Reasoning about XML Constraints based on XML-to-relational mappings[*]

Matthias Niewerth[†]
University of Bayreuth
Lehrstuhl AI VII
matthias.niewerth@uni-bayreuth.de

Thomas Schwentick
TU Dortmund University
Lehrstuhl Informatik 1
thomas.schwentick@tu-dortmund.de

ABSTRACT

The paper introduces a simple framework for the specification of constraints for XML documents in which constraints are specified by (1) a mapping that extracts a relation from every XML document and (2) a relational constraint on the resulting relation. The mapping has to be generic with respect to the actual data values and the relational constraints can be of any kind. Besides giving a general undecidability result for first-order definable mappings and a general decidability result for MSO definable mappings for restricted functional dependencies, the paper studies the complexity of the implication problem for XML constraints that are specified by tree pattern queries and functional dependencies. Furthermore, it highlights how other specification languages for XML constraints can be formulated in the framework.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design

General Terms

Algorithms, Design, Theory

1. INTRODUCTION

Constraint languages for XML and their associated reasoning problems have been an active research area during the last ten years [4, 2, 9, 15, 10, 1, 6, 7, 8]. However, this research has not yet converged to a universally accepted model. A common feature of many existing constraint languages is that they have some means to extract a set of "groups" of nodes and data values from documents and pose restrictions on this set. We propose a simple abstraction for this two-step approach to XML constraints: we define XML constraints as a combination of (1) a mapping which defines, for each XML document, a relation over nodes and values (called X2R-mapping in the following), and (2) a relational constraint on this relation. We refer to such constraint languages as X2R-based constraint languages.

A very simple example of an X2R-based constraint is given in Figure 1. The X2R-mapping in this example uses the tree pattern in (a) to extract tuples of nodes from an XML tree. For these nodes, the target relation might contain the node itself or the data value associated with the node. In Figure 1(c), the nodes bound to x_2, x_3, x_4 and the data values of x_3 and x_4 are chosen for the target relation. A possible relational constraint, violated in the example tree, could be that the value of x_3 determines the node x_4.

This simple approach allows to study and compare constraint languages in a uniform framework, by combining appropriate mappings and (relational) constraint languages. In this paper, we initiate a systematic study of the complexity of the implication problem for X2R-based constraint languages, that is, for a given set Σ of constraints, possibly a schema D, and another constraint τ, whether all documents that satisfy all constraints from Σ (and are valid with respect to D) also satisfy τ. Besides the choice of the language for mappings, relational constraints and schemas, there is a further parameter which can influence this complexity: whether null values are allowed in relations (stemming, e.g., from partial embeddings of tree patterns).

In this paper, we only study the implication problem for X2R-based constraints with (relational) functional dependencies (FDs) and without null values. We introduce null values nevertheless, as we need them in Section 5 to show how related approaches compare to our framework. In our framework, a functional dependency is an expression of the form $Y \to B$, where Y is a set of node terms (x) and data terms ($x.@$) and B is a single node or data term. The FD of the above example would thus be written as $\{x_3.@\} \to x_4$. We show that, not surprisingly, for strong mapping languages, containing first-order logic, the implication problem is undecidable, even in the absence of a schema. However, the restriction to FDs $Y \to B$, in which B must be a node variable, is decidable for the very strong mapping language monadic second-order logic (MSO), even in the presence of a regular tree language as schema. We refer to functional dependencies of the latter kind as XKFDs (for *XML key functional dependencies*) as they can be considered as a kind of key

[*]We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599

[†]Supported by grant number MA 4938/21 of the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).

10.5441/002/icdt.2014.11

constraints.

Not surprisingly either, the complexity of the implication problem for constraints with MSO-mappings and XKFDs is non-elementary. We therefore mainly concentrate our study to simpler mapping languages based on tree patterns with, in the general case, child and descendant axis and wildcards. As schema language we only allow simple DTDs in the sense of [2]. The complexity results are summarized in Table 1.

It turns out that for X2R-constraints with general tree pattern mappings and functional dependencies the implication problem is undecidable in the presence of simple DTDs. It becomes decidable if the descendant axis is disallowed in tree patterns and in the schemaless setting without wildcards in patterns. Whether it is decidable for general patterns in the absence of schemas and for patterns without wildcard in the presence of schemas remains open.

As in the case of more general mappings, the restriction to XKFDs makes the implication problem much more feasible. It is decidable and can be solved with polynomial space for general patterns, even in the presence of simple DTDs. If wildcard or descendant axis are disallowed in the patterns or in the absence of a schema it becomes solvable in co-NP. If the descendant axis is disallowed, the problem is solvable in polynomial time in the absence of schemas or if wildcards are disallowed as well.

Besides these complexity results, presented in Section 4, we also discuss, in Section 5, how our framework compares to other proposals of XML constraint languages. Basic definitions are given in Section 2 and X2R-constraints are defined in Section 3.

We would like to thank Gaetano Geck for fruitful discussions and valuable feedback.

2. PRELIMINARIES

In this section, we fix our notation for trees, schemas and relational constraints.

Trees.

In this paper, we consider labelled directed trees with data values. To this end, we assume pairwise disjoint, infinite sets $\mathcal{V}$ of nodes, $\mathcal{D}$ of data values and $\mathcal{L}$ of labels. An *XML tree* t is a tuple $(V, E, \text{lab}, \text{dv}, <_c)$, where

- $V \subseteq \mathcal{V}$ is a finite set of *nodes*,
- $E \subseteq V \times V$ is a set of *edges*,
- $\text{lab} : V \to \mathcal{L}$ is a *labeling function*,
- $\text{dv} : V \to \mathcal{D}$ is a function assigning to every node a data value, and
- $<_c$ is a partial order that orders the children of each node linearly.

We further require that t has a unique root, denoted $\text{root}(t)$, and that all edges are directed away from $\text{root}(t)$.

We refer to the set of labels of a tree t by $\text{lab}(t)$, and to the set of data values by $\text{dv}(t)$. We write $u \sim v$ if $\text{dv}(u) = \text{dv}(v)$ for two nodes u and v.

If $(u, v) \in E$ then u is the *parent* of v and v is a child of u. The *descendant* relation is the transitive closure of E and the *ancestor* relation the reversal of the descendant relation. We denote the set of all possible trees with $\mathcal{T}$.

Our tree data model is very close to [2], but for our approach, it is convenient to represent attributes by attribute nodes as in the XML data model, such that every node has (at most) one associated data value.

Schemas.

For our investigations of the implication problem for X2R-constraints we will consider two kinds of schema languages for XML-documents. As a schema language with large expressiveness we use the class Reg of regular tree languages. However, very often schemas for XML documents only restrict the set of allowed elements in a content model in a simple fashion. We mainly concentrate on a setting where the order of siblings in an XML document is ignored and thus we use the following important restriction of DTDs: simple DTDs. We use the definition of [9], which we basically repeat here.

Given an alphabet Γ, a regular expression over Γ is called *simple*, if it is of the form $s_1 \cdots s_n$, where for each s_i, there is a letter $a_i \in \Gamma$ such that s_i is either a_i, $a_i?$, a_i^+ or a_i^* and for $i \neq j$, $a_i \neq a_j$. A *simple DTD* (sDTD) is a DTD where the right-hand-side of each production is simple.

For all our lower bound results even the following further restriction of simple DTDs suffices. In an *extremely simple DTD* (esDTD), only the set of allowed labels is fixed, that is every content model has the same regular expression $(a_1 + \cdots + a_\ell)^*$, where $\{a_1, \ldots, a_l\}$ is the set of allowed labels.

Simple DTDs have unique minimal models in the following sense as already observed and used in [9].

Lemma 2.1. *Let D be an* sDTD *and ℓ a label that occurs in some (finite) tree that conforms to D. Then there exits a unique (with respect to structure and labels) minimal tree t_ℓ such that for every tree t and every induced subtree t' with a root node labelled ℓ,*

- *t_ℓ can be obtained by removing some nodes from t', and*
- *if t' is replaced by t_ℓ in t, the resulting tree still conforms to D.*

We refer to the trees of the form t_ℓ as *minimal D-trees.* It should be noted that if a label c occurs in a minimal D-tree t_ℓ then its induced subtree in t_ℓ is just t_c.

It can easily be tested whether for some label ℓ from an sDTD the tree t_ℓ actually exists and, therefore, ℓ can occur in a (finite) model of D. We therefore assume throughout that an sDTD only contains useful labels. Furthermore, we associate with every label ℓ the set $D(\ell)$ of labels that occur in the children of the root of t_ℓ, that is, $D(\ell)$ is the set of necessary labels below ℓ-labelled nodes.

For our reasoning algorithms, we are interested in small (representations of) counter-example trees[1]. It is easy to see that t_ℓ can be of exponential size in the size of D. Thus, an sDTD alone can already enforce minimal models of exponential size. We will therefore use a compact representation of trees conforming to an sDTD D to be defined next.

For a given tree t and sDTD D we define the *D-expansion* $\widehat{t}_D$ as the tree resulting from t by application of the following process. If there is a node v with label ℓ with a child u that has a label ℓ' that is disallowed below an ℓ-node by D then $\widehat{t}_D$ is undefined. Otherwise, as long as there are nodes v with

[1]The exact framework of our reasoning algorithms will be introduced later.

some label ℓ such that for some $\ell' \in D(\ell)$ v has no child with label ℓ', a copy of $t_{\ell'}$ (as guaranteed by Lemma 2.1), in which all nodes have new, pairwise distinct data values, is added below v. If it exists, $\widehat{t}_D$ is the unique minimal tree conforming to D and containing t as a subtree. We note that in $\widehat{t}_D$, every node v can uniquely be identified by a pair (u, w), where u is a node from t and w a (possibly empty) sequence of labels from D of length at most $|D|$.

Relational constraints.

In the following, we denote sets of attributes by upper case letters Y, Z and single Attributes by upper case letters B, C.

As usual, a *functional dependency* $\rho = Y \to B$ consists of a set Y of attributes and a single attribute[2] B. It is satisfied by a relation R, if all tuples in R, which agree on the attributes in Y also agree on the attribute B.

By FD we refer to the set of functional dependencies, as a relational constraint language.

When dealing with relations which may contain null values, it can be helpful to use the more powerful ficticious functional dependencies. A *ficticious functional dependency* $\rho = Y \xrightarrow{Z} B$ consists of two sets of attributes Y, Z and a single attribute B. It is satisfied by a relation R, if all tuples, which are non-null on all the attributes of Y and Z and agree on the attributes in Y also agree on B. We note, that it is allowed that the attribute B is null in both tuples.

Another useful kind of constraints in the presence of null values are non-null constraints. A *non-null constraint* $\rho = \mathrm{NN}(Y, Z)$ consists of two sets of attributes Y and Z. It holds in a relation R, if all tuples, which are non-null in all attributes of Y are non-null in all attributes of Z.

We refer to the set of ficticious functional dependencies by FFD and to the set of non-null constraints by NN. Implication of ficticious functional dependencies and non-null constraints on relations has been investigated in [3].

3. XML-TO-RELATIONAL CONSTRAINTS

In general, an X2R-constraint (m, ρ) consists of two parts: a mapping m that maps trees to relations and a relational constraint ρ that refers to the relations yielded by m. To keep our framework flexible[3], we allow the mapping m to return null values $\perp$, where $\perp \notin \mathcal{V} \cup \mathcal{D} \cup \mathcal{L}$. For every set S we denote $S \cup \{\perp\}$ by $S_\perp$.

Informally, we require that the mapping is independent of actual data values in the sense that any (not necessarily injective) renaming of data values commutes with the mapping.

More formally, a *XML2Relational-Mapping (short: X2R-mapping)* is a function $m : \mathcal{T} \to \mathcal{P}(\mathcal{V}_\perp^\ell \times \mathcal{D}_\perp^n)$, for some ℓ and n, such that for every t and every (not necessarily injective) mapping $\delta : D \to D$ it holds that

- $m(t) \subseteq \mathcal{P}(V_\perp^\ell \times \mathrm{dv}(t)_\perp^n)$; and
- $m(\delta(t)) = \delta(m(t))$, where $\delta(t)$ results from t by renaming all data values according to δ.

[2]As usual, we could allow a set of attributes instead of the single attribute B but such FDs can always be rewritten as a set of FDs with singleton attributes.

[3]The attentive reader will notice that we do not use partial mappings and null values in the technical part of this paper. However, for the general framework we consider them important and therefore involve them in our definitions (and also in some remarks in the forthcoming secions).

Here $\mathcal{P}$ is the powerset operator. Remember that $\mathrm{dv}(t)$ is the set of data values used by t.

A tree t is *valid* with respect to an X2R-constraint $\sigma = (m, \rho)$ if $m(t) \models \rho$. In that case, we write $t \models \sigma$ and also say that t satisfies σ.

A constraint instance Σ is a set of constraints. We write $\Sigma \models \sigma$, if for every tree t for which $t \models \Sigma$ holds, also $t \models \sigma$ holds. We write $\Sigma \models_D \sigma$, where D is some schema[4], if it holds that $t \models \sigma$ whenever $t \models \Sigma$ and $t \models D$.

In Section 5 we discuss how existing notions of XML constraints can be viewed in the framework of X2R-constraints. In the rest of the paper, we will then concentrate on the specialization of the framework, where m is defined by a tree pattern and ρ is a functional dependency. In the following, we give the necessary definitions for these investigations and fix notation.

Tree patterns and tree pattern mappings.

A *tree pattern* $p = (X, A, \mathrm{lab})$ consists of

- a set X of *variables*,
- an edge relation $A = A_/ \cup A_{//}$ on variables, and
- a labeling function $\mathrm{lab} : X \to \mathcal{L} \cup \{*\}$,

such that (X, A) is a directed tree with a unique root, denoted $\mathrm{root}(p)$, such that all edges are directed away from $\mathrm{root}(p)$. In the remainder, we will often use the synonyms *tree* for XML tree and *pattern* for tree pattern, respectively.

We call edges in $A_/$ *short edges* and edges in $A_{//}$ *long edges* (depicted as double lines in figures). Intuitively, they correspond to the child axis and the descendant axis in the sense of XPath. The *wildcard* symbol $*$ is intended to match every label.

For a pattern $p = (X, A, \mathrm{lab}_p)$ and a tree $t = (V, E, \mathrm{lab}, \mathrm{dv})$ a function $\pi : X \to V$ is a *(full) embedding* of p in t if it fulfills the following conditions, for every $x, y \in X$:

1. if $\mathrm{lab}(x) \neq *$ then $\mathrm{lab}_p(x) = \mathrm{lab}(\pi(x))$;
2. if $(x, y) \in A_/$ then $\pi(x)$ is the parent of $\pi(y)$ in t;
3. if $(x, y) \in A_{//}$ then $\pi(y)$ is a descendant of $\pi(x)$ in t;
4. $\pi(\mathrm{root}(p)) = \mathrm{root}(t)$.

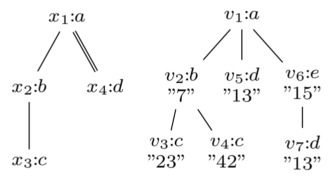
As we do not deal with partial embeddings in the main part of the paper we leave out their definition.

In the presence of an sDTD D, we can represent an embedding π of a pattern p into the expansion $\widehat{t}_D$ of a tree t, by specifying nodes in $\widehat{t}_D$ as pairs (u, w), where u is a node of t and w a label sequence as defined in Section 2. We say that such an embedding uses *relative node addresses.*

We will use compact XPath notation to denote tree patterns. For example the pattern in Figure 1(a) can be abbreviated as $/a[/b/c]//d$ or $/a[//d]/b/c$. Note that we do not care about sibling order.

Variables x in a tree pattern refer to nodes in trees, therefore we also call them *node terms.* To refer to the data value

[4]The precise kinds of schemas that we consider will be defined later on.



(a) Pattern p (b) Tree t

x_2	x_3	$x_3.@$	x_4	$x_4.@$
v_2	v_3	23	v_5	13
v_2	v_4	42	v_5	13
v_2	v_3	23	v_7	13
v_2	v_4	42	v_7	13

(c) mapping $m_p(t)$

Figure 1: Example for a mapping

of a node, we use *data terms* of the form $x.@$. A *variable term* B is a node term or a data term, its *underlying variable* is denoted by $\mathrm{var}(B)$. That is, $\mathrm{var}(x) = x$ and $\mathrm{var}(x.@) = x$. We denote the set of all data terms for a variable set X by $X^@ \stackrel{\text{def}}{=} \{x.@ \mid x \in X\}$.

If π is an embedding of a tree pattern p in a tree t, we use the abbreviation $\pi(x.@) \stackrel{\text{def}}{=} \mathrm{dv}(\pi(x))$.

With a tree pattern p one can associate an X2R-mapping in a straightforward fashion: every variable x of p can give rise to two attributes in the resulting relation, one for the node v matching x and one for its data value $\mathrm{dv}(v)$. However, in the interest of more flexibility and, often, smaller relations, we allow that the target relation consists of a subset of all attributes.

A *tree pattern mapping* $\mu = (p, W)$ consists of a tree pattern $p = (X, A, \mathrm{lab}_p)$ and a set $W \subseteq X \cup X^@$. With an embedding π of p in a tree $t = (V, E, \mathrm{lab}, \mathrm{dv}, <_c)$ we associate the tuple $\theta_{\pi,\mu}$ defined as $\theta_{\pi,\mu}(x) \stackrel{\text{def}}{=} \pi(x)$, for every $x \in W$.

For a tree pattern mapping μ and a tree t we let

$$\mu(t) \stackrel{\text{def}}{=} \{\theta_{\pi,\mu} \mid \pi \text{ a full embedding of } p \text{ in } t\}.$$

In other words, for every possible full embedding π of p in T, the relation $\mu(t)$ has one tuple corresponding to π. Figure 1 gives an example mapping for a pattern p and a tree t. For space reasons, the relation does not contain all attributes.

We denote the set of all mappings that can be specified in this way by TP (or by TP[/, //, *], if we want to stress the availability of the axes and the wildcard symbol). We denote fragments of TP by TP[/, *], TP[/, //] and TP[/], with the obvious meaning.

Tree-pattern based X2R-constraints.

As already mentioned, we will study tree-pattern based X2R-constraints in the main part of this paper. A tree-pattern based X2R-constraint $\sigma = (m, \rho)$ consists of a mapping $m = (p, Y)$ and a (possibly ficticious) functional dependency ρ.

We make use of the following (hopefully) intuitive notation. We specify p by an XPath expression in simplified syntax. The pattern positions that correspond to (node or data) variables in Y are succeeded by a variable name in brackets. The set Y contains both the node and the data variable for every variable name occuring in the expression.

Example 1. We use the tree of Figure 2 as a small example document, which contains user names of persons. A possible constraint that one might want to require is, that each user-id uniquely identifies a person, i.e. there are no two persons with the same user-id. In our framework, we can express

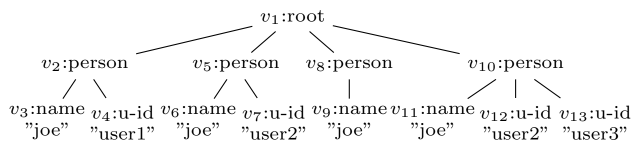


Figure 2: Example XML-Document

	x_p	x_u	$x_u.@$
1:	v_2	v_4	user1
2:	v_5	v_7	user2
3:	v_{10}	v_{12}	user2
4:	v_{10}	v_{13}	user3

Figure 3: Mapping result of σ_{user}

this constraint as

$$\sigma_{\text{user}} = (//\text{person}\langle x_p\rangle/\text{u-id}\langle x_u\rangle, \{x_u.@\} \to x_p).$$

The pattern selects all pairs (v_1, v_2), where v_2 is a child with label u-id of a node v_1 with label person. Figure 3 shows the relevant part of result of the mapping. The constraint is not satisfied due to tuples 2 and 3.

For readability, we drop the set notation from functional dependencies in the (very common) case of singleton sets. That is, we can denote the example constraint by $\sigma_{\text{user}} = (//\text{person}\langle x_p\rangle/\text{u-id}\langle x_u\rangle, x_u.@ \to x_p)$.

When we deal with functional dependencies satisfaction is defined with respect to full embeddings only, whereas for ficticious functional dependencies and non-null constraints satisfaction is defined with respect to partial embeddings. That is, an FD ρ is satisfied by a tree t if $\mu(t) \models \rho$ and an FFD or NN is satisfied by a tree t if $\mu_\perp(t) \models \rho$, where $\mu_\perp$ is defined wrt partial embeddings instead of full embeddings. Intuitively $\mu_\perp$ maps non-existing sub-trees to null values, where μ just ignores such tuples altogether.

We note that an FD $\sigma = (m, Y \to B)$, with $m = (p, Z)$ is equivalent to the FFD $\sigma = (m, Y \xrightarrow{Z} B)$. Thus we can evaluate FDs, FFDs and NNs together by converting all FDs to FFDs.

We call an FD $\sigma = (m, Y \to B)$, in which B is a node variable a *XML-key functional dependency*[5] *(XKFD)*. The set of all XKFDs is denoted by XKFD.

For an X2R-mapping language $\mathcal{M}$ and a relational constraint language $\mathcal{C}$ we denote the resulting set of X2R-constraints by $\mathrm{XC}(\mathcal{M}, \mathcal{C})$. For example, XC(TP, FD) stands for the class of constraints, yielded by tree patterns and functional dependencies.

4. REASONING

In this section, we investigate the complexity of the implication problem for X2R-constraints.

For a set Σ of X2R-constraints and a single X2R-constraint τ we write $\Sigma \models \tau$ if for every tree t, $t \models \Sigma$ implies $t \models \tau$. If D is a schema, we write $\Sigma \models_D \tau$ if for every tree t with $t \models D$, $t \models \Sigma$ implies $t \models \tau$.

Of course, the complexity may depend on the actual choice of the allowed kinds of X2R-mappings, relational constraints

[5]The name stems from the fact that these FDs very closely correspond to XML key constraints.

	TP[/]		TP[/, ∗]		TP[/, //]		TP[/, //, ∗]	
	XKFD	FD	XKFD	FD	XKFD	FD	XKFD	FD
without DTD	in P	in P	in P	in P	co-NP	co-NP	co-NP	co-NP-hard
simple DTD	in P	in P	co-NP	co-NP-hard in EXPTIME	co-NP	co-NP-hard	PSPACE	undecidable

Table 1: Complexity Results for the implication Problem. Complexities shown in gray color are implied by other entries of the table.

and schema languages, therefore the implication problem has three parameters, $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{S}$.

XC-Imp($\mathcal{M}, \mathcal{C}, \mathcal{S}$)	
Given:	A set Σ of constraints and a single constraint σ from XC($\mathcal{M}, \mathcal{C}$), and a schema D from schema language $\mathcal{S}$.
Question:	Does $\Sigma \models_D \tau$?

We will also consider the implication problem (that is, whether $\Sigma \models \tau$) in which no schema is given. We denote it by XC-Imp($\mathcal{M}, \mathcal{C}$).

A *counter-example* for an instance (Σ, τ, D) is a finite tree t with $t \models \Sigma$, $t \not\models \tau$, and $t \models D$.

We will restrict to implication problems, where the relational constraints are functional dependencies and we also study the special case of XKFDs.

We start with general upper and lower bounds, using first-order logic (FO) and monadic second-order logic (MSO) as the mapping language and the regular tree languages $\mathcal{S}$ as schemas. We consider MSO logic over a signature with the edge relation S, the children order $<_c$, and a unary relation P_a, for every symbol a. For FO logic we assume also the binary descendant relation.

An MSO formula Ψ over trees with free variables defines a mapping

$$m_\Psi(t) = \{(x_1, x_1.@, \ldots, x_n, x_n.@) \mid t \models \Psi(x_1, \ldots, x_n)\},$$

where $x_1, \ldots, x_n$ are the free variables of Ψ.

As MSO formulas can *not* refer to data values MSO-defined mappings are X2R-mappings.

Theorem 4.1.

(a) XC-Imp *(MSO,XKFD,Reg) is decidable.*

(b) XC-Imp *(FO,FD) is undecidable.*

A proof sketch is given at the end of Subsection 4.1.

Theorem 4.1 shows that the restriction to XKFDS yields a decidable implication problem, even for very powerful mapping languages. However, the complexity of XC-Imp(MSO, XKFD, Reg) is non-elementary, as this already holds for the satisfiability problem for first-order logic on strings [13].

In the remainder of the paper, we restrict our attention to more tractable instances of the implication problem, based on tree pattern mappings. We investigate the complexity of XC-Imp(TP, FD, sDTD) and XC-Imp(TP, XKFD, sDTD) as well as of implication problems based on more restricted tree patterns and/or without schemas.

More precisely, we prove the complexity results stated in the following theorem and summarized in Table 1. All lower bounds (including the undecidability result) in the presence of schemas already hold for esDTDs.

Theorem 4.2.

(a) XC-Imp(*TP*[/], *FD*, sDTD) *and* XC-Imp(*TP*[/, ∗], *FD*) *can be solved in polynomial time.*

(b) The following implication problems are complete for co-NP:

- XC-Imp(*TP*[/, //], *FD*),
- XC-Imp(*TP*[/, //], *XKFD*),
- XC-Imp(*TP*[/, //], *XKFD*, sDTD),
- XC-Imp(*TP*[/, ∗], *XKFD*, sDTD), *and*
- XC-Imp(*TP*, *XKFD*).

(c) XC-Imp(*TP*[/, ∗], *FD*, sDTD) *is co-NP-hard and can be solved in exponential time.*

(d) XC-Imp(*TP*, *XKFD*, sDTD) *is PSPACE-complete.*

(e) XC-Imp(*TP*, *FD*, sDTD) *is undecidable.*

All upper bounds stated in Theorem 4.2 are based on counter-examples. In some cases counter-examples are computed by chase algorithms, in others they are non-deterministically guessed and the bound follows by a "small or simple" counter-example property. We prove the upper bounds based on chase algorithms in Subsection 4.2, those based on counter-example properties in Subsection 4.3, and the lower bounds in Subsection 4.4. As a tool for both kinds of upper bounds we introduce the notion of witness pairs in Subsection 4.1 and show that they can be computed in polynomial time.

4.1 Witness pairs

Informally, a witness pair (π_1, π_2) for a tree t and a pattern based X2R-constraint $\sigma = (p, Y \to B)$ is a pair of embeddings of p into t that shows that σ does not hold in t.

In the following we denote the subpattern of a tree pattern p that is rooted at some node z of p by p_z.

Let $\sigma = (p, Y \to B)$ be an X2R-constraint, z a node of p, and t a tree. Let π_1, π_2 be two embeddings of p_z in t and let Z be the set of variable terms from $Y \cup B$, whose underlying variables occur in p_z. We call (π_1, π_2) a *z-witness pair for σ in t* if

- for every $C \in Y \cap Z$ it holds $\pi_1(C) = \pi_2(C)$ and
- if $B \in Z$, then $\pi_1(B) \neq \pi_2(B)$.

A *witness pair for σ in t* is a root(p)-witness pair for σ in t.

Note that in a z-witness pair for subpatterns not containing B, both embeddings of the subpattern may be identical.

The significance of witness pairs is illustrated by the following lemma which is straightforward to show.

LEMMA 4.3. *For a tree t and an X2R-constraint σ it holds $t \models \sigma$ if and only if there does* not *exist any witness pair for σ in t.*

In the presence of an sDTD D, witness pairs for a tree of the form $\widehat{t}_D$, for some tree t, are specified by embeddings with relative node addresses.

The following lemma will be useful both for chase-based as well as for counter-example based algorithms. It shows that even for the most general kind of X2R-constraints considered, (1) it can be checked in polynomial time whether a constraint holds in a given tree, and (2) if the constraint does *not* hold, a witness pair can be computed in polynomial time.

LEMMA 4.4. *There is a polynomial time algorithm that tests whether $t \models \sigma$ for trees t and constraints $\sigma \in XC(TP, FD)$ and computes a witness pair (π_1, π_2) if $t \not\models \sigma$.*

PROOF. The algorithm is an adaptation of the algorithm in [12], which computes whether a tree pattern can be embedded in a tree t and follows a simple dynamic programming approach. It computes, in a bottom-up fashion, a ternary relation W that contains all triples (u, v, z) of nodes u, v of t and a node z of p, for which there exists a z-witness pair (π_1, π_2) such that $\pi_1(z) = u$ and $\pi_2(z) = v$.

We explain, how $(u, v, z) \in W$ can be decided, once W is computed for all triples (u', v', z') with nodes u' below u, v' below v and pattern nodes z' below z. The tuple (u, v, z) is added to W, if all the following conditions hold.

(1) $\text{lab}(z) = *$ or $\text{lab}(u) = \text{lab}(v) = \text{lab}(z)$.

(2) If Y contains z then $u = v$.

(3) If Y contains $z.@$ then $u \sim v$.

(4) If B is z then u and v are different nodes.

(5) If B is $z.@$ then u and v carry different data values, that is $u \not\sim v$.

(6) for every $A_{/}$-child z' of z, there is a child u' of u and a child v' of v such that $(u', v', z') \in W$.

(7) for every $A_{//}$-child z' of z, there are nodes u' strictly below u and v' strictly below v such that $(u', v', z') \in W$.

It is easy to prove by induction on the depth of subpatterns that the final relation W exactly contains those triples (u, v, z) of nodes u, v of t and a node z of p, for which there exists a z-witness pair (π_1, π_2) such that $\pi_1(z) = u$ and $\pi_2(z) = v$.

This algorithm can be performed in $\mathcal{O}(|t|^4|p|)$ steps and thus in polynomial time.

Therefore, by Lemma 4.3, $t \not\models \sigma$ holds, if and only if $(\text{root}(t), \text{root}(t), \text{root}(p)) \in W$. With the help of W it is straightforward to construct a witness pair (π_1, π_2) in a top down fashion from W if $t \not\models \sigma$. □

We note that the running time of the above algorithm can be improved by computing another relation W' containing all triples (u, v, z) for which there exists a z-witness pair (π_1, π_2) such that $\pi_1(z) = u'$ and $\pi_2(z) = v'$, for some nodes u' below u and v' below v.

The following lemma will be often used in proofs. We call a tree t *π-diverse*, for a witness pair $\pi = (\pi_1, \pi_2)$ for some σ in t if all nodes outside the range of π carry pairwise distinct data values that are different from the data values of the nodes in the range of π.

LEMMA 4.5. *If t is a counter example tree for some instance (Σ, τ, D) of* XC-IMP(*TP*, *XKFD*, sDTD) *with a witness pair π with respect to τ, then, by changing data values in t, a π-diverse counter-example t' for (Σ, τ, D) can be obtained.*

PROOF SKETCH. Let t' be an arbitrary π-diverse tree obtained from t by changing data values outside the range of π. As π is not changed it remains a witness pair for τ in t'. On the other hand, as no new equalities between data values are introduced, all XKFDs from Σ still hold in t'. □

PROOF OF THEOREM 4.1. Statement (b) can be shown by an easy reduction from XC-IMP(TP, FD, sDTD), which is undecidable by Theorem 4.2 (e). From a given tree-pattern based instance (Σ, τ, D) it constructs an instance (Σ', τ') as follows: the patterns from Σ and τ are simply translated into FO formulas. That only valid trees for D are considered can be enforced by some additional constraint with an FO formula $\varphi(x)$ that selects all nodes if D is not satisfied and no node if D is satisfied and the constraint $(\emptyset \to x)$ that ensures that D is either satisfied or t has at most one node.

Towards (a), let (Σ, τ, S) be an instance of XC-IMP (MSO, XKFD,Reg), where $S \in \text{Reg}$ is a regular tree language.

Let us assume that t is a counter-example for Σ and τ and that $\pi = (\pi_1, \pi_2)$ is a witness pair for $t \not\models \tau$ and, by Lemma 4.5, that t is π-diverse. That is, if $\Sigma \not\models_S \tau$, then there is a counter-example in which at most n data values (in the range of π_1 and π_2) may occur twice, all other data values occur exactly once.

It is easy to construct an MSO formula φ that expresses that a given tree t (without data values) can be extended to a counter-example t' for Σ and τ by assigning at most n *special* data values that occur at more than one node and an arbitrary number of data values that occur only once. To this end, φ, existentially quantifies sets $X_1, \ldots, X_n$ with the understanding that a node in X_i carries the i-th special data value and a node that is in none of these sets carries a data value that occurs nowhere else. Furthermore, φ expresses that $t' \models D$, $t' \models \Sigma$ and $t' \not\models \tau$, each of which is straightforward with the help of the sets X_i.

The decidability of XC-IMP (MSO,XKFD,Reg) thus follows from the decidability of the finite satisfiability problem for MSO logic on trees [14]. □

4.2 Upper bounds based on the chase

Our chase algorithms directly operate on trees, not on relations. A discussion of an alternative approach that is based on the relations obtained by X2R-mappings will be given at the end of this subsection.We next describe a basic algorithm in which the tree is always explicitly available. It will be used for the polynomial time upper bound for XC-IMP(TP[/, *], FD). Later on, we will also use algorithms that involve implicit representations of trees. The basic tree chase algorithm might be of interest on its own, for example to repair XML trees violating constraints.

The basic chase algorithm consists of three parts: (1) the computation of an initial tree, (2) the actual tree chase and (3) a subprocedure for the propagation of the merge of two nodes for (2). As the definition of the initial tree can be

Algorithm 1 Tree Chase

```
1: function CHASE($t, \Sigma$)
2:    while $\exists \sigma = (p, Y \to B) \in \Sigma.\ t \not\models \sigma$ do
3:       $(\pi_1, \pi_2) \leftarrow$ witness-pair($t, \sigma$)
4:       merge($t, \pi_1(B), \pi_2(B)$)
5:    end while
6: end function
```

Algorithm 2 Merge two nodes

```
1: function MERGE($t, v_1, v_2$)
2:    if $v_1 = v_2$ then return
3:    if $v_1 = \text{root}(t) \vee v_2 = \text{root}(t)$ then fail
4:    merge($v_1.@, v_2.@$)
5:    if $\text{lab}(v_1) \neq \text{lab}(v_2)$ then
6:       if $\text{lab}(v_1) = \#$ then $\text{lab}(v_1) \leftarrow \text{lab}(v_2)$
7:       else if $\text{lab}(v_2) = \#$ then $\text{lab}(v_2) \leftarrow \text{lab}(v_1)$
8:       else fail
9:    end if
10:   merge($v_1, v_2$)
11:   merge($t$, parent($v_1$), parent($v_2$))
12: end function
```

easily decribed with the help of (3), we postpone it until after the description of (2) and (3).

Let $I = (\Sigma, \tau)$ be an instance of XC-IMP(TP[/, *], FD) with $\tau = (p, Y \to B)$. We already note that the initial tree, t_τ for the basic tree chase might have nodes labelled by # that indicate that the label of that node has not yet been fixed by the algorithm and still may match any (but only one) label.

Algorithm 1 implements (2) and uses the merge algorithm given as Algorithm 2 for (3).

The tree chase algorithm works similarly as the relational chase. Starting from a tree that does not satisfy τ, it applies chase steps as long as there exists a dependency $\sigma = Y \to B \in \Sigma$, that is not satisfied by the current tree. Whether σ is satisfied in the current tree is tested by the algorithm of Lemma 4.4. A single tree chase step either merges two nodes or identifies two data values, depending on whether B is a node or a data term, and based on the witness pair yielded by the test algorithm.

The identification of two data values d_i and d_j is simply done by replacing all occurrences of d_j by d_i, and it does not matter which is replaced by which. The merge of two different nodes v_1 and v_2 requires a bit more care: First of all, it is only possible if the labels of v_1 and v_2 are compatible, which is the case if they are equal or one of them is the wildcard label #. If the labels are compatible, the nodes can combined into one node which gets all children of v_1 and v_2 as children. However, unless v_1 and v_2 have the same parent, their parents have to be merged recursively. Otherwise the structure would no longer be a tree. This is exactly the point, where the tree chase differs from the relational chase. It should be noted that, as we apply the tree chase only in the context of tree patterns without descendant axis, only nodes of the same depth need to be merged.

Next, we define the initial tree t_τ for the basic tree chase for a given instance $I = (\Sigma, \tau)$ with $\tau = (p, Y \to B)$. Intuitively, it is minimal with the property $t_\tau \not\models \tau$. To this end, let t_1 and t_2 be two copies of p (which use node ids from $\mathcal{V}$ instead of variables from X) in which all data values are

Algorithm 3 Algorithm for XC-IMP(TP[/, *], FD)

```
1: Compute initial tree $t_\tau$
2: if chase($t, \Sigma$) fails then Output "Yes"
3: $t :=$ chase($t, \Sigma$)
4: if $t \models \tau$ then Output "Yes" else Output "No"
```

distinct (every data value occurs at most once in $t_1 \cup t_2$) and π_1 and π_2 be the canonical embeddings of p in t_1 and t_2, respectively. Every node in t_1 and t_2 whose corresponding node in p has a wildcard label $*$ is labelled by #. The tree t_τ results by merging the roots of t_1 and t_2 and all pairs $(\pi_1(z), \pi_2(z))$, for which z occurs as a node term in Y and it identifies all pairs of data values $(\pi_1(z).@, \pi_2(z).@)$, for which $z.@$ is a data term in Y. By applying the node merges the embeddings π_1 and π_2 yield two embeddings π_1' and π_2' such that (π_1', π_2') is a witness pair for t_τ and τ.

The decision algorithm for XC-IMP(TP[/, *], FD) is given as Algorithm 3.

Example 2. An example run of the chase algorithm is depicted in Figure 4. Starting from the initial tree, the run corresponds to testing the implication of $\{\sigma_1, \sigma_2\} \models \sigma_3$, where all dependencies use the tree pattern

$$p = /\text{root}/\text{person}\langle x_p \rangle[/\text{name}\langle x_n \rangle]/\text{u-id}\langle x_u \rangle$$

and the functional dependencies are $\sigma_1 = (p, x_u.@ \to x_u)$, $\sigma_2 = (p, x_p \to x_n)$ and $\sigma_3 = (p, x_u.@ \to x_n.@)$.

Intuitively, σ_1 expresses that u-ids are unique, σ_2 that every person only has one name, and σ_3 whether every u-id has exactly one associated name. Note that the constraints only apply to person, name and u-id nodes, where the person has at least one name and at least one uid, as other nodes are not contained in the mapping.

The chase merges v_4 and v_8, as σ_1 enforces them to be equal. The recursive call in Line 11 of the merge function unifies v_2 and v_6 to restore the tree structure. Finally v_3 and v_7 and their data values are identified, as σ_2 is now violated. Note while in this case there exists only one possible run of the chase algorithm, in general there can be many runs, which only differ in the order in which the rules are applied. In the resulting tree, $\{\sigma_1, \sigma_2\}$ is satisfied, as well as σ_3. We will see in the proof of Proposition 4.7 that this implies $\{\sigma_1, \sigma_2\} \models \sigma_3$.

Before we state the complexity result for XC-IMP(TP[/, *], FD), we first state the correctness of Algorithm 3.

PROPOSITION 4.6. *Algorithm 3 terminates for every instance $I = (\Sigma, \tau)$ of* XC-IMP(*TP*[/, *], *FD*). *It answers "Yes" if and only if $\Sigma \models \tau$.*

The proof is by an induction on the number of chase steps. and will be given in the full version of the paper. We get the following easy corollary.

PROPOSITION 4.7. XC-IMP(*TP*[/, *], *FD*) *can be solved in polynomial time.*

PROOF. As the algorithm is correct and witness pairs can be computed in polynomial time (Lemma 4.4) and there are at most linearly many merge steps the algorithm always terminates and only needs polynomial time. □

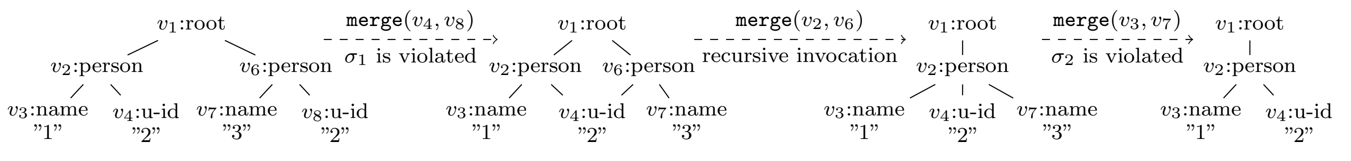


Figure 4: Example run of the chase algorithm.

The tree chase can be extended in the presence of sDTDs, however the definition of the initial tree has to be adapted[6], as this should conform to D. This modification might involve replacing leave nodes with a label ℓ by trees t_ℓ but also the insertion of additional trees of the form t_ℓ below inner nodes of t_τ and the merge of two sibling nodes if the sDTD only allows one child with their label. Another difference to the schema-free case is that we apply the chase to a set T of trees that results from the initial tree by replacing #-labels in all possible ways. If the modified initialization is successful then during the tree chase only D-valid trees will be constructed and the correctness proof is similar to the one of Proposition 4.6.

Proposition 4.8. *For every instance* $I = (\Sigma, \tau, D)$ *of* XC-Imp($TP[/,*], FD$, sDTD)*, Algorithm 3 with the modified initialization terminates for some tree t in T and answers "Yes", if and only if* $\Sigma \models_D \tau$.

However, as T might consist of an exponential number of trees of exponential size (in the size of D and τ), Proposition 4.8 does not immediately yield a polynomial time algorithm. However, we get the following result.

Proposition 4.9. XC-Imp($TP[/,*], FD$, sDTD) *can be solved in exponential time.*

For XC-Imp(TP[/], FD, sDTD) we can do better by using a condensed representation of trees that avoids the exponential blowup that might be caused by the sDTD.

Proposition 4.10. XC-Imp($TP[/], FD$, sDTD) *can be solved in polynomial time.*

At the end of this subsection, we want to discuss an alternative approach to the tree based chase. We now sketch an (exponential time) chase, which works on the produced relation(s) instead of the tree.

For simplicity, we assume, that we only have to deal with one relation R, because all functional dependencies use the same tree pattern p. Without proof, we note that this can be enforced by converting the FDs to FFDs and merging all patterns to one "universal" pattern.

The chase based on R needs to incorporate the following constraints, which are implicit, due to the tree structure of our data model:

- every tree has a unique root
- every node (except the root) has a unique parent
- every node has a unique data value
- join dependencies corresponding to branchings in the pattern[7]
- inclusion dependencies corresponding to inclusion of subpatterns

[6]And we will see soon that there is more than one initial tree.
[7]We will not introduce join and inclusion dependencies formally.

The first 3 constraints can be described by relational functional dependencies. For details see [9]. For the other constraints, we just give two examples. Let p be the tree pattern $/a[/b]/c$. Due to the branching structure of trees, the join dependency $\sigma_{\bowtie} = (/a\langle x\rangle[/b\langle y\rangle]/c\langle z\rangle, \{x,y\} \bowtie \{x,z\})$ holds for all trees. Let now p be the pattern $/a[/b]/b/c$. Due to the inclusion of sub-patterns of p, the inclusion dependency $\sigma_{\subseteq} = (/a[/b\langle x\rangle]/b\langle y\rangle/c, y \subseteq x)$ holds for all trees.

The standard chase algorithm has an exponential worst case running time when these constraints are added. In [9] it is shown how the join dependencies can be incorporated into the chase without an exponential blow-up. However, the incorporation of the implicit inclusion dependencies seems to be harder, which is one reason, why we chase directly on trees.

4.3 Upper bounds based on simple counter examples

For counterexample based proofs the following lemma is useful. It establishes small counterexample properties for various kinds of constraints. By leaves(t), we denote the set of leaves of a tree t.

Lemma 4.11. *Let* $\Sigma \subseteq XC(TP, FD)$ *be a set of constraints and* $\tau = (p, Y \to B)$ *be a constraint. If there is a tree t with* $t \models \Sigma$ *and* $t \not\models \tau$ *then there is a tree* t' *with*

(1) $t' \models \Sigma$ *and* $t' \not\models \tau$*;*

(2) $|\operatorname{leaves}(t')| \leq 2|\operatorname{leaves}(p)|$*;*

(3a) if all tree patterns are from $TP[/,*]$ *then* $\operatorname{depth}(t') \leq \operatorname{depth}(p)$*;*

(3b) if all tree patterns are from $TP[/,//]$ *then* $\operatorname{depth}(t') \leq 8\operatorname{depth}(p)$*;*

(3c) if all FDs are XKFDs then $\operatorname{depth}(t') \leq 8m \operatorname{depth}(p)$*, where m is the maximal depth of all patterns in* Σ*.*

Conditions (1), (2) and (3b) can even be guaranteed in the presence of esDTD*s.*

Proof. Let t be a tree with $t \models \Sigma$ and $t \not\models \tau$. Then there is some witness pair (π_1, π_2) for $t \not\models \tau$. Let P be the set of nodes of t to which some pattern node is mapped via π_1 or π_2.

We first describe the construction of a tree t'_1 fulfilling (1) and (2) and, if all patterns are from TP[/, $*$] also (3a).

Let t'_1 be the tree obtained from t by removing all nodes that are not in P and not ancestors of nodes in P. It is straightforward that $t'_1 \models \Sigma$, $t'_1 \not\models \tau$ and $|\operatorname{leaves}(t'_1)| \leq 2|\operatorname{leaves}(p)|$. Furthermore, if all patterns are from TP[/, $*$], then $\operatorname{depth}(t') \leq \operatorname{depth}(p)$. Thus, t'_1 fulfils (1), (2) and (3a). The construction of t'_1 is not affected if an esDTD has to be respected, as it does not change the set of labels of the tree (unlike the following two constructions).

If all patterns are from TP[/, //] we can construct another tree t'_2 from t'_1 as follows. Let P' contain all nodes from P and all nodes of t that are lowest common ancestors of at least two nodes of P. Clearly $|P'| \leq 2|P| \leq 4|p|$. To obtain t'_2, we replace in t'_1 all maximal paths of nodes that are not in P' by a path of length 2 whose single intermediate node carries a new label # that does not occur in any pattern of Σ. By construction, $|\text{leaves}(t'_2)| \leq |\text{leaves}(t'_1)| \leq 2|\text{leaves}(p)|$ and $t'_2 \not\models \tau$. On the other hand, if all patterns in Σ are from TP[/, //], every embedding of a pattern in t'_2 is also an embedding in t and therefore $t'_2 \models \Sigma$. This is, because an embedding of a pattern without wildcards can only "bridge" the gaps introduced by the new symbols # with the help of decendant edges. Finally, the depth of t'_2 is at most twice the size of P' and thus $\text{depth}(t'_2) \leq 8\,\text{depth}(p)$. Thus, t'_2 fulfils (1), (2) and (3b).

Let t'_3 be the tree obtained from t'_1 by replacing every maximal paths of length $> m$ of nodes that are not in P' by a path of length m in which every node gets a separate new data value and is labelled with a new label # that does not occur in any pattern of Σ. It is easy to see that this transformation does not introduce any violations of any XKFDs from Σ (as the new paths do not match any subpatterns that were not matched before by the replaced path), and thus, t'_3 is a counter-example tree of depth $\leq 8m\,\text{depth}(p)$. □

By combining Lemmas 4.4 and 4.11 we get the following upper bounds.

PROPOSITION 4.12. *The following implication problems are in co-NP.*

(a) XC-IMP(*TP*[/, //], *FD*)

(b) XC-IMP(*TP*[/, ∗], *FD*, ESDTD)

(c) XC-IMP(*TP*, *XKFD*)

PROOF SKETCH. Let in the following always $I = (\Sigma, \tau)$ be an instance of the implication problem at hand with $\tau = (p, Y \rightarrow B)$ and D an ESDTD, in case of (b). Lemma 4.11 guarantees for all three cases (a), (b), and (c) that, if there is a counter-example tree t to I at all, there is one of depth in $\mathcal{O}(|\Sigma||\tau|)$ and with a number of leaves in $\mathcal{O}(|\tau|)$. This yields immediate NP-algorithms for the complement of each of the three implication problems: guess a tree t that obeys the depth and width bounds of Lemma 4.11 and verify whether it is a counter-example to I. Thus, the co-NP upper bound follows in all three cases. □

By applying more involved proofs we get the following upper bounds.

PROPOSITION 4.13.

(a) XC-IMP(*TP*[/, //], *XKFD*, SDTD) *is in co-NP.*

(b) XC-IMP(*TP*, *XKFD*, SDTD) *is in PSPACE.*

PROOF IDEA. Both upper bounds rely on the same basic idea. First, it is easy to see that if an instance (Σ, τ, D) has a counter-example then it has one of the form $\widehat{t}_D$, for some tree t with at most $2|\tau|$ leaves. However, it is not a priori clear there is such a t of polynomial size. We associate with a counter-example tree t and a witness pair $\pi = (\pi_1, \pi_2)$ a *skeleton* $s_{I,\pi}(t)$ which basically consists of all nodes from the range of π and all branching nodes and, for every path between these *special nodes*, additional information on the possible matches of certain partial patterns (and, actually, pairs of partial patterns) of Σ. We show that,

- for every $\sigma \in \Sigma$, $s_{I,\pi}(t)$ has a witness pair for σ (in a sense that will be defined later) if and only if t has a witness pair for σ, and
- $s_{I,\pi}(t)$ is of polynomial size in $|\Sigma|$, $|\tau|$ and $|D|$.

The algorithm for (b) is a non-deterministic polynomial space algorithm that guesses t, computes $s_{I,\pi}(t)$ and verifies that $s_{I,\pi}(t) \models \Sigma$ and $s_{I,\pi}(t) \not\models \tau$.

For (a) we show by an automata theoretic argument that in the absence of wildcards in patterns, the length of paths between special nodes in a minimal counter-example can be polynomially bounded. The details of the proof will be given in the full version of the paper. □

4.4 Lower bounds

The co-NP lower bounds in the following proposition are all by reduction from SAT to the complement of the respective implication problem.

PROPOSITION 4.14. *The following implication problems are co-NP-hard.*

(a) XC-IMP(*TP*[/, ∗], *XKFD*, ESDTD)

(b) XC-IMP(*TP*[/, //], *XKFD*)

PROOF. Both proofs are by reductions from 3-SAT to the complement of the implication problem. The algorithmic problem 3-SAT asks whether a given propositional formula in 3-CNF is satisfiable. A propositional formula in 3-CNF is a conjunction $\varphi = C_1 \wedge \cdots \wedge C_m$ of clauses, over some variables $y_1, \ldots, y_n$, where each clause $C_i = \ell_{i1} \vee \ell_{i2} \vee \ell_{i3}$ is a disjunction of three literals.

Let a 3-CNF formula $\varphi = C_1 \wedge \cdots \wedge C_m$ with variables $y_1, \ldots, y_n$ and clauses of the form $C_i = \ell_{i1} \vee \ell_{i2} \vee \ell_{i3}$ be given. An implication instance (D, Σ, τ) for the reduction for (a) is constructed from φ as follows. The idea for the reduction is to associate truth assignments θ with 0-1-labelled paths such that $\theta(y_i) = 1$ iff the i-th symbol is 1. Thus, first of all, the ESDTD D enforces the alphabet $\{0, 1\}$.

For every clause C_i, we add a XKFD σ_i with pattern p_i to Σ that states that the last node of a path of length n is non-branching if the truth assignment of that path *fails* to satisfy C_i. That is, if there is a path of length n that does not match any pattern p_i (and thus its corresponding truth assignment satisfies all constraints) then a counterexample to $\Sigma \models \tau$ can be constructed by branching at its n-th node. The overall effect is that τ is implied by Σ if and only if there is no satisfying truth assignment for φ.

We now describe the construction in more detail.

The target dependency τ is defined as the XKFD

$$\tau \stackrel{\text{def}}{=} (/{*}/{*}/\ldots/{*}\langle x\rangle/{*}\langle y\rangle, x \rightarrow y)$$

with $n+1$ consecutive $*$ positions, stating that a node at depth n can have only one child node. For every i, let σ_i be the XKFD

$$\sigma_i = (/\alpha_{ij}/\ldots/\alpha_{in}\langle x\rangle/{*}\langle y\rangle, x \rightarrow y),$$

where α_{ij} is 0 if y_j occurs in C_i, 1 if $\neg y_j$ occurs in C_i and $*$, otherwise.

The reduction can be carried out in polynomial time. It remains to prove that φ is satifiable, if and only if $\Sigma \not\models_D \tau$.

(if): Let us assume $\Sigma \not\models_D \tau$. By the proof of Lemma 4.11 and as D allows that 0-labelled and 1-labelled nodes can be leaves, there is a tree $t = (V, E, \text{lab}, \text{dv}, <_c)$ with $V = \{r, v_1, \ldots, v_n, w_1, w_2\}$ and $E = \{(r, v_1), (v_n, w_1), (v_n, w_2)\} \cup \{(v_i, v_{i+1}) \mid i \in \{1, \ldots, n-1\}\}$ such that $t \models D$, $t \models \Sigma$ and $t \not\models \tau$.

Let θ be the truth assignment induced from t, that is, $\theta(y_j)$ is the label of v_j, for every j.

Towards a contradiction, let us assume that, for some $i \leq m$, $\theta \not\models C_i$. Then the pattern p_i of σ_i matches the two paths of t of length $n+1$ and thus σ_i does not hold. This is a contradiction from which we can conclude that $\theta \models \varphi$ and that, in particular, φ is satisfiable.

(only if): Let us assume that φ is satisfiable via some truth assignment θ. Let t be the tree with the same set V of vertices and set E of edges as the tree in the (if)-part and let node v_j carry label $\theta(y_j) = 1$, for every j. Let w_1, w_2 be labelled with 1, for concreteness. As v_n has two children, $t \not\models \tau$. On the other hand, as $\theta \models C_i$, for every i, none of the patterns p_i of the constraints in Σ matches t and thus $t \models \Sigma$, as desired. Therefore, t is a counter-example for $\Sigma \models_D \tau$.

The proof of (b) is also by reduction from 3-SAT to the complement of the implication problem and follows a similar approach, but the encoding of truth assignments is different: For every $i \leq n$, there are two symbols, a_i and b_i, both of which have to occur in any path matching τ. The corresponding truth assignment θ is defined by $\theta(y_i) = 1$ if a_i is a descendant of b_i and $\theta(y_i) = 0$ otherwise. More detail will be given in the full version of the paper. □

The two remaining lower bounds are both by reduction from tiling problems. Their proofs will be given in the full version of this paper.

Proposition 4.15.

(a) XC-Imp(*TP*, *XKFD*, esDTD) *is PSPACE-hard.*

(b) XC-Imp(*TP*, *FD*, esDTD) *is undecidable.*

The proof of Proposition 4.15 (a) is by a reduction from the corridor tiling problem, where the corridor width m is specified in unary encoding. The constraints constructed in this reduction do not refer to data values at all, instead it only uses structural constraints. In a nutshell, tiles in the same column of successive rows are compared by (unary) tree patterns of linear length. The undecidability proof of Proposition 4.15 (b) is by a reduction from the unbounded tiling problem. Here, data values are used to encode row "identities" and column "identities".

5. EXPLORING THE FRAMEWORK

The framework of X2R-constraints can be instantiated with an arbitrary X2R-mapping language $\mathcal{M}$ and an arbitrary relational constraint language $\mathcal{C}$. In the remainder of this section, we are going to sketch ways in which XML constraint languages that are used in practice or were proposed in the literature can be viewed as particular instantiations of the X2R-mapping based framework.

```
 1: <xs:element name="root">
 2:   [...]
 3:   <xs:key name="uid">
 4:     <xs:selector xpath="./person"/>
 5:     <xs:field xpath="u-id"/>
 6:   </xs:key>
 7:   <xs:keyref name="files" refer="uid">
 8:     <xs:selector xpath=".//file"/>
 9:     <xs:field xpath="u-id"/>
10:   </xs:keyref>
11: </xs:element>
```

Figure 5: XML Schema Key and Foreign Key Constraint

5.1 Relative Key Constraints

Arenas, Fan and Libkin [1] investigated relative key constraints, that is key constraints that hold on subtrees.

Consider for example a company with several establishments. In this case user-ids might be local to establishments, i.e. persons from different establishments are allowed to use the same user-id. This constraint can be written (using our syntax) as

$$\sigma = (/\text{establishment}\langle x\rangle/\text{person}\langle y\rangle/\text{u-id}\langle z\rangle, \{x, z.@\} \rightarrow y).$$

Using the node variable x on the left side changes the constraint to be local to establishments, as tuples referring to different establishments cannot conflict any more. In general, relative key constraints can be expressed as XC(TP[/], XKFD) constraints in our framework.

5.2 XML Schema Integrity Constraints

To compare XML Schema integrity constraints with our framework, we need to introduce some terminology.

For every tree t valid wrt to an XSD X, X assigns a type to every node v of t. For every possible type α of an XSD X, the set of nodes matched by α can be described by a regular language L_α over ancestor strings [11]. A node v belongs to the type α, if and only if the ancestor string of v is in L_α.

XML Schema [5] describes three kinds of integrity constraints: *unique constraints*, *key constraints* and *foreign key constraints*. Every XML Schema integrity constraint is specified relative to an element definition, that is XML Schema integrity constraints are relative constraints, like the constraints investigated in [1].

Figure 5 gives an example for an XML key constraint roughly equivalent to σ_{user} from Section 3 in XML Schema notation, leaving out the tail of the schema description.

Line 1 starts an element declaration for elements named root. We leave out the structural part of the type definition. Line 3 starts the definition of the key constraint and specifies a name for it, which is relevant, e.g., for foreign key constraints. Line 4 specifies the selector path (./person), which is a restricted XPath-expression that is evaluated relative to nodes matched by the element declaration: in this example it is evaluated relative to nodes of label root. Note that the element declaration not necessarily matches all elements of label root. Line 5 specifies the field of the constraint (u-id in the example). This XPath-expression is evaluated relative to nodes matched by the expression from Line 4. In general, there may be arbitrarily many field expressions $F_1, \ldots, F_n$.

We only give a simplified description of integrity constraints in XML Schema, as they are quite complex in general. For a tree t, to satisfy a key constraint, the following conditions have to be met by every node v matched by the surrounding element declaration:

(1) for every node v', that is matched by the selector path, it holds that every field expression F_i matches exactly one node v_i, and

(2) for every two nodes v_{t_1} and v_{t_2} matched by the selector path, the vector of data values of the nodes matched by the field specifications are not identical.

Let us assume for the moment, that the element declaration in Figure 5 only matches the root node. From Conditions (1) and (2) we then get, that the key constraint from Figure 5 corresponds to three constraints in our framework. From (1) we get the constraints $(/\text{root}/\text{person}\langle y\rangle/\text{u-id}\langle z\rangle, y \rightarrow z)$ and $(/\text{root}/\text{person}\langle y\rangle/\text{u-id}\langle z\rangle, \text{NN}(y,z))$, saying that every person-node (directly below the root) should have at most one (respectively at least one) u-id node as child.

From (2) we get our intended constraint

$$(/\text{root}\langle x\rangle/\text{person}\langle y\rangle/\text{u-id}\langle z\rangle, \{x, z.@\} \rightarrow y),$$

which is equivalent to σ_{user}. We note that constraints relative to the root node are equivalent to absolute constraints.

This looks like key constraints could be described by a subset of XC(TP [/,//],XKFD). This is true, if the structural part of (the relevant part of) the XML Schema can be described by a DTD. However in general, the element declaration could be enclosed inside a complex type declaration. In this case we have to ensure, that an XML Schema integrity constraint definition is only applied to nodes matched by the element declaration.

There are two straightforward ways to accomplish this. First, we could use tree patterns which can talk about regular paths, second, we could allow tree patterns to match nodes according to their type.

XML integrity constaints, which are defined over regular paths have been investigated in [1]. However, these constraints do not fully cover XML Schema integrity constraints, as the field expressions are restricted to paths of length one.

Let L_{root} be the regular language describing all possible ancestor strings for elements matched by the element declaration and R be a regular expression with $L(R) = L_{\text{root}}$. Note that in our example $L(R) = \{\text{root}\}$. Then the constraints can be described using the tree pattern $p = /R/\text{person}/\text{u-id}$.

The second approach has the advantage, that we get the types of nodes for free, when a tree is validated against a schema, as in the validation process the types have to be computed anyway. These types can then be used to match nodes of a tree pattern using existing algorithms.

XML Schema unique constraints have the same syntax as XML Schema key constraints, only the semantic differs. Unique constraints do not enforce that every field matches at least one node, i.e. it could match zero nodes. Accordingly, (2) is modified, that it only enforces the vector of data values to be different, when all fields match one node. In our framework, the difference is, that unique constraints do not enforce non-null constraints.

Foreign key constraints again use a very similar syntax. An example is given in Figure 5 Lines 7 to 11. The only difference in syntax is, that foreign keys reference a key constraint, in the example the uid constraint from above. The example foreign key specifies, that the u-id of files (described somewhere in the XML tree) should exist, i.e. there should be a person with this u-id.

For space reasons, we do not describe the semantics here, but just note that foreign key constraints can be expressed by inclusion constraints (over tree pattern mappings) in our framework.

5.3 XML functional dependencies (XFDs)

The literature has several different definitions of functional dependencies for XML data, e.g., [2, 9, 6, 15, 10]. We concentrate here on XFDs as introduced by Arenas and Libkin [2] and further examined by Kot and White [9]. An XFD $\sigma = Y \rightarrow Z$ consists of two sets of paths specifying the attributes of the functional dependency. As shown[8] in [9], XFDs can be canonically expressed using XC(TP[/], FFD), where the tree pattern p is the (unique) smallest tree pattern (with respect to the number of nodes) that contains all paths from Y and Z. However, tree patterns for XFDs need to be duplicate free[9], that is they do not contain two edges (x,y) and (x,z) with $\text{lab}(y) = \text{lab}(z)$ and $y \neq z$. Thus, XFDs have the same expressiveness as functional dependencies over duplicate-free tree patterns.

While it is immediatly clear, that the restriction not to use the descendant axis limits the expressivity, as constraints over recursive parts of schemas cannot be expressed, the restriction to duplicate free patterns is more subtle. Note that the dependency $(/r/a\langle x_a\rangle[/b/c\langle x_c\rangle]/b/d\langle x_d\rangle, \{x_c, x_d\} \rightarrow x_a)$ cannot be expressed with duplicate free patterns. Especially it is different from $(/r/a\langle x_a\rangle/b[/c\langle x_c\rangle]/d\langle x_d\rangle, \{x_c, x_d\} \rightarrow x_a)$.

Kot and White give an axiomatization of XFDs [9]. The axiomatization includes FFDs and NNs. They also present a chase-algorithm to decide the implication problem in polynomial time.

Another (more general) definition of XFDs has been proposed by Hartmann and Link [6], allowing XFDs to compare complete subtrees. For example, they can specify the dependency that there are no two a-labelled nodes that have equivalent (meaning isomorphic) subtrees. Dependencies of this kind cannot be expressed in our framework as they are second order constraints, i.e. they can compare sets of nodes.

5.4 Structural Constraints

Structural constraints are usually given by schemas. Popular schema languages for XML include XML Schema and Document Type Definitions (DTDs). Simple DTDs, as defined in Section 2 are an important subclass of DTDs.

It has been observed before (e.g. [9]), that simple DTDs imply certain integrity constraints as follows. Let D be a simple DTD. We define $\Sigma_D \subseteq \text{TP}_\perp(/,//,\text{FD},\text{NN})$ as $\Sigma_D \overset{\text{def}}{=}$

$$\left\{ (//a\langle x_a\rangle/b\langle x_b\rangle, x_a \rightarrow x_b) \;\middle|\; \begin{array}{l} a \rightsquigarrow \gamma b \gamma' \in D \vee \\ a \rightsquigarrow \gamma b? \gamma' \in D \end{array} \right\} \cup$$
$$\left\{ (//a\langle x_a\rangle/b\langle x_b\rangle, \text{NN}(x_a, x_b)) \;\middle|\; \begin{array}{l} a \rightsquigarrow \gamma b \gamma' \in D \vee \\ a \rightsquigarrow \gamma b^+ \gamma' \in D \end{array} \right\}$$

The first row contains all functional dependencies enforced by D, as there is at most one child with a particular label. The second row contains not null constraints enforced by D, as there is at least one child with a particular label.

[8]We note that Kot and White define the mapping of a tree pattern using unfolding of nested relations. The definition is equivalent to our definition using embeddings.

[9]Duplicate free tree patterns have been considered in [12]

LEMMA 5.1 ([9]). *For every simple DTD D, every tree t with $t \models D$ satisfies Σ_D.*

In [9] it is stated, that for implication of functional dependencies under a given simple DTD D, it is possible to replace D by the set Σ_D of dependencies (Theorem 4 in [9]). However, this is not entirely correct, as for functional dependencies, that use labels not present in D, are trivially satisfied under D. The same holds for functional dependencies that involve, say, b-children of a-nodes, where D disallows b for children of a-labelled nodes.

An inference algorithms can easily deal with this subtlety. However, on the formal level, it seems to require the addition of an unusual kind of relational constraints enforcing some attributes to be null (thus expressing that some nodes should not exist in the tree).

6. CONCLUSION

We have introduced a framework for XML integrity constraints and solved some basic complexity questions for the case, where the mappings are defined by tree patterns.

It can be observed that the impact of the descendant axis is very strong in most cases. It increases the complexity from polynomial time to (at least) co-NP and from co-NP to PSPACE or even to undecidability.

We believe, that our upper bounds already hold for ficticious functional dependencies, which are more expressive than functional dependencies, especially when dealing with incomplete data. However, the proofs will get more technical, due to the introduction of null values.

Among many open questions, the following seem to be especially interesting:

- What is the impact on the complexities, if we add other relational constraints, especially foreign key constraints?
- Is there a generic approach towards axiomatization by combining axiom systems for relational constraints with general tree axioms and axioms derived from the query language at hand?
- Similarly, is there a generic approach to combine the relational chase with tree-specific rules to get a chase procedure for X2R-constraints?

7. REFERENCES

[1] M. Arenas, W. Fan, and L. Libkin. On the complexity of verifying consistency of XML specifications. *Siam J. Comp.*, 38(3):841–880, 2008.

[2] Marcelo Arenas and Leonid Libkin. A normal form for XML documents. *ACM TODS*, 29:195–232, 2004.

[3] Paolo Atzeni and Nicola M. Morfuni. Functional dependencies and constraints on null values in database relations. In *Inf. Control*, volume 70(1), pages 1–31, 1986.

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

[5] S. Gao, C. M. Sperberg-McQueen, H.S. Thompson, N. Mendelsohn, D. Beech, and M. Maloney. W3C XML Schema Definition Language (XSD) 1.1 part 1: Structures. Technical report, W3C, April 2009. http://www.w3.org/TR/2009/CR-xmlschema11-1-20090430/.

[6] Sven Hartmann and Sebastian Link. More functional dependencies for XML. In Leonid Kalinichenko, Rainer Manthey, Bernhard Thalheim, and Uwe Wloka, editors, *Advances in Databases and Information Systems*, volume 2798 of *LNCS*, pages 355–369. Springer Berlin / Heidelberg, 2003.

[7] Sven Hartmann, Sebastian Link, and Klaus-Dieter Schewe. Functional dependencies over XML documents with DTDs. *Acta Cybernetica*, 17(1):153–171, 2005.

[8] Sven Hartmann, Sebastian Link, and Thu Trinh. Solving the implication problem for XML functional dependencies with properties. In Anuj Dawar and Ruy J. G. B. de Queiroz, editors, *WoLLIC*, volume 6188 of *LNCS*, pages 161–175. Springer, 2010.

[9] Lucja Kot and Walker M. White. Characterization of the interaction of XML functional dependencies with DTDs. In *ICDT*, pages 119–133, 2007.

[10] Mong Lee, Tok Ling, and Wai Low. Designing functional dependencies for XML. In *EDBT*, pages 145–158. 2002.

[11] W. Martens, F. Neven, T. Schwentick, and G.J. Bex. Expressiveness and complexity of XML Schema. *ACM TODS*, 31(3):770–813, 2006.

[12] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.

[13] L. Stockmeyer. The complexity of decision problems in automata and logic, 1974. Ph.D. Thesis, MIT, 1974.

[14] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[15] Millist W. Vincent, Jixue Liu, and Chengfei Liu. Strong functional dependencies and their application to normal forms in XML. *ACM TODS*, 29(3):445–462, September 2004.