On Tree Pattern Constraints for XML Documents

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

Abstract

In this paper, we introduce the concept of tree pattern constraints (XTPCs) for XML documents as a pattern-based schema formalism founded on XPath. XTPCs provide an effective means to specify conditions on path or tree patterns that XML documents have to satisfy. Conditions include implication, absence, and co-occurrence of patterns. XTPCs can be used as stand-alone schema formalisms or in conjunction with a DTD or XML Schema. We study in detail the (bounded) implication and consistency problems for XTPCs, give a sound and complete axiomatization as well as complexity results. If used in conjunction with a DTD, we study the consistency problem for XTPCs in the presence of DTDs.

1 Introduction

Although XML documents do not necessarily require any schema formalism, schemas are known to be useful in order to effectively process, query, and manage XML documents. Several schema formalisms have been proposed, in particular Document Type Definitions (DTD) and XML Schema. Such grammar-based schema formalisms, however, fall short when it comes to the specification of complex structural relationships among patterns that go beyond admissible parent-child and sibling relationships in XML documents. For example, consider the case where in a document fragment of a specific type (element) the existence of a tree pattern emanating from that type requires (or precludes) the existence of one or more other patterns in the same or other fragment. Using existing schema formalisms, such pattern relationships can only be added through either type specializations (e.g., [18]) or making all type specializations explicit in the schema (DTD or XML schema), resulting in large, complex, and non-intuitive schema specifications.

Pattern-based schema formalisms, in contrast, offer expressive models (typically based on XPath [19]) to specify patterns and relationships among patterns, ranging from simple path to complex tree patterns. Interestingly, there are only very few pattern-based schema formalisms, most notably the not-yet-formalized Schematron [12]. Pattern-based schema languages, of course, do not provide rich typing mechanisms or data types. From a schema design point of view, it is thus desirable to combine grammar- and pattern-based schema formalisms.

In this paper, we propose tree pattern constraints (XTPCs) for XML documents that address the shortcomings mentioned above. The concept of XTPCs is inspired by recent proposals for absolute and relative keys, foreign keys, and general functional dependencies as part of XML specifications (see, e.g., [3, 9, 11, 13]). While these types of constraints focus on admissible relationships among values associated with nodes in an XML document, XTPCs concentrate on admissible relationships among structures in

documents, thus providing a framework orthogonal to constraints studied in the above works. As our primary focus is on admissible patterns, we use a fragment of XPath as basis for the specification of XTPCs.

To get a good understanding of tree pattern constraints as schema formalism, in this paper our primary focus is on using XTPCs as stand-alone XML specification where types and type relationships are defined in the form of node, path, and tree patterns that are allowed to occur in an XML document.

As it is the case for any type of XML specification, e.g., keys in the presence of a DTD or the combination of keys and foreign keys, *consistency* (or satisfiability) of a specification is an important property (e.g., [1, 4, 10, 11]). In this paper, we study in detail the (bounded) consistency and implication problems for XTPCs. We present a sound and complete set of inference rules as well as an algorithm to derive new constraints that are implied by a given set of XTPCs. The idea of deriving new (implied) constraints is to incrementally add tree patterns to a so-called *implication closure pattern (ICP)*. Such an ICP then describes the structure of an XML document that conforms to all explicit and implied XTPCs, if such a document exists.

The latter aspect relates to the consistency problem of XTPCs, that is, is there an XML document that conforms to a given set of XTPCs. We show that the (bounded) consistency and implication problems for XTPCs as stand-alone schema formalism are decidable. Our main conclusion is that XTPCs are an expressive and easy to use schema formalism that can not only be used as alternative stand-alone XML specification, but also in addition to existing XML specifications, including DTD, XML Schema, and key constraints. A major concern in using XTPCs in conjunction with other types of schema formalisms is that the possibility of creating inconsistencies in specifications increases because of the complex interaction among XTPCs and other types of schema components, especially when specifications are written in stages (schemas evolve as new requirements are discovered). It is shown in this paper that the consistency problem for an XML specification consisting of a DTD and an XTPC specification is decidable.

Related Work. Abiteboul's and Vianu's work on path constraints [2] has inspired numerous proposals on extending schema formalisms for XML by key, foreign key, and inverse constraints [3, 9, 11, 13]. The interaction, i.e., implication and consistency, among such constraints in isolation and in combination with DTDs has been studied in detail by Fan et al. [1, 4, 10, 11]. The objectives of these works can be considered orthogonal to ours. Whereas our approach is concerned with pattern-based structural constraints as stand-alone or add-on schema formalism, the above works focus almost exclusively on key and foreign key models for node values in the context of existing schema formalisms.

There are only a few works that deal with rich typing mechanisms (in addition to a given schema), which can be considered pattern-based constraints. Cali et al. [8] introduce the concept of extended local constraints to specify required edges in semistructured data. Papakonstantinou and Vianu [18] propose specialized DTDs to represent richer type structures in addition to "normal" DTDs. To the best of our knowledge, Schematron [12] 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 reasoning about a pattern-based schema formalism for XML.

Organization. In the following section, we introduce some fundamental concepts such as the XML tree model, tree patterns, and XP expressions. In Section 3, we present the syntax and semantics of our tree pattern constraint language (XTPCs). The (bounded) implication and consistency problems for XTPCs as stand-alone XML specification are studied in detail in Section 4. Section 5 illustrates the interaction

among XTPCs and DTDs. We conclude the paper in Section 6 with a summary and outline of future work. We assume that the reader is familiar with XPath and basic properties of its fragments [5] as well as the fundamental work on query containment (e.g., [15, 16]).

2 Background and Definitions

XML Tree Model. We model XML documents as unordered, node-labeled trees over an infinite alphabet $\mathbf{E}$ of node labels (element names). The set of all trees over $\mathbf{E}$ is denoted $T_{\mathbf{E}}$. For a tree $t \in T_{\mathbf{E}}$, the set of nodes in t is denoted $nodes(t)$, and the label of a node $v \in nodes(t)$ is denoted $label(v)$. the depth of t in terms of path from the root is denoted $depth(t)$. We adopt a view similar to the data model underlying XPath 2.0 [19] in which the root node of an XML tree is called the *document node*, labeled by the root symbol "/" (see Fig. 1). A document node contains exactly one child node, known as *root element*. For the sake of simplicity, attribute nodes and text values, which can be handled similarly to element nodes, are not considered in this paper.

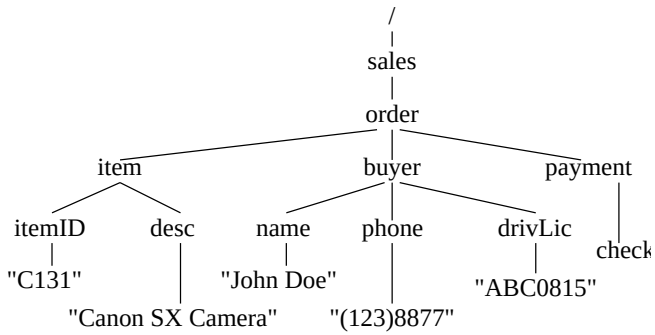

Figure 1: Example XML Tree

XP Expressions. We use a fragment of XPath as basis for our tree pattern constraint language. This fragment includes the child axis (/), context node (.), descendant axis (//), wildcards (*), and branching ([ ]). Note that XPath expressions with upward axis (e.g., parent and ancestor axis) can be transformed into equivalent upward-axis-free ones [17], and are thus excluded from our discussions. Expressions based on this fragment are called *XP expressions* and are based on the following grammar.

$$p ::= \ \bot \mid p/p \mid p/\!\!/p \mid e \mid * \mid . \mid p[p], \quad e \in \mathbf{E}$$

The symbol $\bot$ denotes an XP expression that cannot be satisfied by any XML tree. It is not part of XPath, but we include it here to simplify proofs and discussions. The semantics of XP expressions in terms of evaluation conforms to the XPath 2.0 specification [19]. XP expressions are relative expressions, that is, a context node is assumed. Here we assume the context node to be the document node, unless specified otherwise.

Tree Patterns. Tree patterns [15] are a graphical representation of XP expressions. As our constraint language specifies relationships between tree patterns that must (not) occur in XML trees, such patterns offer a way to better visualize XP expressions. They also provide us with a basis for the notions of tree embedding and subpatterns.

A tree pattern τ is an unordered tree over alphabet $\mathbf{E} \cup \{*, .\}$[1] with a subset of edges called *descendant edges* and a distinguished *result node*. Descendant edges are represented by double-lines and other edges are called *child edges*. The result node is marked by a circle (see Fig. 2). The set of nodes and the root node of a pattern τ are denoted $nodes(\tau)$ and $root(\tau)$, respectively. We use $depth(\tau)$ and $resdepth(\tau)$ to denote the depth of a pattern τ and the distance between $root(\tau)$ and the result node. The construction of patterns from XP expressions (and vice versa) is straightforward. An embedding e from τ to $t \in T_{\mathbf{E}}$ is a function $e : nodes(\tau) \rightarrow nodes(t)$, which is root preserving, respects node labels and edge relationships [15].

We now introduce the notion of *subpattern*. Let $range(e)$ be the set of nodes a tree embedding e maps to. For two tree patterns τ and ρ, τ is a subpattern of ρ, denoted $\tau \sqsubseteq \rho$ if the following holds. For all $t \in T_{\mathbf{E}}$ and for all tree embeddings e_ρ from ρ to t, there exists an embedding e_τ from τ to t such that $range(e_\tau) \subseteq range(e_\rho) \cup \{root(t)\}$. Let p and q be the XP expressions corresponding to the patterns τ and ρ, respectively. We sometimes will write $p \sqsubseteq q$ iff $\tau \sqsubseteq \rho$, as there is a clear correspondence between tree patterns and XP expressions. Examples of subpatterns are given in Fig. 2.

Although based on different definitions, subpattern and *boolean pattern containment*[15] are indeed equivalent. Boolean patterns are simply tree patterns with no result node and are evaluated to true if an embedding exists and false otherwise. For an XP expression p, let $p(t)$ denote the set of nodes determined by evaluating p aginst tree $t \in T_{\mathbf{E}}$. For two boolean patterns p and q, p is *contained* in q, denoted $p \subseteq q$, iff for all $t \in T_{\mathbf{E}}$, $p(t) \Rightarrow q(t)$.

Proposition 2.1 Given two XP expressions p and q. Let p_b and q_b be the corresponding boolean patterns. Then $p \sqsubseteq q$ iff $q_b \subseteq p_b$.

Proof: The "only-if" direction is obvious. We will prove the "if" direction by contradiction. Suppose that $\forall t : q_b(t) \Rightarrow p_b(t)$ and $p \sqsubseteq q$ is false. It follows that an embedding e from q to t exists for some t and there exists no embedding e' from p to t such that $range(e') \subseteq range(e)$. Let t^* be the fragment of t covered by the range of e. That is, t^* is the tree pattern with nodes that are not in the range of e deleted. A document node is inserted to t^* if it is missing. Since $q(t^*)$ is non-empty, by assumption, $p(t^*)$ is also non-empty. Thus, there is an embedding e' from q to $nodes(t^*) = range(e)$. Contradiction. ■

It is important not to confuse the concept of subpattern with *query containment* where an XP expressions p is contained by q, denoted $p \subseteq q$, iff $p(t) \subseteq q(t)$ for all XML trees $t \in T_{\mathbf{E}}$. Two expressions can still be related by subpattern even if their query results are disjoint (see Fig. 2). In this paper, we safely assume the existence of algorithms to check for subpatterns and query containment. Note that containment for XP expressions is in general coNP-complete and in PTIME for expressions that do not contain branching [15, 16].

3 Tree Pattern Constraints

A tree pattern constraint for XML, called XTPC, consists of two components, a *context* and a *pattern relationship*, and has the general form $c : p_1 \circ p_2$. Conceptually, the context c of an XTPC specifies where in an XML tree the constraint is applicable. For example, a context of `sales/order` means that the constraint is only applicable under nodes with a rooted path `sales/order`. The idea of having a context for a constraint is similar to the concept of relative key constraints [3] or functional dependencies [13].

[1] Note that Miklau and Suciu [15] do not include the context node (.) in their pattern model.

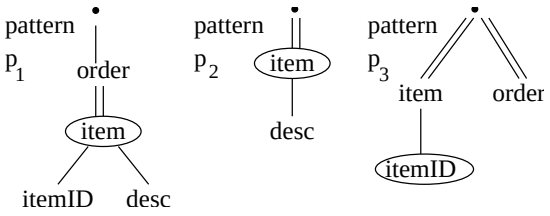

Figure 2: The following holds for the above patterns: $p_2 \sqsubseteq p_1$, $p_3 \sqsubseteq p_1$, $p_1 \subseteq p_2$, and $p_1 \not\subseteq p_3$.

Following the context is a pattern relationship between the two patterns, p_1 and p_2, emanating from the context c. Three types of relationships are supported: pattern implication, absence, and co-occurrence. The formal syntax and semantics of XTPCs are as follows.

Definition 3.1 *(Pattern Implication, Absence, Co-occurrence)*
Let c, p_1, and p_2 be XP expressions. An XML tree t satisfies

- the *pattern implication* constraint $c : p_1 \rightarrow p_2$, denoted $t \models c : p_1 \rightarrow p_2$, if $c[p_1](t) \subseteq c[p_2](t)$,
- the *pattern co-occurrence* constraint $c : p_1 \leftrightarrow p_2$, denoted $t \models c : p_1 \leftrightarrow p_2$, if $c[p_1](t) = c[p_2](t)$, and
- the *pattern absence* constraint $c : p_1 \not\rightarrow p_2$, denoted $t \models c : p_1 \not\rightarrow p_2$, if $c[p_1](t) \cap c[p_2](t) = \emptyset$.

In simple terms, a pattern implication constraint states that if under a node determined by an expression c the pattern p_1 occurs, then the pattern p_2 must occur under that node, too. In contrast, a pattern absence constraint precludes the existence of the pattern p_2 if p_1 occurs. Finally, pattern co-occurrence is simply a two-way pattern implication. All three cases are described in terms of the nodes selected by respective combined XP expressions $c[p_1]$ and $c[p_2]$.

Example 3.2 Consider the following XTPCs.

$\sigma_1 = . : . \rightarrow$ `sales`
$\sigma_2 = */$`order` $: . \rightarrow .//$`item` [`itemID`][`desc`]
$\sigma_3 = .//* :$ `payment/check` $\rightarrow$ `buyer/drivLic`
$\sigma_4 = .//$`buyer` : `drivLic` $\rightarrow .//$`phone`
$\sigma_5 = .//$`payment` : `creditCard` $\leftrightarrow$ `expDate`
$\sigma_6 = .//$`payment` : `creditCard` $\not\leftrightarrow$ `check`
$\sigma_7 =$ `sales`$//$`payment` : `cash` $\rightarrow \perp$

XTPCs σ_1 and σ_2 specify required patterns (`sales` and $.//$`item`[`itemID`][`desc`]) under the specified contexts. σ_3 and σ_4 specify implications between patterns. For instance, σ_3 states that under any context, the existence of the pattern `payment/check` implies `buyer/drivLic`. σ_5 states that under any node labeled `payment`, `creditCard` and `expDate` must co-occur, whereas σ_6 states that `creditCard` and `check` must not co-occur. σ_7, which states that `cash` cannot occur under the context, shows how to use $\perp$ together with pattern implication to specify patterns that are not allowed. All of the above constraints are satisfied by the tree in Fig. 1.

Despite the difference in syntax, both pattern absence and co-occurrence constraints can be easily rewritten into pattern implication constraints. As mentioned earlier, pattern co-occurrences are essentially two-way pattern implications, meaning that $c : p_1 \leftrightarrow p_2$ is equivalent to the combination $c : p_1 \rightarrow p_2$ and $c : p_2 \rightarrow p_1$.

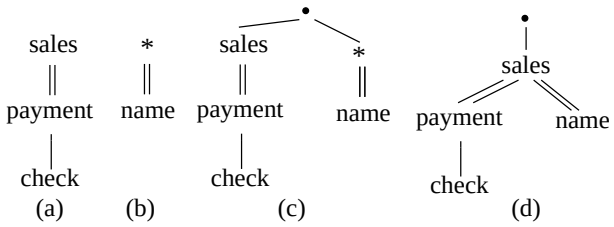

Figure 3: Tree pattern (c) is the composition of (a) and (b). (d) is the tree pattern obtained by applying the MERGEROOT function to pattern (c) (discussed in Section 4.1)

To rewrite a pattern absence into pattern implication, we utilize the notion of *pattern composition*, an operation for merging two tree patterns. Given tree patterns τ and ρ, the composition of τ and ρ, denoted $\tau \otimes \rho$, is obtained by creating a root node labeled "." (the current node symbol) and attaching both τ and ρ to this root (see Fig. 3). Composition of XP expressions is defined similarly. Note that, given a pattern absence constraint $c : p_1 \not\rightarrow p_2$, if p_1 cannot co-occur with p_2, then the pattern composition of p_1 and p_2 can never occur. Therefore, $c : p_1 \not\rightarrow p_2$ can be rewritten as a pattern implication $c : p_1 \otimes p_2 \rightarrow \bot$.

Clearly, XTPCs are less expressive than DTDs or XML Schema since they do not address ordering or cardinality of (groups of) elements. However, it easy is to see that structural conditions specified in, e.g., the content models of a DTD, such as optional child nodes, required siblings or alternative groups of child elements, can be formulated using XTPCs.

4 Bounded Logical Implication and Consistency

XTPCs can be viewed as a rule-based system where document structures are defined by rules in a form similar to rewriting rules. Consider the XTPC $.//* : . \rightarrow$ `e`. Obviously, such a rule causes a recursive specification because the existence of a root element leads to an infinite chain of `e` elements, and no document of finite size can conform to it. Note that this strongly resembles non-termination properties studied in the context of rewriting systems and XSLT in particular (e.g., [7]). In both cases, the termination problem is known to be undecidable. For the consistency problem for XTPCs, this observation suggests an expensive lower bound, or even undecidability. While the general implication and consistency problems for XTPCs are left as open problems, we study a practically useful setting where the depth of XML trees is bounded.

As indicated in the introduction, reasoning about XML specifications plays an important role in different settings. The *bounded logical implication and consistency problems* discussed in this section are a restricted form of the logical implication and consistency problems, with the domain restricted to the set of XML trees that have a depth no larger than a constant B, called the *depth bound*.

Definition 4.1 *(Bounded Logical Implication and Consistency)*
Given a set of XTPCs Σ and a depth bound $B > 0$. An XTPC σ is *implied* by Σ, denoted $\Sigma \models_B \sigma$, iff for any XML tree t with $depth(t) \leq B$, $t \models \Sigma$ implies $t \models \{\sigma\}$. Σ is said to be *consistent* if there exists an XML tree t such that $depth(t) \leq B$ and $t \models \Sigma$. Σ is *inconsistent* otherwise. □

Example 4.2 Assume Σ consists of two constraints, σ_1=.//`payment`: `creditCard` $\leftrightarrow$ `expDate` and σ_2=.//`payment`: `creditCard` $\not\rightarrow$ `check`. Since the node `creditCard` must co-occur with `expDate`, if

`check` contradicts `creditCard`, then `check` must also contradict `expDate`. Therefore, $\Sigma \models_B .\!/\!/\texttt{payment}: \texttt{expDate} \not\rightarrow \texttt{check}$. Furthermore, Σ is consistent, as the XML tree in Fig. 1 satisfies Σ.

Typically, the study of the logical implication problem involves the development of a decision procedure and/or a sound and complete set of inference rules. Below, both approaches for bounded logical implication of XTPCs are presented.

4.1 Inference Rules

Axiomatization is an important concept in reasoning about constraints as inference rules can concisely capture the essential properties of the constraint language under study. In this section, the set of inference rules for XTPCs is presented.

Recall that $\perp$ denotes an XP expression that always evaluates to an empty result. Rules 1a-1c illustrate the cases with $\perp$ in either the context or the pattern relationship.

Rule 1a. (empty-context) $\perp: x \rightarrow y$ for any x, y.

Rule 1b. (empty-antecedent) $z :\perp\rightarrow x$ for any z, x

Rule 1c. (empty-consequence) If $z : x/y \rightarrow\perp$, then $z/x : y \rightarrow y'$ for any $y' \in XP$.

Rule 1a and 1b are trivial from the definition of pattern implication. For Rule 1c, notice that the pattern $z/x/y$ can never occur in an XML tree that satisfies the constraint $z : x/y \rightarrow\perp$. Thus, it does not matter what the consequence of the pattern implication $z : x/y \rightarrow y\prime$ is — the constraint always holds. Besides, it also does not matter where x is placed, as part of context or as part of pattern relationship. That is, x can be moved to the context ($z/x : y \rightarrow y'$) without altering the semantics of the constraint. The next two rules show some intuitive properties of pattern implications.

Rule 2a. (subpattern) If $y \sqsubseteq x$, then $z : x \rightarrow y$ for any $z \in XP$.

Rule 2b. (transitivity) If $z : x_1 \rightarrow x_2$ and $z : x_2 \rightarrow x_3$, then $z : x_1 \rightarrow x_3$.

Since tree patterns are the basic building blocks for XTPCs, one way to derive constraints is by combining or restricting such patterns. Tree pattern manipulations form the basis for rules 3a and 3b.

Rule 3a. (context-containment) If $z : x \rightarrow y$ and $z' \subseteq z$, then $z' : x \rightarrow y$.

Rule 3b. (pattern-composition)If $z : x_1 \rightarrow x_2$ and $z : x_3 \rightarrow x_4$, then $z : x_1 \otimes x_3 \rightarrow x_2 \otimes x_4$.

Rule 4 describes a way to derive constraints by shifting part of the context expression to the pattern relationship.

Rule 4. (context-shifting) If $z_1/z_2 : x \rightarrow y$, then $z_1 : z_2/x \rightarrow z_2/y$.

Note that in the above rule, z_2 is distributed to the two expressions in the pattern relationship. Such distribution allows the two z_2's map to different nodes in an XML tree. Therefore, trees that satisfy $z_1 : z_2/x \rightarrow z_2/y$ may not satisfy $z_1/z_2 : x \rightarrow y$. In other words, the converse of rule 4 does not hold.

Next, we describe rules that rely on the fact that XML trees have exactly one root element. Consider the XP expression $.[*/\texttt{b}][\texttt{e}.\!/\!/\texttt{d}]/*/\texttt{f}$. Given that the context of the expression is the document node, the two wildcards "*" and the `e` element must be mapped to the root element for any valid embedding. Therefore, the root element of any tree that contains a match for this pattern must be `e`, and the expression can be rewritten to $\texttt{e}[\texttt{b}][.\!/\!/\texttt{d}]/\texttt{f}$. The action of combining the nodes that represent the root element in a tree pattern into one node is accomplished by the function MERGEROOT. Clearly, if the nodes representing root element in an expression have different element names (for example, the `a` and `e` in the expression

.[a/b][e //d]/*/f, such expression is unsatisfiable. The mergeRoot function is defined as follows, and is illustrated by an example in Fig. 3.

Algorithm 4.3 The MERGEROOT function is shown as follows.
Input: Tree pattern τ
Output: Tree pattern τ' with merged root elements
MERGEROOT(τ)
let $\tau' := \tau$
if $label(root(\tau)) \neq .$ **then return** τ'
let $c_1, \cdots, c_k$ be children of $root(\tau')$ connecting to $root(\tau)$ by child edges
if $\cup_{i=1}^{k} label(c_i)$ contains ≤ 1 element name
then merge $c_1, \cdots, c_k$ into a node n s.t.
(a) n has the children of $c_1, \cdots, c_k$,
(b) $label(n) = e$ **if** $label(c_j) = e$ for some $j \in [1, k]$, and
(c) $label(n) = *$ **if** $\forall j \in [1, k]$, $label(c_j) = *$
return τ'
else return $\perp$

Rule 5a is based on the fact that for a pattern τ, MERGEROOT(τ) is equivalent to τ under the context ".", i.e., the document node. In contrast to rule 4, rule 5b states that one can shift nodes that are required to map to the root element to the context.

Rule 5a. (root-merging) $. : x \leftrightarrow$ MERGEROOT(x) for any $x \in XP$.

Rule 5b. (single-root) If $. : e/x \rightarrow */y$ for some $e \in \mathbf{E}$, then $e : x \rightarrow y$.

Finally, rule 6 describes the case where the depth of the pattern in an implication constraint exceeds the depth bound. This rule is a departure from rules 1-5 in the sense that it is a by-product of the depth bound restriction, rather than a general property of XTPCs.

Rule 6. (out-of-bound) Let $d_z = depth(z)$, $d_{z,x} = resdepth(z) + depth(x)$, and $d_{z,y} = resdepth(z) + depth(y)$. If $z : x \rightarrow y$ and $\max\{d_z, d_{z,x}, d_{z,y}\} > B$, then $z : x \rightarrow \perp$.

Here $\max\{d_z, d_{z,x}, d_{z,y}\}$ is the minimum depth of an XML tree that contains z, z/x, and z/y. If it is greater than B, then no tree within the depth bound can contain the pattern z/x, i.e., $z : x \rightarrow \perp$.

Rules 1-6 build a sound and complete set of inference rules for bounded logical implication of XTPCs (the soundness and completeness proof is given in Sec. 4.5).

4.2 Implication Closure Pattern

Given a set of XTPCs Σ, a constraint σ, and a depth bound B, checking whether the implication $\Sigma \models_B \sigma$ holds utilizes the concept of *implication closure pattern* (ICP) and is defined as follows.

Definition 4.4 *(Implication Closure Pattern)*
Given a set of XTPCs Σ and non-empty $c, p \in XP$. The implication closure pattern of p under context c and constraint specification Σ, denoted $ICP_{c,\Sigma}(p)$ or simply $ICP_c(p)$, if Σ is clear from the context, is a tree pattern such that

(1) $\Sigma \models_B c : p \rightarrow ICP_c(p)$,
(2) for any p', $\Sigma \models_B c : p \rightarrow p'$ implies $p' \sqsubseteq ICP_c(p)$, and
(3) $ICP_c(p) = \perp$ if $\Sigma \models_B c : p \rightarrow \perp$. □

Intuitively, with two non-empty XP expressions c and p, the ICP of p under the context c can be considered as the largest tree pattern that p implies under c, where any tree pattern that p implies under c is a subpattern of, if not equivalent to, the ICP. The ICP is *empty* ($ICP_c(p) = \perp$) if no XML tree within the depth bound can satisfy Σ and contains c/p at the same time. The algorithm to construct an ICP follows. In the algorithm, we use the notion $evlp(n)$ to denote all nodes in a given tree pattern minus all nodes that are descendants of node n.

Algorithm 4.5 Computation of $ICP_c(p)$
Input: Σ, B, and non-empty $c, p \in XP$
Output: $ICP_c(p)$

```
   /* first, rewrite constraints in Σ */
(1)  for each σ ∈ Σ
(2)    if σ ≡ c : p1 ↛ p2 then σ := c : p1 ⊗ p2 →⊥
(3)    if σ ≡ c : p1 ↔ p2
(4)    then replace σ by c : p1 → p2 and c : p2 → p1
(5)    if σ ≡ . : e1/p1 → e2/p2 then /* rule 5b */
(6)      if e1 = e2 or e1 = *
(7)      then σ := e1 : p1 → p2
(8)      else if e2 = * then σ := e2 : p1 → p2
(9)      else σ := e1/p1 : . →⊥
(10)   if σ ≡ c : p →⊥ then σ := c/p : . →⊥ /* rule 1c */
(11)  /* Initialize ICP with root node and p */
(12)  let τ be the tree pattern "."
(13)  τ := ATTACH(p, root(τ))
(14)  /* main loop body */
(15)  repeat until no more nodes can be added to τ
(16)    N := nodes(τ)
(17)    for each n ∈ N
(18)      let t_n := tree(n) and c_n := c/evlp(n)
(19)      for each c' : p'1 → p'2 in Σ
(20)        if c_n ⊆ c' and p'1 ⊑ t_n then
(21)          if p'2 ⋢ t_n then ATTACH(p'2, n)
(22)          if p'2 = ⊥ then return ⊥ /* rule 1c */
(23)          if max{d_c, d_{c,p'2}, d_{c,τ}} > B /* rule 6 */ then return ⊥
(24)  if c = . then τ := MERGEROOT(τ) /* rule 5a */
(25)  return τ

ATTACH(p, n)
  if root(p) = . then
    merge root(p) and n into a new node n' s.t. n' has the same label as node n
  else connect p to n by a child edge
```

Algorithm Details. The algorithm starts with a rewrite of some constraints in Σ. In particular, all pattern absence and co-occurrence constraints are rewritten into implications. Next, τ is initialized with a root node labeled ".". Then, the initial pattern p is added to τ using the function ATTACH. The main part of the algorithm is a loop. Inside the loop, each node in τ is checked against the set of constraints in Σ. For each node n in τ and constraint $c' : p'_1 \rightarrow p'_2$, if $c/evlp(n)(= c_n)$ is contained in the context c' and the pattern p'_1 is a subpattern of $tree(n)$, the context and the condition of the pattern implication are satisfied. The consequence of the implication, p'_2, is then attached to node n if it is not already a subpattern of $tree(n)$. The next few lines verify if ICP is empty ($\perp$). The loop terminates if a full iteration on the set of nodes of τ does not yield any new pattern. Finally, if the context node is the document node ($p = .$), the nodes in the ICP required to map to a root element are merged using function MERGEROOT (see rule 5b).

Example 4.6 Consider the constraints $\sigma_1 = .\!/\!/\texttt{order}$: $\texttt{payment/check} \leftrightarrow \texttt{buyer/drivLic}$, and $\sigma_2 = .\!/\!/\texttt{order/buyer}$: $\texttt{drivLic} \rightarrow \texttt{phone}$. The construction of $ICP_{.\!/\!/\texttt{order}(.\!/\!/\texttt{check})}$, illustrated in Fig. 4, occurs as follows. First, τ is initialized with the pattern (a). In the first pass of the repeat-until loop,

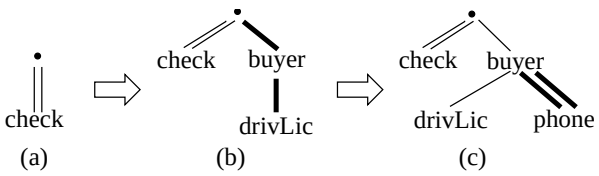

Figure 4: Construction of $ICP_{.\!/\!/\texttt{order}(.\!/\!/\texttt{check})}$

the set N contains two nodes, the root "." and `check`, and `buyer/drivLic` is attached to τ due to σ_3, as shown in (b). In the second pass, the set N is expanded to five nodes, and a new node `phone` is attached to `buyer` due to constraint σ_4. As no more new pattern can be attached to τ in the following pass, the pattern (c) is returned.

We now establish some preliminary results that lead to the correctness of the algorithm. Note that an ICP is constructed incrementally by attaching tree patterns to it. The following lemma states that at any point during a run of the algorithm, the pattern constructed at that point is implied by p under the context c.

Lemma 4.1 Given a set of XTPCs Σ and XP expressions c and p. Let $R = \langle \rho_1, \rho_2, \cdots \rangle$ be the sequence of patterns attached to τ in the computation of $ICP_c(p)$, and τ_i be the pattern obtained after attaching $\rho_1, \cdots, \rho_i$, where $i \geq 1$. For all $\rho_i \in R$, $\Sigma \models_B c : p \rightarrow \tau_i$.

Proof: We will prove the claim by induction as follows. (Base) In the algorithm, the first pattern attached to τ is p, thus $\tau_1 = p$. Obviously, $c : p \rightarrow \tau_1$ holds by the subpattern rule. (Induction step) Assume $c : p \rightarrow \tau_i$ for all $i \leq k$. We want to prove that $c : p \rightarrow \tau_{k+1}$ holds. From the transitivity rule and the assumption that $c : p \rightarrow \tau_k$, it is sufficient to show $c : \tau_k \rightarrow \tau_{k+1}$. Let $evlp_i(n)$ and $tree_i(n)$ be the envelope of n and the subtree of rooted n w.r.t. the pattern τ_i. Assume the pattern ρ_{k+1} is attached to the node n, where $n \in nodes(\tau_k)$ after a match to a constraint $\sigma = c' : p'_1 \rightarrow \rho_{k+1}$ in Σ. Then, it must be true that $c/evlp_k(n) \subseteq c'$ and $p'_1 \sqsubseteq tree_k(n)$. The steps to prove $c : \tau_k \rightarrow \tau_{k+1}$ are as follows.

$\gamma_1 = c/evlp_k(n) : p_1' \rightarrow \rho_{k+1}$ (σ and rule 3a)
$\gamma_2 = c/evlp_k(n) : tree_k(n) \rightarrow p_1\ '$ (rule 2a)
$\gamma_3 = c/evlp_k(n) : tree_k(n) \rightarrow \rho_{k+1}$ (γ_1, γ_2, rule 2b)
$\gamma_4 = c/evlp_k(n) : tree_k(n) \rightarrow tree_k(n) \otimes \rho_{k+1}$ (γ_3 and rule 3b)
$\gamma_5 = c : evlp_k(n)/tree_k(n) \rightarrow evlp_k(n)/(tree_k(n) \otimes r_{k+1}$ (γ_4 and rule 4)

Since $evlp_k(n)/tree_k(n) = \tau_k$ and $evlp_k(n)/(tree_k(n) \otimes r_{k+1} = evlp_{k+1}(n)/tree_{k+1}(n) = \tau_{k+1}$, we can conclude that $c : \tau_k \rightarrow \tau_{k+1}$. ■

The following lemma shows that Algorithm 4.5 terminates on any input.

Lemma 4.2 The algorithm halts for any set of XTPCs Σ, depth bound $B > 0$, and non-empty XP expressions c and p.

Proof: Note that the size of τ in the algorithm is bounded. Specifically, each node in τ can have at most $|\Sigma|$ patterns attached to it, and the depth of τ is no large than $B - resdepth(c)$ (see the out-of-bound rule in Section 4.1). The algorithm terminates either when no new tree pattern can be attached or when the sum of the depth of τ and the result depth of the context exceeds B. ■

Theorem 4.7 Algorithm 4.5 correctly computes the implication closure pattern.

Proof: In Lemma 4.2, we have shown that the algorithm always terminates, i.e., it stops on any input. There are two possible outcomes after a run of the algorithm. The algorithm either returns a non-empty pattern or an empty pattern. In the first case, we need to show that the pattern τ returned is indeed $ICP_c(p)$ by showing that, first, $\Sigma \models_B c : p \rightarrow \tau$, and second, for any p', if $\Sigma \models_B c : p \rightarrow p'$ then $p' \sqsubseteq \tau$.

By Lemma 4.1 and Rule 5a, we have $\Sigma \models_B c : p \rightarrow \tau$. We now claim that τ is indeed $ICP_c(p)$, i.e., the pattern that includes every possible pattern p' that p implies under c.

Claim 1: If $\Sigma \models_B c : p \rightarrow p'$ with non-empty c, p, and p', then $p' \sqsubseteq \tau$.

Assume $p' \not\sqsubseteq \tau$. We will prove $\Sigma \not\models_B c : p \rightarrow p'$ by showing that there is an XML tree t such that $t \models_B \Sigma$, $c[p](t) \neq \emptyset$ and $c[p'](t) = \emptyset$. Let ρ_c be the tree pattern for c. An XML tree $t_c \in T_{\mathbf{E}}$ is called a canonical model of ρ_c (see also [15]) if $\rho_c(t_c)$ is non-empty and has the same shape as t_c. We construct t_c from ρ_c as follows. First, if $root(\rho_c) = .$, we replace the root by the document node (which has the label "/"). Then, we replace all the descendant edges by child edges. Finally, we change the label of all wildcard nodes to an element name $e \in \mathbf{E}$. Since $p' \not\sqsubseteq \tau$, there exists a tree t_p such that $\tau(t_p) \neq \emptyset$ and $p'(t_p) = \emptyset$. Let $n \in c(t_c)$, and t be an XML tree obtained by connecting n in t_c to $root(t_p)$. Note that if $c = .$, the child nodes of $root(\tau)$ are merged (line (24)) and thus, t is a valid XML tree. Because t has nodes that match $c[p]$ but not $c[p']$, we can conclude that $\Sigma \not\models_B c : p \rightarrow p'$.

Hence, the algorithm is correct if a non-empty tree pattern is returned. Next, we will prove that if an empty tree pattern is returned, $ICP_c(p)$ must also be empty.

Claim 2: $ICP_c(p) = \perp$ if the algorithm returns $\perp$.

Case 1) Suppose that the algorithm terminates with empty consequence after attaching k patterns to τ (line (22)). Then, by Lemma 4.1, we have Σ implies $c : p \rightarrow \tau_k$ and τ_k contains a node labeled $\perp$. Since no node can match the empty expression $\perp$, $ICP_c(p)$ must be empty. Case 2) Suppose that the algorithm terminates with out-of-bound implication (line (23)). Let τ' be the pattern constructed before $\perp$ is returned. If there is a tree t that matches τ' under context c, t must have a depth greater than B. Since, by Lemma 4.1, $c : p \rightarrow \tau'$, no tree within the depth bound B can have a node that matches the

pattern c/p, and therefore, $ICP_c(p) = \perp$. Case 3) Suppose that, in line (24), MERGEROOT(τ) returns an empty pattern due to the multiple-root problem. Then the given context c must be ".", which has to map to the document node for any tree embedding. Let e_1 and e_2 be the labels of the nodes connecting to the $root(\tau)$ by child edges, where $e_1 \neq e_2$. By Lemma 4.1, $. : p \rightarrow \tau$ and thus $. : p \rightarrow .[e_1]/[e_2]$ since $.[e_1]/[e_2] \sqsubseteq \tau$. In other words, the existence of the pattern p under the document node implies two root elements e_1 and e_2. Since an XML tree can have only one root element, no XML tree can satisfy Σ if the pattern p occurs under the document node. Hence, $ICP_c(p) = \perp$.

We can now conclude that Algorithm 4.5 correctly computes $ICP_c(p)$. ■

It should be noted that given a fixed ordering of constraints in Σ, Algo. 4.5 is deterministic. ICPs are also unique up to equivalence of tree patterns.

Complexity Analysis. Given $c, p \in XP$ and a set of XTPCs Σ, let $|ICP|$ be the number of nodes in $ICP_c(p)$. We use $order(q)$ to denote the maximum number of children per node in a given pattern q. Note that if a tree pattern ρ is attached to τ in Algorithm 4.5, ρ is either the tree pattern of p (attached to τ during initialization) or the consequence of an implication $\sigma \in \Sigma$. Let A be the maximum order among all tree patterns that can be attached to τ, i.e., $A = \max\{order(q) \mid (\sigma \in \Sigma \wedge \sigma = c' : p'toq) \vee q = p\}$. Note that at most $|\Sigma|$ patterns can be attached to a node in τ and each pattern attached has an order no greater than A; thus, the order of $ICP_c(p)$ is at most $A|\Sigma|$. Since the depth of an ICP is bounded by B, the size of ICP is at most $\sum_{k=0}^{k=B-1}(A|\Sigma|)^k = \frac{1-(A|\Sigma|)^B}{1-A|\Sigma|} = O((A|\Sigma|)^B)$.

Because at least one node is attached to τ in every pass, the repeat-until loop is executed at most $|ICP|$ times. The two for-each loops are executed at most $|ICP| * |\Sigma|$ times. The first two if-statements do containment/subpattern checks at most three times. Let Q be the union of $\{c, p\}$ and the set of XP expressions that occur in Σ. Let s be the maximum pattern size among the patterns in Q, w be the maximum number of *-labeled nodes in $q \in Q$ forming a path consisting exclusively of child edges, and d be the maximum number of descendant edges in $q \in Q$, containment/subpattern checks can be decided in $O(s^2(1+w)^{d+1})$ time (Theorem 4 in [15]). Therefore, ICP can be computed in $O((A|\Sigma|)^{2B}|\Sigma|s^2(1+w')^{d+1})$ $= O(A^{2B}\ |\Sigma|^{2B+1}\ s^2(1+w')^{d+1})$ time.

4.3 Bounded Logical Implication

Since the tree pattern $ICP_{c,\Sigma}(p_1)$ is the largest pattern p_1 implies under the constraint specification Σ and context c, if the implication $\Sigma \models_B c : p_1 \rightarrow p_2$ holds, p_2 must be a subpattern of $ICP_c(p_1)$. The relationship between ICPs and the bounded implication problem is summarized in the following theorem, which can easily be transformed into a decision procedure.

Theorem 4.8 $\Sigma \models_B \{c : p_1 \rightarrow p_2\}$ iff one of the following conditions holds: (1) $c = \perp$, (2) $p_1 = \perp$, (3) $ICP_c(p_1) = \perp$, or (4) $p_2 \sqsubseteq ICP_c(p_1)$.

Proof: ($\Leftarrow$) By Rule 1a and 1b (Section 4.1), $c : p_1 \rightarrow p_2$ holds if $c = \perp$ (Condition (1)) or $p_1 = \perp$ (Condition (2)). If $ICP_c(p_1) = \perp$ (Condition (3)), then $c[p_1](t) = \perp \subseteq c[p_2](t)$ for all $t \in T_{\mathbf{E}}$ and thus, $c : p_1 \rightarrow p_2$. If $p_2 \sqsubseteq ICP_c(p_1)$ (Condition (4)), by the definition of ICP, we have $c : p_1 \rightarrow p_2$. Therefore, we have $\Sigma \models_B \{c : p_1 \rightarrow p_2\}$ if any of the four conditions holds. ($\Rightarrow$) Now, let us assume that none of the conditions holds. Then $c \neq \perp$, $p_1 \neq \perp$, $ICP_c(p_1) \neq \perp$ and $p_2 \not\sqsubseteq ICP_c(p_1)$. By the definition of ICP, p_1 does not imply p_2 under c, and hence, the implication $\Sigma \models_B \{c : p_1 \rightarrow p_2\}$ does not hold. ■

Complexity Analysis. The decision procedure for the bounded implication problem involves the computation of $ICP_c(p)$, a few comparisons that require only constant time, and a subpattern test for $p_2 \sqsubseteq ICP_c(p_1)$. Therefore, the bounded implication problem has the same complexity bound as the ICP computation, i.e., $O(A^{2B}\ |\Sigma|^{2B+1}\ s^2(1+w)^{d+1})$.

4.4 Bounded Consistency

Similar to the implication problem, ICPs are again used as main vehicle to determine the consistency of a set of XTPCs Σ. The major challenge here is to check whether it is possible to construct a tree that is within the depth bound and satisfies Σ. Since an XML tree needs at least a document node and a root element, consistency of Σ can be determined by computing $ICP_{\cdot}(*)$, i.e., the required pattern under the document node with the presence of a root element. In particular, if $ICP_{\cdot}(*)$ is non-empty, one can easily construct an XML tree t from the ICP such that t satisfies Σ. Obviously, Σ is inconsistent if $ICP_{\cdot}(*)$ is empty.

Theorem 4.9 A set of XTPCs Σ is consistent under a depth bound B iff $ICP_{\cdot,\Sigma}(*) \neq \perp$.

Proof: ($\Rightarrow$) If $ICP_{\cdot}(*) = \perp$, obviously no XML tree t with a document node and a root element can satisfy Σ, and therefore, Σ is inconsistent by definition. ($\Leftarrow$) Assume that $ICP_{\cdot}(*) \neq \perp$. We will show that Σ is consistent by constructing an XML tree that satisfies Σ. Let t be an XML tree obtained by (1) changing the root label of $ICP_{\cdot}(*)$ to the document node symbol "/", (2) replacing all *-labeled nodes in $ICP_{\cdot}(*)$ by an element name e, where $e \in \mathbf{E}$ does not appear in any of the constraints in Σ, and (3) replacing all descendant edges with child edges. It is clear that t satisfies all constraints in Σ because t contains the nodes necessary for an XML tree, the document node and the root element, as well as all the required patterns under them as specified by Σ. Finally, we must stress that e must be an element name that does not occur in Σ. The reason is to make sure that nodes in t labeled e match only context expression c which ends with a wildcard. ■

Complexity Analysis. Determining bounded consistency for a set of XTPCs requires the computation of ICP. Therefore, the bounded consistency problem is decidable in $O(A^{2B}\ |\Sigma|^{2B+1}\ s^2(1+w)^{d+1})$ time.

4.5 Axiomatization Proof

Rules 1-6 build a sound and complete set of inference rules for implication of XTPCs.

Theorem 4.10 Rules 1-6 are sound and complete for the bounded logical implication problem for XTPCs.

Proof: (Soundness) Let $\Sigma \subseteq XTPC$ and $\varphi \in XTPC$. We also let $\langle \sigma_1, \cdots, \sigma_n \rangle$ be the sequence of proof steps in a proof that shows $\Sigma \models_B \varphi$. The soundness of the inference rules can be proven by an easy induction on $\sigma_i, i \in [1, n]$, given that each rule is sound.

(Completeness) We need to show that if $\Sigma \models_B \varphi$, then there exists a proof to justify the implication. Let $\varphi = c : p_1 \rightarrow p_2$. If $c = \perp$, then $\sigma_1 = \varphi$ by Rule 1a. If $p_1 = \perp$, then $\sigma_1 = \varphi$ by Rule 1b. Otherwise, let α be a run of the Algorithm 4.5 to compute $ICP_c(p_1)$. If constraints $\tau_1, \cdots, \tau_l$ are rewritten to $\tau'_1, \cdots, \tau'_l$, respectively, then for each rewritten constraint τ_i, $1 \leq i \leq l$, $\sigma_{0,i} = \tau'_i$ and is justified by τ_i and rule 5b or 1c.

Let $R = \langle \rho_1, \cdots, \rho_k \rangle$ be the sequence of patterns attached to τ in α before it exits the repeat-until loop. We denote by τ_i the pattern constructed by the algorithm after attaching $\rho_1, \cdots, \rho_i$, where $1 \leq i \leq k$. In the following, a sub-proof $\sigma_i = \langle \sigma_{i,1}, \cdots, \sigma_{i,j} \rangle$ is constructed for each attachment of ρ_i to τ_{i-1} to show that $c : p_1 \rightarrow \tau_i$. Let

$\sigma_{1,1} = c : p_1 \rightarrow p_1 = c : p_1 \rightarrow \tau_1$ (Rule 2a)

For each $1 < i \leq k$, let n be the node ρ_i attached to and $c_n = c/evlp(n)$. Then,

$\sigma_{i,1} = c_n : tree(n) \rightarrow \rho_i$ (Rule 3a and $\in \Sigma \cup \{\sigma_{0,i} | 1 \leq i \leq l\}$)

$\sigma_{i,2} = c_n : tree(n) \rightarrow tree(n)$ (Rule 2a)

$\sigma_{i,3} = c_n : tree(n) \rightarrow \rho_i \otimes tree(n)$ (Rule 3b)

$\sigma_{i,4} = c : evlp(n)/tree(n) \rightarrow evlp(n)/(\rho_i \otimes tree(n)) = c : \tau_{i-1} \rightarrow \tau_i$ (Rule 4)

$\sigma_{i,5} = c : p_1 \rightarrow \tau_i$ (Rule 2b, $\sigma_{i,4}$ and σ_{i-1})

If $c \neq .$ and τ_k is non-empty, we know that p_2 is a subpattern of τ_k from the definition of ICP and we can complete the proof by:

$\sigma_{k+1} = c : \tau_k \rightarrow p_2$ (Rule 2a)

$\sigma_{k+2} = c : p_1 \rightarrow p_2$ (Rule 2b, $\sigma_{k,5}$, and σ_{k+1})

If $c = .$ and let $\text{MERGEROOT}(\tau_k) \neq \perp$, then the rest of the proof is as follows.

$\sigma_{k+1} = c : \tau_k \rightarrow \text{MERGEROOT}(\tau_k)$ (Rule 5a)

$\sigma_{k+2} = c : p_1 \rightarrow \text{MERGEROOT}(\tau_k)$ (Rule 2b, $\sigma_{k,5}$ and σ_{k+1}),

$\sigma_{k+3} = c : \text{MERGEROOT}(\tau_k) \rightarrow p_2$ (Rule 2a)

$\sigma_{k+4} = c : p_1 \rightarrow p_2$ (Rule 2b, σ_{k+2}, and σ_{k+3})

In the following cases, the ICP returned by the algorithm is empty.

Case i) If $c = .$ and $\text{MERGEROOT}(\tau_k) = \perp$, then

$\sigma_{k+1} = c : p_1 \rightarrow \text{MERGEROOT}(\tau_k) = c : p_1 \rightarrow \perp$ (Rule 5a, $\sigma_{k,5}$)

Case ii) If τ_k contains $\perp$. Since $q/\perp \equiv \perp/q \equiv \perp$ for any pattern q and $p_2 \sqsubseteq \tau_k$, p_2 must be empty, we have:

$\sigma_{k+1} = c : p_1 \rightarrow \perp$ ($\sigma_{k,5}$)

Case iii) if $\max\{d_c, d_{c,p_1}, d_{c,\tau_k}\} > B$, then,

$\sigma_{k+1} = c : p_1 \rightarrow \perp$ (Rule 6, $\sigma_{k,5}$)

For the above cases, we can complete the proof by:

$\sigma_{k+2} = c : p_1 \rightarrow p_2$ (Rule 1c, σ_{k+1}) ■

5 Interaction with DTDs

In this section, we investigate the interaction between XTPCs and DTDs. In particular, we study the bounded consistency problem for an XML specification consisting of a set of XTPCs and a DTD.

5.1 Conflicts between XTPCs and DTD

In the following, we first illustrate how XTPCs and DTDs together can form an inconsistent specification. Recall that a DTD consists of a set of element definitions, and each element definition has a *content model*, i.e., a regular expression describing the admissible sequences of child elements. Consider the following DTD D (descriptions of elements whose type is PCDATA are omitted).

```
<!ELEMENT sales (order+)>
<!ELEMENT order (item+, buyer, payment)>
<!ELEMENT buyer (name, (phone, drivLic)?, note?)>
<!ELEMENT item (itemID, details?, note?)>
<!ELEMENT details (desc, price)>
<!ELEMENT payment ((creditCard, expDate) | check )>
```

Suppose that we impose the set of XTPCs from Example 3.2 to the above DTD. Note that σ_2 (*/`order`: . $\rightarrow$ .//`item`[`itemID`][`desc`]) requires the element `desc` to be a child of the element `item`. However, the DTD requires `desc` to be a child of `details`, which, in turn, is a child of `item`. Clearly, there is a conflict between the XTPCs and the above DTD. No XML document can satisfy both the DTD and the set of XTPCs. The consistency problem thus is stated as follows.

Definition 5.1 *(Bounded Consistency for XTPCs and DTD)*
Given a depth bound $B > 0$. An XML specification $\chi = \{\Sigma, D\}$ consisting of a DTD D and a set of XTPCs Σ is *consistent* iff there exists an XML tree t with $depth(t) \leq B$ such that $t \models \Sigma$ and t conforms to D. □

5.2 Bounded Consistency for |-Free DTD and Wildcard-Free XTPCs

Before discussing the general bounded consistency problem for XTPCs and DTD, we first focus on a simpler case, which assumes that the given DTD has no alternation operator (|) and the XTPCs have no wildcard. In the following, *|-free content models* refer to content models that do not contain any alternation operator, and *|-free DTDs* refer to DTDs that contain only |-free content models (see also [14]). Furthermore, *wildcard-free XTPCs* are XTPCs with no wildcard element "*" and descendant axis, and *wildcard-free XTPC specifications* are sets of XTPCs containing only wildcard-free XTPCs. For a given |-free content model m, let $L(m)$ be the set of element sequences defined by m, and $min(m)$ to be the shortest element sequence in $L(m)$, which is computed by listing all the elements in m that are not in a *- or ?-group, from the left to the right. We now introduce the notion of *minTree*.

Definition 5.2 *(minTree)*
Let D be an |-free DTD. The *minTree* of D, denote $minTree(D)$, is an XML tree such that
(1) the root of $minTree(D)$) is the document type defined by D, and
(2) for each $e \in minTree(D)$, $\min(m_e)$ is the sequence of child elements of e, where m_e is the content model of e. □

Intuitively, the minTree of a |-free DTD D represents the "minimal document structure" required for any document conforming to D. In the following, we use $m(\tau)$ to denote the canonical model of τ, i.e., the XML tree that has the same shape and node labels as τ. Note that $m(\tau)$ is unique because of the absence of wildcards in τ. The bounded consistency of a wildcard-free XTPC specification Σ and a |-free DTD D can be determined with the algorithm described below.

Algorithm 5.3 Bounded Consistency of wildcard-free XTPC specification Σ and |-free DTD D
Input: $\chi = \{\Sigma, D\}$, B
Output: return *true* iff $\chi = \{\Sigma, D\}$ is consistent

let $\tau := minTree(D)$
repeat until $depth(\tau) > B$ or $\tau = \perp$
  $\tau := ICP_{.,\Sigma}(\tau)$
  if $\tau \neq \perp$ and $depth(\tau) \leq B$ and $m(\tau)$ conforms to D
  then return *true*
  extend τ s.t. $m(\tau)$ conforms to D or
  return *false* **if** no such extension exists
return *false*

Algorithm Details. In the algorithm, $ICP_{.,\Sigma}(minTree(D))$ represents the "minimal pattern" required by Σ in the presence of $minTree(D)$. If this pattern is non-empty and within the depth bound, and $m(\tau)$ conforms to D, then $m(\tau)$ satisfies both Σ and D and therefore, χ is consistent. Otherwise, we extend the pattern τ such that $m(\tau)$ conforms to D, and recompute the ICP using the extended pattern. If no such extension exists, then χ is inconsistent. The basic idea of the algorithm is to attempt to construct an XML tree that satisfies χ by inserting elements required by both D and Σ to τ in an iterative manner. It should be noted that the algorithm is non-deterministic, because there is usually more than one way to extend τ such that $m(\tau)$ conforms to D. Although in a content model the symbol * ($^+$) is used to denote more than zero (one) occurrences of an element grouping, in order to limit the number of extensions to τ, we do not consider child sequences containing element groupings with multiple occurrences.

Example 5.4 Let D' be a DTD that has the same set of element definitions as the DTD D given in Section 5.1, except that the element `payment` is defined as follows.

```
<!ELEMENT payment (check)>
```

Let $\Sigma = \{$ `sales/order`: `payment/check` $\rightarrow$ `buyer/drivLic` $\}$. The minTree of D' is shown in Fig. 5(a). The ICP $\tau = ICP_{.,\Sigma}(minTree(D'))$ is shown in Fig. 5(b). Because $m(\tau)$ does not conform to D', we extend τ to a form that conforms to D', as shown in (c). Since the extended pattern corresponds to an XML tree that satisfies both Σ and D', the specification D' and Σ is consistent.

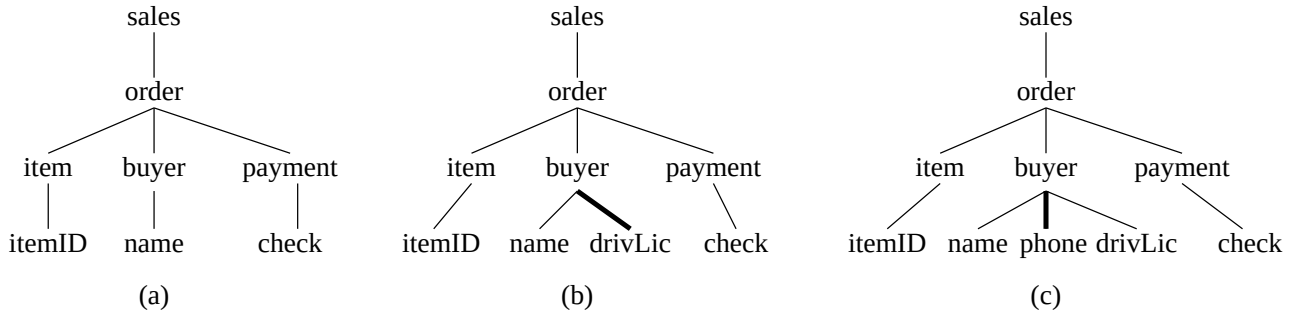

Figure 5: (a) The minTree of D'. (b) The minimum document structure required by Σ in the presence of $minTree(D')$ ($\tau = ICP_{.,\Sigma}(minTree(D'))$). (c) Extension of τ with a structure that conforms to D'.

5.3 Bounded Consistency for DTDs and XTPCs

In our approach, the bounded consistency of a DTD D and a set of XTPCs Σ is determined by first splitting D into a set of |-free DTDs $\{D_1, \cdots, D_k\}$ and expanding Σ into a set of wildcard-free XTPC

specifications $\{\Sigma_1, \cdots, \Sigma_m\}$. Then, we apply Algorithm 5.3 to each $\{\Sigma_i, D_j\}$ pair, where $i = 1, \cdots, k$ and $j = 1, \cdots, m$. We will conclude that $\{D, \Sigma\}$ is consistent if and only if one of the $\{\Sigma_i, D_j\}$ pairs is consistent.

In the following, we first formalize the notion of splitting content models and extend this notion to DTDs.

Definition 5.5 *(Split Content Model)*
The split of a content model m is a set of |-free content models, denoted $split(m)$, that is defined recursively by the following rules:
$split(m*) = \{g* | g \in split(m)\}$,
$split(m+) = \{g+ | g \in split(m)\}$
$split(m?) = \{g? | g \in split(m)\}$
$split((m_1, \cdots, m_n)) = \{(g_1, \cdots, g_n) | g_i \in split(m_i)\}$
$split((m_1 | \cdots | m_n)) = split(m_1) \cup \cdots \cup split(m_n)$
$split(e) = \{e\}$, for an element e
$split(\#\texttt{PCDATA}) = \{\#\texttt{PCDATA}\}$ □

Definition 5.6 *(Split DTD)*
The split of a DTD D, denoted $split(D)$, is a set $\{D_1, D_2, \cdots, D_k\}$ of |-free DTDs, where $D_i = \{\{m'_1, \cdots, m'_k\} | m'_i \in split(m_i)\}$ for $i = 1, 2, \cdots, k$. □

Example 5.7 The split of the example DTD D given in Sec. 5.1 consists of two |-free DTDs, D_1 and D_2, which share the same set of element definitions with the exception of the element `payment`. The definitions for the element `payment` for D_1 and D_2 are
`<!ELEMENT payment (creditCard, expDate)>` and
`<!ELEMENT payment (check)>`,
respectively.

The following lemma states that a DTD D is consistent iff at least one of the |-free DTD in $split(D)$ is consistent. The proof is trivial and is thus omitted.

Lemma 5.1 For any XML tree t and DTD D, t conforms to D iff there exists $D_i \in split(D)$ such that t conforms to D_i. ■

Next, we describe how the path information given in a DTD can be used to expand all the wildcards (including * and descendant axis) that occur in an XTPC specification. Given a DTD D and an XP expression p. Let P_D be the set of admissible paths defined by D. For a given XP expression p, we use $expand_D(p)$ to denote the set of wildcard-free paths that match p, given that p is also admissible in documents conforming to D, i.e., $expand_D(p) = \{p' \in P_D | p' \subseteq p\}$. We extend such notion to an XTPC (specification) as follows.

Definition 5.8 *(expand XTPC)*
Let D be a DTD. The wildcard expansion of an XTPC σ, denoted $expand_D(\sigma)$, is a set of XTPC specifications, $\{\Gamma_1, \cdots, \Gamma_k\}$ such that $\Gamma_i = \{c' : p'_1 \rightarrow p_{2i} | c' \in expand_D(c) \wedge p'_1 \in expand_D(p_1)\}$ and $expand_D(p_2) = \{p_{21}, \cdots, p_{2n}\}$. □

Definition 5.9 *(expand XTPC specification)*
Let D be a DTD. The wildcard expansion of an XTPC specification $\Sigma = \{\sigma_1, \cdots, \sigma_r\}$, denoted $expand_D(\Sigma)$,

is a set of XTPC specifications $\{\Sigma_1, \cdots, \Sigma_m\}$, where $\Sigma_i = \{\Gamma_1 \cup \cdots \cup \Gamma_r \mid \Gamma_j \in expand_D(\sigma_j)\}$ for $i = 1, \cdots, m$. □

Example 5.10 Let D be the example DTD, $\sigma_1 = .\!/\!/\texttt{details} : \texttt{price} \rightarrow \texttt{desc}$ and $\sigma_2 = * : . \rightarrow .\!/\!/\texttt{note}$. Then the wildcard expansions of σ_1 and σ_2 are

$expand_D(\sigma_1)$={{sales/order/item/details: price → desc}}, and
$expand_D(\sigma_2)$={{sales:. → order/buyer/note}, {sales: . → order/item/note}}.

The wildcard expansion of $\Sigma = \{\sigma_1, \sigma_2\}$, $expand_D(\Sigma)$, is thus

{{ sales/order/item/details: price → desc, sales:. → order/buyer/note},
{ sales/order/item/details: price → desc, sales: . → order/item/note}}.

The following lemma shows that an XTPC specification Σ is consistent if and only if one of its wildcard expansions is consistent.

Lemma 5.2 Let Σ be a set of XTPCs and D be a DTD. For any XML document t that conforms to D, $t \models \Sigma$ iff there exists $\Sigma_i \in expand_D(\Sigma)$ such that $t \models \Sigma_i$.

Proof: ($\Leftarrow$) Assume that t conforms to D and $t \models \Sigma_i$. Let $\sigma \in \Sigma$ and $\sigma = c : p_1 \rightarrow p_2$. Suppose that $\Gamma \in expand_D(\sigma)$ is a subset of Σ_i. Since $t \models \Sigma_i$, we have $t \models \Gamma$, where $\Gamma = \bigcup_{i=1}^{k} c_i : p_{1i} \rightarrow p'_2$, $expand_D(c) = \{c_1, \cdots, c_k\}$, $expand_D(p_1) = \{p_{11}, \cdots, p_{1k}\}$, and $p'_2 \in expand_D(p_2)$. For $n \in c[p_1](t)$, since t conforms to D, it must be true that $n \in c_i[p_{1i}](t)$ for some $i = 1, \cdots, k$. It follows that $n \in c_i[p_2](t)$, and therefore, σ is satisfied. ($\Rightarrow$) Assume t conforms to D and $t \models \Sigma$. Let $\sigma \in \Sigma$ where $\sigma = c : p_1 \rightarrow p_2$. Since every path in t must be in P_D, there exists $\Gamma_j \in expand_D(\sigma)$ such that $t \models \Gamma_j$. Note that $expand_D(\Sigma)$ consists of all different combinations of XTPC rules from $expand_D(\sigma)$, $\sigma \in \Sigma$, there must be a combination Σ_i that is satisfied by t. ■

The bounded consistency problem for an XML specification can now be summarized as follows.

Theorem 5.11 For a given XML specification χ consisting of a DTD D and a set of XTPCs Σ, χ is consistent iff there exists $\Sigma_i \in expand_D(\Sigma)$ and $D_j \in split(D)$ such that $\chi' = \{\Sigma_i, D_j\}$ is consistent.

Proof: The theorem follows directly from Lemma 5.1 and Lemma 5.2. ■

6 Conclusions and Future Work

Comprehensive and practically useful specification frameworks for XML documents play an important role to better manage, query, and process XML documents. We have presented the concepts underlying tree pattern constraints (XTPCs) for XML documents as a pattern-based schema formalism founded on XPath. Using results on query containments, we have shown that the bounded implication and consistency problems for XTPC specifications are decidable. We have also outlined consistency issues that arise if XTPCs are not used as stand-alone schema formalism but in the context of DTDs.

Our ongoing research extends the framework presented in this paper in two directions. First, we are investigating the addition of value conditions to XTPCs. Such additions would allow the specification of value-structure relationships, e.g., if in an XML document some nodes in a pattern have some given values, then this document must (not) exhibit another pattern, or values in this pattern. The second direction is to complete the study of XTPCs in the presence of other XML formalisms, including XML Schema and functional dependencies.

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

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

[4] P. Buneman, S. Davidson, W. Fan, C. Hara, W. Tan: Reasoning about Keys for XML. *Information Systems*, 28(8), 1037–1063, 2003.

[5] M.Benedikt, W.Fan, G.M.Kuper: Structural Properties of XPath Fragments. In *Database Theory (ICDT 2003)*, LNCS 2572, Springer, 79–95, 2003.

[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] G.Bex, S.Maneth, F.Neven: A Formal Model for an Expressive Fragment of XSLT. In *Information Systems* 27(1):21–39, 2002.

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

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

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

[11] W. Fan, J. Simeon: Integrity Constraints for XML. In *Journal of Computer and System Sciences*, 66(1):254–291, 2003.

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

[13] M.L. Lee, T.W. Ling, W.L. Low: Designing Functional Dependencies for XML In *Advances in Database Technology, EDBT 2002*, Springer, LNCS 2287, 124–141, 2002.

[14] S.Lu, Y.Sun, M.Atay, F.Fotouhi: A Sufficient and Necessary Condition for the Consistency of XML DTDs. In *1st Intl. Workshop on XML Schema and Data Management*, Springer, LNCS 2814, 2003.

[15] G. Miklau, D. Suciu: Containment and Equivalence of Tree Patterns. In *21th ACM Symposium on Principles of Database Systems*, 65–76, 2002.

[16] F. Neven, T. Schwentick: XPath Containment in the Presence of Disjunction, DTDs, and variables. In *Database Theory (ICDT 2003)*, LNCS 2572, Springer, 315–329, 2003.

[17] D.Olteanu, H.Meuss, T.Furche, F.Bry: XPath: Looking Forward. In *XML-Based Data Management and Multimedia Engineering, EDBT Workshop*, LNCS 2490, 109-127, 2002.

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

[19] XML Path Language (XPath) 2.0. W3C Working Draft, www.w3c.org/TR/xpath20, Nov 2003.