence of path p_1 in a fragment requires the existence of path p_2 in that fragment. Obviously, if the first element in the path p does not correspond to the document's root label, the document can never satisfy the constraint. As an abbreviation, we use the notation $p_1 \leftrightarrow p_2$ to denote a *co-occurrence constraint* stating that both paths p_1 and p_2 have to be present in a document. A document d satisfies $p_1 \leftrightarrow p_2$ if and only if it satisfies $p_1 \rightarrow p_2$ and $p_2 \rightarrow p_1$.

Definition 2.3 (Path Absence) Let p_1 and p_2 be two paths such that $lcp(p_1, p_2) = p \neq \epsilon$. A *path absence constraint* has the pattern $p_1 \perp p_2$ and is said to be *satisfied* by a document d, denoted $d \models p_1 \perp p_2$, if and only if for all nodes $e \in dext_d(p, p_1)$, $e \notin dext_d(p, p_2)$. □

In comparison to a path implication constraint, a path absence constraint requires that in a context p, if the path p_1 occurs in a document fragment matching p, then p_2 must not occur in this fragment.

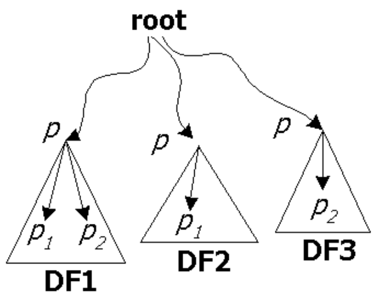

Figure 3: Consider the 3 types of document fragments (DF1-3) that are reachable through the path p. Then $p_1 \leftrightarrow p_2$ is satisfied for documents containing only DF1 under the context p, $p_1 \perp p_2$ for documents without DF1, and $p_1 \rightarrow p_2$ for documents without DF2.

Example 2.4 The following path implication and absence constraints are all satisfied by the document shown in Figure 1. The first three constraints are the specifications of the constraints stated informally in the motivating example. For the sake of readability and to emphasize the context of a path relationship, a constraint $p_1 \circ p_2$, $\circ \in \{\rightarrow, \leftrightarrow, \perp\}$, is written as p: $p'_1 \circ p'_2$, where $p = lcp(p_1, p_2)$ and p'_1, p'_2 are the paths suffixes in p_1, p_2, respectively, trailing p.

C_1 :`auctions.auction` : `seller.type.store` $\rightarrow$ `price.tax`
   ($\equiv$ `auctions.auction.seller.type.store` $\rightarrow$ `auctions.auction.price.tax`)
C_2 : `auctions.auction` : `seller.type.personal` $\perp$ `price.tax`
C_3 : `auctions.auction` : `payment.paypal` $\rightarrow$ `buyer.contact.email`

The following are path co-occurrence constraints:

C_4: `auctions.auction.seller.contact`: $\epsilon \leftrightarrow$ `phone`
C_5: `auctions.auction.seller.contact`: $\epsilon \leftrightarrow$ `email`

The relationships between paths specified by path implication and absence constraints exhibit some important properties. These properties will be utilized in the following section in order to reason about consistency and implication of XSCs. In particular, we have the following properties.

Proposition 2.5 Path implication is a binary reflexive and transitive relation.

Proof: It is clear that path implication is reflexive (i.e., $p \rightarrow p$ for any p). We now show that path implication is also transitive. Assume a document d that satisfies the two constraints $p_1 \rightarrow p_2$ and $p_2 \rightarrow p_3$. By definition, $p_1 \rightarrow p_2$ implies that $dext_d(l, p_1) \subseteq dext_d(l, p_2)$ and $dext_d(l', p_2) \subseteq dext_d(l', p_3)$, where $l = lcp(p_1, p_2)$ and $l' = lcp(p_2, p_3)$. Since both l and l' are prefixes of p, $lcp(l, l')$ equals either l or l'. Case 1) Assume $lcp(l, l') = l'$. Then $l' = lcp(p_1, p_3)$. Let $e_1 \in dext_d(l', p_1)$. Then there exists a descendant node of e_1, namely e_2, such that $e_2 \in dext_d(l, p_1)$. It follows that $e_2 \in dext_d(l, p_2)$ which then implies $e_1 \in dext_d(l', p_2)$. Thus, $e_1 \in dext_d(l', p_3)$, and we can conclude that $dext_d(l', p_1) \subseteq dext_d(l', p_3)$ and $p_1 \rightarrow p_3$. Case 2) with $lcp(l, l') = l$ can be shown in an analogous fashion. ■

It is obvious that path absence is a symmetric relation since both $p_1 \perp p_2$ and $p_2 \perp p_1$ deliver the same message, i.e., p_1 and p_2 must never co-occur in a document. For a document d and paths p_1 and p_2 with $p = lcp(p_1, p_2)$, $dext_d(p, p_1) \cap dext_d(p, p_2) = \emptyset$ would be an alternative definition for path absence. In either case, we say that p_1 and p_2 are *contradictory* if $p_1 \perp p_2$ holds. Although path absence does not specify a transitive relation, the following interaction between path implication and absence holds.

Proposition 2.6 If $p \rightarrow p'$ and $p' \perp q$, then $p \perp q$ if and only if $lcp(p', q)$ is a prefix of $lcp(p, p')$.

Proof: Suppose $p \rightarrow p'$ and $p' \perp q$, and let $l_1 = lcp(p', q)$ and $l_2 = lcp(p, p')$. ($\Leftarrow$) Assume l_1 is the prefix of l_2. By definition, $dext_d(l_2, p) \subseteq dext_d(l_2, p')$ and $dext_d(l_1, p') \cap dext_d(l_1, q) = \emptyset$. Since l_1 is the prefix of l_2, $lcp(p, q) = l_1$. Suppose $e \in dext_d(l_1, p)$. Then there exists a node e' which is a descendant of node e such that $e' \in dext_d(l_2, p)$. It follows that $e' \in dext(l_2, p')$. Thus, $e \in dext_d(l_1, p')$ and $e \notin dext_d(l_1, q)$. ($\Rightarrow$) Assume $l_1 = lcp(p', q)$ is not the prefix of $l_2 = lcp(p, p')$. Since both l_1 and l_2 are prefixes of p', it follows that l_2 is the prefix of l_1. We now show that $p \perp q$ is false by constructing a document d such that $p \perp q$ does not hold. First, d has a node e with path l_2. The node e has three descendants e_p, e'_p, and e_q, whose paths are p, p', and q, respectively. The lowest common ancestor of e'_p and e_q is e. Hence, d is an instance where $p \rightarrow p'$ and $p' \perp q$ hold but not $p \perp q$. ■

In summary, the relation properties shown in Figure 4 hold for path implication, path absence, and path co-occurrence constraints.

3 Consistency and Implication

In this section we study the consistency and implication problems for XSCs as a stand-alone schema formalism. In Section 3.1, we formally introduce the notions of consistency and implication.

	reflexive	symmetric	transitive
Path implication	$\surd$		$\surd$
Path co-occurrence	$\surd$	$\surd$	$\surd$
Path absence		$\surd$	

Figure 4: Summary of properties of XML Structural Constraints

Section 3.2 details the concept of constraint graphs and how such graphs are used to determine inconsistencies and implications among XSCs. The latter aspects are covered in Sections 3.3 and 3.4. A set of sound and complete inference rules for logical implication of XSCs is presented in Section 3.5.

3.1 Formal Definitions

Given a set Σ of path implication, absence, and co-occurrence constraints, it is natural to ask whether there exists any document that satisfies Σ, and whether other XSCs are implied. These two problems are generally known as the consistency and (logical) implication problems and have been covered in the context of relational databases in great detail [2].

Definition 3.1 (Logical Implication) Let Σ and Φ be two sets of XSCs. Σ *(logically) implies* Φ, denoted $\Sigma \models \Phi$, if for every XML document d, $d \models \Sigma$ implies $d \models \Phi$. $\square$

Defining consistency for XSCs is a little bit more tricky. Typically, a set of constraints Σ is said to be consistent if there exists an XML document that satisfies Σ. However, if XSCs are used as stand-alone schema formalism, this notion of consistency turns out to be insufficient. The reason for this is that for any set of XSCs Σ, one can easily come up with a document containing only a root node labeled with an element name that does not occur in any constraint in Σ. Such a document trivially satisfies Σ. For a more meaningful definition of consistency, it is thus natural to impose additional conditions on what elements and/or paths must occur in a document satisfying the constraints, if such a document exists.

Definition 3.2 (P-Consistency) Given a set of paths P and a set of XSCs Σ. Σ is said to be *P-consistent* iff there exists a document d such that $d \models \Sigma$, and for each $p \in P$, d contains p as path. If no such d exists, Σ is said to be *P-inconsistent*. $\square$

Since P can be an arbitrary set of paths, what are proper sets of paths in order to determine P-consistency? In the presence of a DTD (or other type of schema), the set of admissible rooted document paths implied by the DTD should be an obvious candidate. Without a DTD, one can at least assume P to be the set of paths that occur in Σ. In the following we focus on the natural case of P-consistency where P is the set of all paths mentioned in Σ, denoted P_Σ. That is, $p \in P_\Sigma$ if and only if there exists a constraint $\sigma \in \Sigma$ where $\sigma = p \circ q$ or $\sigma = q \circ p$, and $\circ \in \{\leftrightarrow, \rightarrow, \perp\}$.

Example 3.3 Consider the set $\Sigma = \{C_1, \ldots, C_5\}$ from Example 2.4. Obviously, $\Sigma \models$ `auctions.auction.seller.contact:  email` $\leftrightarrow$ `phone` by transitivity. It is also trivial that if a path p' is a prefix of a path $p \in P_\Sigma$, then $\Sigma \models p \rightarrow p'$. With $P = P_\Sigma$, Σ is P-consistent since, e.g., the document shown in Figure 1 satisfies Σ. However, Σ is no longer P-consistent if one adds `auctions.auction.seller.contact: email` $\perp$ `phone` to Σ. ◇

3.2 Constraint Graph

In the following, we introduce the notion of a constraint graph as essential construct to compute implied XSCs and to decide consistency. Given a set of XSCs Σ, the constraint graph, $G = (V, E_i, E_c)$, for Σ is a directed graph with V being a set of vertices, and E_i and E_c being two sets of edges. Edges in E_i and E_c are called *i-edges* (for implication) and *c-edges* (for contradiction), respectively. The algorithm to construct G is as follows.

Algorithm 3.4

```
CONSTRAINTGRAPH($\Sigma$)
for each $p \circ q \in \Sigma$, where $\circ \in \{\rightarrow, \leftrightarrow, \perp\}$
    if $p \notin V$ then { $V := V \cup \{p\}$; InsertPrefix($p$) }
    if $q \notin V$ then { $V := V \cup \{q\}$; InsertPrefix($q$) }
    if $p \rightarrow q$ then $E_i = E_i \cup \{(p, q)\}$
    else if $p \leftrightarrow q$ then $E_i = E_i \cup \{(p, q), (q, p)\}$
    else if $p \perp q$ then $E_c = E_c \cup \{(p, q), (q, p)\}$

INSERTPREFIX($q$)
/* a path $q$ implies all its prefixes */
let $r := q$ and $s := q$
remove the last element from path $s$
while $s \neq \epsilon$ and $s \notin V$ do
    $V := V \cup \{s\}$
    $E_i := E_i \cup \{(r, s)\}$
    $r := s$ and remove the last element from path $s$
if $s \neq \epsilon$ then $E_i := E_i \cup \{(r, s)\}$
```

Intuitively, a constraint graph can be considered as an abstract representation of the set Σ and some other constraints implied by Σ. Paths in a possible document satisfying Σ are abstracted as nodes (carrying the path as label), and edges between nodes represent the relationships among these paths. In particular, i-edges (c-edges) are inserted if two paths are related by implication or co-occurrence (absence). Other than the paths in P_Σ, all the prefixes of these paths are also included in G through the procedure INSERTPREFIX. In this procedure, all pairs of document paths representing a child-parent condition are connected with i-edges. Figure 5 shows an example of a constraint graph. Let n be the number of constraints in Σ, and d be the maximum length of paths in P_Σ. Then, the number of vertices in G is at most $n * d$. Furthermore, the number of edges inserted directly from Σ is at most $2n$, whereas the number of edges inserted between prefixes is no more than $n * d - 1$. The size of G is thus bounded by $O(n * d)$. Assuming that a suffix tree is used to keep track of the existence of label paths in G, one can compute a constraint graph in time $O(n * d + K)$, where K is the total length of all paths in P_Σ.

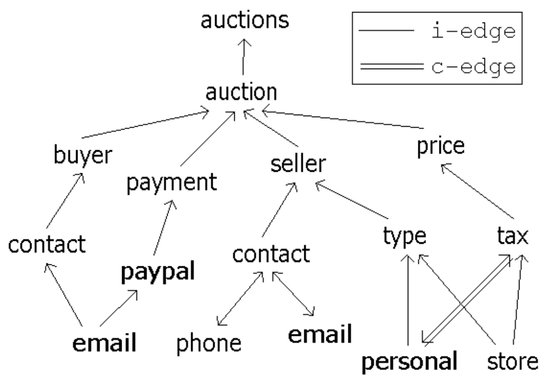

Figure 5: Constraint graph for $\Sigma = \{C_1, \cdots, C_5\}$. For readability, only the last element of each label path is shown.

Given a constraint graph G, in order to determine constraints implied by Σ and to decide consistency of Σ, we utilize the notion of *implication path*.

Definition 3.5 (Implication Path) Let G be a constraint graph, and p and q two nodes in V. An *implication path* from p to q, denoted i-path(p, q), is a sequence of i-edges $\langle (p, p_1), (p_1, p_2), \cdots, (p_k, q) \rangle$ in G from p to q. For $p = q$, i-path(p, q) is defined to be the empty sequence. i-path(p, q) is undefined if no such path exists. □

3.3 Consistency Problem

Given a set Σ of XSCs as a stand-alone schema formalism, it is essential to determine whether there exists a document that satisfies Σ. The answer to this question can be decided using a constraint graph.

Definition 3.6 (Inconsistency of constraint graph) A constraint graph G is said to be *inconsistent* iff at least one of the following conditions is satisfied:

(1) There exist nodes $p_i, p_j, p_k \in V$ such that i-path(p_i, p_j) and i-path(p_i, p_k) exist, and $(p_j, p_k) \in E_c$.

(2) More than one node in G has a path label of length 1.

(3) There exists $(p, q) \in E_c$ such that $length(lcp(p, q)) = 1$. □

P-consistency of a set of XSCs Σ is determined by three factors. First, if a path p implies two paths q_1 and q_2, then q_1 and q_2 must not be contradictory. Furthermore, all paths mentioned in Σ must have a common first element, since it is impossible to incorporate two paths in the same document if they have different roots. Lastly, the longest common prefix (lcp) of two contradictory paths must not be the root path. For instance, no document with both $a.b$ and $a.c$ can be constructed if $a.b \perp a.c \in \Sigma$. Notice that all three criteria are guaranteed if the constraint graph for Σ is consistent. We thus have the following.

Theorem 3.7 A set of XSCs Σ is P-inconsistent iff G is inconsistent.

Proof: Assume G is inconsistent and there exists a document d that satisfies Σ. Since G is inconsistent, at least one of the three conditions (1)-(3) is satisfied. Since it is trivial that (2) or (3) cause inconsistency, we only provide proofs for the inconsistency caused by (1). Assume there exist paths p_i, p_j, and p_k such that i-path(p_i, p_j) and i-path(p_i, p_k) exist, and $(p_j, p_k) \in E_c$. From the transitivity and reflexivity property of path implication, it is easy to see that $p_i \to p_j$, $p_i \to p_k$, and $p_j \perp p_k$. It follows that, for any document d that satisfies Σ, $dext_d(l_1, p_i) \subseteq dext_d(l_1, p_j)$, $dext_d(l_2, p_i) \subseteq dext_d(l_2, p_k)$ and $dext_d(l_3, p_j) \cap dext_d(l_3, p_k) = \emptyset$, where $l_1 = lcp(p_i, p_j)$, $l_2 = lcp(p_i, p_k)$, and l_3 equals l_1 or l_2. W.l.o.g., let $l_3 = l_1$. Then, l_1 is a prefix of l_2. Note that $dext_d(l_1, p_j) \cap dext_d(l_1, p_k) = \emptyset$ implies $dext_d(l_1, p_i) \cap dext_d(l_1, p_k) = \emptyset$. Let $x \in dext_d(l_2, p_i)$. Then $x \in dext_d(l_2, p_k)$, and there exists an ancestor of x, x_a, such that $x_a \in dext_d(l_1, p_i)$ and $x_a \in dext_d(l_1, p_k)$, which gives a contradiction. Therefore, we can conclude that Σ is P-inconsistent if (1) is satisfied.

We now prove the other direction by showing that if G is consistent, then Σ is P-consistent. Assume that G is consistent. We will show the existence of a document that satisfies Σ. For this,we transform G into a tree structure, $H = (V_H, E_H)$, that can be viewed as an XML document d_H such that $d_H \models \Sigma$. Given a document path p, we use the notation $parent(p)$ to denote a path q such that $lcp(p, q) = q$ and $length(q) = length(p) - 1$. The transformation algorithm is as follows.

Algorithm 3.8 (Document construction)
Input: constraint graph G
Output: tree H that represents d_H s.t. $d_H \models \Sigma$

```
/* determine prefixes of subgraphs to be copied */
let $W := \{lcp(p,q) | (p,q) \in E_c\}$
for each $r \in W$
    if $\exists r' \in W$ such that $lcp(r, r') = r'$ and $r' \neq r$
    then $W := W - \{r\}$
/* transformation begins */
for each $r \in W$
    let $t_0 = tree(r)$ be the subgraph of $G$ containing
    all and only nodes with label prefixed by $r$
    let $p_0, \cdots, p_k$ be nodes in $tree(r)$ s.t. $(p_i, p_j) \in E_c$
    /* create copies of the subgraph */
    create $k$ copies of $tree(r)$, called $t_1, ..., t_k$
    /* remove c-edges from each copy */
    for $i = 0$ to $k$
        for each $p_j$ such that $(p_i, p_j) \in t_i$
            remove $p_j$ and $p_s$ if $\exists$ i-path($p_s$,$p_j$) in $t_i$(*)
            remove all edges involving the deleted nodes
        for each c-edge $(p_l, p_j)$ left in $t_i$, $l < j$
            remove $p_j$ and $p_s$ if $\exists$ i-path($p_s$,$p_j$) in $t_i$(**)
            remove all edges involving the deleted nodes
    /* connect the copies to the rest of the graph */
    for each edge between $p$ and $p'$, $p'$ not in $tree(r)$
        for each $t_i$ that contains $p$
```

```
            insert same type of edge between $p'$ and $p$ in $t_i$
/* reduce the graph into a tree structure */
remove all edges $(p, q)$ if $parent(p) \neq q$
```

Algorithm Details. The main idea of the algorithm is to eliminate all c-edges from the graph G by duplicating subgraphs of G. The subgraphs to be duplicated are disjoint (each with a different common prefix r) and determined at the beginning of the algorithm. From the failure of condition (3), $length(r)$ must be greater than 1 for all $r \in W$, which guarantees that the root node is never duplicated. Each of these subgraphs, called $tree(r)$, contains one or more c-edges, which are resolved by creating enough number of copies and removing edges from these copies. In particular, for $k+1$ nodes $(p_0, \cdots, p_k)$ in $tree(r) = t_0$ encountering c-edges, k extra copies $\{t_1, \cdots, t_k\}$ of $tree(r)$ are created. For each t_i, at least one node p_i must be kept in the copy, and a minimal set of nodes is removed from t_i to ensure that no c-edge remains while preserving path implications inside t_i. Such operation is possible because of the dissatisfaction of condition (1). Edges between $tree(r)$ and the rest of the graph are then inserted for the corresponding nodes in the copies. At the end of the algorithm, a graph is obtained that is free of c-edges. Finally, the graph is reduced to a tree structure by removing all edges except those between nodes with document paths representing a parent-child relationship. The document tree represented by H for the constraint graph in Figure 5 is shown in Figure 6.

There are a few claims we need to justify to show that d_H is indeed an XML document that can show P-consistency of Σ: 1) H is a tree structure; 2) H contains all the paths in P_Σ; and, 3) all constraints in Σ are satisfied by d_H.

1) We will prove that H is a tree structure by contradiction. Assume H is not a tree. Then, it either contains cycles or is disconnected. Suppose there is a cycle $p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_k \rightarrow p_1$ in H. Then, $p_1 = parent(p_2)$, $\cdots$, $p_k = parent(p_1)$. It follows that p_1 is the ancestor of itself. Contradiction. Thus, H must be disconnected. Since G is consistent, there exists exactly one node, r, with length 1 in H. If H is not connected, then there is a node q where no implication path from q to r exists. Then, from the construction algorithm, since the root node is never duplicated, it must be true that q has a root label, r', different from r, and $r' \in V$. Then, G is inconsistent by condition (2). Contradiction.

Now, with H being a tree structure, if we drop the all the labels except for the last for all paths in V_H, H can easily be viewed as the tree model of an XML document. The document represented by H is denoted d_H.

2) Suppose $p \in P_\Sigma$ and $p \notin V_H$. Then it must be removed in either step (*) or (**) during the transformation. p does not appear in any copy $t_0, \cdots, t_k$ because there exist some nodes p_j and p_l in the copies such that $(p_j, p_l) \in E_c$, and i-path(p, p_j) and i-path(p, p_l) exist. It follows that G satisfies condition (1) and thus is inconsistent. Contradiction.

3) Suppose d_H does not satisfy $p \perp q \in \Sigma$. Then there exists a node n in d_H with path $r = lcp(p, q)$ and $n \in dext_{d_H}(r, p) \cap dext_{d_H}(r, q)$. Since it must be true that p and q are connected by a c-edge between the respective nodes in G, in steps (*,**) of the algorithm, either p or q must be removed in all copies. Thus, the node n must not exist. Contradiction. Suppose d_H does not satisfy $p \rightarrow q \in \Sigma$. Then, there exists a node n in d_H with path p, which has an ancestor n' with path

$r = lcp(p,q)$ and $n' \notin dext_{d_H}(r,q)$ ($n = n'$ if $r = path(n)$). Clearly, if no nodes are removed during the transformation, then all path implications must be satisfied. We know that in steps (*,**), when a node (q) is to be removed, every node that has an implication path to it (including p) will also be removed. Since $(p,q) \in E_i$, it is impossible for the absence of node with path q under n. Therefore, we can conclude that d_H satisfies Σ. Hence, Σ is P-consistent if G is consistent. ■

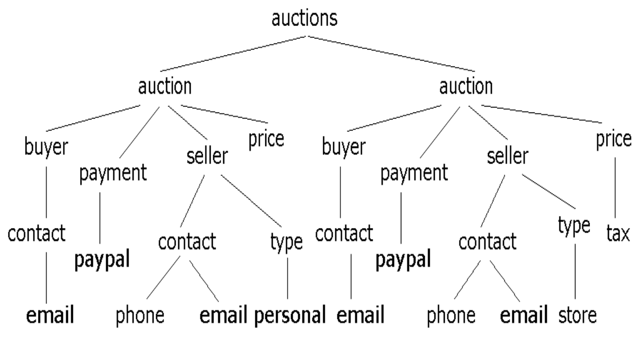

Figure 6: The document tree represent by H for the constraint graph in Figure 5

It should be noted that the construction of an XML document satisfying all constraints in Σ not only serves the purpose of providing a means to prove Theorem 3.7, but it can also be used as a tool in designing a schema using XSCs. While the user specifies path implication and absence constraints, besides checking for implication and consistency, a tool can provide the user with document templates that satisfy the specified constraints.

The algorithm for deciding P-consistency can be summarized as follows.

Algorithm 3.9 (Deciding P-consistency)

Input: Constraint Graph G
Output: Yes if and only if G is consistent

```
if there exist $p_1, p_2 \in V$ such that the path labels associated with $p_1$ and $p_2$
        have length one and $p_1 \neq p_2$
then return no ($\equiv$ condition (2))
if $(p,q) \in E_c$ and $length(lcp(p,q)) = 1$
then return no ($\equiv$ condition (3))
for each $p \in V$
        mark all $q$ for which an implication path i-path$(p,q)$ exists
        if there exists $(q_i, q_j) \in E_c$ and $q_i$ and $q_j$ are marked
        then return no ($\equiv$ condition (1))
        un-mark all nodes in $V$
return yes
```

Let $|V|$,$|E_i|$, and $|E_c|$ be the number of vertices, i- and c-edges in the constraint graph, respectively. Since conditions (2) and (3) can be checked in time $O(|V| + |E_c|)$ and condition (1) in time $O(|V| *$

$(|E_i| + |E_c|))$, the complexity of the P-consistency problem is $O((n * K)^2)$, where n is the number of XSCs in Σ and K is the maximum path length.

Theorem 3.10 The P-consistency problem for XSCs is decidable in quadratic time. ■

3.4 Implication Problem

We now show how a constraint graph constructed from a set Σ of XSCs is used for checking whether a constraint is implied by Σ. The following theorems specify the properties of the graph in order to satisfy certain (logical) implications.

Theorem 3.11 Let Σ be a set of P-consistent XSCs. Then, $\Sigma \models p \to q$ if and only if there exists i-path(p, q) in G.

Proof: 1) $(\Leftarrow)$ Assume an i-path$(p, q) = \langle (p_1, p_2), (p_2, p_3), \cdots, (p_{k-1}, p_k) \rangle$ in G where $p = p_1$ and $q = p_k$. For $(p_i, p_{i+1}) \in$ i-path(p, q), $1 \leq i < k$, either $\mathit{parent}(p_i) = p_{i+1}$ or $p_i \to p_{i+1} \in \Sigma$. $p_1 \to p_k$ is then followed from the transitivity of path implication (Prop. 2.5). $(\Rightarrow)$ Assume G does not contain i-path(p, q). It is then sufficient to show that there exists a document containing p but not q, while satisfying Σ. Such a document can be obtained by a simple modification to the tree H constructed in Algo. 3.8. From H, we remove 1) all nodes with path q, and 2) all nodes q' where i-path(q', q) exists. Note that any node labeled p is not among the nodes to be removed, given the assumption that i-path(p, q) does not exist. ■

Logical implication of path co-occurrence follows directly from Theorem 3.11.

Corollary 3.12 Given a set of P-consistent XSCs Σ. $\Sigma \models p \leftrightarrow q$ if and only if there exist i-path(p, q) and i-path(q, p) in G. ■

Theorem 3.13 Given a set of P-consistent XSCs Σ, $\Sigma \models p \perp q$ if and only if there exist p' and q' such that 1) i-path(p, p') and i-path(q, q') exist, 2) $(p', q') \in E_c$, and 3) $\mathit{lcp}(p', q')$ is the prefix of $\mathit{lcp}(p, q)$.

Proof Sketch: Suppose that there exist p' and q' and the conditions (1)-(3) hold. Then we have $\Sigma \models \{p \to p', q \to q', p' \perp q'\}$. Note that $\mathit{lcp}(p, p')$ and $\mathit{lcp}(q, q')$ are also prefixes of $\mathit{lcp}(p', q')$. Then, it follows from Proposition 2.6 and condition 3) that $p \perp q'$, and hence $p \perp q$. Assume there doesn't exist p' and q' for the conditions to hold. We now show that $\Sigma \not\models p \perp q$ by giving an instance that satisfies all constraints in Σ but not $p \perp q$. Consider a document d_H constructed in Algorithm 3.8. If d_H does not satisfy $p \perp q$, then we are done; otherwise, we suggest a modification of the algorithm which results in a dissatisfaction of $p \perp q$ in d_H. The modification can be described informally as follows: Let $r = \mathit{lcp}(p, q)$. Then for all $\mathit{tree}(r)$ (subtrees rooted r) in H, $\mathit{tree}(r)$ does not contain both p and q. For this to happen, there must be nodes p' and q' in G such that i-path(p, p') and i-path(q, q') exist, and both p' and q' are ends of some c-edges. Note that $r' = \mathit{lcp}(p', q')$ must be a prefix of r (which corresponds to condition 2) because r would not have been duplicated and neither p nor q would have been removed from $\mathit{tree}(r)$ otherwise. According to the algorithm, in at least one of the copies t_i, the node p is kept while other c-edges are being resolved in the copy. If q' and q are removed from t_i in step (*), it means that $(p', q') \in E_c$, which contradicts our initial assumption. Thus, q' and q must be removed from t_i in step (**). In step (**), for an occurrence of a c-edge (p_l, p_j) in t_i, $l < j$, p_j is chosen to be the one to be removed. It is obvious that one can as well resolve all contradictions left in the copy by removing p_l instead of p_j. Hence, a statement can be inserted to check whether $p_j = q'$. If this is so, instead of removing p_j and its dependents, we remove p_l from t_i as well as all p_s if i-path(p_s, p_l) exists. With such modification, the resulting tree H contains at least a subtree t_i which contains a node labeled r with descendants p and q. Hence, the modified d_H satisfies Σ but not $p \perp q$. ■

The above theorems and corollary state a straightforward algorithm to check whether a given constraint is a logical implication of Σ. Assume that G has been constructed. The algorithm for the implication problem is as follows.

Algorithm 3.14 (Checking implied constraints)
Input: constraint graph G, XSC τ
Output: Yes if and only if $\Sigma \models \tau$

```
/* check for path implication */
if $\tau = p \rightarrow q$
then if there exists i-path$(p, q)$
          then return yes
/* check for path co-occurrence */
else if $\tau = p \leftrightarrow q$
then if there exist i-path$(p, q)$ and i-path$(q, p)$
          then return yes
/* check for path absence */
else if $\tau = p \perp q$
then mark $p$ and $q$ in $G$
     mark node $r$ if i-path$(p, r)$ or i-path$(q, r)$ exist
     for each pair of marked nodes $r_i$ and $r_j$, $i \neq j$
          if there exists $(r_i, r_j) \in E_c$ and $lcp(r_i, r_j)$ is the prefix of $lcp(p, q)$
          then return yes
return no
```

Note that the algorithm for checking logical implication terminates in time $O(|E_i|+|E_c|) = O(n*K)$ where again K is the maximum length of a path.

Theorem 3.15 The implication problem for XSCs is decidable in linear time. ■

3.5 Inference Rules for XSCs

In addition to the development of algorithms to determine logical implication, axiomatization is another important aspect in the study of dependency theory. Typically, if a class of constraints Σ can be concisely characterized by a finite set of inference rules, Σ is said to be axiomatizable and the set of inference rules is the axiomatization of Σ. In this section, we will show that XSCs are indeed axiomatizable. The inference rules for XSCs are as follows.

R1: If $lcp(p_1, p_2) = p_1$, then $p_2 \rightarrow p_1$.

R2: If $p_1 \rightarrow p_2$ and $p_2 \rightarrow p_3$, then $p_1 \rightarrow p_3$.

R3: If $p_1 \perp p_2$, then $p_2 \perp p_1$.

R4: $p_1 \rightarrow p_2$ and $p_2 \rightarrow p_1$ if and only if $p_1 \leftrightarrow p_2$.

R5: If $p_1 \rightarrow p_2$, $p_1 \perp p_3$ and $lcp(p_2, p_3)$ is a prefix of $lcp(p_1, p_2)$, then $p_2 \perp p_3$.

As shown by the following theorem, this set of inference rules forms an axiomatization for XSCs.

Theorem 3.16 The set of inference rules {R1-R5} is sound and complete for logical implication of XSCs.

Proof: R1-R3 are direct properties of the respective types of constraints. R4 follows from the definition of path co-occurrence. R5 is proven in Prop. 2.6. Therefore, all five rules are sound. For completeness, we need to show that for any XSC τ where $\Sigma \models \tau$, τ can be proven using R1-R5. (Case 1) Let $\tau = p \rightarrow q$. If $lcp(p, q) = q$, $p \rightarrow q$ can be proven simply using R1. Otherwise, we construct the constraint graph G. By Theorem 3.11, there exists an implication path from p to q in the constraint graph. Let $\langle (n_0, n_1), \cdots, (n_{k-1}, n_k) \rangle$ be the sequence of k i-edges that leads from $p(n_0)$ to $q(n_k)$. Then the proof begins with $s_i = n_i \rightarrow n_{i+1}$, $i \in [1..k]$, which is justified by rule R1 if $lcp(n_i, n_{i+1}) = n_{i+1}$, or $s_i \in \Sigma$ otherwise. Finally, we complete the proof for $p \rightarrow q$ using the transitivity property of path implication by adding the following steps:

$s_{k+1} = n_0 \rightarrow n_2$ (by R2 using s_1 and s_2)

$s_{k+2} = n_0 \rightarrow n_3$ (by R2 using s_{k+1} and s_3)

$\vdots$

$s_{2k-1} = n_0 \rightarrow n_k$ (by R2 using s_{2k-2} and s_k)

(Case 2) Let $\tau = p \leftrightarrow q$. The proof of τ consists of the proofs for $p \rightarrow q$ and $q \rightarrow p$. It is then concluded by rule R4.

(Case 3) Let $\tau = p \perp q$. By Theorem 3.13, in the constraint graph G, there exist p' and q' such that i-path(p, p') and i-path(q, q') exist, $(p', q') \in E_c$, and $lcp(p', q')$ is the prefix of $lcp(p, q)$. Note that $lcp(p, p')$ and $lcp(q, q')$ are also prefixes of $lcp(p', q')$. First, we provide the proofs for $p \rightarrow p'(s_1)$ and $q \rightarrow q'(s_2)$ (details are given in case 1). It is then followed by the steps

$s_3 = p' \perp q'$ ($\in \Sigma$ and by R3 if necessary)

$s_4 = p \perp q'$ (by R5 using s_1 and s_3)

$s_5 = q' \perp p$ (by R3 using s_4)

$s_6 = p \perp q$ (by R5 using s_2 and s_5).

Hence, R1-R5 are sound and complete for logical implication of XSCs. ■

4 XSCs and DTDs

Other than being an extension to a grammar-based schema formalism, here DTD, XSCs can also be derived from a DTD. In Section 4.1, we detail an algorithm to compute XSCs from a DTD. In

Section 4.2, we discuss the construction of a specialized DTD from a given DTD and set of XSCs.

4.1 Deriving XSCs from a DTD

Document Type Definition (DTD) is one of the most common schema formalisms for XML. A DTD basically specifies admissible elements, element nesting, and element attributes in form of an extended context-free grammar [7]. For this, with each element a *content model* in form of a regular expression is associated. A DTD is said to be *recursive* if there are elements $e_0, e_1, \dots, e_n$ such that e_{i+1} occurs in the content model of $e_i, i < n$, and e_n occurs in the content model of e_0.

Given a DTD, XSCs implied by the DTDs are defined in the obvious sense.

Definition 4.1 (DTD-implied XSCs) An XSC τ is said to be *implied* by a DTD D iff for any document d that conforms to D, $d \models \tau$. $\square$

Consider the DTD given in Section 1. It is obvious that any document conformant to that DTD also satisfies, among others, the following three simple XSCs:

(1) `auctions.auction:` $\epsilon \rightarrow$ `item`,
(2) `auctions.auction: seller` $\leftrightarrow$ `buyer`, and
(3) `auctions.auction.payment: creditCard` $\perp$ `moneyorder`.

We now describe a three step approach to compute XSCs from a given DTD. First, individual content models for elements in the DTD are investigated. Then, a so-called *XTrie* is computed that captures admissible paths according to the DTD. Finally, the XTrie is transformed into a constraint graph that sufficiently describes the set of all XSCs the DTD implies.

1. DTD Preprocessing. In this step, the content model for each element in a given DTD D is investigated. Information about what elements are required/optional for a given element and how the child elements are related is recorded in an *Element Relationship Table (ERT)*.

Assume a content model m_e for element $e \in \mathbf{E}$. Based on the specification of e's content model in D, child elements can be optional, required, and/or can occur in a choice group. An element e' is said to be a *required element* in m_e iff for every document that conforms to D, there exists at least one child element e' for every occurrence of e. Otherwise, e' is said to be an *optional element.* The other type of information in an ERT describes the relationships among the elements in e's content model. Assume a content model for e where e' and e'' are siblings. e' implies e'', denoted $e' \rightarrow e''$ iff for every document that conforms to D and for every occurrence of e' under e, there exists a sibling element e'' for e'. e' contradicts e'' iff for every document and for every occurrence of e' under e, there exists no sibling e'' for e'. e' and e'' co-occur, denoted $e' \leftrightarrow e''$, if they imply each other. The following example lists some entries of the ERT for the DTD shown in the motivating example in Section 1. (r) and (o) stand for required and optional, respectively.

Element	Children	Constraints
seller	*contact(r),type(r)*	*contact* $\leftrightarrow$ *type*
contact	*name(r), email(o), phone(o)*	*email* $\rightarrow$ *name*, *phone* $\rightarrow$ *name,*
type	*personal(o),store(o)*	*personal* $\perp$ *store*

Obviously, if a DTD has n elements, then the ERT has n entries. For a content model $m_i, i = 1, \ldots, n$ with k_{m_i} elements specified in m_i, at most $k^2_{m_i}$ comparisons are necessary to determine the relationships among all k_{m_i} elements. Let $k = \max\{k_{m_i} \mid i = 1, \ldots, n\}$. Then the time complexity for constructing an ERT is $O(n * k^2)$.

2. XTrie Construction. Similar to a (strong) Dataguide [15], an *XTrie*, $XT(V_x, E_x)$, is a tree structure that summarizes path information of XML documents. Nodes (V_x) are the admissible element names and edges (E_x) are created for pairs of child-parent elements. The major difference between an XTrie and a Dataguide is that the latter is constructed from source data, whereas an XTrie is derived from a DTD. Note that if a DTD is recursive, there exist infinitely many admissible paths, which then implies an infinitely large XTrie. Thus, loops must be introduced to an XTries to make the construction possible. In the following, we only consider non-recursive DTDs.

The XTrie constructed from the auction DTD is shown in Fig. 7.

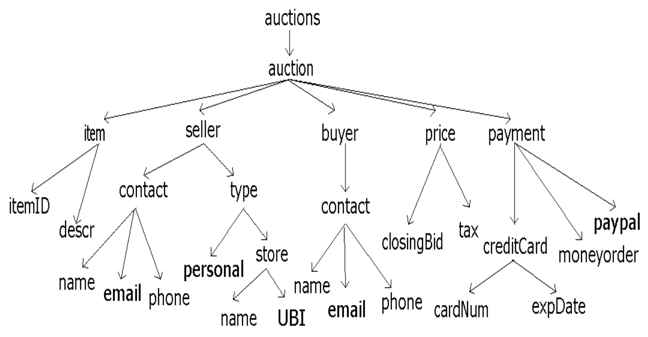

Figure 7: XTrie for the auction DTD in Section 1

3. Constraint Graph for a DTD. In the following, we detail the construction of a constraint graph $G = (V, E_i, E_c)$ from a DTD D. G is obtained by transforming the XTrie computed from D using information from the ERT. Then, all XSCs implied by the DTD are properly encoded in the constraint graph. That is, one can efficiently enumerate all XSCs implied by the DTD from the graph, or alternatively, check whether a given XSC is implied by the DTD using the theorems regarding logical implication given in Section 3. The construction of the constraint graph from an XTrie and ERT is as follows.

Assume the XTrie XT and ERT for a given DTD have been constructed. We first replace the label l of every node $n \in V_x$ by its corresponding "full path", i.e., the sequence of labels starting from the root leading to n. V_x is then the set of vertices for G. Next, we reverse the direction of the edges in E_x, which then form the initial set of i-edges in G. In order words, $(n_1, n_2) \in E_i$ iff $(n_2, n_1) \in E_x$. The reversion is necessary because we know that by inference rule R2 (see Section 3.5), any path implies its parent path, not child path. Let us denote the set of paths with e being the end label $P(e)$. Then, for each entry in the ERT describing the property of an element e, if e' is a required child for e (marked "r"), $(p_e, p_e.e')$ is inserted into E_i for all $p_e \in P(e)$. Finally, for each element constraint σ in the ERT entry for e, if σ is an implication constraint, $e_1 \rightarrow e_2$, we add an i-edge $(p_e.e_1, p_e.e_2)$ to E_i; and, if σ specifies a contradiction between p_{e_1} and p_{e_2}, two c-edges $(p_e.e_1, p_e.e_2)$ and $(p_e.e_2, p_e.e_1)$ are added to E_c, for all $p_e \in P(e)$. An excerpt of the constraint graph for the auction DTD is shown in Figure 8.

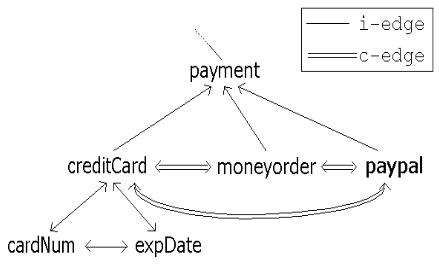

Figure 8: Excerpt of the constraint graph for the auction DTD. For readability, only the last element of label paths are shown.

Definition 4.2 (Soundness and Completeness) A set of XSCs Γ derived from a DTD D is *sound* iff for every document d that conforms to D, $d \models \Gamma$, and *complete* iff there does not exist an XSC τ such that $\Gamma \not\models \tau$ and $\Gamma \cup \tau$ is sound. □

Theorem 4.3 Given a DTD D. The set of XSCs represented by the constraint graph for D is sound and complete.

<u>Proof:</u> Assume the constraint graph G derived from D is properly constructed and let Γ be the set of XSCs implied by G. (Soundness) Note that there are three types of edges in G: 1) child-parent edges connecting nodes having a child-parent relationship, 2) required child edges for required child elements, and 3) sibling edges for relationships derived from a content model between sibling nodes. Child-parent edges are trivially sound from inference rule R1. Assume that required child and sibling relationships within a content model are correctly computed. For element type e with a required child e', having paths p and p' respectively, $p \rightarrow p'$ is obviously implied. For sibling element types e_1 and e_2 whose paths are p_1 and p_2 respectively, it is also clear that $e_1 \rightarrow e_2$ ($e_1 \perp e_2$) implies $p_1 \rightarrow p_2$ ($p_1 \perp p_2$). Therefore, the XSCs represented by the edges of G are sound. Other XSCs inferred by G but not directly represented as edges are also sound from the soundness of the inference rules (Section 3.5).

(Completeness) Assume that there exist τ such that $\Gamma \not\models \tau$ and $\tau \cup \Gamma$ is sound. Suppose $\tau = p \rightarrow q$. Since $\Gamma \not\models \tau$, there exists no implication path from p to q by Theorem 3.11. Case 1) q is a prefix path of p (i.e., $lcp(p, q) = q$). From the construction of G, there is an i-edge from every path p (except for the root) to the respective parent path $parent(p)$, thus there must be an implication path from p to q. Case 2) p is a prefix path of q and $p \neq q$. If p is a direct ancestor of q, then for any conformant document d, all elements with a path p have a child with path q. Therefore, the element corresponding to q is marked as "required" in the ERT, and there must be an i-edge from p to q in

G. Now suppose q is not a direct descendant of p. Let $p = a_0.\cdots.a_j$ and $q = a_0.\cdots.a_j.y_1.\cdots.y_n$, $n > 1$. Recall that a DTD specifies the document structure by giving a legitimate list of children for every element type. In order to have $p \rightarrow q$, y_1 must be a required child of a_j, and y_i must be a required child of y_{i-1} for all $i > 1$. Therefore, there must be an implication path from p to q in G. Case 3) p and q are sibling paths (i.e., share the same parent path). Let $p = a_0.\cdots.a_j.x$ and $q = a_0.\cdots.a_j.y$. Then, it must be true that $x \rightarrow y$ is present in the entry a_j of the ERT. It follows that G contains an i-edge from p to q.

Case 4) Neither of the cases (1)-(3) hold. Let $p = a_0.\cdots.a_j.x_1.\cdots.x_m$ and $q = a_0.\cdots.a_j.y_1.\cdots.y_n$. The *lcp* of p and q is denoted $l = a_0.\cdots.a_j$. (i) Suppose y_i is a required child of y_{i-1} for all $i > 1$. If y_1 is a required child of a_j, or $x_1 \rightarrow y_1$ is in the entry a_j, then there must be an implication path from p to q in G. (ii) Suppose there exists $1 < k \leq n$ such that y_k is optional in the respective entry in the ERT. We will show that $p \rightarrow q$ cannot be true by giving a document instance that conforms to D but doesn't satisfy $p \rightarrow q$. Let d be a document that conforms to D such that d contains a path p. Let $v \in dext_d(l, p)$. If $v \notin dext_d(l, q)$, then d doesn't satisfy $p \rightarrow q$. Assume that $v \in dext_d(l, q)$. Since y_k is optional, there exists an element sequence w that conforms to the content model of y_{k-1} but contains no y_k. For each descendant u of v such that $u \in extent_d(l.y_1.\cdots.y_{k-1})$, we modify its (direct and indirect) descendants such that the sequence of children u contained is exactly w. The resulting document does not satisfy $p \rightarrow q$. (iii) Assume y_i is a required child of y_{i-1} for all $i > 1$. Also assume y_1 is an optional child of a_j and $x_1 \rightarrow y_1$ is absent from the entry a_j in the ERT. Similar to (ii), we can raise a contradiction by showing the existence of a document that conforms to D but doesn't satisfy $p \rightarrow q$. Consider the document d described above. Note that there exists an element sequence w that conforms to the content model of a_j such that w contains x_1 but no y_1. Thus, we modify the descendants of v such that the list of children element of v is w. Hence, we have $d \nvDash p \rightarrow q$

Suppose $\tau = p \perp q$. Since $\Gamma \nvDash \tau$, τ cannot be derived from G. It is clear that $p \perp q$ is impossible for case (1) and (2) where p and q are ancestors and descendants. Case 3) p and q are sibling elements. Let $p = a_0.\cdots.a_j.x$ and $q = a_0.\cdots.a_j.y$. For τ to hold, it must be true that $x \perp y$ is in the entry a_j of the ERT. Thus, there is a c-edge between p and q in G, and consequently, $\Gamma \models \tau$. Case 4) Neither of the cases (1)-(3) hold. Let $p = a_0.\cdots.a_j.x_1.\cdots.x_m$ and $q = a_0.\cdots.a_j.y_1.\cdots.y_n$. The *lcp* of p and q is denoted $l = a_0.\cdots.a_j$. If $x_1 \perp y_1$ holds in the entry a_j of the ERT, then there is a c-edge between $l.x_1$ and $l.y_1$. Note that there are implication paths from p to $l.x_1$ and from q to $l.y_1$. It follows that $\Gamma \models p \perp q$ by Theorem 3.13. Suppose that $x_1 \perp y_1$ is absent from the entry a_j of the ERT. We now show that τ is not sound by giving a document that conforms to D but does not satisfy τ. Let d be a document conforming to D containing a path p. Let $v \in dext_d(l, p)$. If $v \in dext_d(l, q)$, then d doesn't satisfy τ. Otherwise, we let w to be an element sequence that conforms to the content model of a_j such that w contains both x_1 and y_1. Then we modify the descendant nodes of v according to w. The modified document d now fails $p \perp q$.

■

It should be noted that any extra XSC can easily be incorporated into the constraint graph for a DTD. Therefore the mechanism of deriving XSCs from DTDs described in this section in addition to the theorems stated in Section 3.4 indeed can be used to test for logical implication of XSCs in the presence of DTD.

4.2 Computing a Specialized DTD from a DTD and XSCs

Specialized DTDs are an extension of DTDs, yielding more expressiveness using of homomorphic functions. Context-sensitive element types are expressed through specializations. Specialized DTDs have been introduced and studied in detail in [21, 22] and are known to be equivalent to type mechanisms in XML Schema. It is not difficult to see that specialized DTDs are strictly more expressive than the combination of DTDs and XSCs (e.g., specialized DTDs allow the specification of cardinalities). In this section, we show how one can create a specialized DTD from a given DTD and set of XSCs, if such a specialized DTD exists. If a specialized DTD does not exist, one can conclude that the original DTD and the XSCs are inconsistent. This approach provides a convenient way to map a grammar-based and a pattern-based schema specification to a single grammar-based schema.

Definition 4.4 **(Specialized DTD)** A *specialized DTD* is a 4-tuple $< \mathbf{E}, \mathbf{E}', w, \mu >$ where 1) $\mathbf{E}$ and $\mathbf{E}'$ are finite alphabets; 2) w is a DTD over $\mathbf{E}'$; and, 3) μ is a mapping from $\mathbf{E}'$ to $\mathbf{E}$. □

The set $\mathbf{E}'$ provides specializations for each element (type) in $\mathbf{E}$. Through a specialized DTD, it is possible to specify subtypes, or different content models for an element type under different contexts. μ induces a homomorphism on elements, words, and trees over $\mathbf{E}'$. Let $T(w)$ be the set of document trees that conform to w. For a specialized DTD D', a document d satisfies D' iff d is the homomorphic image of a tree $t \in T(w)$.

Given a DTD D and a set of XSCs Σ, a specialized DTD D' such that all documents conformant to D' also conform to D and satisfy Σ can be constructed in an obvious manner. The construction algorithm is illustrated by an example below. Initially, we assume $\mathbf{E}' = \mathbf{E}$. $\mathbf{E}'$ and w are modified iteratively according to the given XSCs. Consider, for example, the auction DTD and the path co-occurrence constraint C_1: `auctions.auction:  seller.type.store` $\leftrightarrow$ `price.tax`.

1. First, we create two specializations, $\texttt{auction}_{\texttt{sto}}$ and $\texttt{auction}_{\overline{\texttt{sto}}}$, to denote the two subtypes of `auction`, one contains a descendant path `seller.type.store` and the other does not. We also create specializations, $\texttt{price}_{\texttt{tax}}$ and $\texttt{price}_{\overline{\texttt{tax}}}$, for the element `price`, $\texttt{seller}_{\texttt{sto}}$ and $\texttt{seller}_{\overline{\texttt{sto}}}$ for `seller`, and $\texttt{type}_{\texttt{sto}}$ and $\texttt{type}_{\overline{\texttt{sto}}}$ for `type`.

2. Then, we replace `price` (`seller`) in the content model of $\texttt{auction}_{\texttt{sto}}$ by $\texttt{price}_{\texttt{tax}}$ ($\texttt{seller}_{\texttt{sto}}$) and in the content model of $\texttt{auction}_{\overline{\texttt{sto}}}$ by $\texttt{price}_{\overline{\texttt{tax}}}$ ($\texttt{seller}_{\overline{\texttt{sto}}}$). Further, we replace `type` in the content model of $\texttt{seller}_{\texttt{sto}}$ by $\texttt{type}_{\texttt{sto}}$ and in the content model of $\texttt{seller}_{\overline{\texttt{sto}}}$ by $\texttt{type}_{\overline{\texttt{sto}}}$.

3. Let m be the content model for `price`. We then compute two modified forms of m, m' and $\overline{m'}$, such that without adding new words to the language, m' is the maximum model that has `tax` as a required element, and $\overline{m'}$ is the maximum model that contains no `tax` element. More specifically, if $L(m)$ denotes the language specified by m, then $L(m') = L(m) \cap L(\Sigma'^{*}\ \texttt{tax}\ \Sigma'^{*})$ and $L(\overline{m'}) = L(m) - L(m')$. m' and $\overline{m'}$ are the content models for $\texttt{price}_{\texttt{tax}}$ and $\texttt{price}_{\overline{\texttt{tax}}}$, respectively. An empty model of m' implies that the XSC is inconsistent with the DTD in the case of path implication (co-occurrence), or is trivial in the case of path absence. The computation of m' and $\overline{m'}$ has the same complexity as the intersection/difference problem of regular expressions. Similar computations are applied for $\texttt{seller}_{\texttt{sto}}$, $\texttt{seller}_{\overline{\texttt{sto}}}$, $\texttt{type}_{\texttt{sto}}$, and $\texttt{type}_{\overline{\texttt{sto}}}$.

4. Finally, we modify all content models in w that contain the element type `auction` by substituting `auction` for the alternation of its specialized types, i.e., $\texttt{auction}_{\texttt{sto}} + \texttt{auction}_{\overline{\texttt{sto}}}$. These steps are analogoulsy performed for `price`, `seller`, and `type`.

Example 4.5 The following shows a portion of the specialized DTD for the auction DTD and the XSC `auctions.auction: seller.type.store` $\leftrightarrow$ `price.tax`. For the sake of readability, the mapping μ from $\mathbf{E}'$ to $\mathbf{E}$ is implicit. Elements of the form a_b are mapped to a, and others are mapped to themselves.

$\mathsf{auctions} : \mathsf{auction}_{\mathsf{sto}} + \mathsf{auction}_{\overline{\mathsf{sto}}}$
  $\mathsf{auction}_{\mathsf{sto}} : \mathsf{item}\ \mathsf{seller}_{\mathsf{sto}}\ \mathsf{buyer}\ \mathsf{price}_{\mathsf{tax}}\ \mathsf{payment}$
  $\mathsf{auction}_{\overline{\mathsf{sto}}} : \mathsf{item}\ \mathsf{seller}_{\overline{\mathsf{sto}}}\ \mathsf{buyer}\ \mathsf{price}_{\overline{\mathsf{tax}}}\ \mathsf{payment}$
  $\mathsf{seller}_{\mathsf{sto}} : \mathsf{contact}\ \mathsf{type}_{\mathsf{sto}}$
  $\mathsf{seller}_{\overline{\mathsf{sto}}} : \mathsf{contact}\ \mathsf{type}_{\overline{\mathsf{sto}}}$
  $\texttt{type}_{\texttt{sto}} : \texttt{store}$
  $\texttt{type}_{\overline{\texttt{sto}}} : \texttt{personal}$
  $\mathsf{price}_{\mathsf{tax}} : \mathsf{closingBid}\ \mathsf{tax}$
  $\mathsf{price}_{\overline{\mathsf{tax}}} : \mathsf{closingBid}$

Consider the case where all of the XSCs relate to the specializations of different elements under the same context. That is, $e : a_1 \circ b_1,\ e : a_2 \circ b_2,\ \cdots,\ e : a_n \circ b_n$, where $\circ \in \{\rightarrow, \leftrightarrow, \perp\}$ and $a_i \neq a_j$ for all $i \neq j$. Then, for Σ with size n, there is a total of 2^n combinations regarding whether a_i is present/absent, and consequently, 2^n specializations for the element referred to by e are required. Such potential exponential blowups of DTDs from incorporating XSCs using the specialization mechanism further confirms the advantages of using XSCs as an add-on to DTD, i.e., providing a schema with extra expressive power while keeping the simplicity and readability of DTDs.

5 Conclusions and Future Work

In this paper, we presented a structural constraint language (XSCs) for specifying path conditions XML documents have to satisfy. We have shown how XSCs can be used as a stand-alone schema formalism, allowing to incrementally develop a schema using paths. Important computational aspects such as logical implication and consistency are efficiently decidable and, together with a sound and complete axiomatization of XSCs, provide important tools for a path-based schema design.

We also showed how XSCs interact with a grammar-based schema formalism, in particular how XSCs can be derived from DTDs or mapped to specialized DTDs. From a practical point of view, XSCs can easily be validated using XSLT or XPath, or during the conformance check in the presence of a DTD.

Although paths in XSCs are not as expressive as, e.g., XPath expressions, the proposed work provides a first step towards studying the usage and complexity of this alternative schema formalism. We are currently investigating suitable subsets of XPath as path patterns to be used in XSCs. Preliminary results show that even a precise semantics is debatable. For example, there are several possible interpretations of a path implication of the pattern $//e \rightarrow //d$. Obviously, due to wildcards in such expressions, the complexity of the logical implication and consistency problems increase drastically (mainly due to expensive path containment checks [19]). As a guidance, we focus in particular on features present in Schematron [17] as most frequently used pattern-based schema formalism used in practice.

References

[1] M. Arenas, W. Fan, L. Libkin: On Verifying Consistency of XML Specifications. In *20th ACM Symposium on Principles of Database Systems*, 259–270, ACM Press, 2002.

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

[3] S. Abiteboul, V. Vianu: Regular Path Queries with Constraints. In *16th ACM Symposium on Principles of Database Systems*, 122–133, 1997.

[4] P. Buneman, S. Davidson, W. Fan, C. Hara: Keys for XML. In *10th International World Wide Web Conference*, 201–210, ACM, 2001.

[5] P. Buneman, W. Fan, S. Weinstein: Path Constraints on Semistructured and Structured Data. In *17th ACM Symposium on Principles of Database Systems*, 129–138, ACM Press, 1998.

[6] P. Buneman, W. Fan, S. Weinstein: Interaction between Path and Type Constraints. In *18th ACM Symposium on Principles of Database Systems*, 56–67, ACM Press, 1999.

[7] T. Bray, J. Paoli, C. Sperberg-McQueen: Extensible Markup Language (XML) 1.0. W3C Recommendation, February 1998.

[8] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Simeon, M. Stefanescu: XQuery 1.0: An XML Query Language. W3C Working Draft, 2002.

[9] A. Cali, D. Calvanese, M. Lenzerini: Semistructured Data Schemas with Expressive Constraints. In *7th Int. Workshop on Knowledge Representation meets Databases*, 3–16, 2000.

[10] J. Clark, S. DeRose. XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999.

[11] J. Clark: RELAX NG. Working Draft, June 2002. www.oasis-open.org/committees/relax-ng

[12] W. Fan, G. M. Kuper, J. Simeon: A Unified Constraint Model for XML. In *10th International World Wide Web Conference*, 179–190, ACM Press, 2001.

[13] W. Fan, L. Libkin: On XML Integrity Constraints in the Presence of DTDs. In *20th ACM Symposium on Principles of Database Systems*, 114–125, ACM Press, 2001.

[14] W. Fan, J. Simeon: Integrity Constraints for XML. In *19th ACM Symposium on Principles of Database Systems*, 23–34, ACM Press, 2000.

[15] R. Goldman, J. Widom: Dataguides: Enabling Query Formulation and Optimization in Semistructured Databases. In *23rd International Conference on Very Large Data Bases*, 436–445, Morgan Kaufmann, 1997.

[16] H. Hosoya, B. C. Pierce: XDuce: A typed XML processing language. In *Proc. of the 3rd Int. Workshop on the Web and Databases*, 226–244, LNCS 1997, Springer, 2000.

[17] R. Jelliffe. The Schematron: An XML Validation Language using Patterns in Trees. www.ascc.net/xml/resource/schematron/.

[18] G. Kuper, J. Simeon: Subsumption for XML Types. In *8th Int. Conference on Database Theory* , 331–345, LNCS 1973, Springer, 2001.

[19] G. Miklau, D. Suciu: Containment and Equivalence of Tree Patterns. In *21th ACM Symposium on Principles of Database Systems*, 2002.

[20] F. Neven, T. Schwentick: Automata- and logic-based pattern languages for tree-structured data. Manuscript, 2000.

[21] Y. Papakonstantinou, P. Velikhov: Enhancing Semistructured Data Mediators with Document Type Definitions. In *15th Int. Conference on Data Engineering* , 136–145, IEEE, 1999.

[22] Y. Papakonstantinou, V. Vianu: DTD Inference for Views of XML Data. In *19th ACM Symposium on Principles of Database Systems*, 35–46, ACM Press, 2000.

[23] XML Schema. W3C Recommendation, 2001.