Static Analysis of XML Processing with Data Values*

Luc Segoufin
INRIA and Université Paris 11
http://www-rocq.inria.fr/~segoufin

1 Introduction

Most theoretical work on static analysis for XML and its query languages models XML documents by labeled ordered unranked trees, where the labels are from a finite set. Attribute values are usually ignored. This was quite successful for many applications like, to mention only some of them, the study of the navigational fragment of XPath known as *Core-XPath* [20], a complete picture of the complexity of satisfiability and inclusion of XPath queries without joins [4], XML-schema design with no integrity constraints [28, 31], type-checking [40]. The reader is invited to have a look at the Database Theory columns [32, 37, 40, 39] for an introduction on this rich production.

The success of this model has basically two reasons, which are not independent. First, it allows to apply tree automata based techniques, providing a solid tool in order to obtain nice complexity results. See for instance the surveys [32, 39].

Second, extending the model by attribute values (data values) quickly leads to languages with undecidable static analysis (see, for instance [2, 4, 19, 33]).

Nevertheless, there are many examples of decidable scenarios involving attribute values. For instance inclusion of fragments of XPath with joins and XML-schema validation in the presence of integrity constraints. In this paper we survey some of the known positive, negative results and open problems in this direction.

We start with a generic approach based on extending the successful tree automata techniques to trees over infinite alphabet. Even though this approach leads to undecidability in general there exists at least one interesting decidable fragment. We show how this fragment can be used to obtain decidability results in the context of XPath queries containment, XML-schema validation in the presence of joins and integrity constraints.

We show next the limitation of this generic approach and survey many other positive results obtained using ad-hoc techniques.

*Database Principles Column. Editor: Leonid Libkin, School of Informatics, University of Edinburgh, libkin@inf.ed.ac.uk

2 Data trees

There exist many papers proposing extensions of the classical framework of languages over a finite alphabet to languages over an infinite alphabet. Some where mostly interested in the formal language point of view [3, 22, 35, 23, 18, 38, 11, 24, 25, 36]. Some where aiming at applications in program verification [7, 8, 10, 9, 15, 14]. Others had applications in XML and database in mind [34, 5, 6]. Most of the papers cited above deal with *words* over an infinite alphabet with the notable exception of [25, 5] that dealt with *trees*. The tree case is considerably harder than the word case.

For XML applications the tree model is of course the most useful and this is the one we present here. We use a presentation and a terminology introduced in [10] and reused later in [5, 6]. It combines both a finite alphabet and an infinite one, instead of having just an infinite alphabet. This makes it easier to use for XML applications.

2.1 The model

Let Σ be a finite alphabet and D be an infinite set. We could choose any infinite set for D, as we will deal with formalisms that can compare values only with respect to equality.

In this paper we consider unranked, ordered, labeled trees with data values. A *data tree t over* Σ has a set of *nodes*, where every node v has a *label* $v.l \in \Sigma$ and a *data value* $v.d \in D$.

Data trees can be used to encode XML documents as trees in a way which closely corresponds to the XPath data model [1]. We then let the finite alphabet Σ to be the set of all attribute and tag names occurring in the document. The attributes of a node v are represented by attribute nodes (labeled by the attribute name) which are children of v. I.e., the B-attribute value of a node v is given by the value of its (unique) child labeled with B. An example of this encoding is presented in Figure 1.

Note that there are other ways to code XML documents in data trees as we will see in Section 3.2.

```
<schedule>
 <course ID="5">
   <lecturer faculty="12">  </lecturer>
   <building nr="1"> </building>
 </course>
</schedule>
```

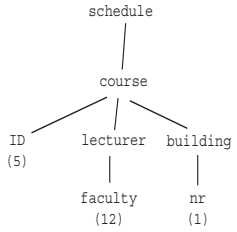

Figure 1: An XML document and its data tree encoding. In the encoding, data values are in parentheses. Data values for non-attribute nodes are not used.

2.2 Logics

A data tree can be seen as a model for a logical formula. The universe of this structure is the set of nodes of the tree, moreover, there are the following predicates available:

- For each possible label $a \in \Sigma$, there is a unary predicate $a(x)$, which is true for all nodes that have the label a.
- The binary predicate $x \sim y$ holds for two nodes if they have the same data value.
- The binary predicate $E_{\rightarrow}(x, y)$ holds for two nodes if x and y have the same parent node and y is the immediate successor of x in the order of children of that node.
- The binary predicate $E_{\downarrow}(x, y)$ holds if y is a child of x.
- The binary predicates $E_{\Rightarrow}$ and $E_{\Downarrow}$ are the transitive closures of $E_{\rightarrow}$ and $E_{\downarrow}$, respectively.

We write $\mathrm{FO}(\sim, <, +1)$ for first-order logic with all these predicates and $\mathrm{FO}(\sim, +1)$ the logic without $E_{\Rightarrow}$ and $E_{\Downarrow}$. Given a formula ϕ, we write $L(\phi)$ for the set of data trees that satisfy the formula ϕ. A formula satisfied by some data tree is *satisfiable*.

We write FO^k for formulas using at most k variables. Note that the following examples use 2 variables only.

Example 1 • Our first example shows how key constraints can be expressed in $\mathrm{FO}^2(\sim, <, +1)$.

The formula φ_a says that all a's are in different classes:

$$\varphi_a = \forall x \forall y (x \neq y \ \wedge a(x) \wedge a(y)) \longrightarrow x \not\sim y \ .$$

• In the same spirit we can define inclusion constraints.

The formula $\psi_{a,b}$ says that each class with an a also contains a b:

$$\psi_{a,b} = \forall x \exists y \big(a(x) \longrightarrow (b(y) \wedge x \sim y)\big) \ .$$

• Our last example presents a formula ϕ such that the Σ-projection of $L(\phi)$, i.e. the set of trees of $L(\phi)$ obtained by dropping the data values, is not a regular tree language.

Let $\varphi = \varphi_a \wedge \varphi_b \wedge \psi_{a,b} \wedge \psi_{b,a}$. Any data tree satisfying φ has is such that the numbers of a and b-labeled positions are equal.

This example can be easily extended to describe data trees with an equal number of a's, b's and c's, hence a language with a non-Context-Free projection. This illustrates the difficulty of handling data values: intrinsically we go beyond regular and context-free languages.

Unlike the classical setting, where the alphabet is finite, satisfiability for $\mathrm{FO}(\sim, <, +1)$ is undecidable. Actually this is already the case only in the restricted fragment using only three variables. Note that with three variables one can define $E_{\rightarrow}$ from $E_{\Rightarrow}$ and $E_{\downarrow}$ from $E_{\Downarrow}$, therefore the following result also implies the undecidability of the three-variable fragment of $\mathrm{FO}(\sim, <, +1)$ that uses only $<$.

Theorem 1 *[6] Satisfiability of* $\mathrm{FO}^3(\sim, +1)$*, and therefore of* $\mathrm{FO}^3(\sim, <, +1)$*, is undecidable over data trees.*

It turns out that restricting $\mathrm{FO}(\sim, +1)$ to its two-variables fragment yields decidability. We will see in the next section that this is sufficient to obtain many useful results in the context of XML.

Theorem 2 *[5] Satisfiability of* $\mathrm{FO}^2(\sim, +1)$ *is decidable over data trees.*

The complexity however is quite high. The algorithm presented in [5] is in 3NEXPTIME. This is possibly not optimal. The current best known lower-bound is NEXPTIME-hard.

What about $\mathrm{FO}^2(\sim, <, +1)$? The status of its decidability is still an open question. We only know that showing that satisfiability of $\mathrm{FO}^2(\sim, <, +1)$ is decidable is likely to be a difficult problem as it would imply deciding multi-counter automata on trees and the linear logic MELL, which are known as open issues in their respective fields (see [13] and the references therein).

Open problem 1 *Is satisfiability of* $\mathrm{FO}^2(\sim, <, +1)$ *decidable over data trees?*

Let $\mathrm{EMSO}^2(\sim,+1)$ be the extension of $\mathrm{FO}^2(\sim,+1)$ consisting of all formulas starting with a sequence of existential quantifiers over unary predicates (i.e., set variables) followed by a $\mathrm{FO}^2(\sim,+1)$ formula. That is formulas of the form $\exists R_1 \cdots R_n \varphi$ where $\varphi \in \mathrm{FO}^2(\sim,+1)$.

The following is an obvious consequence of Theorem 2.

Corollary 1 *Satisfiability of* $\mathrm{EMSO}^2(\sim,+1)$ *is decidable over data trees.*

2.3 Automata

By *trees* we denote the usual notion of (unranked ordered) trees labeled with just a finite alphabet. A tree can therefore be seen as a projection of a data tree by ignoring the data value of each node. Logics over trees are defined as logics of data trees without the predicate $\sim$.

When the alphabet is finite there exists several notions of tree automata for unranked ordered trees. We use the presentation of [12, 29] because it is easier to see the connection with logics. A *nondeterministic automaton over unranked ordered trees* has a finite set Q of states, subsets I and J of Q, along with relations

$$\delta_h, \delta_v \subseteq Q \times \Sigma \times Q \ ,$$

which are called the *horizontal* and *vertical* transition relations respectively. A *run* of such an automaton over a Σ-tree t is a labeling $\rho : V \to Q$ of the tree's nodes with states such that for every node v with label a we have:

- If v is a leaf then $\rho(v) \in I$.
- If v has no horizontal predecessor then $\rho(v) \in J$.
- If v has a horizontal successor w, then the triple $(\rho(v), a, \rho(w))$ belongs to the horizontal transition relation δ_h.
- If v has no horizontal successor and its parent is w, then the triple $(\rho(v), a, \rho(w))$ belongs to the vertical transition relation δ_v.

A run is *accepting* when the state and label of the root belong to the designated *accepting* set $F \subseteq Q \times \Sigma$. A tree is *accepted* if it admits an accepting run. A set of unlabeled trees is called *regular* if it is recognized by an automaton.

The reader may be more accustomed to a different definition of automata on unranked ordered trees (see for instance [32]). In the other definition, there is a finite set of rules of the form

$$a, L \models q \quad a \in \Sigma, q \in Q \ , L \subseteq Q^* \text{ regular} \ .$$

The idea is that a tree with a in the root and subtrees $t_1, \cdots, t_n$ can be assigned a state q if the subtrees can be assigned states $q_1, \cdots, q_n$ such that the word $q_1 \cdots q_n$ belongs to L. It is not difficult to see that this type of automaton is equivalent the one presented above.

With the definition of tree automata as presented above, the classical coding of automata into MSO immediately yields the following result.

Fact 1 For every regular tree language there is an equivalent formula in $\mathrm{EMSO}^2(+1)$ and vice-versa.

3 Application to XML reasoning

3.1 Integrity Constraints

XML documents usually come with a specification, often stated in XML Schema, which tells what to expect in the document. It contains a structural part which includes a mechanism for assigning types to nodes of the tree and possibly a set of integrity constraints such as *keys* and *inclusion constraints*. It is natural to ask whether a specification is consistent and whether a set of integrity constraints is minimal or not (*implication problem*).

Basically, the two standard XML schema languages, DTD and XML Schema, are able to define only sets of documents that are regular tree languages (but not all regular tree languages!). In the following, we thus assume that the allowed set of documents is described by a tree automaton A. The *type* of a node v is the state of A on v in an accepting run. The specification of XML Schema are such that the type of each node is uniquely determined. This implies that the automaton A has a unique accepting run, thus it is unambiguous.

A *key constraint* is an expression of the form $\tau[X] \longrightarrow \tau$ where τ is a type of a node and X a set of attributes of that node. It says that the X-attributes of a node of type τ uniquely determine the node. Stated in other terms, for each combination of attribute values there is at most one node of type τ having these values.

An *inclusion constraint* is an expression of the form $\tau[X] \subseteq \tau'[Y]$ where τ and τ' are two node types and X and Y are sequences of attributes of the same cardinality. It says that for each node u of type τ there is a node v of type τ' such that the X-attributes of u have the same (corresponding) values as the Y-attributes of v. Key and inclusion constraints are said to be *unary* if $|X| = |Y| = 1$.

Note, that by combining key and inclusion constraints also foreign key constraints can be covered.

The *consistency problem* for unary keys and unary inclusion constraints relative to a regular tree language is as follows. Given an (unambiguous) tree automaton

A and a set K of unary key and inclusion constraints[1], it asks whether there is a tree t which is accepted by A and fulfills the constraints. The *implication problem* asks, given A and sets K_1, K_2 of constraints, whether each tree accepted by A which fulfills K_1 also fulfills K_2.

The general problem turns out to be undecidable.

Theorem 3 *[17] The consistency and implication problems for keys and inclusion constraints relative to a regular tree language are undecidable.*

However several interesting decidable scenarios have been identified. In this section, we will see that it follows quite directly from Theorem 1 that the consistency and the implication problem for unary keys and unary inclusion constraints are decidable, even relative to structural constraints given by a regular tree language (a similar result was first shown in [17]). Other scenarios and approaches will be discussed in the next section.

One of the advantages of a logic-based approach to decidability is the compositionality of logic. This holds in particular for $\mathrm{FO}^2(\sim, +1)$, which is closed under all Boolean operations and, as far as satisfiability is concerned, under existential set quantification. This compositionality is well illustrated in the proof of the following theorems.

Theorem 4 *[5] The consistency and implication problems for unary key and unary inclusion constraints relative to a regular tree language are decidable.*

proof : (sketch) We only consider the more general, implication problem. We use the coding of XML documents as data trees presented in Section 2.1. Consider now a XML Schema with key and inclusion constraints. That is we are given a tree automaton A and two sets K and K' of, respectively, unary key and unary inclusion constraints, where the states of A induce the types in the constraints. We want to know whether all trees accepted by A and satisfying the constraints in K, also satisfy the constraints in K'.

By Fact 1, from A we can derive an $\mathrm{EMSO}^2(+1)$ formula equivalent to A. This formula is of the form $\exists R_1, \ldots, R_n \;\; \varphi_A$. As in Example 1 we can derive for each constraint U in $K \cup K'$ an equivalent formula φ_U in $\mathrm{FO}^2(\sim, +1)$: this is done by using the existentially quantified variables of the formula of A whenever the corresponding type is used.

When this is done, the implication problem now reduces to satisfiability of the formula of $\mathrm{EMSO}^2(\sim, +1)$:

$$\exists R_1, \ldots, R_n \;\; \varphi_A \wedge \bigwedge_{U \in K} \varphi_U \wedge \bigwedge_{U \in K'} \neg \varphi_U.$$

[1]Recall that the types used in these constraints are states of A.

□

Note that even though the reduction above is simple, it does not provide the precise complexity of the implication problem, we only have the upper-bound given by the analysis of the proof of Theorem 2: 3NEXPTIME.

3.2 XPath Containment

We study two problems: satisfiability and containment. Let L be a query language, for instance XPath, and C be a class of XML-schema, for instance DTD or DTD with integrity constraints. The *containment problem* of L in the presence of C, asks whether two queries q and q' of L are such that for any XML document t in C, $q(t) \subseteq q'(t)$. The *satisfiability* problem for L in the presence of C asks whether for a query $q \in L$, there exists a tree $t \in T$ such that $q(t)$ is non empty. Containment reduces to satisfiability when L is closed under negation, which is going to be the case in this section.

Core-XPath, the fragment of XPath capturing its navigational behavior by removing all numerical operations was introduced by Gottlob et al. [20], is by now well understood. In particular, it corresponds to $\mathrm{FO}^2(<, +1)$ on unranked ordered trees [30]. Satisfiability and inclusion of Core-XPath are decidable even in the presence of DTDs and the complexity of many of its fragments has been studied in the literature. We refer to [37, 4, 19] and the references therein for a comprehensive survey.

In the presence of data values a simple extension of Core-XPath is to allow equalities of the form $p/@A = q/@B$ inside qualifiers, meaning that the value of the A attribute of some node accessible by a path matching p equals the B attribute value of some node accessible by a path matching q. We denote this fragment by Core-Data-XPath. It turns out that this fragment is already too expressive to obtain decidability.

Theorem 5 *[19] Satisfiability (and containment) of Core-Data-XPath is undecidable over XML documents.*

As satisfiability (and therefore containment) of Core-Data-XPath is undecidable, it is natural to consider fragments of XPath.

It is easy to verify that the expressive power of $\mathrm{FO}^2(\sim, <, +1)$ is strictly contained in the expressive power of Core-Data-XPath. The translation of $\mathrm{FO}^2(\sim, <, +1)$ into XPath is done as in the translation of $\mathrm{FO}^2(<, +1)$ into unary-TL over words [16]. This inclusion is strict because $\mathrm{FO}^2(\sim, <, +1)$ cannot express the test $\texttt{Self/@A} = \texttt{Self//b//c/@A}$.

As $\mathrm{FO}^2(\sim, <, +1)$ is not yet known to be decidable we define in this section a fragment of XPath, which we call *XPath-local* , which can be captured by $\mathrm{FO}^2(\sim, +1)$.

More precisely, satisfiability and containment test for unary queries expressed in these fragments, possibly in the presence of integrity constraints and schema, can be reduced to satisfiability of $\mathrm{FO}^2(\sim, +1)$.

The language XPath-local, when compared with Core-Data-XPath, has two restrictions: (1) navigation is not allowed along the "transitive" axes as `Descendant` and `FollowingSibling` and (2) in an equality on attribute values either one of the location paths has to be absolute (i.e., starting from the root), or both (relative) location paths are strongly limited.

In XPath-local only the following axes are allowed:

$$\begin{aligned}\texttt{Axis} := \quad & \texttt{Child} \mid \texttt{Parent} \mid \texttt{NextSibling} \mid \\ & \texttt{PreviousSibling} \mid \texttt{Self} \mid \texttt{ElseWhere}\end{aligned}$$

Every axis corresponds to a binary relation on tree nodes. For instance, the `Child` axis is true for node pairs (v, w) where w is a vertical successor of w. The other axes are defined analogously. The new `ElseWhere` axis corresponds to the relation of pairs (v, w) of nodes, where $v \neq w$. It is added in order to allow at least some kind of global navigation.

The rest of the syntax of XPath-local is given by the following grammar.

$$\begin{aligned}
\texttt{LocPath} &:= \texttt{RelLocPath} \mid \texttt{AbsLocPath} \\
\texttt{AbsLocPath} &:= \texttt{'/'RelLocPath} \\
\texttt{RelLocPath} &:= \texttt{Step} \mid \texttt{RelLocPath'/'Step} \\
\texttt{Step} &:= \texttt{Axis :: NameTest Predicate}^* \\
\texttt{NameTest} &:= \texttt{Name} \mid \texttt{'*'} \\
\texttt{Predicate} &:= \texttt{'['PredExpr']'} \\
\texttt{PredExpr} &:= \texttt{LocPath} \mid \\
&\quad \texttt{LocPath'/'Attr EqOp AbsLocPath'/'Attr} \mid \\
&\quad \texttt{Self :: NameTest'/'Attr EqOp Step'/'Attr} \mid \\
&\quad \texttt{PredExpr and PredExpr} \mid \\
&\quad \texttt{PredExpr or PredExpr} \mid \texttt{not PredExpr} \\
\texttt{Attr} &:= \texttt{'@'Name} \\
\texttt{EqOp} &:= \texttt{'='} \mid \texttt{'!='}
\end{aligned}$$

An expression derived from `LocPath` defines a binary relation on tree nodes (a set of paths), while an expression derived from `PredExpr` defines a unary relation (a set of tree nodes). These are defined using mutual recursion.

To obtain decidability, we restrict (in-)equalities of the form `Self :: NameTest'/'Attr EqOp Step'/'Attr`, which we call relative equalities, a bit further. We say that an attribute name B is *associated* to a label a in an XPath expression e if the pair $(a, @B)$ or $(*, B)$ occurs as (`NameTest`, Attr) pair in a relative (in-)equality of e.

A set of expressions is *safe* if the set of induced associations is a function from labels to attribute names. In particular if the wildcard $*$ is present, there is a unique attribute name occurring in all relative sub-expressions.

Example 2 The following (safe) expression selects a node v if all of its children with label b have the same data value as v:

$$\texttt{Child :: b/@B} \ != \ \texttt{Self :: */@B}.$$

The following expression is also safe:

$$\texttt{Child :: b/@B}_1 \ = \ \texttt{Self :: a/@B}_2.$$

Theorem 6 *[5] Satisfiability and containment for XPath-local safe expressions is decidable. This holds even relative to a schema consisting of a regular tree language and unary key and unary inclusion constraints.*

proof : (sketch) The proof is, of course, by translating the expressions into $\mathrm{FO}^2(\sim, +1)$ formulas. We encode XML documents as in Section 2.1 with a small extension that we will introduce later. As long as expressions do not compare attribute values, there is no need to restrict the location paths: We can just use the standard inclusion of Core-XPath into $\mathrm{FO}^2(<, +1)$ of [30].

This easily extends to equality expressions with at most one relative location path by, intuitively, first simulating the relative path, then jumping to a node with the same data value and checking that this node satisfies its absolute path constraint by simulating the path backwards to the root. Note that it seems crucial here that the second path is absolute and thus does not start at the current node, as the two variables are needed for the navigation and thus the current node can not be remembered. As an example the expression

$$\begin{aligned}\texttt{Child :: } a/\texttt{Child ::} b/@B_1 = \\ /\texttt{Child :: } c/\texttt{NextSibling :: } d/@B_2.\end{aligned}$$

is translated into the following equivalent formula $\varphi(x)$:

$$\begin{aligned}
&\exists y E_{\downarrow}(x, y) \wedge a(y) \wedge \\
&\quad \exists x E_{\downarrow}(y, x) \wedge b(x) \wedge \\
&\quad\quad \exists y E_{\downarrow}(x, y) \wedge B_1(y) \wedge \\
&\quad\quad\quad \exists x \; x \sim y \wedge B_2(x) \wedge \\
&\quad\quad\quad\quad \exists y E_{\downarrow}(y, x) \wedge d(y) \wedge \\
&\quad\quad\quad\quad\quad \exists x E_{\rightarrow}(x, y) \wedge c(x) \wedge \\
&\quad\quad\quad\quad\quad\quad \exists y E_{\downarrow}(y, x) \wedge \neg \exists x E_{\downarrow}(x, y).
\end{aligned}$$

It only remains to explain how we can deal with relative (in-)equalities. To this end, we exploit the fact

that the encoding of XML documents used so far only needs data values in attribute nodes. Thus, we can use the data values of element nodes for our purpose. Note, that the safety restriction on relative (in-)equalities ensures that for each element only one attribute is used in relative (in-)equalities. Therefore, we use data trees in which this attribute value (if any) is stored. Note, that an additional $\mathrm{FO}^2(\sim, +1)$ formula can check that the data values in element nodes are consistent with those in the attribute nodes.

As an example, if $(\texttt{Child} :: \texttt{b}/@\texttt{B}_1 = \texttt{Self} :: \texttt{a}/@\texttt{B}_2)$ is a subexpression of our XPath expression at hand, then we consider data trees in which the data value of a-nodes is interpreted as the B_2-attribute and the data value of b-nodes as the B_1-attribute. Thus, the expression is equivalent to the formula $a(x) \wedge \exists y E_{\downarrow}(x, y) \wedge b(y) \wedge x \sim y$.

It is now straightforward to combine the techniques described so far with those of Section 3.1 to obtain the second statement of the theorem. □

It should be noted that satisfiability of a similar fragment of XPath with all axes besides `Following` and `Preceding` can be reduced to satisfiability of $\mathrm{FO}^2(\sim, <, +1)$. Unfortunately, we do not know if satisfiability of $\mathrm{FO}^2(\sim, <, +1)$ is decidable, see Open problem 1.

3.3 Discussion of the approach

The first obvious limitation of this approach is the lack of expressive power. The main obstacle being the inclusion of the descendant axis, see Open problem 1. As we already mentioned, this issue is likely to be difficult to solve.

There might be a way around this. Indeed it is natural for a logician to consider two-variable fragments in order to obtain decidability (see the survey [21]). But as we have illustrated above, even though $\mathrm{FO}^2(<, +1)$ correspond to Core-XPath, $\mathrm{FO}^2(\sim, <, +1)$ is not really suitable for Core-Data-XPath anymore. It seems that XPath and $\mathrm{FO}^2(\sim, <, +1)$ are two different conceptual objects. Temporal logics seems a lot more relevant for dealing with XPath. Finding decidable temporal logics over data trees is therefore a natural direction for future research.

A natural extension of LTL (with only the future temporal operator NEXT and UNTIL) manipulating data values by the mean of one register was studied in [15, 14]. This logic was shown to be decidable but with a non-elementary lower bound complexity. This extension of LTL with one register is incomparable in term of expressive power with $\mathrm{FO}^2(\sim, <, +1)$.

This direction look promising but so far there does not exist any extension of this work to data trees, where CTL and CTL* are obvious starting points.

The second natural limitation of this approach is the high complexity. This is a classical feature for generic approaches: for each particular problem, the generic approach gives a higher complexity bound when compared with ad-hoc methods. Note that we do not know the exact complexity of decidability of $\mathrm{FO}^2(\sim, +1)$. The current proof of Theorem 2 is 3NEXPTIME. It is likely that this could be improved, possibly to match the current best lower-bound: NEXPTIME-hard. If this were the case the generic approach would give a complexity comparable to the current known ad-hoc ones.

4 More about XPath and Schema validation

4.1 XPath containment in the presence of data values

It is difficult to compare the result presented in Theorem 6 with those existing in the literature. The main reason is that this result holds even in the presence of schema containing key and foreign-key constraints while we are aware only of studies of XPath containment in the presence of DTDs without any constraints. Therefore the rest of this section assumes only DTDs without constraint.

The results presented here can be found in [4, 19]. We only restrict our attention to fragments of XPath containing equality tests between values of attributes pointed by XPath expressions as explained above.

Most of the known results concern satisfiability of XPath fragments only and not containment. Recall that satisfiability can be reduced to containment but, unless the language is closed under negation, the opposite reduction doesn't hold in general.

As we already mentioned in Theorem 5, when all the navigational axes are present, satisfiability is undecidable even when no DTDs are present. Actually a much sharper result can be shown. Let $\mathrm{XPath}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, =)$ be the restriction of Core-Data-XPath that uses only the vertical navigational axes: `Child`, `Descendant`, `Parent` and `Ancestor` (no horizontal navigation). This fragment is closed under negation, but satisfiability in the presence of DTDs is already undecidable.

Theorem 7 *[4] In the presence of DTDs, satisfiability (and containment) for* $\mathrm{XPath}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, =)$ *is undecidable.*

When no DTDs are present the satisfiability of $\mathrm{XPath}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, =)$ is still an open issue.

Let XPath$(\downarrow, =)$ be the restriction of Core-Data-XPath that uses only one axis predicate: `Child`. This fragment is closed under negation.

Theorem 8 *[4] In the presence of DTDs (with no constraints) containment for XPath$(\downarrow, =)$ is decidable in* NEXPTIME.

Note that XPath$(\downarrow, =)$ is incomparable in expressive power with the fragment introduced in Section 3.2: It does not allow horizontal navigation nor upward navigation; moreover $\text{FO}^2(\sim, <, +1)$ is not powerful enough to capture this fragment, as already mentioned in Section 3.2.

Let us denote by posXPath(=) the restriction of Core-Data-XPath where negation is not allowed (all navigational axis are permitted). This fragment is not closed under negation.

Theorem 9 *In the presence of DTDs (with no constraints) satisfiability for posXPath(=) is* NP*-complete.*

Note that this result could also be obtained (with a much higher complexity) by reduction to satisfiability of $\text{EMSO}^2(\sim, +1)$. Indeed as posXPath(=) does not contain negation it is contained in the existential fragment of $\text{FO}(\sim, <, +1)$ which in turn can easily be coded in $\text{EMSO}^2(\sim, +1)$.

The status of the containment problem for posXPath(=) is still an open issue.

4.2 XML-schema validation

In this section we revisit the main results presented in the survey [17] and compare them with Theorem 4.

As in the XPath case above, there is a mismatch between the consistency and the implication problems. Consistency was studied in great details while less is known about the implication problem. The generic technique presented in Section 3 doesn't suffer from this problem as $\text{FO}^2(\sim, +1)$ is closed under complement.

The second difference is that all the results of [17] are stated for DTDs the typing system of which captures only a restricted fragment of regular tree language. This could make a difference but the authors of [17] believe that they can extends all their results to regular tree languages [27].

The advantage of the generic approach is that it readily extends to any kind of integrity constraints as long as it is definable in $\text{FO}^2(\sim, +1)$, while the techniques below are likely to be difficult to extend.

One advantage of the specific approach presented in [17] is that it gives a better complexity. For instance the 3NEXPTIME complexity result obtained in Theorem 4 is not optimal for consistency of unary keys and unary inclusions constraints as shown below.

Theorem 10 *[17] The consistency problem for unary keys and unary inclusion constraints relative to a DTD is* NP*-complete.*

*The implication problem for unary keys and unary inclusion constraints relative to a DTD is co*NP*-complete.*

The techniques developed in [17] actually show a stronger result. The NP-completeness result of Theorem 10 can be generalized to key constraints of arbitrary arity (but unary inclusion constraints) assuming they are *primary*: no two key constraints for the same type.

Another interesting result obtained in [17] concerns the case when only integrity key constraints are present, but allowing an arbitrary arity for those. Note that $\text{FO}^2(\sim, +1)$ cannot express binary key constraints.

Theorem 11 *[17] The consistency and implication problems for arbitrary key constraints relative to a DTD is decidable in linear time.*

Remark: It would be tempting to extend the decidability result of Theorem 2 to the logic $\text{FO}^2(\sim_1, \sim_2, \sim_3, ..., +1)$, extending $\text{FO}^2(\sim, +1)$ with arbitrary many equivalence relations. This logic would then easily express arbitrary key constraints. But it would also express arbitrary inclusion constraints and therefore by Theorem 3 the logic $\text{FO}^2(\sim_1, \sim_2, \sim_3, ..., +1)$ is undecidable. It turns out that the situation is even more dramatic than that because the logic $\text{FO}^2(\sim_1, \sim_2, \sim_3)$, three equivalence relations, no navigation capability, is already undecidable [26].

5 Conclusion

Static analysis of XML processing in the presence of data values is an important and challenging problem. We have presented here only a subset of the existing work in this area.

The logical approach, based on the decidability of $\text{FO}^2(\sim, +1)$, was a means to obtain a generic approach on the existing decidable scenarios. We have already mentioned the caveat and the problems raised by this approach: High complexity and limited expressive power. Finding the precise complexity of $\text{FO}^2(\sim, +1)$ is left for future work. The decidability of $\text{FO}^2(\sim, <, +1)$ is a challenging open problem.

Another possibility for a generic approach would be to consider temporal logic on trees.

In this survey we considered only the simple case where data values can only be tested for equality. It turns out that if we would allow a little bit of arithmetic, checking whether one data value is smaller than

another one, then the two-variable first-order logic becomes undecidable [6].

Acknowledgment. We thank Michael Benedikt, Leonid Libkin and Thomas Schwentick for all their comments on an earlier version of this paper.

References

[1] XML Path Language (XPath). Available at http://www.w3.org/TR/2002/WD-xpath20-20020816.

[2] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with Data Values: Typechecking Revisited. *JCSS*, 66(4):688–727, 2003.

[3] J.-M. Autebert, J. Beauquier, and L. Boasson. Langages des alphabets infinis. *Discrete Applied Mathematics*, 2:1–20, 1980.

[4] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS 2005*, pages 25–36.

[5] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-Variable Logic on Data Trees and XML Reasoning. In *PODS 2006*.

[6] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-Variable Logic on Words with Data. In *LICS 2006*.

[7] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic Verification of Recursive Procedures with one Integer Parameter. *TCS*, 295, 2003.

[8] P. Bouyer. A logical characterization of data languages. *IPL*, 84(2):75–85, 2002.

[9] P. Bouyer and A. Petit. A Kleene/Büchi-like theorem for clock languages. *Journ. of Automata, Lang. and Combin.*, 7(2):167–186, 2002.

[10] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003.

[11] E. Cheng and M. Kaminski. Context-Free Languages over Infinite Alphabets. *Acta Inf.*, 35(3):245–267, 1998.

[12] J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In *FCT 2005*.

[13] P. de Groote, B. Guillaume, and S. Salvati. Vector Addition Tree Automata. In *LICS 2004*, pages 64–73.

[14] S. Demri and R. Lazić. LTL with the Freeze Quantifer and Register Automata. In *LICS 2006*.

[15] S. Demri, R. Lazić, and D. Nowak. On the freeze quantifier in Constraint LTL. In *TIMES*, 2005.

[16] K. Etessami, M. Vardi, and T. Wilke. First-Order Logic with Two Variables and Unary Temporal Logic. *Inf. Comput.*, 179(2):279–295, 2002.

[17] W. Fan, and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3): 368–406, 2002.

[18] N. Francez and M. Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *TCS*, 306(1-3):155–175, 2003.

[19] F. Geerts and W. Fan. Satisfiability of XPath Queries with Sibling Axes. In *DBPL 2005*, pages 122–137.

[20] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *VLDB 2002*.

[21] E. Grädel and M. Otto. On logics with two variables. *TCS*, 224(1-2):73–113, 1999.

[22] J. Idt. Automates a pile sur des alphabets infinis. In *STACS*, 1984.

[23] M. Kaminski and N. Francez. Finite memory automata. *TCS*, 134(2):329–363, 1994.

[24] M. Kaminski and T. Tan. Regular Expressions for Languages over Infinite Alphabets. *Fundam. Inform.*, 69(3):301–318, 2006.

[25] M. Kaminski and T. Tan. Tree Automata over Infinite Alphabets. Poster at the 11th International Conference on Implementation and Application of Automata, 2006.

[26] E. Kieronski and M. Otto. Small substructures and decidability issues for first-order logic with two variables. In *LICS*, pages 448–457, 2005.

[27] L. Libkin. Personal communication.

[28] W. Martens, F. Neven, T. Schwentick, and G.-J. Bex. Expressiveness and complexity of XML Schema. *ACM TODS*, 2007.

[29] W. Martens and J. Niehren. Minimizing tree automata for unranked trees. In *DBPL 2005*, volume LNCS 3774.

[30] M. Marx. First order paths in ordered trees. In *ICDT 2005*.

[31] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.

[32] F. Neven. Automata theory for xml researchers. *SIGMOD Record*, 31(3):39–46, 2002.

[33] F. Neven and T. Schwentick. XPath Containment in the Presence of Disjunction, DTDs, and Variables. In *ICDT 2003*.

[34] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 15(3):403–435, 2004.

[35] F. Otto. Classes of regular and context-free languages over countably infinite alphabet. *Discrete and Apllied Mathematics*, 12:41–56, 1985.

[36] H. Sakamoto and D. Ikeda. Intractability of decision problems for finite-memory automata. *TCS*, 231:297–308, 2000.

[37] T. Schwentick. Xpath query containment. *SIGMOD Record*, 33(1):101–109, 2004.

[38] Y. Shemesh and N. Francez. Finite-State Unification Automata and Relational Languages. *Inf. Comput.*, 114(2):192–213, 1994.

[39] D. Suciu. The XML typechecking problem. *SIGMOD Record*, 31(1):89–96, 2002.

[40] V. Vianu. A web odyssey: from Codd to XML. *SIGMOD Record*, 32(2):68–77, 2003.