Reasoning about XML Update Constraints*

Bogdan Cautis
INRIA Futurs & U. Paris Sud
bogdan.cautis@inria.fr

Serge Abiteboul
INRIA Futurs & U. Paris Sud
serge.abiteboul@inria.fr

Tova Milo
Tel Aviv U.
milo@cs.tau.ac.il

ABSTRACT

We introduce in this paper a class of constraints for describing how an XML document can evolve, namely *XML update constraints*. For these constraints, we study the implication problem, giving algorithms and complexity results for constraints of varying expressive power. Besides classical constraint implication, we also consider an instance-based approach. More precisely, we study implication with respect to a current tree instance, resulting from a series of unknown updates. The main motivation of our work is reasoning about data integrity under update restrictions in contexts where owners may lose control over their data, such as in publishing or exchange.

Categories and Subject Descriptors

H.2.3 [**Database management**]: Languages; H.2.0 [**Database management**]: General—*Security, integrity, and protection*

General Terms

Algorithms, Languages, Theory

Keywords

Semi-structured data, XML, update constraints, implication, data integrity.

1. INTRODUCTION

Restricting the ways in which data is modified and transformed is often a necessity and the basis for reasoning about data validity. When data is under centralized control, arbitrarily complex update constraints can be actively enforced inside the boundaries of the data owner, who can monitor changes. But in distributed, loose environments, for instance when data is published or exchanged, it becomes much harder to control updates. The enforcement of update restrictions can be passively achieved via cryptographic techniques, and by consequence only simpler update restrictions can be imposed. Dealing with simpler update limitations has, however, an advantage, since it allows users to do more reasoning about data properties, beyond "no illegal update occurred", understanding what could have happened and how.

To illustrate, consider an XML document that is exchanged between three parties, Source, Broker and User (Figure 1). Assume Broker is allowed to modify data he receives from Source, but only in a controlled manner. For instance, advertisements may be introduced but only in certain well-defined areas. Also, some information may be filtered out, but again in well-defined areas. For instance, Broker may be allowed to remove a private phone number but not to replace it by another one. In particular, the rules of the game should be precise enough so that (a) the Source can choose the right restrictions on how data can be modified and can specify them in a clear way, and (b) based on the given update restrictions and the data to which they apply, the User has the means to decide on the validity of the data that interests her.

This paper introduces a constraint model that allows data owners to specify restrictions on allowed updates for XML trees. Starting from this model, we focus on inference techniques that help data owners choose the right restrictions and help users reason about the integrity properties of data. In short, the constraints we consider allow stating that a set of selected XML nodes representing the result to some path query should always grow, or shrink, or should not change at all. Then we study two inference problems, namely, the *constraint implication problem* and the *instance-based implication problem*.

The *constraint implication problem* is defined as follows. Given a set of update constraint $\mathcal{C}$ and a constraint c, is it true that each pair (I, J) of consecutive data instances (i.e., a tree instance before and after updates) satisfying $\mathcal{C}$ also satisfies c?

In the *instance-based implication problem*, J is known.

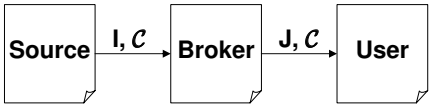

Figure 1: Exchange with update constraints ($\mathcal{C}$).

*This work has been partially supported by the ANR grant DocFlow, the EC project EDOS and the Israel Science Foundation.

PODS'07, June 11–14, 2007, Beijing, China.

The problem becomes: is it true that for each I such that (I, J) satisfies $\mathcal{C}$, (I, J) also satisfies c. Thus this may be viewed as a question over the past. For instance, knowing that $\mathcal{C}$ is enforced, can we derive from the instance we received that no new product has been inserted. A symmetrical problem (not studied here) is obtained by giving I instead of J and questioning the future. Instance-based implication may be viewed as a foray into the more general problem of temporal queries under update constraints.

As our constraint language closely captures what cryptographic techniques can support, we believe that such a simple model, that talks only about increasing or decreasing sets of XML nodes, is best suited to express update restrictions on data with limited owner control and no log or history of updates. Hence, it has not only theoretical but also practical value, as it can be effectively enforced in non-centralized environments. However, the focus of this paper is not on the actual enforcement, but on reasoning about integrity properties of data under update constraints. We only remind here that although classic signing techniques prevent any kind of modification on signed data, more flexible approaches have been provided lately [26, 15, 1, 8, 21], in which some restricted modifications may still occur, without causing the invalidation of the data. For example, by digital signatures, one can impose that a certain collection of items can only increase (or decrease).

To the best of our knowledge, this work is the first to consider update constraints for tree structured data. Perhaps the work that is closest in spirit is the one of Miklau and Suciu [23] which, for relational data, models integrity guarantees of digital signature schemes as embedded dependencies and considers query related issues that can be solved by the relational chase [2]. As we will see, both for constraint implication and instance-based implication, new issues are raised when considering trees. While integrity constraints for XML data have received a lot of attention lately [14, 18, 11], we study here update constraints, defined in terms of XPath expressions, which talk about how a document can be changed. Nevertheless, our work gives also new insight into XML integrity constraints in general.

The paper is organized as follows. In Section 2 we define the constraint language and the implication problems studied in the paper. In Section 3 the two implication problems are related to previous works on query containment and constraints for XML data. Constraint implication is studied in Section 4 and instance-based implication in Section 5. We briefly discuss a model extension, namely relative constraints, in Section 6. In Section 7 we discuss other related works and we conclude. For space reasons, we only give the intuition of proofs.

2. XML AND UPDATE CONSTRAINTS

Given two infinite domains, the domain of node identifiers ($\mathcal{N}$) and the domain of labels ($\mathcal{L}$), we define XML trees as follows.

DEFINITION 2.1. *An (unordered)* data tree *is an expression (T, λ), where $T = (N, E)$ is a finite unordered tree, with set of nodes $N \subset \mathcal{N}$, directed edges $E \subset N \times N$, and $\lambda : N \to \mathcal{L}$ is a labeling function over nodes.*

By the above definition, we intend to capture XML data which, besides labels ($\mathcal{L}$), have unique node Ids ($\mathcal{N}$). Hence a node is a pair in $\mathcal{N} \times \mathcal{L}$ and from here on, when we speak of an individual node, we mean such a pair[1].

In the specification of update constraints, we rely on XPath queries from the fragment $XP\{/, [], //, *\}$, generally referred to as *unary tree pattern queries*. More precisely, the XPath expressions used in this paper are generated by the following grammar:

$$\begin{aligned} path &::= /step \mid //step \mid path\ path \\ step &::= label\ pred \\ pred &::= \epsilon \mid [path] \mid pred\ pred \\ label &::= L \mid * \end{aligned}$$

By / we denote *child axis* navigation. By // we denote *descendant axis* navigation. L denotes labels and $*$ is the wildcard label. The path inside brackets is called a *predicate*. Queries have one distinguished *output* node. For example, the `b` node is the output node of the query `/a//b[/c]`. Notice that the root of the document is treated differently from other nodes, as predicates cannot be defined on the root. The reason for this is that we are mainly interested in queries that test properties of individual nodes, and not of entire documents.

The semantics of such queries is defined in the standard way (see, for instance, [29, 9]). For example, the previous query returns the set of `b` nodes having both a `c` child and an `a` ancestor which is child of the document root. For some node n in a data tree I and path query q, $q(n, I)$ denotes the result of q evaluated on the subtree of I rooted at n. We write $q(I)$ for $q(root, I)$. We stress that the result of a query is set of pairs $(Id, label)$. The evaluation of $XP\{/, [], //, *\}$ queries can be done in polynomial time [19].

By *concrete path*, we denote a path that has the output node labeled by a concrete label (not wildcard). In order to simplify the presentation, we will only discuss in this paper concrete XPath queries. However, all the results can be extended and reformulated to deal also with non-concrete paths.

Throughout this paper, we use of the notions of query *containment* (denoted by $\subseteq$) and *equivalence* (denoted by $\equiv$), both defined in a standard way. We also consider the *intersection* of queries (denoted by $\cap$). Informally, we say that $q \equiv q_1 \cap \cdots \cap q_k$ if for all instances I, we have $q(I) \equiv q_1(I) \cap \cdots \cap q_k(I)$.

As in [28], an *update* on an XML tree is defined as a sequence of node insertions deletions, moving and modifications of labels. In this paper, we will simply abstract an update as a pair of data trees (I, J), where I (resp. J) is the before (resp. after) update tree.

We next define *update constraints*.

DEFINITION 2.2 (SYNTAX). *An* XML update constraint *is an expression (q, σ), where q is an XPath query called the* range *and σ is the* constraint type*, i.e., one of* no-insert *(in short $\downarrow$) or* no-remove *(in short $\uparrow$) .*

DEFINITION 2.3 (SEMANTICS). *We say that a pair of trees (I, J) is* valid *with respect to some constraint $c = (q, \sigma)$ (denoted $(I, J) \vdash c$) if we have $q(I) \subseteq q(J)$ (resp. $q(J) \subseteq q(I)$) when σ is* no-remove *(resp.* no-insert*).*

[1]Other aspects of the XML data model [30] such as data values (text content) or attribute values can be considered as being part of the node label.

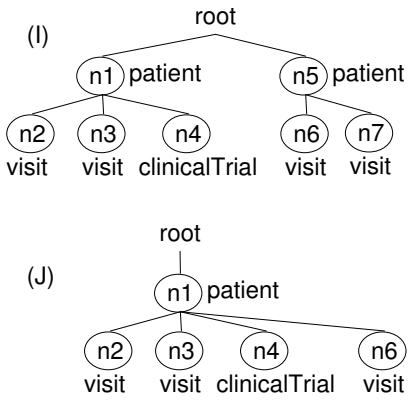

Figure 2: Pair of instances (I, J).

A pair (I, J) is *valid* for a set of constraints if it is valid for each of them. Note that we can express *immutability* restrictions by simply pairing no-remove and no-insert conditions. As a shorthand, we will use $(q, \top)$ to denote such a pair of no-remove and no-insert constraints.

EXAMPLE 2.1. *For example, consider the (I,J) instances of Figure 2 and the following constraints:*

- $c_1 = (/\texttt{patient}[/\texttt{visit}], \downarrow)$
- $c_2 = (/\texttt{patient}[/\texttt{clinicalTrial}], \top)$
- $c_3 = (/\texttt{patient}/\texttt{visit}, \uparrow)$,

The first constraint states that the set of `patients` *having a* `visit` *can only shrink. The second one says that the set of* `patients` *having a* `clinicalTrial` *cannot change at all, while the third one says that the overall set of* `visits` *can only grow. The pair of instances (I, J) in Figure 2 is valid for c_1 and c_2 but not for c_3. This is because the* `visit` *node n_7 has been deleted.*

We will briefly consider one extension to this constraints, namely *relative update constraints*, in Section 6. These are constraints that specify update restrictions relative to some *scope*, e.g., a particular constraint should hold for each `patient` in the medical document that is sent. Other possible extensions and directions for future work are discussed in Section 7.

Finally, note that in our model constraints are always consistent and, in particular, a pair (I, I) of identical instances is valid for any set of update constraints. This would no longer be the case if we consider arbitrary inclusions such as $(q_1(I) \subseteq q_2(J))$, that would actually "force" modifications to happen.

2.1 Implication problems

We are now ready to formally define the problems we study. First, we consider the implication problem for update constraints:

DEFINITION 2.4 (GENERAL IMPLICATION). *Given a set of update constraints $\mathcal{C}$ and an update constraint c, we say that $\mathcal{C}$ implies c (denoted $\mathcal{C} \models c$) if for any pair of tree instances I, J, we have $(I, J) \vdash \mathcal{C} \Rightarrow (I, J) \vdash c$.*

In Example 2.1, the constraint

$$c = (/\texttt{patient}[/\texttt{visit}][/\texttt{clinicalTrial}], \downarrow)$$

is implied by $\{c_1, c_2\}$. We briefly explain why: *in order to violate c, by adding in some instance I a node in the result of* `/patient[/visit][/clinicalTrial]`*, one should either (a) add some new* `patient` *node, along with* `visit` *and* `clinicalTrial` *children (but this would violate c_1 and c_2), or (b) just add some* `visit` *and / or* `clinicalTrial` *children to a node that did not qualify for at least one of these predicates before (but this would again violate at least one of c_1 or c_2).*

We also consider implication when the current tree instance (J), to which previous updates lead, is available. The corresponding implication problem is called *instance-based implication*.

DEFINITION 2.5 (INSTANCE-BASED IMPLICATION). *Given a set of update constraints $\mathcal{C}$, an instance J and a constraint c, we say that $\mathcal{C}$ implies c for J (denoted $\mathcal{C} \models_J c$) if for any tree instance I, we have $(I, J) \vdash \mathcal{C} \Rightarrow (I, J) \vdash c$.*

Considering the J instance and constraints of Example 2.1, the constraint

$$c = (/\texttt{patient}[/\texttt{clinicalTrial}]/\texttt{visit}, \uparrow)$$

is implied by $\{c_3\}$ and J. Let us consider what could have happened on an initial I instance in order to violate c: *one could have completely removed from I a* `visit` *of some* `patient` *with* `clinicalTrial` *(this would violate c_3); and one could have moved a* `visit` *of a* `patient` *with* `clinicalTrial` *below another patient without* `clinicalTrial` *(this is not possible because there is no such* `patient` *in the current instance, J). Observe that c would not be implied by c_3 alone, i.e., for any pairs of instances.*

The two implication problems capture different scenarios for data integrity. General implication is relevant in situations of data exchange or publishing when a publisher wants to decide a priori, regardless of the published data, what update restrictions should be imposed. Instance-based implication is relevant when someone obtains a document with update constraints and wants to understand the integrity properties of this data. It is easy to observe that the general constraint implication implies the instance-based one.

These problems abstract more practical ones such as deciding if some update can be safely performed or understanding the integrity properties of a query result. Instance-based implication can be viewed as verifying properties of the past, i.e., questioning the past evolution. Similarly, we could consider inference when the I instance is given, i.e., questioning the future evolution. The problem becomes somewhat analogous to the one we consider here, in some sense the symmetric in the future of the problem we consider about the past. Besides instance-based implication, other validity questions may be relevant when we consider data. For example, we could simply ask if a certain node could have been added to the result of a query, or could have been removed. Solutions to the above implication problem represent a first step towards inferring such richer, fine-grained assertions. A detail study of such aspects is left for future work.

On sequences of instances Observe that the current definition of validity and implication considers only pairs of instances and can only capture contexts of exchange among

three parties (such as the one of Figure 1). However, in many other scenarios, it is also worth considering sequences of instances.

Let us consider a sequence $(I_0, \ldots, I_k)$. We call it *pairwise valid* if each of its pairs (I_i, I_j), $i < j$, are valid according to Definition 2.3. When the last instance is fixed, we can give a more data-oriented definition of validity, one that only takes into account this last instance: we say $(I_0, \ldots, I_k)$ is *valid for* I_k if the pair (I_0, I_k) is valid. Then, implication (in both flavors) for sequences would be defined based on the corresponding notions of validity. All the results presented in this paper, referring to pairs of instances, remain valid also for sequences. This becomes immediate from the definitions of *pairwise validity* and *validity for* I_k. So, without any loss of generality, in the rest of the paper we will only refer to pairs of instances.

Notation We consider during our analysis various sub-fragments of $XP\{/, [], //, *\}$. They will be represented by the navigational primitives that are allowed. For instance, by $XP\{/, //\}$ we denote paths without predicates and wildcards. To denote implication in restricted contexts, we use the notation $\models_{\sigma}^{X}$, where X is the XPath fragment we assume and $\sigma \in \{\uparrow, \downarrow\}$. For example, $\models_{\uparrow}^{XP\{/,[]\}}$ denotes the constraint implication problem for no-remove constraints expressed in $XP\{/, []\}$. Similarly, we use $\models_{J,\sigma}^{X}$ for instance-base implication in restricted contexts.

3. RELATED PROBLEMS

To further clarify the implication problems outlined in the previous section, we first consider some initial results relating them to previous work on topics such as the equivalence of path queries, regular XML keys and XML Integrity Constraints (XICs).

Query equivalence Since we rely on XPath queries to express update constraints, it comes as no surprise that XPath query containment and equivalence [27] are tightly related to our implication problems. Regarding the relationship between query equivalence and implication, we can prove the following:

THEOREM 3.1. *Given two constraints* $c_1 = (q_1, \sigma)$ *and* $c_2 = (q_2, \sigma)$*, expressed in* $XP\{/, [], //, *\}$*, we have* $c_1 \models c_2$ *(and symmetrically* $c_2 \models c_1$*) iff* $q_1 \equiv q_2$*.*

Proof: [Sketch] Without loss of generality we assume that σ is $\uparrow$; the opposite type is analogous by symmetry. From the definition of implication, one directions is obvious ($c_1 \models c_2 \Leftarrow q_1 \equiv q_2$). Now suppose $c_1 \models c_2$. Suppose also that $q_2 \not\subseteq q_1$, then there exists some tree I and node n such that $n \in q_2(I)$ and $n \notin q_1(I)$. We can easily obtain a contradiction to $c_1 \models c_2$ by the pair instances $(I, I[n \to n'])$, where by $I[n \to n']$ we denote the instance obtained by replacing n with a new node n' with the same label.

Regarding the other containment, if we assume $q_1 \not\subseteq q_2$, then there exists some tree T' and node n' such that $n' \in q_1(T')$ and $n' \notin q_2(T')$. We can then obtain a contradiction to $c_1 \models c_2$ by a transformation as the one illustrated in Figure 3. In this figure, T is any tree with some node n in $q_2(T)$. By putting together T and T' (i.e., merging their root nodes), the presence of n and n' in range queries is not affected in any way. In the transformation $I \to J$ we simply interchange n and n'. Since n and n' have the same label, by this transformation, we remove the node n from the result of q_2, without removing anything from the result of q_1. ■

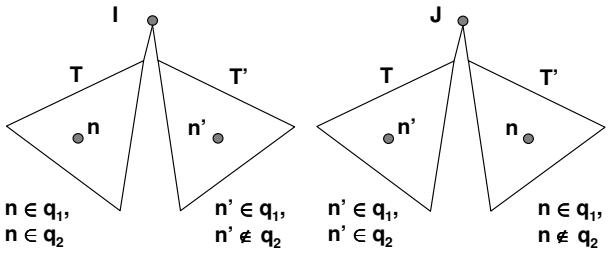

Figure 3: Counterexample pair of instances (I, J)**.**

A similar result can be proven for instance-based implication. So, in general, the implication problem, in both flavors, is at least as hard as query equivalence and containment[2].

To illustrate further the relationship of constraint implication with query containment and equivalence, let us consider a restricted setting in which all constraints have the same type (say $\uparrow$). As a sufficient approach for testing implication, we can state the following proposition, which follows from the definition of implication.

PROPOSITION 3.1. *A constraint* $(q, \uparrow)$ *is implied by a a set of* $\uparrow$ *constraints* $\mathcal{C}$ *if there exist some constraints* $c_1, ..., c_k$ *in* $\mathcal{C}$*, with their respective range queries* $q_1, \ldots, q_k$*, s.t.* $q \equiv q_1 \cap ... \cap q_k$*.*

We will see further that, in some restricted cases, this condition of equivalence is also necessary (but only in restricted cases).

Regular XML key constraints Previous works addressed the implication problem for inclusion dependencies on semi-structured data [11, 4, 18] and XML keys/foreign keys [7, 17], taking into account also schema information (such as DTDs). In particular, *regular XML keys* [7] have the form $\beta.\tau[X] \to \beta.\tau$, for β being a regular expression over schema types and wildcard, τ being a schema type and X being a set of node attributes[3]. The interpretation is that the tuple of X attributes represents a key for the path β, i.e., they uniquely determine nodes which are found on the path β. Similarly, a *regular foreign key* has the form $\beta_1.\tau_1[X] \subseteq \beta_2.\tau_2[Y]$, for X and Y being sequences of attributes of the same cardinality. Such constraints are said to be *unary* if they only refer to one attribute.

Given a set of key and foreign key regular constraints and a Document Type Definition (DTD) ([30]), the *consistency problem* (defined in [7]) is asking whether there exists an XML document that conforms both to the DTD and the regular key constraints. (The exact definitions can be found in [17, 7].)

Although this constraint formalism is generally not comparable to our XPath-based formalism, regular key constraints can be used to express some of our update constraints. More precisely, they can express constraints described by only linear paths (i.e., no use of predicates). We

[2]Query equivalence and containment were shown to be coNP-hard in [22] (see also [27]) for $XP\{/, [], //, *\}$; the proof of this lower bound uses only concrete paths.

[3]Attributes can be seen as nodes that are uniquely identified by their label for each parent node.

can see node identifiers as being the only node attribute and pairs (I, J) as being the two main branches of a document. We need an unary key constraint to enforce uniqueness for node identifiers and one unary foreign key constraint for each update constraint. In fact, we show in Section 4 how constraint implication can be reduced to consistency in the presence of DTDs and unary regular constraints, even for queries with predicates.

XML Integrity Constraints (XICs) This is probably the richest formalism that has been proposed for expressing integrity constraints for XML data in terms of XPath expressions [14]. An XIC is defined as follows:

$$\forall\ x_i, \ldots, x_n\ \ A(x_1, \ldots, x_n) \rightarrow$$

$$\exists\ y_1, \ldots, \exists\ y_m\ \ B(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

where A, B are conjunctions of atoms $u = v$ or $u\ p\ v$, with p being a path step (such as $/label$, $//label$ or $/@attribute$) evaluated at u which returns v.

While the implication problem for XICs has not been fully explored yet, implication under some limitations and query containment in the presence of such constraints have been considered in [14]. In general, implication of XICs was shown to be undecidable. It becomes decidable for a tighter class, namely *bounded* XICs, which are XICs for which usage of // and attributes are not allowed under existential quantifiers, and the technique used is a classical inference technique, the *chase* [2].

First, our update constraints can be fully expressed by XICs, even in the instance-based setting. For that, it suffices to see a pair (I, J) of instances as a virtual document divided into two main branches, I and J. We just have to use an Id attribute, with no two nodes under the same main branch (I or J) having the same Id value.

Unfortunately, the XICs needed to capture update constraints are *not bounded* because of both // axis and the Id attribute (it is existentially quantified). Indeed, we can exhibit very simple examples of constraint implication where the chase technique fails to terminate. So our contribution is also to show decidability and give complexity bounds for constraint implication in a family of unbounded XICs.

4. CONSTRAINT IMPLICATION

In this section we study *constraint implication*. *Instance-based implication* is the topic of the next section. We first give some intuition on how constraints of opposite types may interact (Section 4.1). Then, in Section 4.2, we study the complexity of the constraint implication problem. We first show that constraint implication is decidable, but with high complexity (NEXPTIME upper bound), although the tightness of this bound remains open. We then study complexity when restricting the expressivity of constraints. More precisely, there are two directions in which one can restrict constraints: (a) restrictions on the XPath fragment used, and (b) restrictions on the update types (for instance, only $\downarrow$ or only $\uparrow$), and we will consider both. The results of this section are summarized in Table 1.

4.1 Interacting types

As we will see, the interaction between *no-insert* and *no-remove* constraints is surprising. For each such constraint type σ ($\downarrow$ or $\uparrow$), and each set $\mathcal{C}$ of constraints, let $\mathcal{C}^\sigma$ denote the constraints in $\mathcal{C}$ of type σ. A property that may seem rather intuitive is the following:

[Same-type property] *For a constraint c of type σ and any set of constraints $\mathcal{C}$, we have $\mathcal{C} \models c$ iff $\mathcal{C}^\sigma \models c$.*

It turns out that this is not true in general, as can be witnessed in the following example.

EXAMPLE 4.1. *The following constraints:*

- (c_1) : $(//\texttt{a}//\texttt{c}, \uparrow)$
- (c_2) : $(//\texttt{b}//\texttt{c}, \uparrow)$
- (c_3) : $(//\texttt{a}//\texttt{b}//\texttt{c}, \downarrow)$
- (c_4) : $(//\texttt{a}//\texttt{b}//\texttt{a}//\texttt{c}, \uparrow)$
- (c_5) : $(//\texttt{b}//\texttt{a}//\texttt{b}//\texttt{c}, \uparrow)$

imply $c = (//\texttt{b}//\texttt{a}//\texttt{c}, \uparrow)$, while the no-remove constraints alone do not. For space reasons, the detailed explanation of this example is omitted.

So, when looking at the implication of constraints we need to take into account both update types. A restricted case where we can indeed limit to one type only is when we disallow the // axis. We can prove the following.

THEOREM 4.1. *For any constraint c of type σ and any set of constraints $\mathcal{C}$, all expressed in $XP\{/, [], *\}$, we have $\mathcal{C} \models c$ iff $\mathcal{C}^\sigma \models c$.*

The intuition behind the proof of this result is that we can exhibit a pair of instances that witnesses non-implication using some tree transformations.

4.2 Complexity of constraint implication

We first show that constraint implication is decidable in NEXPTIME. A coNP lower bound follows immediately from the fact that constraint implication is at least as hard as query equivalence (Theorem 3.1).

THEOREM 4.2. $\models^{XP\{/,[],//,*\}}$ *is in* NEXPTIME *and* coNP*-hard.*

Proof: [Sketch] We solve the constraint implication problem by reducing it to the consistency problem for unary regular constraints and DTDs (described in Section 3). The main difficulty is to take into account predicates, which are not handled by such constraints. The crux is to transform normal labeled trees into *annotated* trees, where the label of a node describes precisely the predicates that can be matched below that node. (The predicates to be considered are only those occurring in constraints.)

In this way, instead of evaluating tree patterns with predicates on the normal tree, one can evaluate linear paths on the annotated tree and obtain the same result (modulo the modified labels). To avoid that annotations "lie" about the patterns that can be matched at a node, we control the correspondence between a node's annotations and its content by a *specialized DTD* (an extension of DTDs that decouples node labels and types; see, for instance, [25]). Regarding its general structure, this DTD describes trees having 3 main branches, $\{I, J, witness\}$, where the *witness* gives one node that is removed or inserted in order to violate c.

	$XP\{/,[],*\}$	$XP\{/,[],//\}$	$XP\{/,//,*\}$	$XP\{/,[],//,*\}$
only one update type	in PTIME	in coNP	in PTIME$^{(1)}$	coNP-complete
arbitrary update types	in PTIME	in NEXPTIME coNP-hard	in NP$^{(1)}$ coNP-hard	in NEXPTIME coNP-hard

(1): if the number of constraints and the maximal number of wildcards between consecutive //'s are bounded by constants.

Table 1: Upper and lower bounds for the implication of constraints.

We reduce constraint implication to an instance of the XML consistency problem in which: (a) the DTD has exponential size, (b) the number of unary regular constraints is polynomial in the number of unary constraints, and (c) the size of unary regular constraints is exponential but each constraint can still be described by a deterministic finite-state automaton of only exponential size.

The consistency problem was shown to be solvable in 2-NEXPTIME [5]. More precisely, the complexity is (1) non-deterministic doubly-exponential in the number of regular constraints, and (2) only non-deterministic polynomial in the size of the product of their deterministic finite-state automata, and in the size of the DTD.

While this would lead to an 2-NEXPTIME upper-bound, the problem remains in NEXPTIME due to the limited type of inclusions we must handle (i.e., only between two main branches, I and J). ■

Given the high complexity of the above decision procedure, we consider in the following various restrictions on the expressivity of constraints, tracing a fairly tight borderline between tractable and intractable cases.

XPath fragment restrictions We start by restricting the XPath language. First, when predicates are not used, the general upper bound can be refined to NP, under two restrictions:

THEOREM 4.3. $\models^{XP\{/,//,*\}}$ *is in* NP *if the number of constraints and the maximal number of wildcards between two consecutive //'s are bounded by constants.*

Proof: [Sketch] We use the same reduction from the XML consistency problem under unary regular constraints, which is simpler in the absence of predicates. As before, the DTD describes trees having the structure $\{I, J, witness\}$, where the *witness* contains one node that is removed or inserted in order to violate c. Now, our update constraints can be modeled as a deterministic automata of only polynomial size[4]. Again, the upper bound benefits from the limited type of inclusions we consider (i.e., only between I and J), instead of arbitrary ones. ■

Next, moving from an XPath fragment without predicates to one without // axis or wildcard, we can prove that finding an equivalence with some intersection of ranges is not only a sufficient condition (by Theorem 3.1) but also a necessary one. This will translate into a PTIME decision algorithm for $XP\{/,[],*\}$. We remind that we already know that, for this fragment, we can safely limit to only one constraint type, $\downarrow$ or $\uparrow$ (by Theorem 4.1).

THEOREM 4.4. *Given a set of constraints $\mathcal{C}$ all with update type σ and a constraint $c = (q, \sigma)$, all expressed either in $XP\{/,[],*\}$ or $XP\{/,[],//\}$ we have $\mathcal{C} \models c$ iff there exist constraints $c_1, ..., c_k \in \mathcal{C}$ with respective ranges $q_1, \ldots, q_k$, s.t. $q \equiv q_1 \cap ... \cap q_k$.*

Proof: [Sketch] We can assume w.l.o.g. no-remove constraints; the opposite type is analogous by symmetry. One direction (sufficiency) was already discussed (Proposition 3.1). The other direction is proven by induction on the number of no-remove constraints. (We already proved the statement for one no-remove constraint in Theorem 3.1.) ■

Once we know that equivalence is also a necessary condition, a naive decision procedure is to just look for a combination of range queries having their intersection equivalent to the to-be-implied range. We can avoid such an expensive search by taking all the range queries q_i such that $q \subseteq q_i$ (testable in polynomial time [22]), and then test if $q \equiv \cap_i q_i$. While the fragment $XP\{/,[],*\}$ is closed under intersection and the intersection of queries can be computed in linear time, it remains open if the same can done for the fragment $XP\{/,[],//\}$. It can be easily checked that the latter fragment is not closed under intersection.

From Theorems 4.1 and 4.4 we obtain that:

THEOREM 4.5. $\models^{XP\{/,[],*\}}$ *is in* PTIME.

Finally, regarding lower-bounds for the discussed XPath fragments, we can show that implication remains coNP-hard when predicates or wildcard are combined with descendant axis.

THEOREM 4.6. $\models^{XP\{/,[],//\}}$ *and* $\models^{XP\{/,//,*\}}$ *are* coNP-*hard.*

We reduce from the unsatisfiability of 3SAT formulas.

One type restrictions We next consider restricting the type of constraints, more precisely assuming that all constraints have one same type. Observe that the problems $\models_\downarrow$ and $\models_\uparrow$ are equivalent by symmetry, so in general only one of them will be discussed (we denote this by the generic notation $\models_\sigma$). We first show that under the one-type restriction, constraint implication becomes solvable in coNP time.

THEOREM 4.7. $\models_\sigma^{XP\{/,[],//,*\}}$ *is in* coNP.

[4]The deterministic finite-state automaton that describes a linear path has exponential size only in the maximal number of wildcard nodes between two consecutive // edges (see [20]).

Proof: [Sketch] We show that if there exists a pair (I, J) that is a contradiction for implication, we can find another one, (I', J'), of polynomial size in $\mathcal{C}$ and c with the same property (we call this the *small instance* property). The size of (I', J') will depend on that of c and $\mathcal{C}$, and in particular on the maximal *star-length* of ranges[5]. We first apply a number of pruning steps on J, while maintaining the witness property. After obtaining a J' instance of polynomial size in size of c and the maximal star-length from ranges, we apply similar pruning steps to the I instance. The pruning steps are complex and for space reasons not presented here. In conclusion, we can guess in polynomial time a pair of instances that witnesses non-implication. ■

We next show that if we now restrict the XPath fragment, by disallowing predicates, constraint implication becomes solvable in PTIME. We show the following:

THEOREM 4.8. $\models_{\sigma}^{XP\{/,//,*\}}$ *is in* PTIME *if the number of constraints and the maximal number of wildcards between consecutive //'s are bounded by constants.*

Proof: [Sketch] The proof is based on finite state automata techniques. First, if we assume that implication does not hold and a counterexample exists, it is immediate that we can limit to looking for pairs of trees contradicting $\mathcal{C} \models c$ that are made only of linear paths (i.e., nodes have at most one child). Then, for any constraint c, the range can be modeled as a deterministic automaton of polynomial size (denoted $\mathcal{A}_c$). Hence, the complement of a range can also be modeled by a deterministic automaton of polynomial size (denoted $\mathcal{A}_{\neg c}$). We then look at products of such automata, and there are polynomially many products to be considered if the number of update constraints is considered fixed. ■

5. INSTANCE-BASED IMPLICATION

We study in this section *instance-based implication.* Given a set of constraints and a current tree instance, we want to check if other constraints are implied. Unless specified otherwise, when we mention complexity, we mean combined complexity, i.e., in the size of c, $\mathcal{C}$ and J. The results of this section are summarized in Table 2.

First, similar to constraint implication, we can show that instance-based implication is at least as hard as query containment and equivalence. So when all navigational primitives are used, even with only one update type, instance-based implication is coNP-hard. Next, we show that this lower bound is tight, by showing that we can guess a counterexample pair of instances in polynomial time.

THEOREM 5.1. $\models_{J}^{XP\{/,[],//,*\}}$ *is* coNP*-complete.*

Proof: [Sketch] We show that we have a *small instance* property; more precisely, if there exists an instance I such that the pair (I, J) is a contradiction for implication, we can find an instance I' of polynomial size in $|J|$ and $|\mathcal{C}|$ with the same property. We apply a number of pruning steps on I, making sure to preserve at each step the witness property of the pair. The size of the resulting I' will depend in particular on the maximal star-length of ranges. ■

[5]The *star-length* of a path denotes the maximal length of a chain of wildcards occurring in the path (this notion was introduced by [22]).

Next, we show that, even for classes where the implication problem is tractable, instance-based implication may become intractable.

We show that when no-remove and no-insert constraints are used together, the problem becomes coNP-hard for fragments on which general implication was in PTIME. (We note that some of the hardness results of fragments with wildcard are obtained without making use of this primitive. For brevity of presentation, the cases with and without wildcard are grouped together in Table 2.)

THEOREM 5.2. $\models_{J}^{XP\{/,[]\}}$ *and* $\models_{J}^{XP\{/,//\}}$ *are* coNP*-hard.*

The proof is by reduction from the unsatisfiability of 3SAT formulas. Intuitively, the hardness comes from the fact that we can affect c (without affecting the constraints of $\mathcal{C}$) by permuting data from J, and the moves to consider are exponentially many. Hence we have an exponential search space in which we need to find a "previous" instance I such that $(I, J) \vdash \mathcal{C}$ but $(I, J) \nvdash c$.

In order to trace the tractability boundary, we state without proof that for the most restricted fragment, $XP\{/\}$, instance-based implication is in PTIME.

Since one cannot obtain tractability by imposing reasonable XPath restrictions, we next consider restricting the update types. More precisely, we consider instance-based implication when all constraints (both of $\mathcal{C}$ and c) have the same constraint type. We begin with no-insert constraints, looking first at the fragment $XP\{/, [], *\}$.

THEOREM 5.3. $\models_{J,\ \downarrow}^{XP\{/,[],*\}}$ *is in* PTIME.

Proof: [Sketch] Let $\mathcal{C} = \{(q_i, \downarrow)\}$ and $c = (q, \downarrow)$. We will construct in PTIME an instance $F(J)$ such that $\mathcal{C} \nvDash_J c$ iff $(F(J), J)$ contradicts $\mathcal{C} \vdash c$. This instance will contain all the certain facts that can be obtained from data and constraints. We build $F(J)$ as follows. Initially, $F(J)$ is just *root*. Then, for each constraint $(q_i, \downarrow)$ and each node $n \in q_i(J)$, we add to $F(J)$ a tree having the structure of the range q_i. This tree will have the actual n as the distinguished node and nodes with fresh identifiers for the other query nodes. We put a fresh label (z) for wildcard nodes.

We then take the trees obtained at the previous step, identify nodes with the same Id, and merge respectively their ancestors. We use the following policy for the node identifiers: when two merged nodes have both fresh identifiers, the merged node gets one of them arbitrarily. When one of them has an original label (i.e., a label appearing in $\mathcal{C}$), the merged node gets this label. If one of the nodes has an original Id (i.e. one from J) the merged node gets this Id. It should be noted that no conflicts may arise in the above merging process, namely it cannot be the case that we need to merge two nodes that both have original identifiers, merged nodes will not have different original labels, and no paths of different structures will need to be merged.

Once $F(J)$ is obtained, we rely on the following claim: $\mathcal{C} \nvDash_J c$ iff $q(J)$ contains some node that does not belong to $q(F(J))$. For space reasons, the proof of the claim is omitted. ■

The same upper bound can be obtained for $XP\{/, //, *\}$ under restrictions similar to those of Theorem 4.8:

THEOREM 5.4. $\models_{J,\ \downarrow}^{XP\{/,//,*\}}$ *is in* PTIME*, if the number of constraints and the maximal number of wildcards between consecutive //'s are bounded by constants.*

	$XP\{/\}$	$XP\{/,[],*\}$	$XP\{/,//,*\}$	$XP\{/,[],//,*\}$
only update type $\downarrow$	in PTIME	in PTIME	in PTIME$^{(1)}$	coNP-complete
only update type $\uparrow$	in PTIME	in PTIME$^{(2)}$ coNP-hard	in PTIME$^{(2)}$ coNP-hard	in PTIME$^{(2)}$ coNP-hard
arbitrary update types	in PTIME	coNP-complete	coNP-complete	coNP-complete

(1): if the number of constraints and the maximal number of wildcards between consecutive //'s are bounded by constants.
(2): if the size of c is bounded by a constant.

Table 2: Upper and lower bounds for the instance-based implication of constraints.

The proof is almost identical to that for general implication in this fragment (Theorem 4.8).

We now consider the opposite update type, no-remove. We show that, under no fragment restrictions, we obtain a tractable solution in the size of J and $\mathcal{C}$, though exponential in that of c.

THEOREM 5.5. $\models^{XP\{/,[],//,*\}}_{J,\uparrow}$ *can be decided in time polynomial in the size of* J *and* $\mathcal{C}$*, exponential in that of* c.

Proof: [Sketch] Let $c = (q, \uparrow)$. We show that we can test implication while limiting to I instances that have the "shape" of q. (The construction is similar to the one used in the proof of Theorem 4.7.)

Given an instance I, query q such that $q(I)$ is not empty, and a node $n \in q(I)$, we say a node $n' \in I$ is redundant for n and q if $n \in q(I')$, where I' is the tree obtained from I by removing the subtree rooted at n'.

We say that an instance I is a *possible embedding* of q if it satisfies the following conditions (stated informally): (1) there is a homomorphism from q to I that preserves the parent-child (for / axis) and ancestor-descendant (for // axis) relationships among nodes, the labels of query nodes that are not wildcards, and the root node, (2) for some $n \in q(I)$, there are no redundant nodes for n and q, (3) the nodes of I that are matched (in the evaluation of q yielding n) only to query nodes labeled by wildcards have a fresh label z, and (4) all the paths in I that are matched in this evaluation to a // in the query are sequences of (m + 1) nodes labeled z, for m being the maximal star-length of ranges.

The proof is based of the following observation:

$\mathcal{C} \not\models_J c$ iff there is a previous tree instance I such that

- $(I, J) \vdash \mathcal{C}$ *(i.e. all the range sets are subsets of the ones in J),*
- I *is a* possible embedding *of q in which we assign to nodes that are not labeled z either Ids from J or fresh Ids,*
- *and* $q(I) \not\subseteq q(J)$.

We can enumerate all the possible embeddings by taking the tree pattern representation of q and (1) assigning to wildcard nodes either a label occurring in data or constraints, or the special z label, (2) merging some nodes having the same label, and (3) deciding on how the remaining ones are ordered (i.e., choosing the child/parent and ancestor/descendant relationships that are not already given by q and don't contradict q).

The number of nodes that are not labeled z is at most $|q|$. The number of possible embedding depends only on q (it does not depend on maximal star-length), and is at most exponential.

From the above observation we can derive a naive enumeration algorithm for testing if $\mathcal{C} \models_J c$. The number of Id assignments is bounded by $(|J|+1)^{|q|}$, so polynomial in $|J|$, although exponential in $|q|$. ■

For the same setting, i.e., only no-remove constraints, we can also prove coNP hardness even for the sub-fragments $XP\{/,[]\}$ and $XP\{/,//,*\}$.

THEOREM 5.6. $\models^{XP\{/,[]\}}_{J,\uparrow}$ $\models^{XP\{/,//,*\}}_{J,\uparrow}$ *are* coNP*-hard.*

The proof uses a construction similar to that of Theorem 5.2. Details are omitted.

6. RELATIVE CONSTRAINTS

We briefly consider in this section *relative constraints*. For instance, while we imposed in Example 2.1 that the overall set of `visits` can only increase, we cannot require that the `visits` of each individual `patient` element can only increase as well. This form of update restriction can be expressed by introducing a *scope* over which constraints are specified. More precisely, we would express the above restriction on `visits` by the relative constraint:

$$(/\texttt{patient}, /\texttt{visit}, \uparrow)$$

As already noted in previous works on XML integrity constraints, such relative constraints are particulary suited for hierarchically structured data. For instance, keys that are relative to a node type were introduced in [17].

A possible model extension to relative update constraints could be the following:

DEFINITION 6.1 (SYNTAX). *A* relative XML update constraint *is an expression of the form (q_s, q_r, σ), where q_s, q_r are queries called the* scope *and the* range *, and σ is the* constraint kind*, i.e., one of* no-insert *(in short $\downarrow$) or* no-remove *(in short $\uparrow$).*

DEFINITION 6.2 (SEMANTICS). *We say a pair of trees $(I,\ J)$ is* valid *with respect to some relative constraint $c = (q_s, q_r, \sigma)$ (denoted by $(I, J) \vdash c$) if for all x in $q_s(I) \cap q_s(J)$*

we have $q_r(x, I) \subseteq q_r(x, J)$ *(resp.* $q_r(x, J) \subseteq q_r(x, I)$*) when* σ *is* no-remove *(resp.* no-insert*).*

We next briefly discuss what changes or may change when we have relative constraints. First of all, we can easily exhibit examples (see Example 6.1 below) showing that the *same-type property* of Section 4.1 is no longer true even for $XP\{/, []\}$ (it was proven true for the fragment $XP\{/, [], *\}$ in Theorem 4.1).

EXAMPLE 6.1. *Given the constraints* $\mathcal{C}$*:*

- $c_1 = (/\texttt{patient}, \downarrow)$
- $c_2 = (/\texttt{patient}, /\texttt{visit}, \downarrow)$
- $c_3 = (/\texttt{patient/visit}, \uparrow)$,

the constraint $c = (/\texttt{patient}[/\texttt{visit}], \uparrow)$ *is implied by* $\mathcal{C}$*, although it is not implied by the only no-remove constraint* (c_3) *alone.*

Also, with relative constraints, implication for sequences and for pairs are not necessarily equivalent, as we can easily exhibit sequences in which consecutive pairs are valid but the overall sequence is not. To address this drawback, some soundness requirements could be imposed on constraints.

Regarding complexity, although in the case of constraint implication one should expect complexity to increase, we believe that upper-bounds for instance-based implication should not be affected, and that most of our proofs can be extended to deal with scopes. It seems on the other hand that such an extension with scopes may have a very serious impact on the constraint implication problem. Indeed, it is not even clear whether the problem is still decidable. Implication for relative constraints remains mostly open and is a direction we intend to follow in future work.

7. CONCLUSION

We introduced in this paper a family of update constraints for XML data. We studied general constraint implication, i.e., for all possible instances, and instance-based implication, i.e., in the presence of a current tree instance. Our work was motivated by contexts of exchange or publishing where data has update constraints but no history of updates is maintained. In particular, the constraints considered here could be enforced by existing digital signing techniques [26, 15, 1, 8, 21]. In the reminder of the section we consider additional related work and open questions.

Instance-based implication is related to representing and querying incomplete information (e.g. [3]). One may consider representing a document which is subject to controlled modifications by a combination of certain and possible facts, describing which parts may have changed and how. However, the results of [3] can not be applied directly to our XPath-based setting - the work in [3] uses a different query semantics where queries return, besides the result nodes, also the full root to node path. The instance-based flavor of implication is also related to query answering in XML data exchange [6]. Intuitively, one could see update constraints as source-to-target dependencies. The analogy does not fully go through because we do not consider DTDs and we can also have target-to-source dependencies.

Some of our constraints can be expressed by first-order formulas with 2 variables over unranked, ordered trees with data values. The logic $FO^2(\sim, <)$[6] over trees was proven decidable in [10]. However, this logic captures only update constraints without // axis and the complexity of implication is high.

Several open problems and future work directions were identified throughout the paper. In particular, we highlighted a number of open issues regarding relative constraints. Also, tighter complexity bounds are open for some of the studied problems. Other future directions have to do with richer XPath fragments and "static" integrity constraints (such as keys and foreign keys [18, 11]) and schema information, which are often available for XML data. It is important to understand the role they play in update constraint implication. Note that we assume an update language where nodes may also be moved around. So, in two "consecutive" instances, the same node may appear in totally different parts of the document. However, in some settings, parent-child relations may not be modifiable, and as a consequence one cannot move a node from one parent to another. We leave the study of such "no-move" constraints for future work.

Regarding the instance-based context, besides the studied implication, a more fine-grained language for data validity assertions and temporal reasoning may be worth considering. Also, for both implication problems, as some cases where proven difficult, it may be useful to consider sound approaches. For instance, we could build on the sufficient test of general implication, by taking into consideration only some data properties.

Acknowledgments. We would like to thank Alin Deutsch and Luc Segoufin for helpful discussions and comments.

8. REFERENCES

[1] S. Abiteboul, B. Cautis, A. Fiat, and T. Milo. Digital signatures for modifiable collections. In *ARES*, 2006.

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

[3] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. In *PODS*, 2001.

[4] S. Abiteboul and V. Vianu. Regular path queries with constraints. *J. Comput. Syst. Sci.*, 58(3), 1999.

[5] M. Arenas. Design principles for XML data. PhD thesis, 2005.

[6] M. Arenas and L. Libkin. XML data exchange: consistency and query answering. In *PODS*, 2005.

[7] M. Arenas, W. Fan, and L. Libkin. Consistency of XML specifications. In *Inconsistency Tolerance*, 2005.

[8] M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. *Lecture Notes in Computer Science*, 1994.

[9] M. Benedikt and C. Koch. XPath leashed. Technical report, 2006.

[10] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS*, 2006.

[11] P. Buneman, S. Davidson, W. Fan, C. Hara, and

[6] $\sim$ is a predicate that checks equality of two data values (such as the *Id* attribute), while $<$ refers to two relations, for parent/child and sibling.

W. C. Tan. Reasoning about keys for XML. *Inf. Syst.*, 28(8), 2003.

[12] B. Cautis, S. Abiteboul, and T. Milo. Reasoning about XML update constraints. In *BDA*, 2006.

[13] C. Y. Chan, W. Fan, P. Felber, M. N. Garofalakis, and R. Rastogi. Tree pattern aggregation for scalable XML data dissemination. In *VLDB*, 2002.

[14] A. Deutsch and V. Tannen. XML queries and constraints, containment and reformulation. *Theor. Comput. Sci.*, 336(1), 2005.

[15] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. In *IFIP Workshop on Database Security*, 2000.

[16] W. Fan. XML constraints: Specification, analysis, and applications. In *DEXA Workshops*, 2005.

[17] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 2002.

[18] W. Fan and J. Siméon. Integrity constraints for XML. *J. Comput. Syst. Sci.*, 66(1), 2003.

[19] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and xml typing. *J. ACM*, 52(2), 2005.

[20] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata. In *ICDT*, 2003.

[21] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *CT-RSA*, 2002.

[22] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. of ACM PODS*, 2002.

[23] G. Miklau and D. Suciu. Modeling integrity in data exchange. In *SDM*, 2004.

[24] F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2 (3:1), 2006.

[25] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS*, 2000.

[26] R. L. Rivest. Two signature schemes., 2000. Manuscript.

[27] T. Schwentick. XPath query containment. *SIGMOD Record*, 33 (1), 2004.

[28] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *SIGMOD*, 2001.

[29] P. Wadler. A formal semantics of patterns in XSLT, 1999.

[30] Extensible Markup Language 1.0 (2nd Edition). `http://www.w3.org/TR/REC-xml`.

[31] The XML Schema specification. `http://www.w3.org/TR/XML/Schema`.