Journal of Computer and System Sciences 75 (2009) 336–358

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Reasoning about XML update constraints ☆

Bogdan Cautis [a,*], Serge Abiteboul [b], Tova Milo [c]

[a] Telecom ParisTech, Computer Science Department, 46, rue Barrault, 75013 Paris, France
[b] I.N.R.I.A. Futurs, 4 Rue J. Monod, Parc Club Université, 91893 Orsay Cedex, France
[c] Tel Aviv University, School of Computer Science, Schreiber Building, Ramat Aviv, Tel Aviv 69978, Israel

ARTICLE INFO

Article history:
Received 19 April 2007
Received in revised form 20 February 2009
Available online 3 March 2009

Keywords:
Semi-structured data
XML
Update constraints
Implication
Data integrity

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 in which we take into account data. 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.

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 (Fig. 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*.

☆ This work has been partially supported by the ANR grant DocFlow, the EC project EDOS and the Israel Science Foundation.
* Corresponding author.
E-mail addresses: cautis@telecom-paristech.fr (B. Cautis), serge.abiteboul@inria.fr (S. Abiteboul), milo@cs.tau.ac.il (T. Milo).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2009.02.001

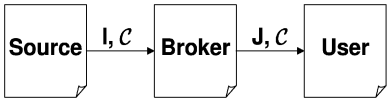

Fig. 1. Exchange with update constraints ($\mathcal{C}$).

The *constraint implication problem* is defined as follows. Given a set of update constraints $\mathcal{C}$ 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. 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 [1,8,21,22], 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).

An extended abstract of this work appeared in the proceedings of the 26th ACM Symposium on Principles of Database Systems [12]. The present paper provides a comprehensive description of our results, including additional examples and detailed proofs which together bring better insight into reasoning about update constraints for XML.

To the best of our knowledge, this work is the first to consider update constraints for tree structured data. A limited class of update constraints, namely dynamic functional dependencies, and their interaction with integrity constraints have been considered, for relational data, in [28]. Perhaps the work that is closest in spirit is the one of Miklau and Suciu [24] 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 [11,15,17], 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.

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 *Id*s ($\mathcal{N}$). 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:

$$path ::= /step \mid //step \mid path\ path$$

$$step ::= label\ pred$$

$$pred ::= \epsilon \mid [path] \mid pred\ pred$$

$$label ::= L \mid *$$

[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.

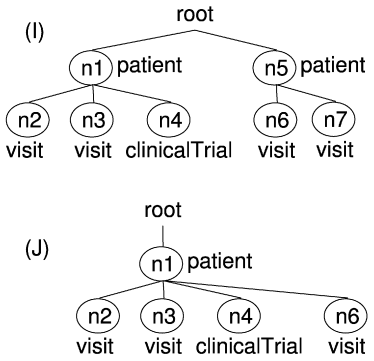

Fig. 2. Pair of instances (I, J).

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, [9,29]). 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 [18].

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 $\sqsubseteq$) 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 [27], 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*).

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 Fig. 2 and the following constraints:

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

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 Fig. 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.

2.2. 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 Fig. 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.

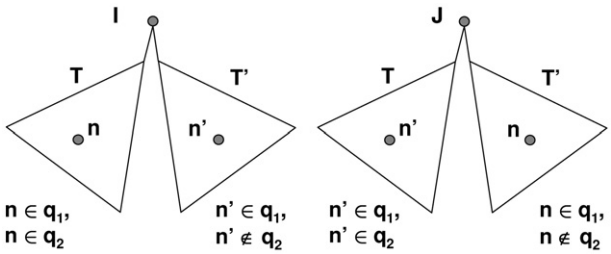

Fig. 3. Counterexample pair of instances (I, J).

2.3. 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 $\models^{X}_{\sigma}$, where X is the XPath fragment we assume and $\sigma \in \{\uparrow, \downarrow\}$. For example, $\models^{XP\{/,[]\}}_{\uparrow}$ denotes the constraint implication problem for no-remove constraints expressed in $XP\{/, []\}$. Similarly, we use $\models^{X}_{J,\sigma}$ 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).

3.1. Query equivalence

Since we rely on XPath queries to express update constraints, it comes as no surprise that XPath query containment and equivalence [26] 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. 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 $q_2 \not\sqsubseteq q_1$, 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 $q_1 \not\sqsubseteq q_2$, 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 Fig. 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. □

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 set of $\uparrow$ constraints $\mathcal{C}$ if there exist some constraints $c_1, \ldots, c_k$ in $\mathcal{C}$, with their respective range queries $q_1, \ldots, q_k$, s.t. $q \equiv q_1 \cap \cdots \cap q_k$.*

[2] Query equivalence and containment were shown to be coNP-hard in [23] (see also [26]) for $XP\{/, [], //, *\}$; the proof of this lower bound uses only concrete paths.

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

3.2. Regular XML key constraints

Previous works addressed the implication problem for inclusion dependencies on semi-structured data [4,11,17] and XML keys/foreign keys [6,16], taking into account also schema information (such as DTDs). In particular, *regular XML keys* [6] have the form $\beta.\tau[X] \rightarrow \beta.\tau$, 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 [6]) 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 [6,16].)

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 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 (see Example 3.1). 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.

Example 3.1. Consider the update constraint $c = (/\mathtt{a}/\mathtt{b}, \uparrow)$. Assume the straightforward transformation φ which from any pair of trees (I, J) builds another tree with two main branches corresponding to I and J, mapping labels different from a or b into a new label z. We can then capture c by a simple DTD (D) and a set of regular key constraints R_c as follows. Let the D be the following:

$$root :- I, J$$

$$I :- (a|b|z)^*$$

$$J :- (a|b|z)^*$$

$$a, b, z :- Id, (a|b|z)^*$$

$$Id :- @id$$

The set of regular constraints R_c consists of two 2 key constraints and a foreign key constraint:

$$root.I.(a|b|z)^+.Id@id \rightarrow root.I.(a|b|z)^+.Id \tag{1}$$

$$root.J.(a|b|z)^+.Id@id \rightarrow root.J.(a|b|z)^+.Id \tag{2}$$

$$root.I.a.b.Id@id \subseteq root.J.a.b.Id@id \tag{3}$$

(The symbol '.' denotes concatenation. For $(a|b|z)^+$, we could have used wildcard, '_' by the notation of [6], writing it as "$\_^+$.")

Constraints (1) and (2) say that no two nodes can have the same *Id* in the I or J branch, while constraint (3) says that the set of b's in the I branch is a subset of those in the J branch. It is straightforward to check that a pair of trees (I, J) satisfies c iff the corresponding tree in D satisfies R_c. More details on this translation will be given in Section 4.

3.3. 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 [15]. An XIC is defined as follows:

$$\forall x_i, \ldots, x_n \quad A(x_1, \ldots, x_n) \quad \rightarrow \quad \exists y_1, \ldots, \exists y_m \quad B(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

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

While the implication problem for XIC expressions has not been fully explored yet, implication under some limitations and query containment in the presence of such constraints have been considered in [15]. In general, implication of XICs was

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

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.

Example 3.2. The constraint $(/\mathtt{a}[//\mathtt{b}]/\mathtt{c}, \uparrow)$ can be expressed using XICs as follows.

For each label l in $\{a, b, c\}$, the following constraints can enforce node *Id*s in the branch I:

$$\forall x_I, x \quad (/I\, x_I) \wedge (x_I//l\, x) \quad \rightarrow \quad \exists v \quad (x/@id\, v)$$

(i.e., every node labeled l has *at least* one *id* attribute)

$$\forall x_I, x, v, v' \quad (/I\, x_I) \wedge (x_I//l\, x) \wedge (x/@id\, v) \wedge (x/@id\, v') \quad \rightarrow \quad v = v'$$

(i.e., every node labeled l has *at most* one *id* attribute)

$$\forall x_I, x, x', v \quad (/I\, x_I) \wedge (x_I//l\, x) \wedge (x/@id\, v) \wedge (x_I//l\, x') \wedge (x'/@id\, v) \quad \rightarrow \quad x = x'$$

(i.e., two distinct nodes cannot have the same value for the *id* attribute[4]).

Similar XICs can be used for the branch J. In the same style, the existence and uniqueness of the two branches, I and J, must be enforced.

The update constraint can be then captured by the following XIC:

$$\forall x_I, x_a, x_b, x_c, v \quad \big[(/I\, x_I) \wedge (x_I/a\, x_a) \wedge (x_a//b\, x_b) \wedge (x_a/c\, x_c) \wedge (x_c/@id\, v)\big] \quad \rightarrow$$
$$\exists x_J, x'_a, x'_b, x'_c \quad \big[(/J\, x_J) \wedge \big(x_J/a\, x'_a\big) \wedge \big(x'_a//b\, x'_b\big) \wedge \big(x'_a/c\, x'_c\big) \wedge \big(x'_c/@id\, v\big)\big].$$

Unfortunately, the XICs needed to capture update constraints are *not bounded* because of both // axis and the existential quantification on the *id* attribute (in the first constraint). 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. The following simple example shows that the chase approach may not always terminate, entering into a loop of generating new facts.

Example 3.3. For the two constraints:

- $(c1)$: $(/\mathtt{a}/\mathtt{b}/\mathtt{c}, \uparrow)$
- $(c2)$: $(/\mathtt{a}/\mathtt{b}[\mathtt{c}], \downarrow)$

if we test implication for $(/\mathtt{a}/\mathtt{b}/\mathtt{c}/\mathtt{d}, \uparrow)$ using the chase, we get into a non-terminating chase sequence by applying $(c1), (c2), (c1), (c2), (c1), \ldots$, at one step adding something under the J branch (namely, a subtree $a/b/c$ with a fresh *Id* for the b node and an *Id* from the I branch for the c node), then adding something under the I branch (namely, a subtree $a/b/c$ with a fresh *Id* for the c node and the previously introduced *Id* for the b node), and so on indefinitely. Here, non-termination is a direct consequence of having existentially quantified *id* attributes. The reader can find detailed explanations on how the chase technique works in [15], and they are omitted here.

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:

[4] In the XIC model, the equality operator means *same node* for node variables and *same value* for attribute variables.

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

	$XP\{/, [], *\}$	$XP\{/, [], //\}$	$XP\{/, //, *\}$	$XP\{/, [], //, *\}$
only one update type	in PTIME	coNP-complete	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.

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

imply $c = (//\mathtt{b}//\mathtt{a}//\mathtt{c}, \uparrow)$, while the no-remove constraints alone do not. We next detail this implication. Let I be some instance from which we try to remove a c-labeled node n_c from the range of c. Let n_b and n_a be the "closest" two nodes labeled b and a on the path from root to n_c (i.e., there are no other nodes labeled a or b in between n_b and n_a). Now, if n_b has some a-labeled ancestor, then n_c cannot be removed from $//b//a//c$ (by c_4). Similarly, if there is some b-labeled node on the path between n_a and n_c, then n_c cannot be removed from $//b//a//c$ (by c_5). For the remaining cases, observe first that for now $n_c \notin //a//b//c(I)$ (i.e., n_c is not in the range of c_3). By c_1 and c_2, we can still remove n_c only by permuting n_b and n_a, but this would break c_3.

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 $\mathcal{C}^\sigma \models c$.*

Proof. We assume w.l.o.g. that $\sigma = \uparrow$. Let c be $(q, \uparrow)$. We say that a pair of trees (I, J) *violates* c, noted $(I, J) \nvDash c$, if $q(I) \not\subseteq q(J)$.

One implication is obvious ($\mathcal{C} \models c \Leftarrow \mathcal{C}^\sigma \models c$). For the other implication, we show that if $\mathcal{C}^\sigma \nvDash c$ then $\mathcal{C} \nvDash c$. More precisely, we show that from a pair witnessing $\mathcal{C}^\sigma \nvDash c$ we can always build one witnessing $\mathcal{C} \nvDash c$. Let $c = (q, \uparrow)$, let $(q_i^\uparrow, \uparrow)$ be the set of no-remove constraints and let $(q_j^\downarrow, \downarrow)$ be the set of no-insert constraints. Let (I, J) be a pair of trees such that:

- $q(I) \not\subseteq q(J)$ (some node was removed from q),
- $q_i^\uparrow(I) \subseteq q_i^\uparrow(J)$, for all $q_i^\uparrow$ (no-remove constraints are not violated)

(i.e. (I, J) is a witness for $\mathcal{C}^\sigma \nvDash c$).

Let $n \in q(I), n \notin q(J)$ be one of the nodes removed from q's result. First, if we assume that n is in none of the specified no-remove ranges ($n \notin q_i^\uparrow(I)$, for all i), we can obtain $\mathcal{C} \nvDash c$ by constructing from I a pair (I', J') as depicted in Fig. 4. More precisely, we add to I a copy[5] of n and its subtree. Both n and n' are in $q(I')$ and, as assumed, they are in none of the $q_i^\uparrow(I')$ results. We obtain J' by removing n and placing its subtree under n'. Hence only n is removed; no-remove constraints are not affected by this change since n was not in their ranges, while no-add constraints cannot be affected either.

Let us assume next that there exist some $q_i^\uparrow$ such that $n \in q_i^\uparrow(I)$. We first deal with a particular case, and then show how the remaining cases can be reduced to it. The particular case is when assuming the following:

- for all $q_j^\downarrow$, we do not have $n \notin q_j^\downarrow(I)$ and $n \in q_j^\downarrow(J)$, hence n is not inserted in some no-insert range.

With this assumption, we can build a witness pair of instances for $\mathcal{C} \nvDash c$ as illustrated in Fig. 5. Let us clarify the details of this construction:

[5] By *copy* of a tree we denote a tree having the exact structure and labels, but fresh *IDs*.

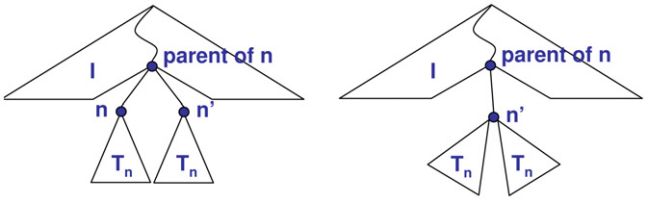

Fig. 4. Counterexample pair of instances (I', J').

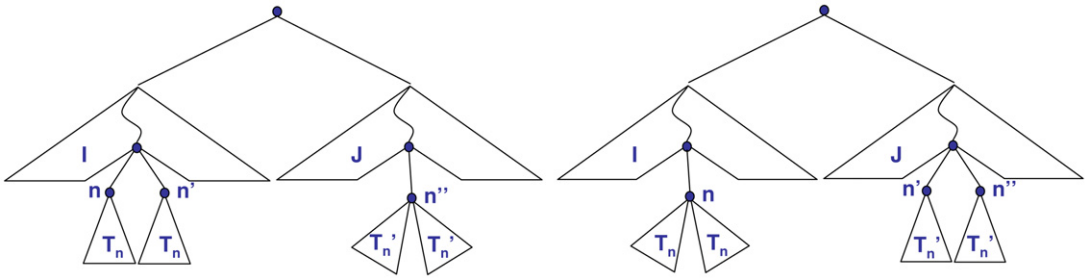

Fig. 5. Counterexample pair of instances (I', J').

- we first modify I by adding a copy of the tree rooted at n (including n) in I (n' is the copy of n),
- we then modify J by duplicating the tree rooted at n (without n),
- we obtain I' by putting under the same root copies of the modified I and J (n'' is the copy of n from J),
- J' is obtained from I' by moving the n' copy from the left subtree to the right subtree.

It is straightforward to check that this transformation does not violate $\mathcal{C}$ while it violates c.

Now consider the remaining case where there are some $q_j^{\downarrow}$ ranges such that $n \notin q_j^{\downarrow}(I)$ and $n \in q_j^{\downarrow}(J)$.

All we need to do in this case is to find an instance J_0 that allows us to build the same counterexample as in Fig. 5. To this end, we construct an instance J_0 such that n is in some range in J_0 iff it is in that range in both I and J. It is straightforward to check that the construction of Fig. 5 would then work for the pair (I, J_0). In short, J_0 will be an intersection of I and J w.r.t. node n, where we keep only the common patterns for which n qualifies (a similar construction was described in [14] as the least-upper bound of tree patterns). The construction of J_0 is straightforward and we can next apply the counterexample construction of Fig. 5. As a last observation, this construction may not be possible if the descendant axis is used.

We next discuss the construction of J_0 from I and J. First, for both trees and for each node n' on the root-to-n path we double the following (child) node n'' found on this path towards n. Obviously, this will not introduce new ranges in the picture.

From this, we start constructing the actual J_0. Its root-to-n path is obtained by taking the two root-to-n paths and intersect them. First, notice that they should have the same length. For intersection, in each position, if the 2 nodes have different labels we put one with label z, with z being a fresh label. Otherwise, we keep their label. Then, for each node on this root-to-n path we look at I and J and the sets of all predicates that can be matched at that node in I, respectively in J. Then, we take their intersection. The patterns in this intersection are plugged under this node.

It is rather straightforward to check that all non-common ranges will be eliminated. This is because there must be some predicate that in one of the trees is matched while in the other is not.

Note that all predicates that might have been matched before at some node are still matched (since we doubled each node on the root-to-n path). This is why we argue that after this step, the common ranges are all preserved. This is because these common ranges for sure asked for $*$ in that position. No new ranges are introduced. □

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.*

The proof of this result is quite long, so we first give a rough outline. In short, we solve the constraint implication problem by reducing it to the consistency problem for unary regular constraints and DTDs (described in Section 3). We start by giving the reduction for linear paths only (i.e., $XP\{/, //, *\}$). In this case, the reduction to the setting of [6] is easier.

We then consider predicates, which are more difficult to capture in the framework of [6]. 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 roughly 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 annotations). To avoid that annotations "lie" about the patterns that can be matched at a node, we control by the DTD the correspondence between a node's annotations and its content.

Regarding its general structure, the 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.

Although the upper-bound for the consistency problem is 2-NEXPTIME [5], the problem instance we obtain is solvable in only non-deterministic exponential time due to the limited type of inclusions we must handle (i.e., only between two main branches, I and J).

Proof. Linear paths. We start with the case of linear patterns (no predicates). This will in fact represent the proof of Theorem 4.3, which is stated in Section 4.3.

Let $\mathcal{C}, c$ be the constraints and (I, J) some pair of instances witnessing $\mathcal{C} \not\models c$. Let $l_1, \ldots, l_k$ denote the labels occurring in these constraints and let z denote a new label. Also, let n be some node which breaks c. First, it is straightforward to prove that if a pair (I, J) witnessing non-implication exists, then the pair (I_z, J_z) obtained from (I, J) by replacing labels different from $l_1, \ldots, l_k$ by the z label is also a witness. So, we can safely assume that only the labels $l_1, \ldots, l_k, z$ occur in a witness.

We build a DTD D, set of regular path constraints Σ and a simple bijective transformation φ such that for any tuple (I, J, n), the pair of instances witnesses non-implication (by n) iff the tree $\varphi(I, J, n)$ is consistent w.r.t. D and Σ.

Let D be the following:

$$root :- I, J, witness$$

$$I :- (l_1|\ldots|l_k|z)^*$$

$$J :- (l_1|\ldots|l_k|z)^*$$

$$witness :- Id$$

$$l_1, \ldots, l_k, z :- Id, \quad (l_1|\ldots|l_k|z)^*$$

$$Id :- @id$$

(The *Id* type is only needed in order to respect the syntax of regular constraints from [6].)

The φ transformation will build a tree valid w.r.t. to D by simply taking I and putting it in the I-branch (node *Id*s become the *id* attribute of the *Id* element), same for J, and then putting n in the witness branch. This transformation is obviously bijective.

We next describe the set of regular constraints Σ. Using the notation of [6], the symbol '.' denotes concatenation. We first introduce 2 key constraints:

$$root.I.(l_1|\ldots|l_k|z)^+.Id@id \to root.I.(l_1|\ldots|l_k|z)^+.Id@id \tag{4}$$

$$root.J.(l_1|\ldots|l_k|z)^+.Id@id \to root.J.(l_1|\ldots|l_k|z)^+.Id@id \tag{5}$$

These constraints say that we cannot have two nodes with the same *id* attribute in the I branch or the J branch.

Next, for each constraint (q, τ), let $reg(q)$ be the regular expression obtained by (a) replacing each / axis by the '.' symbol, (b) replacing each $*$ by "$l_1|\ldots|l_k|z$," and (c) replacing each // by "$(l_1|l_2|\ldots|l_k|z)^*$."

For constraints $(q, \uparrow)$ and $(q, \downarrow)$ respectively, we introduce foreign keys of the form

$$root.I.reg(q).Id@id \subseteq root.J.reg(q).Id@id \tag{6}$$

$$root.J.reg(q).Id@id \subseteq root.I.reg(q).Id@id \tag{7}$$

Next, assume w.l.o.g. that c is no-remove, $c = (q_c, \uparrow)$. The other case is analogous by symmetry. We also put explicitly in the tree a node that is causing the pair (I, J) to break c. For that, we impose the following constraints:

$$root.witness.Id@id \subseteq root.I.reg(q_c).Id@id \tag{8}$$

$$root.\big(witness \mid \big(J.reg(q_c)\big)\big).Id@id \to root.\big(witness \mid \big(J.reg(q_c)\big)\big).Id \tag{9}$$

This basically makes sure the witness *Id* is the range of c in I but not in J. It is straightforward to check that we have a tree $\varphi(I, J, n)$ that is consistent with the DTD D and constraints Σ iff the corresponding pair (I, J) witnesses $\mathcal{C} \not\models c$ by n.

Regarding complexity, we know from [5] that consistency is 2-NEXPTIME for unary regular constraints. More precisely the upper-bound is

- non-deterministic doubly-exponential only in the number of constraints, while
- only non-deterministic exponential in their size (more precisely, non-deterministic polynomial in the size of the product of their DFAs),
- only non-deterministic polynomial in the size of the DTD.

So this would yield a 2-NEXPTIME upper bound. But, for the first item, we gain one order of magnitude by the particularity that we have constraints only between the two main branches, I and J. This is because a particular node *Id* can at most appear twice, once in I and once in J. The doubly-exponential upper bound in [5] comes from having to consider various sets of boolean expressions over regular paths. In our setting, most such possible sets will be empty, only those with at most two boolean expressions are relevant. All other sets of boolean expressions would describe *Id*s found in two different places in the I branch or the J branch (and this is not possible). The overall complexity becomes thus NEXPTIME.

If we further assume that the number of constraints is constant and that the maximal number of wildcards between 2 consecutive //'s is constant (making the size of the DFAs only polynomial) then the complexity drops to NP.

Dealing with predicates. We next extend the above approach to handle predicates, which is the main difficulty since they are not directly expressible by unary regular constraints. The crux is to transform normal labeled trees into *annotated* trees, where the label of a node describes precisely the sub-patterns that can be matched below that node.

In the following, by the *patterns* of $\mathcal{C}$ and c we denote all the query parts of ranges (starting with an edge) generated by */step* or *//step* in the grammar of Section 2, seen as boolean queries (the output is ignored). The patterns to be considered are those occurring in constraints, plus the patterns obtained from these ones as follows:

- for patterns of the form "*//rest-of-the-path*," we also consider the pattern "*/rest-of-the-path*,"
- for patterns of the form "*/∗rest-of-the-path*," we also consider the pattern "*/l rest-of-the-path*," for all $l \in \{l_1, \ldots, l_k, z\}$,
- for patterns of the form "*//rest-of-the-path*," we also consider the pattern "*/l//rest-of-the-path*," for all $l \in \{l_1, \ldots, l_k, z\}$.

Let P denote the set of patterns obtained in this way. The number of patterns is obviously polynomial in the size of the constraint ranges. They are assumed ordered (for instance, in lexicographic order). By an *annotation* we denote a *boolean expressions over* P, i.e., of the form $m \in \{0,1\}^{|P|}$. We say that the pattern $p \in P$ is *included* in an annotation m iff $m(p)$ is 1.

For any label $l \in \{l_1, \ldots, l_k, z\}$ and any annotation in $m \in M$, we introduce the *annotated label* $l \| m$. Intuitively, a node with annotated label $l \| m$ denotes l-labeled nodes for which all the patterns included in m match. Observe that some annotations may not be possible, since qualifying for some predicates may imply qualifying for some others too. For example, a node having an a-labeled child will also have an a-labeled descendant. Similarly, a node qualifying for $//a$ must for sure qualify at least for some $/l//a$ or for $/a$. We denote by M the set of "consistent" configurations, in which no pattern which is not explicitly included is *implied* by the conjunction of the included ones. The set of consistent annotations M can be computed in exponential time, as it amounts to testing boolean query containment in $XP\{/, [], //, *\}$.

As before, for any (I, J) pair witnessing non-implication and node n which breaks c, a transformation φ builds a tree with three main branches I, J and *witness*. The novelty is that for each node of I or J, the label is now replaced by the corresponding annotated label.

A similar translation is first performed on the queries in $\mathcal{C}$ and c, and then combined with the previous reduction steps towards unary regular constraints. We obtain from q the regular expression $reg(q)$ as follows:

- for each node n' in the linear root-to-result path of a constraint, we compute all the patterns $P_{n'}$ that will for sure be matched below. Let $M_{n'}$ denote all the consistent annotations including $P_{n'}$,
- each query node n' on the root-to-result path, labeled by some l_i, will be replaced by the disjunction of annotated labels $l_i \| m$ for all $m \in M_{n'}$,
- similarly, each query node n' on the root-to-result path, labeled by wildcard, will be replaced by the disjunction of extended labels $l \| m$ for all $m \in M_{n'}$ and all $l \in \{l_1, \ldots, l_k, z\}$,
- we remove all the predicates, keeping only the root-to-node path,
- we replace each / axis by the '.' symbol,
- we replace each // by the disjunction of all consistent annotated labels $l \| m$, for all labels l and consistent annotations m.

Let Σ denote the set of regular constraints containing (4) and (5) (obtained as before, but with annotated labels), and (6), (7), (8), (9) (using the new *reg* translation).

Observe that these regular constraints can be exponentially bigger than the update constraints from which they were obtained. However, they can still be described by a DFA of only exponential size, regardless of how the initial pattern looked like and regardless of the number of extended labels for one given label (as long as this number remains only exponential).

For now, it is straightforward to check that for any (I, J) pair and node n, the tuple is a counter-example for implication iff the tree $\varphi(I, J, n)$ is valid for the regular constraints Σ. Conversely, we would like to prove that any such annotated tree with three main branches I, J and *witness* will give us a pair of "normal" trees (I, J) witnessing non-implication. The only problem is that the existence of such a tree does not necessarily translate into the existence of the pair denoting a

counter-example. This is because annotations of nodes may "lie" about the predicates that can be matched at them (after removing the annotations). We next show how this can be controlled by the DTD.

(Our approach is somehow similar to that of [25], for deterministic tree automata recognizing boolean patterns in $XP\{/, [], //, *\}$.)

We start by describing how the first levels of the document are specified in the DTD (D'). We first need to "guess" the exact set of patterns that will be matched in the I and J branches. The first productions are as follows:

$$root :- I, J, witness$$

$$I :- (I\|m_1) \mid \ldots \mid (I\|m_k)$$

$$J :- (J\|m_1) \mid \ldots \mid (J\|m_k)$$

$$witness :- Id$$

$$Id :- @id$$

where each m_i denotes a consistent boolean formula over P. Observe that the size of these productions is at most exponential.

Next, we describe the productions for the $I\|m_i$ types (the ones for the $J\|m_i$ types are similar).

For each m_i, let $L(m_i) \subseteq \{l_1, \ldots, l_k, z\}$ denote the set of labels appearing in patterns included in m_i, of the form "$/l\ldots$"; for each l, the set of such patterns is denoted $Y_l(m_i)$. These patterns give in a way the certain children of the I node. Also, let $N_l(m_i)$ be the patterns not included in m_i, of the form "$/l\ldots$."

Let $Y_l(m_i) = \{p'_1, \ldots, p_{k'_l}\}$, let $N_l(m_i) = \{p''_1, \ldots, p''_{k''_l}\}$. For each $p'_j \in Y_l(m_i)$, let G'_j be the set of sub-patterns children of the l query node. Similarly, for each $p''_j \in N_l(m_i)$, let G''_j be the set of sub-patterns children of the l query node.

For each $I\|m_i$ and each label l which is an immediate child, we have to consider the various choices of grouping the sets $G'_1, \ldots, G'_{k'_l}$.

In order to make the DTD productions less complex, we introduce a level of intermediary types. Each intermediary type will only expand into one annotated node, and will be defined by a disjunction of such individual annotated nodes. The intermediary types will be noted $\tau_l[S, m_i]$, for $S \in Y_l(m_i)$. Such a type will expand into an l node which verifies all the patterns in S and for which all the patterns in $N_l(m_i)$ are not verified.

First, we use as a shortcut notation $\tau_l[m_i]$ for the disjunction

$$\bigvee_{(S_1,\ldots,S_k)} \left(\tau[S_1, m_i]^+, \ldots, \tau[S_k, m_i]^+\right)$$

for all $(S_1, \ldots, S_k)$ such that $Y_l(m_i) = S_1 \cup \cdots \cup S_k$ and $S_1, \ldots, S_k$ are all disjunct. The size of this disjunction is at most exponential.

This disjunction will basically enumerate all the possible configurations of l children, such that one of these configurations must for sure occur in order to satisfy all the patterns in $Y_l(m_i)$.

Using this shortcut, we are now ready to give the production defining $I\|m_i$:

$$I\|m_i :- \left(\bigwedge_{l \in L(m_i)} \tau_l[m_i]\right), any[m_i].$$

Observe that the size of this production is at most exponential.

We next describe the productions for each $\tau_l[S, m_i]$ and for $any[m_i]$. The former is defined as:

$$\tau_l[S, m_i] :- \bigvee_m l\|m$$

for m denoting all the consistent annotations which verify the following:

- assuming $S = \{p'_{i_1}, \ldots, p'_{i_k}\} \subseteq Y_l(m_i)$, all the patterns of $G'_{i_1}, \ldots, G'_{i_k}$ are included in m,
- for all the patterns $p''_j \in N_l(m_i)$, there exists some pattern in G''_j which is not included in m.

The latter is defined as:

$$any[m_i] :- \bigwedge_l \bigwedge_m (l\|m)^*$$

for l denoting all the labels $\{l_1, \ldots, l_k, z\}$ and m denoting all the consistent annotations which verify the following:

- for all the patterns $p''_j \in N_l(m_i)$, there exists some pattern in G''_j which is not included in m.

Then, the DTD continues in similar style for each $l\|m$. Each $l\|m$ node will also have one *Id* child.

(Since we introduce some intermediary nodes, we need to consider them in the constraints. This means we need to plug the disjunction of intermediary types between each pair of consecutive annotated labels.)

Once Σ and D' are constructed, we can prove that there exists a tuple (I, J, n) witnessing non-implication iff there exists an annotated tree satisfying D' and the regular constraints Σ.

In conclusion, we obtain a DTD of exponential size (exponentially many types and exponentially big productions), unary regular constraints also of exponential size, but described by DFAs of only exponential size. The number of constraints remains the same. Hence the overall complexity remains NEXPTIME. □

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.

4.3. 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. We use the same reduction as in the proof of Theorem 4.2. 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.[6] 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 $c_1, \ldots, c_k \in \mathcal{C}$ with respective ranges $q_1, \ldots, q_k$, s.t. $q \equiv q_1 \cap \cdots \cap q_k$.*

One direction (sufficiency) was already discussed (Proposition 3.1). The other direction is proven by induction on the number of no-remove constraints.

Proof. We can assume w.l.o.g. no-remove constraints; the opposite type is analogous by symmetry. We already proved the statement for one no-remove constraint in Theorem 3.1. Next, assuming the statement holds for any set of no-remove constraints of cardinality $k \geqslant 1$, we show that it will also hold for sets of constraints of cardinality $k+1$. In other words, we need to prove that if c is implied, given a set $\mathcal{C}$ of $k+1$ no-remove constraints, then q is equivalent to the intersection of some of their respective ranges.

A simple case. Let us assume that for all q_i, $i = 1 \ldots k+1$, we have $q \sqsubseteq q_i$, hence $q \sqsubseteq \bigcap_i q_i$. If we assume we do not have inclusion in the other direction, there exists a tree T' and node n' such that $n' \in \bigcap_i q_i(T')$ and $n' \notin q(T')$. With this, we obtain a contradiction to $\mathcal{C} \models c$ by a transformation similar to that of Fig. 3.

Remaining case. We are left with the case where there exists a no-remove range query, say q_{k+1}, such that $q \not\sqsubseteq q_{k+1}$. We show that if c is implied, then q is equivalent to the intersection of some range queries among $q_1, \ldots, q_k$. More precisely, with an abuse of notation clarified shortly, we show that

$$(\star) \quad \mathcal{C} \models c \quad \Rightarrow \quad \{c_1, \ldots, c_k\} \models c.$$

Once $(\star)$ is proven, by the induction step, it would imply exactly the needed conclusion (equivalence to the intersection of some range queries). To prove $(\star)$, what needs to be proven is that if (I, J) *violates q then (I, J) violates some of $q_1 \ldots q_k$*. Let us assume this is not true, and let (I, J) be a witnessing pair. We show that this leads to $\mathcal{C} \not\models c$.

We first build an auxiliary tree as follows: let T_q be a tree that has the exact shape of q, with the node corresponding to the result node of q being some $n \in q(I)$. When // or wildcard are present, we will build T_q by replacing wildcard labels by some unused label z, and // edges with a $/z/$ path.

[6] 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 [19]).

It is easy to see that if $q \not\sqsubseteq q_{k+1}$ then $n \notin q_{k+1}(T_q)$. By arguments used before, we know that there must exist some q_i range queries such that $n \in q_i(T_q)$. From the way T_q is constructed, an important property is the following: if $n \in q_i(T_q)$ then $n \in q_i(I)$ also (since we have in I another, possibly "tighter," q-embedding with result n). In particular, by the assumption that the pair (I, J) does not violate any of $q_1, \ldots, q_k$, this means that $n \in q_i(T_q)$ implies $n \in q_i(J)$.

We are now ready to conclude this proof by constructing a counterexample (I', J'), similar to that of Fig. 5, with T_q in the role of I.

Computing the intersection of some $XP\{/, [], *\}$ is straightforward. First, the root-to-result paths in queries have to be compatible, i.e., same length and in each position they have the same concrete label or wildcard. We need to merge this root-to-results paths, at each position keeping the concrete label l if some appears in one of the queries, or wildcard otherwise. All the predicates of nodes are kept below the node into which they merged. □

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 $q \sqsubseteq q_i$ (testable in polynomial time [23]), and then test if $q \equiv \bigcap_i q_i$.

$XP\{/, [], *\}$ is closed under intersection and an intersection $\bigcap_i q_i$ can be computed in linear time, hence the equivalence test above can be done in polynomial time for this fragment. However, for $XP\{/, [], //\}$, this test is more difficult. First, it can be easily checked that this fragment is not closed under intersection. More importantly, a recent result from [13] indicates a coNP lower-bound for testing $q \equiv \bigcap_i q_i$ (see Theorem 4.9).

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.*

Proof. We start with the proof for $XP\{/, [], //\}$. The proof for $XP\{/, //, *\}$ is very similar.

XP{/,[],//} The proof is by reduction to the problem of checking unsatisfiability of 3CNF formulas. Consider a 3CNF formula f with n variables $x_1 \ldots x_n$. For instance, suppose f is $(x_1 \vee \neg x_2 \vee x_3) \wedge \cdots$. We transform this formula into a set of constraints $\mathcal{C}$ and one to-be-implied constraint c, such that $\mathcal{C} \models c$ iff f is not satisfiable.

c is the following:

$$c = (\mathtt{s}/\mathtt{x}_1//\mathtt{x}_2// \ldots //\mathtt{x}_n//\mathtt{m}//\mathtt{x}_1// + // - //\mathtt{x}_2// + // - //\mathtt{x}_3// \ldots //\mathtt{x}_n// + // - //\mathtt{e}, \uparrow)$$

($+$ and $-$ are labels.) This means that in order to break c, one has to remove an $\mathtt{e}$ node from the range path. We next describe the constraints of $\mathcal{C}$, showing how they limit our space of maneuver for performing such a removal. $\mathcal{C}$ will tell us how I and J should look like in order to go through with the removal. In the end, we will conclude that there exists an instance I from which we can remove e iff f is satisfiable.

In the following we denote by p the sub-pattern following s in the range of c.

First group of constraints of $\mathcal{C}$ say that the path to e in I must not have x_i's, $+$'s, $-$'s or m in the "holes" corresponding to //'s. Otherwise we cannot remove e.

- $(/\mathtt{s}[//\mathtt{m}//\mathtt{m}]//\mathtt{p}, \uparrow)$
  This constraint says that in I no two m nodes should be on the path from root to the e node. Otherwise, the removal would not possible. So, there is only one m node, and the root to e path is split into two halves by it.
- $(/\mathtt{s}[//\mathtt{x}_\mathtt{i}//\mathtt{x}_\mathtt{i}//\mathtt{m}//]//\mathtt{p}, \uparrow)$
- $(/\mathtt{s}[//\mathtt{m}//\mathtt{x}_\mathtt{i}//\mathtt{x}_\mathtt{i}]//\mathtt{p}, \uparrow)$
  These constraints say that in I no two x_i nodes should be in the same half on the path from root to the e node. Otherwise, the removal would not possible. The following constraints are similar.
- $(/\mathtt{s}[//\mathtt{x}_\mathtt{i}//\mathtt{x}_{\mathtt{i}-\mathtt{k}}//\mathtt{m}]//\mathtt{p}, \uparrow)$
- $(/\mathtt{s}[//\mathtt{m}//\mathtt{x}_\mathtt{i}//\mathtt{x}_{\mathtt{i}-\mathtt{k}}]//\mathtt{p}, \uparrow)$
- $(/\mathtt{s}[// + //\mathtt{m}]//\mathtt{p}, \uparrow)$
- $(/\mathtt{s}[// - //\mathtt{m}]//\mathtt{p}, \uparrow)$
- $(/\mathtt{s}[//\mathtt{m}//\mathtt{x}_\mathtt{i}// + // + //\mathtt{x}_{\mathtt{i}+1}]//\mathtt{p}, \uparrow)$
- $(/\mathtt{s}[//\mathtt{m}//\mathtt{x}_\mathtt{i}// - // - //\mathtt{x}_{\mathtt{i}+1}]//\mathtt{p}, \uparrow)$

So, for now, we know that the root-to-node path for the e node to be removed from I must be one that is exactly as the one given in the range, modulo the parts jumped by //s, that are not interfering in any way.

The next constraint says that e cannot be completely removed, as it must stay on the general path. So e will be in J on such a root to node path.

$$(\mathtt{s}//\mathtt{x}_1//\mathtt{x}_2// \ldots //\mathtt{x}_n//\mathtt{m}//\mathtt{x}_1//\mathtt{x}_2//\mathtt{x}_3// \ldots //\mathtt{x}_n//\mathtt{e}, \uparrow)$$

So one can affect c only by affecting the $+$, $-$ nodes on its root to node path.

We next add other constraints which limit the choices for root to e path in J.

- $(/\mathtt{s}//\mathtt{m}//\mathtt{m}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//\mathtt{x_i}//\mathtt{x_i}//\mathtt{m}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//\mathtt{m}//\mathtt{x_i}//\mathtt{x_i}//\mathtt{e}, \downarrow)$

These constraints say that no new m or x_i's can appear, so will still have 2 halves and one x_i in each half.

Now, we will enforce that in order to remove e, the only possible way is to "shuffle" the $+, -$ in such a way that: (a) no $+, -$ is removed, (b) for each x_i they are perfectly split between the two halves, and (c) they also obey a constraint saying that each term of f has at least one good assignment in the first half.

- $(/\mathtt{s}//+//+//\ldots//+//\mathtt{e}, \uparrow)$
- $(/\mathtt{s}//-//-//\ldots//-//\mathtt{e}, \uparrow)$

These constraints say that the n $+$'s and $-$'s remain on the path to e.

- $(/\mathtt{s}//\mathtt{x_i}//+//+//\mathtt{x_{i+1}}//\mathtt{m}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//\mathtt{x_i}//-//-//\mathtt{x_{i+1}}//\mathtt{m}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//\mathtt{x_i}//+//-//\mathtt{x_{i+1}}//\mathtt{m}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//\mathtt{x_i}//-//+//\mathtt{x_{i+1}}//\mathtt{m}//\mathtt{e}, \downarrow)$

These constraints say that no two $+$ or $-$ may appear between consecutive x-nodes in the first half.

- $(/\mathtt{s}//\mathtt{m}//\mathtt{x_i}//+//+//\mathtt{x_{i+1}}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//\mathtt{m}//\mathtt{x_i}//-//-//\mathtt{x_{i+1}}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//\mathtt{m}//\mathtt{x_i}//-//+//\mathtt{x_{i+1}}//\mathtt{e}, \downarrow)$

These constraints say that no two signs of the same kind or a $-$ before a $+$ may appear in the same interval in the lower half.

So for now we know that at least one $+$ or $-$ must move in the upper half. We will put now conditions such that this movement triggers a perfect split:

- $(/\mathtt{s}//+//\mathtt{m}//\mathtt{x_j}//+//-//\mathtt{x_{j+1}}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//-//\mathtt{m}//\mathtt{x_j}//+//-//\mathtt{x_{j+1}}//\mathtt{e}, \downarrow)$

So we know that the assignments can only be perfectly split among the 2 halves. Now, we make such that the split is acceptable iff it gives a solution to the 3CNF formula. Assume we have a term $(x_2 \vee \neg x_5 \vee x_9)$. The corresponding constraints will be

- $(/\mathtt{s}//+//\mathtt{m}//\mathtt{x_2}//+//\mathtt{x_3}//\mathtt{x_5}//-//\mathtt{x_6}//\mathtt{x_9}//+//\mathtt{x_{10}}//\mathtt{e}, \downarrow)$
- $(/\mathtt{s}//-//\mathtt{m}//\mathtt{x_2}//+//\mathtt{x_3}//\mathtt{x_5}//-//\mathtt{x_6}//\mathtt{x_9}//+//\mathtt{x_{10}}//\mathtt{e}, \downarrow)$

So at least one the variables of the term must have the good variable assignment in the first half. We can conclude that the only ways in which we can split the assignments are those making f true.

XP{/, //, *} We can rely on a similar construction, using both no-insert and no-remove constraints, to show coNP-hardness for the fragment $XP\{/, //, *\}$. However, the interest of this construction is only limited, as we prove in the following section coNP-hardness for $XP\{/, //, *\}$ under additional restrictions, namely when only one update type is allowed (see Theorem 4.9). For that, we use a more direct proof technique that relies on a recent result from [13]. □

4.4. 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.

Proof. 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). We first apply a number of

pruning steps on J, while maintaining the witness property. After obtaining an instance J' of polynomial size in the size of c and the maximal star-length from ranges, we apply similar pruning steps to the I instance. The size of (I', J') will depend on that of c and $\mathcal{C}$, and in particular on the maximal star-length of ranges.[7]

Without loss of generality we assume no-insert constraints. Let $c = (q, \downarrow)$ and let n be a node in $q(J) - q(I)$. Starting from I and J, we apply the following steps towards the pair (I', J'):

First, we select some embedding of q in J that gives n as result and we mark the nodes of this embedding. The number of marked nodes is of the order of c. We remove all the nodes that do not have a marked descendant. Observe that removing nodes from J is safe.

At a second step, for all unmarked nodes n' (i.e., nodes jumped by // axis), we change the *Id* and the label into some unique, new label (z). In other words, we remove an intermediary node and put a new one with label z instead. If n' was in no constraint range, than constraints are not affected. The only case where we can affect some constraints is if n' is in a range with distinguished node $*$. But we do not have such ranges, since we assume concrete paths. If we would however have them, since n' would be in the same range in I, we can avoid violations by simply adding the new z node as "clone" of n' in I. Observe that adding in I is perfectly safe. By these transformations, we still have a witness pair.

Next, we show that in the remaining part of J, between two consecutive marked nodes, the number of z intermediary nodes can be bounded. We use here the notion of *star length* of a path, denoting the maximal length of a chain of wildcards occurring in that path (this notion was introduced by [23]). Let m be the maximal star length for the ranges. On the z-path between two consecutive marked nodes, first we individuate the nodes that have some other outgoing branch, besides this path. We can have at most $|c|$ such z nodes. We keep those nodes, and for each two consecutive pairs of them, we shrink sequences longer than $m+1$ to $m+1$ nodes (i.e., we remove some of the z nodes). This means that the path between m_1 and m_2 has length at most $(m+1) \times |c|$.

Summing up, the maximal size of the resulting J instance (noted J') will be $(m+1) \times |c|^2$.

In similar style, starting now from this J', we can remove nodes from I, obtaining an instance I' that is polynomial in the size of $\mathcal{C}$ and J'. For more details, we refer the reader to the proof or Theorem 5.1, where similar steps are applied to the I instance starting from a given J instance.

In conclusion, we can guess a pair of instances of polynomial size 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. The proof is based on finite state automata. First, if we assume that implication does not hold and a counterexample exists, it is immediate that we can limit to counterexamples being pairs of linear paths. Then, for any constraint c, the range can be modeled as a deterministic automaton of exponential size (noted $\mathcal{A}_c$).[8] Also, the complement of a range can be modeled by deterministic automaton of exponential size (noted $\mathcal{A}_{\neg c}$). We then look at products of such automata, polynomially many if their number is bounded (for more details on the product automaton, see for instance [20]). By $\times\mathcal{A}_{\neg c_i}$ we denote the product automaton of $\mathcal{A}_{\neg c_1}, \ldots, \mathcal{A}_{\neg c_n}$.

We use the following claim to decide if implication holds:

$\mathcal{C} \not\models c$ *iff* either $\mathcal{A}_c \times (\times\mathcal{A}_{\neg c_i})$ *is not empty*, or there exist some constraints $c_1, \ldots, c_k \in \mathcal{C}$ s.t.

- $\mathcal{A}_c \times \mathcal{A}_{c_1} \times \cdots \times \mathcal{A}_{c_k} \times \mathcal{A}_{\neg c_{k+1}} \times \cdots \times \mathcal{A}_{\neg c_n}$ *is not empty, and*
- $\mathcal{A}_{c_1} \times \cdots \times \mathcal{A}_{c_k} \times \mathcal{A}_{\neg c}$ *is not empty.*

Let us assume no-remove constraints. Assuming that for some (I, J) pair, some node n from the range of c can be removed, we have two cases: (1) this node did not appear in any of the ranges of $\mathcal{C}$ (corresponding to the first case in the claim, $\mathcal{A}_c \times (\times\mathcal{A}_{\neg c_i})$ is not empty), or (2) this node did appear in some ranges of $\mathcal{C}$ (corresponding to the second case in the claim). For case (1), the root-to-node path for n denotes exactly a string which is in $\mathcal{A}_c$ (but also in all the $\mathcal{A}_{\neg c_i}$) automata. Similarly, for case (2) the two root-to-node paths for n (in I and J) denote the two strings which witness the non-emptiness of the two automata products. Similar reasoning can be applied for the other direction.

The alphabet can be considered finite since we can replace all labels that do not appear in constraints by some new label z. In conclusion, for each choice of constraints $c_1, \ldots, c_k$, we need to check the emptiness of automata of exponential size. The problem becomes solvable in polynomial time if the number of constraints and the maximal number of wildcards between two consecutive //'s are bounded by constants. □

Finally, we prove that implication in the fragment $XP\{/, [], //\}$ is coNP-hard, even when only one update type is used.

[7] The *star length* of a path denotes the maximal length of a chain of wildcards occurring in the path (this notion was introduced by [23]).

[8] We get exponential size only in the maximal number of wildcard nodes between two consecutive // edges (see [19]).

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

	$XP\{/\}$	$XP\{/, [], *\}$	$XP\{/, //, *\}$	$XP\{/, [], //, *\}$
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.

Theorem 4.9. $\models_{\sigma}^{XP\{/,[],*\}}$ *is* coNP-*hard.*

Proof. Theorem 4.3 of [13] states that, within this fragment, deciding if a pattern q is equivalent to an intersection of patterns q_i containing q is coNP-hard. By Theorem 4.4, we can reduce this equivalence problem to the implication of update constraints. For that, it suffices to consider q as the range of c, and the queries q_i as the ranges of the constraints from $\mathcal{C}$. We can then conclude that $\mathcal{C}$ implies c if and only if $q \equiv \bigcap_i q_i$. □

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. 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.

Let I be an instance such that (I, J) is a witness for non-implication. Assume c is a no-insert constraint. We apply on I the following steps:

First, for each node $n \in I$ that is also in J, and each no-insert constraint range q in which n is found in J, we chose one embedding of q and mark the nodes of that embedding, i.e., the nodes of I on which q is mapped. Node n is also mapped. At the end of this step, we have at most $|J| \times |\mathcal{C}|$ marked nodes.

At the second step, for unmarked nodes, we changed their label into some unique, new label (z). Let I' be the resulting instance. We argue that (I', J) is still a witness for non-implication. We rely on the following observations:

1. No new qualifications are introduced by changing labels into z.
2. No removed qualifications occurred for J-nodes.
3. Although we did remove some nodes from I, say nodes having label "a," since those nodes were not marked, they were not in J so it was safe for them to be removed. Similar, we may have removed qualifications for nodes not found in J, but this is also safe, by the same argument.
4. Last, we introduced some z-labeled nodes, that will disappear in J. But this cannot break no-remove constraints, since the only ranges for which those nodes may be in the result, must be ending in $/*$. And these constraints would also guard the "a" node, for which it was assumed it was not in J, so safe to remove.

At the third step, we remove all z nodes that have no marked descendant. It is immediate to check that this is safe.

Next, we show that between two consecutive marked nodes, the number of z intermediary nodes can be bounded. In fact, for all chains of z nodes longer than $m + 1$ (the star length, i.e., the maximal length of a chain of wildcards; this notion was introduced by [23]), we will try to shrink them to $m + 1$. Let m_1, m_2 be two consecutive marked nodes (descendant/ancestor, with no other marked nodes between them).

On the z-path between them, first we individuate the nodes that have some other outgoing branch, besides this path. We can have at most $|J| \times |\mathcal{C}|$ such z nodes. We keep those nodes, and for each two consecutive pairs of them, we will also keep at most $m + 1$ other z nodes. We remove all other z nodes. This means that the path between m_1 and m_2 has length at most $(m + 1) \times |J| \times |\mathcal{C}|$.

Summing up, the maximal length of a path in I' will be $m \times |J|^2 \times |\mathcal{C}|^2$. Concerning the width, the number children of the root is at most J.

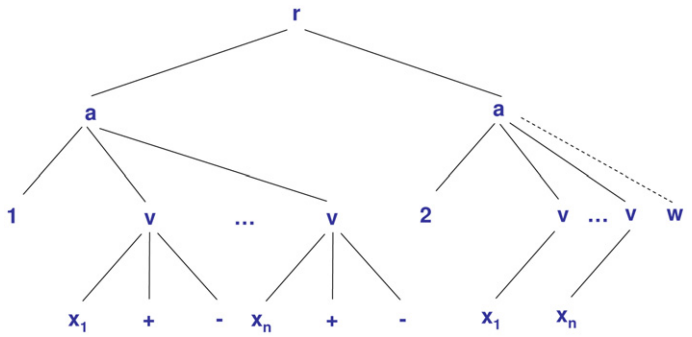

Fig. 6. Current data instance (J).

For c being no-remove, we must also mark n the removed node, and its corresponding range embeddings. The other steps are identical. □

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.*

Proof. We start with the proof for $XP\{/,[]\}$.

XP{/, []} We first discuss the case when the update type of c is *no-insert*($\uparrow$) and then, by a minor modification, adapt the proof to the case of *no-remove*($\downarrow$). The proof is by reduction to the problem of checking unsatisfiability of 3CNF formulas. Consider a 3CNF formula f with n variables $x_1 \dots x_n$. Suppose f of the form $(x_1 \vee \neg x_2 \vee x_3) \wedge \cdots$. We build a tree with update constraints, and show a constraint which is implied iff the formula is unsatisfiable.

We consider that the tree instance J has the form of Fig. 6 (ignore for now the dashed arrow to the w node). Below the root are two a branches, one numbered 1 and the other 2, each containing also some v subtrees, one per variable x_i. For brevity, from now on we refer to the first subtree as a_1 and to the second as a_2. In the a_1 subtree, each variable name x_i has its two possible truth assignment (represent resp. by $+$ and $-$) as siblings. In the a_2 subtree the variables have no siblings (i.e. no truth assignments).

We consider the following constraints in $\mathcal{C}$ (all immutable, hence pairs of $\uparrow$ and $\downarrow$ constraints):

- (/a, ⊤), (/a[/1], ⊤), (/a[/2], ⊤)
- (/a/v, ⊤)
- (/a[/1]/v[/x_1], ⊤) ... (/a[/1]/v[/x_n], ⊤)
- (/a[/2]/v[/x_1], ⊤) ... (/a[/2]/v[/x_n], ⊤)
- (/a[/1][/v[/x_1]], ..., [/v[/x_n]], ⊤)
- (/a[/2][/v[/x_1]], ..., [/v[/x_n]], ⊤)

These constraints guarantee that the general structure, besides the nodes $+$, $-$ was originally the same, and that (1) the root has no other a children, (2) each of the a nodes is numbered either by 1 or by 2, but not by both, (3) each a node has exactly one v child for each variable name, and (4) there are no other v nodes besides those.

- (/a/v[/x_i]/+, ⊤), (/a/v[/x_i]/−, ⊤)

These constraints guarantee that there is exactly one $+$ and one $-$ for each variable name x_i. Note that, while in the current tree the $+$ and $-$ were in the a_1 branch (under the v parent of x_i), in another possible instance they could have been in the a_2 branch.

- (/a[/2][/v[/x_1][/+][/−]], ↑)
  ...,
- (/a[/2][/v[/x_n][/+][/−]], ↑)

Note that, when evaluated on the current tree, all these queries give an empty answer. For that to also be the case in a previous tree, at least one of the + or − siblings of $\mathtt{x_i}$ (or both) must have belonged to the corresponding $\mathtt{x_i}$ subtree in $\mathtt{a_1}$. Or, in other words, in a previous tree, each variable name in $\mathtt{a_1}$ had at least one truth assignment associated with it.

Finally, we have one constraint for each clause in the 3CNF formula. For instance, for the formula $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge \cdots$ we have the following constraints

- $(\mathtt{a[/2][/v[/x_1][/+]][/v[/x_2][/-]][/v[/x_3][+]]}, \uparrow)$
- $(\mathtt{a[/2][/v[/x_1][/-]][/v[/x_4][/+]][/v[/x_5][+]]}, \uparrow)$

  ...

Note that, here again, when evaluated on the current tree, all these queries give an empty answer. for that to also be the case in a previous tree, for each clause of the 3CNF formula, at least one branch was not in the $\mathtt{a_1}$ subtree. Or in other words, in a previous tree, at least one satisfying assignment of each clause appeared in the $\mathtt{a_1}$ subtree.

To conclude, recall that we know (due to the previous constraints above) that, in any tree I such that $(I, J) \vdash \mathcal{C}$, each variable under the $\mathtt{a_1}$ branch had at least one truth assignment associate with it. The other one could be in $\mathtt{a_1}$ or $\mathtt{a_2}$. Note that the only case where the assignments for all variables could be really split (i.e. for each variable, one truth value is under $\mathtt{a_1}$ while the other under $\mathtt{a_2}$), is if the formula is satisfiable. In all other cases there will be at least one variable where both truth value will reside under $\mathtt{a_1}$. It follows that for $c = (\mathtt{/a[/1][/v[/+][-]]}, \downarrow)$, $\mathcal{C} \models_J c$ iff the formula is unsatisfiable. Observe that both data and constraints are polynomial in the size of the formula, while $|c|$ is constant.

We now adapt the above proof to the case when c's update type is $\uparrow$ (no-remove). For a newly introduced label $\mathtt{w}$, we add a new node to J, below a_2 (in dotted arrow). We also add the following constraints to the existing ones:

- $(\mathtt{/a/w}, \top)$
- $(\mathtt{/a[/1][/w][/v[/+][/-]]}, \top)$

We know that there can be only one $\mathtt{w}$ in a previous instance. Note that, when evaluated on the current tree, the second range query gives an empty answer.

By a similar argument, it follows that for $c = (\mathtt{/a[/1][/w]}, \uparrow)$, $\mathcal{C} \models_J c$ iff the formula is unsatisfiable.

XP{/, //, ∗} The proof is identical to that of Theorem 4.6, for this fragment. □

In order to trace the tractability boundary, we state without proof that for the most restricted fragment, $XP\{/\}$, instance-based implication is in PTIME. Intuitively, the tree structure plays no role here, each increase-only or delete-only collection being determined by the root-to-node path.

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. $\models^{XP\{/,[],*\}}_{J,\downarrow}$ *is in* PTIME.

Proof. Let $\mathcal{C} = \{(q_i, \downarrow)\}$ and $c = (q, \downarrow)$. We will construct in PTIME an instance F_J such that $\mathcal{C} \not\models_J c$ iff (F_J, J) contradicts $\mathcal{C} \models c$. 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: $\mathcal{C} \not\models_J c$ iff $q(J)$ contains some node that does not belong to $q(F_J)$.

Assume first that $q(J)$ contains some node that does not belong to $q(F_J)$. We can observe that, in this case, the pair of trees (F_J, J) is precisely a witness for $\mathcal{C} \not\models_J c$.

For the opposite implication, we show next that if c is not implied, then $q(J)$ must contain some node that does not belong to $q(F_J)$. We prove this by contradiction. Let us assume that c is not implied but $q(J) \subseteq q(F_J)$.

Let ID_J be a set of node identifiers consisting of the range nodes in J. Our first observation is that for every tree I such that (I, J) is valid, the tree F_I (constructed as above) "includes" the tree F_J. Namely, there is a homomorphism from F_J to F_I that preserve node labels, parent child relationships, and node identifiers from ID_J. Hence $q(F_J)$, restricted to the identifiers in ID_J, is a subset of $q(F(I))$, restricted to these identifiers. Since we assume that $q(J) \subseteq q(F_J)$, it follows that also $q(J)$, restricted to identifiers in ID_J, is a subset of $q(F_I)$, restricted to these identifiers.

Our second observation is that for every I, $q(F_I)$ cannot contain identifiers in ID_J that are not in $q(I)$. From this and from the fact that $q(J)$, restricted to ID_J, is a subset of $q(F_I)$, it follows that $q(I)$ contains all the identifiers in ID_J that $q(J)$ contains. It follows that for no-insert(q) to not be implied, $q(J)$ must contain some *Id* i not in ID_J. But this contradicts the assumption that $q(J) \subseteq q(F_J)$, because all the identifiers in F_J besides those in ID_J are "fresh" identifiers so this node with *Id* i cannot be in $q(F_J)$. □

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

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

Proof. We use the claim of the proof of Theorem 4.8. We describe how we can get to a setting where the claim can be directly applied. First, let (I, J) be a pair witnessing non-implication. Let n be some node that was added to the result of q, for $c = (q, \downarrow)$. We can build a new instance I' that takes an exact copy of J, except for n which is replaced with a new node n' having the same label as n. We then need to find a linear path leading to n such that n is at least in all the $\mathcal{C}$-ranges in which it was in J but it is not in the result of q. But this is exactly the setting of Theorem 4.8, and the same approach can be applied. □

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. 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 do not 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.*

To prove this result, we can use a construction almost identical to that of Theorem 5.2. While we use exactly the same J instance, since now we only have no-remove constraints, we cannot put conditions on the shape of I as easily as before. However, we manage to say what I should contain via the c constraint, which will now be as big as J. The rest of the proof is similar to that of Theorem 5.2.

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 particularly suited for hierarchically structured data. For instance, keys that are relative to a node type were introduced in [16].

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 $\mathcal{C}$:

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

(c_1 and c_3 use the previous notation, in other words having as scope the root.)

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 (discussed in Section 2.2) 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 (see Example 6.2). To address this drawback, some soundness requirements could be imposed on constraints.

Example 6.2. Suppose no one is allowed to delete appointments from persons qualified as friends, expressed as an update constraint by

$$(/\texttt{person}[/\texttt{friend}], /\texttt{appointment}, \uparrow)$$

In a sequence of updates, participants can delete a friend's `appointment` by first deleting its `friend` node, performing the deletion of the `appointment` (allowed now) and then reintroducing the `friend` qualifier. This would yield a sequence of stepwise valid instances that does not produce a valid overall pair of instances.

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 [1,8,21,22]. In the remainder 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] cannot 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 [7]. 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 $FO^2(\sim, <)$[9] 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 [11,17]) and schema information, which are often available for XML data. It is important to understand the interaction of static and dynamic constraints, and the role they can play in the implication of either update or static constraints. 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, as well as the anonymous referees of PODS 2007 and JCSS for careful proofreading and valuable comments on earlier versions of this manuscript.

References

[1] Serge Abiteboul, Bogdan Cautis, Amos Fiat, Tova Milo, Digital signatures for modifiable collections, in: Proc. of the Intl. Conference on Availability, Reliability and Security (ARES), 2006, pp. 390–399.
[2] Serge Abiteboul, Richard Hull, Victor Vianu, Foundations of Databases, Addison–Wesley, 1995.
[3] Serge Abiteboul, Luc Segoufin, Victor Vianu, Representing and querying XML with incomplete information, ACM Trans. Database Syst. 31 (1) (2006) 208–254.
[4] Serge Abiteboul, Victor Vianu, Regular path queries with constraints, J. Comput. System Sci. 58 (3) (1999) 428–452.
[5] Marcelo Arenas, Design principles for XML data, PhD thesis, 2005.
[6] Marcelo Arenas, Wenfei Fan, Leonid Libkin, Consistency of XML specifications, in: Inconsistency Tolerance, 2005, pp. 15–41.
[7] Marcelo Arenas, Leonid Libkin, XML data exchange: Consistency and query answering, J. ACM 55 (2) (2008).
[8] Mihir Bellare, Oded Goldreich, Shafi Goldwasser, Incremental cryptography: The case of hashing and signing, in: Proc. of the Intl. Cryptology Conference, 1994, pp. 216–233.
[9] Michael Benedikt, Christoph Koch, XPath leashed, ACM Comput. Surv. 41 (1) (2008).
[10] Mikolaj Bojanczyk, Claire David, Anca Muscholl, Thomas Schwentick, Luc Segoufin, Two-variable logic on data trees and XML reasoning, in: Proc. of the ACM Symposium on Principles of Database Systems (PODS), 2006, pp. 10–19.
[11] Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, Wang Chiew Tan, Reasoning about keys for XML, Inf. Syst. 28 (8) (2003) 1037–1063.
[12] Bogdan Cautis, Serge Abiteboul, Tova Milo, Reasoning about XML update constraints, in: Proc. of the ACM Symposium on Principles of Database Systems (PODS), 2007.
[13] Bogdan Cautis, Alin Deutsch, Nicola Onose, XPath rewriting using multiple views: Achieving completeness and efficiency, in: WebDB, 2008.
[14] Chee Yong Chan, Wenfei Fan, Pascal Felber, Minos N. Garofalakis, Rajeev Rastogi, Tree pattern aggregation for scalable XML data dissemination, in: Proc. of the Intl. Conference on Very Large Data Bases (VLDB), 2002, pp. 826–837.
[15] Alin Deutsch, Val Tannen, XML queries and constraints, containment and reformulation, Theoret. Comput. Sci. 336 (1) (2005) 57–87.
[16] Wenfei Fan, Leonid Libkin, On XML integrity constraints in the presence of DTDs, J. ACM 49 (3) (2002) 368–406.
[17] Wenfei Fan, Jérôme Siméon, Integrity constraints for XML, J. Comput. System Sci. 66 (1) (2003) 254–291.
[18] Georg Gottlob, Christoph Koch, Reinhard Pichler, Luc Segoufin, The complexity of XPath query evaluation and XML typing, J. ACM 52 (2) (2005) 284–335.
[19] Todd J. Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, Dan Suciu, Processing XML streams with deterministic automata and stream indexes, ACM Trans. Database Syst. 29 (4) (2004) 752–788.
[20] John E Hopcroft, Jeffrey D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison–Wesley, Reading, MA, 1979.
[21] Robert Johnson, David Molnar, Dawn Xiaodong Song, David Wagner, Homomorphic signature schemes, in: Proc. of the Cryptographers' Track – RSA Conference (CT-RSA), 2002, pp. 244–262.
[22] Silvio Micali, Ronald L. Rivest, Transitive signature schemes, in: Proc. of the Cryptographers' Track – RSA Conference (CT-RSA), 2002, pp. 236–243.
[23] Gerome Miklau, Dan Suciu, Containment and equivalence for a fragment of XPath, J. ACM 51 (1) (2004) 2–45.
[24] Gerome Miklau, Dan Suciu, Modeling integrity in data exchange, in: Proc. of the Secure Data Management Workshop (SDM), 2004, pp. 41–54.

[9] $\sim$ 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.

[25] Frank Neven, Thomas Schwentick, On the complexity of XPath containment in the presence of disjunction, DTDs, and variables, Log. Methods Comput. Sci. 2 (3) (2006).
[26] Thomas Schwentick, XPath query containment, SIGMOD Record 33 (1) (2004) 101–109.
[27] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, Daniel S. Weld, Updating XML, in: Proc. of the ACM SIGMOD Conference, 2001, pp. 413–424.
[28] Victor Vianu, Dynamic constraints and database evolution, in: Proc. of the ACM Symposium on Principles of Database Systems (PODS), 1983, pp. 389–399.
[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.