Two-Variable Logic on Data Trees and XML Reasoning*

[Extended Abstract]

Mikołaj Bojańczyk
Faculty of Mathematics,
Informatics and Mechanics
Warsaw University
Poland

Claire David
LIAFA
Université Paris 7
France

Anca Muscholl
LIAFA
Université Paris 7
France

Thomas Schwentick
Lehrstuhl Informatik I
Universität Dortmund
Germany

Luc Segoufin
INRIA &
Université Paris 11
France

ABSTRACT

Motivated by reasoning tasks in the context of XML languages, the satisfiability problem of logics on *data trees* is investigated. The nodes of a data tree have a label from a finite set and a data value from a possibly infinite set. It is shown that satisfiability for two-variable first-order logic is decidable if the tree structure can be accessed only through the *child* and the *next sibling* predicates and the access to data values is restricted to equality tests. From this main result decidability of satisfiability and containment for a data-aware fragment of XPath and of the implication problem for unary key and inclusion constraints is concluded.

Categories and Subject Descriptors

F.4.1 [**Mathematical logic and formal languages**]: Mathematical logic; H.2.3 [**Database management**]: Languages—*Query languages*

General Terms

Theory

Keywords

First-order logic, data trees, decidability.

1. INTRODUCTION

Most theoretical work on 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 has basically two reasons, which are not independent. First, the modeling allows to apply automata based techniques, as automata operate on trees of this kind. Second, extending the model by attribute values (data values) quickly leads to languages with undecidable static analysis (see, for instance [1, 3, 12, 20]).

Nevertheless, there are examples of decidable static reasoning tasks involving attribute values [2, 6]. The motivation for our work was to find a logical approach for such tasks.

It is immediately clear that full first-order logic is far too powerful for this purpose. Satisfiability for first-order logic with a predicate for data values equality is undecidable already on strings [4]. There are several possible candidates for more appropriate logics, including temporal logics or fragments of first-order logic. In this work, we concentrate on a (classical) fragment of first-order logic, two-variable logic. There are several good reasons to consider this fragment. It is known that on many kinds of structures this fragment is decidable [14]. On ordered, unranked trees, it corresponds in a natural way to the navigational behavior of XPath, and it can express many interesting integrity constraints.

Before we describe the technical contributions of the paper, we first discuss the connections with XML processing in more detail.

Core-XPath, the fragment of XPath capturing its navigational behavior introduced by Gottlob et al. [13], is by now well understood. In particular, it corresponds to $FO^2(<,+1)$ on unranked ordered trees [18]. Here, $FO^2(<,+1)$ is the two-variable fragment of first-order logic that uses the order $<$ and successor $+1$ relations, along with the labels of the nodes. The labels are encoded by unary relations, one for each of the (finitely many) possible labels. The symbol "$<$" refers to two binary predicates: one for comparing the descendant/ancestor relationship of two nodes and one for the preceding/following relationship of two siblings. Similarly, "$+1$" also refers to two binary predicates: one for comparing the parent/child relationship of two nodes and

*Work supported by the French-German cooperation programme PROCOPE, the EU-TMR network GAMES, the Foundation for Polish Science and KBN Grant 4 T11C 042 25.

PODS'06, June 26–28, 2006, Chicago, Illinois, USA.

one for comparing the next/previous sibling relationship of two siblings. Core-XPath is decidable even in the presence of DTDs and the complexity of many of its fragments has been studied in the literature. We refer to [3, 12] 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. As shown in [12], Core-Data-XPath is undecidable and both [3, 12] studied fragments of Core-Data-XPath. To be able to reason about Core-Data-XPath, it is natural to consider the logic $\mathrm{FO}^2(\sim,<,+1)$, the extension of $\mathrm{FO}^2(<,+1)$ with a binary predicate $\sim$ that checks data value equality of two nodes.

It is easy to verify[1] that $\mathrm{FO}^2(\sim,<,+1)$ is strictly contained in Core-Data-XPath and therefore it is natural to wonder whether $\mathrm{FO}^2(\sim,<,+1)$ is decidable.

We do not solve the question whether $\mathrm{FO}^2(\sim,<,+1)$ is decidable, here. Nevertheless, we show that the decidability of $\mathrm{FO}^2(\sim,<,+1)$ is a difficult problem as it would imply deciding multicounter automata on trees and the linear logic MELL, which are known as open issues in their respective fields (see [9] and the references therein).

However, we show that the logic $\mathrm{FO}^2(\sim,+1)$ is decidable. It turns out that this implies the decidability of several reasoning problems for XML which involve data values. We give some examples of applications next.

- Common reasoning tasks for XML are the consistency and the implication problems for integrity constraints. Given a finite set S of constraints and a further constraint φ, one asks whether each document satisfying all constraints in S also satisfies φ. This boils down to testing if there is a document that satisfies all constraints in S but not φ, a satisfiability question. The most common family of integrity constraints are key and inclusion constraints. Many of them involve only one attribute. It is easy to see that such constraints can be expressed in $\mathrm{FO}^2(\sim,+1)$. Our main result implies the decidability of the implication problem for such constraints. This was already known from [6], which shows that the complexity of implication, without schemas, is polynomial.

- An advantage of the logical approach is that reasoning problems can be relativized to documents satisfying schemas. Schemas are usually captured by regular tree languages (where only the labels and not the data values are used). It is known that regular tree languages can be characterized by $\mathrm{EMSO}^2(+1)$ formulas, i.e., formulas where an $\mathrm{FO}^2(+1)$ formula is preceded by a block of existential quantifiers ranging over sets of nodes. When satisfiability is concerned, it is straightforward that decidability of $\mathrm{FO}^2(\sim,+1)$ implies decidability of $\mathrm{EMSO}^2(\sim,+1)$. Thus, by combining formulas in a suitable way, it follows easily that the implication problem for unary key and inclusion constraints (and thus also for foreign key constraints) is decidable also relative to a schema given by a regular tree language. This result was already known from [2], who shows that the satisfiability problem is in NPTIME.

- Furthermore, tree automata can be used to assign *types* to nodes of a document. Integrity constraints can refer to such types. Therefore, as these types can also be expressed by $\mathrm{EMSO}^2(\sim,+1)$ formulas, the implication problem for more involved integrity constraints is still decidable.

- Another application of the logical result considers the containment problem for XPath *with attribute equalities*. We present a fragment of XPath which allows equalities and inequalities on attribute values for which the containment problem can be reduced to satisfiability of $\mathrm{FO}^2(\sim,+1)$. By combining techniques, we obtain decidability of the containment for this XPath fragment even relative to a schema consisting of a regular tree language and unary constraints.

In the following we give an overview of the structure of the paper together with the main contributions. After the introduction and a section which fixes some notation, Section 3 contains the main technical result of the paper, that $\mathrm{FO}^2(\sim,+1)$ (and therefore $\mathrm{EMSO}^2(\sim,+1)$) is decidable. In Section 4, we show that satisfiability and implication for unary key and inclusion constraints is decidable. Section 5 establishes decidability of the containment problem for an XPath fragment with attribute equalities. In Section 6 we give strong evidence that decidability of $\mathrm{FO}^2(\sim,<,+1)$ on unranked trees is a difficult problem by reducing the non-emptiness problem for vector addition tree automata to it. Most of the proofs will be found in the full version of the paper.

An additional contribution of this work is a unified framework for several decidability questions that were studied separately in the past: consistency of integrity constraints and satisfiability of queries involving data.

Related work Closely related to our work are the papers [3, 12] and the references therein. Most of the fragments they consider are inside Core-XPath (i.e., without data values). However several of the fragments are in Core-Data-XPath. In [3] the decidable fragments of Core-Data-XPath that have data equality tests either don't have negation or don't have recursion. They also don't have horizontal navigation and therefore miss an important aspect of XML navigation features. The paper [12] extends the results of [3] by including the horizontal axis but the only decidable fragment presented in this paper does not have negation. The logic $\mathrm{FO}^2(\sim,+1)$ does have negation and it can investigate nodes that are arbitrarily deep in the input tree. Finally the focus of [3, 12] was to have the precise complexity for the decision procedure of the fragment considered while the precise complexity of $\mathrm{FO}^2(\sim,+1)$ is still an open issue: An inspection of the current proof of Theorem 1 gives an upper bound of 3NEXPTIME, and the best lower bound we currently have is NEXPTIME-hardness.

The logic considered in [1] in order to solve the type inference problem is also incomparable to $\mathrm{FO}^2(\sim,+1)$. It uses patterns with variables for the data values together with equality and inequality constraints on the variables in order to extract the relevant pieces of data. It can use arbitrarily many variables in the patterns, something $\mathrm{FO}^2(\sim,+1)$

[1] The inclusion of $\mathrm{FO}^2(\sim,<,+1)$ into XPath is done as in the translation of $\mathrm{FO}^2(<,+1)$ into unary-TL over words [11]. It is strict because $\mathrm{FO}^2(\sim,<,+1)$ cannot express the test `Self/@A = Self//b//c/@A`

cannot do, but there is no recursion in the patterns and therefore it can only inspect the tree up to a given constant depth.

As we have already mentioned, restricting FO to its two-variable fragment is a classical idea when looking for decidability [14]. Over graphs or over any relational structures, FO is undecidable, while its two-variable fragment is decidable [19]. This does not imply anything on the decidability of $\mathrm{FO}^2(\sim,+1)$, since the equivalence relation and the two tree successor relations cannot be axiomatized in FO^2. A recent paper generalized the result of [19] in the presence of one or two equivalence relations [16]. Again this does not apply to our context as we also have two successor relations. However [16] also showed that the two-variable fragment of FO with *three* equivalence relations, without any other structure, is undecidable. This implies that $\mathrm{FO}^2(\sim_1,\sim_2,\sim_3,+1)$ is undecidable and that manipulating more than two different attributes at the same time quickly leads to undecidability. Note that this does not imply anything on XPath, as already in the presence of two equivalence relations the logic $\mathrm{FO}^2(\sim_1,\sim_2,+1)$ is no longer included in XPath. For instance, XPath cannot check that a node, represented by x has the following property: $\exists y\ y \neq x \wedge x \sim_1 y \wedge x \sim_2 y$ expressing that there exists another node with the same two attribute values as x.

Results on consistency of integrity constraints in the presence of DTDs were surveyed in [2]. All of the results were obtained for DTDs where the tag names and the types are coupled (all tag name have the same type), and the extension to decoupled DTDs (also known as *extended DTDs*) was left open. In particular [2] showed that it is decidable whether a set of unary keys and foreign keys is consistent with a DTD. The decidability of $\mathrm{FO}^2(\sim,+1)$ implies that it is decidable whether any set of integrity constraints definable in $\mathrm{FO}^2(\sim,+1)$ is consistent with an extended DTD. In particular, as (absolute) unary keys and foreign keys are definable in $\mathrm{FO}^2(\sim,+1)$ this extends one of the the results of [2].

Another related line of research is to consider logics and automata on words with data values. In [5, 15, 22] automata and logics over words with data values were considered. The automata of [5] had very limited expressive power, while the logics and automata of [15, 22] were undecidable. In [4] it is shown that $\mathrm{FO}^2(\sim,<,+1)$ is decidable on words with data values. In [10] an extension of LTL was given which can manipulate data values using a *freeze* operator. Their decidable fragment is incomparable to $\mathrm{FO}^2(\sim,<,+1)$ as it can only process the word from left-to-right, but can express properties that $\mathrm{FO}^2(\sim,<,+1)$ cannot. In any case those references considered only the case of words, which turns out to be considerably easier.

2. NOTATIONS AND PRELIMINARIES

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**[2] $v.\mathrm{data} \in \mathbb{N}$.

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 $\mathrm{FO}^2(\sim,<,+1)$ for two-variable logic with all these predicates and $\mathrm{FO}^2(\sim,+1)$ for the logic without $E_\Rightarrow$ and $E_\Downarrow$. By $\mathrm{FO}^2(<,+1)$ and $\mathrm{FO}^2(+1)$ we denote the respective logics without $\sim$.

Abusing the notation, we allow ourselves to use these predicates outside of logical formulas, e.g., we write $v \sim w$ for two nodes v, w that have the same data value. We also write $v \sim d$ if the data value of the node v is d.

For a data tree t, the **underlying graph** $G(t)$ of t is the graph induced by $E_\rightarrow$ and $E_\downarrow$. A set of nodes of a data tree t is called **connected** if the induced subgraph is connected.

For a data value d, the ***d*-class** of t is the set of all nodes with data value d. A **class** is a d-class, for some d. A **zone** is a maximal connected set of nodes with the same data value. Two zones are **adjacent** if they are connected by an edge (in the underlying graph). Zones are illustrated in Figure 1.

We associate with every node v in a data tree a **node profile**, which contains the information which of the right neighbor, the left neighbor and the parent of v have the same data value as v. Let Pro denote the set of the eight possible node profiles. For a data tree t over Σ, the **profiled tree** of t is the data tree over $\Sigma \times \mathrm{Pro}$ obtained by adding to each node its profile.

For a data tree t over Σ, the **data erasure** of t is the tree over Σ obtained from t by ignoring the data value $v.\mathrm{data}$ of each node.

3. DECIDABILITY OF $\mathrm{FO}^2(\sim,+1)$ ON TREES

In this section, we present the main result of the paper.

THEOREM 1. *The logic* $\mathrm{FO}^2(\sim,+1)$ *is decidable on unranked data trees.*

The proof is a bit involved and can be found in the full version of the paper.

It consists of three main parts:

- First, the satisfiability problem for $\mathrm{FO}^2(\sim,+1)$ on unranked data trees is reduced to a *puzzle* problem which asks for the existence of a data tree with certain properties.
- Next, it is shown that, every solvable puzzle problem P even has a solution with certain size constraints which only depend on the size of P.
- Finally, it is proved that whether a solution fulfilling these size constraints exists can be tested by a certain kind of (extended) tree automaton which has decidable non-emptiness.

[2]We could choose any other infinite set instead of $\mathbb{N}$, as formulas can compare values only with respect to equality.

We next define automata and puzzles. There are several possible equivalent definitions of automata over unranked trees. We use the one of [7, 17] (also known as hedge automata), which is easier to translate into $\mathrm{EMSO}^2(+1)$.

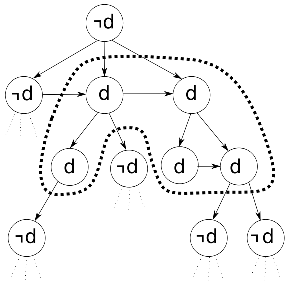
A **nondeterministic automaton over unranked trees** has a set Q of states, along with relations

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

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 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 $(\rho(v), a, \rho(w)) \in \delta_v$ belongs to the vertical transition relation δ_v.

A run is **accepting** when: a) every leaf without horizontal predecessors is labeled with one of the designated **initial states** $I \subseteq Q$; and b) 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.



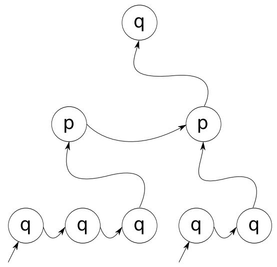


Figure 1: Illustration of zones (each node is represented with its data value) and runs (each node is represented with the state given by the run).

Fact 1. *For every regular tree language there is an equivalent formula of the form $\exists R_1, \ldots, R_n\varphi$ and vice versa, where $R_1, \ldots, R_n$ are set variables and $\varphi \in \mathrm{FO}^2(+1)$.*

Note that Fact 1 talks about trees without data values.

A **puzzle** over Σ is a pair (L, F), where L is a regular language over $\Sigma \times \mathrm{Pro}$ and F is a set of **accepting pairs** of the form $(D, S) \in 2^\Sigma \times 2^\Sigma$ where D and S are disjoint. A data tree t over Σ is a **solution** to (L, F) if

- the data erasure of the profiled tree of t belongs to L, and
- for each class in t there is some accepting pair $(D, S) \in F$ such that all nodes in that class have labels from $D \cup S$ and every label in D occurs exactly once.

We call D the **dog** letters of the pair and S the **sheep** letters.

3.1 Reduction to puzzles

In this subsection we sketch how satisfiability of $\mathrm{FO}^2(\sim, +1)$ can be reduced to solving puzzles.

Proposition 1. *For every formula φ of $\mathrm{FO}^2(\sim, +1)$ one can calculate a puzzle that has a solution if and only if φ is satisfiable.*

Note that the opposite reduction does not hold as a puzzle can express any regular property on trees. However it is easy to see that being a solution of a puzzle can be expressed in $\mathrm{EMSO}^2(\sim, +1)$. In this sense, puzzles can be seen as a normal form for $\mathrm{EMSO}^2(\sim, +1)$.

The proof is by rewriting the formula φ into a normal form, called "data normal form". We begin by defining this normal form and showing how it can be converted into a puzzle.

The idea behind data normal form is that, by introducing existential quantification over predicates, all $\mathrm{FO}^2(\sim, +1)$ formulas can be expressed using boolean combinations of very simple building blocks. These building blocks are called **simple** formulas. There are five kinds of simple formulas:

(a) A **data-blind** property, i.e. one that does not use $\sim$.

(b) "Each class contains at most one node with α."

(c) "Each class with at least one α has no β."

(d) "Each class with at least one α also has a β."

(e) "Each position with α has profile $p \in \mathrm{Pro}$"

The last kind is parameterized by a profile p, and the four latter kinds are parameterized by **types** α, β, which are conjunctions of unary predicates or their negations. A formula is in **data normal form** if it is a disjunction of formulas:

$$\exists R_1 \cdots R_m \bigwedge_{i \in I} \theta_i,$$

where each formula θ_i is simple.

Lemma 1. *For every formula φ in data normal form we one calculate a puzzle that has a solution if and only if φ is satisfiable.*

Proof Idea. Basically, formulas of the forms (b), (c) and (d) can be turned into dog and sheep pairs. Conditions of the forms (a) and (e) are covered by the regular language of the puzzle. $\square$

The following lemma completes our reduction of two-variable logic to puzzles from Proposition 1. A similar result for data strings is shown in [4].

Lemma 2. *Every formula of $\mathrm{FO}^2(\sim, +1)$ can be effectively transformed into an equivalent formula in data normal form with at most doubly exponentially many disjuncts, each of at most exponential size.*

To give the flavor of the proof of this lemma, let us consider the following property, which is not in data normal form: "every class contains either zero or at least two occurrences of the label a". This can be transformed into data normal form as follows:

> There exists a unary predicate R such that each class with at least one a also has an $a \wedge R$ and an $a \wedge \neg R$.

3.2 A Small Model Property

We fix a puzzle $P = (L, F)$ for the rest of this subsection. We assume that P has a solution. Our goal is Proposition 2, which says that this solution can be transformed into a solution where "few" zones are "large".

We also fix a nondeterministic automaton A over unranked, profiled trees without data that recognizes the regular language L and an accepting run ρ of this automaton which associates a state of Q to each node of t. For simplicity, we assume that the state of a node implies its profile – that is two nodes with the same state have the same profile. Any automaton A can be easily transformed into one that satisfies this assumption.

We need some more terminology which we define next.

- The set of all children of some node is called a **siblinghood**. Any contiguous sequence of siblings is called an **interval**.
- A **twin-pair** $p = (v, v')$ is a pair of two consecutive siblings, i.e., v' is the horizontal successor of v.

  The **left interface** of an interval I is the twin-pair (v, v') consisting of the left-most node v' of I and its left neighbor v. If there is no such left neighbor then we set $v = \perp$. Correspondingly, we define the **right interface**.

  An interface, in which the two data values are different is called a **border interface**.
- A node with data value d is also called a ***d*-node**. An interval consisting solely of d-nodes is called ***d*-pure**. If the exact value of d does not matter, we simply call it **pure**. An interval is called **complete** if both its interfaces are border interfaces.
- If the parent of an interval (node, siblinghood) has value d we call it a ***d*-parent** interval (node, siblinghood).
- For a data value d, a ***d*-path** is a set of d-valued nodes connected by the *vertical* successor relation. A **data path** is a d-path for some d.

Figure 2 illustrates some of these terms.

$$\underbrace{d_1|d_1\ \overbrace{\overbrace{|d_2\ \underbrace{|d_2|d_2|d_2|}_{\text{pure interval}}\ d_2|}^{\text{complete pure interval}}\ d_3|d_4\ \underbrace{|d_4|d_5|d_5|}_{\text{interval}}\ d_5|}^{\text{complete interval}}\ d_6|d_7|d_7|}_{\text{siblinghood}}$$

Figure 2: Different types of intervals

Given $M, N \in \mathbb{N}$, a data tree is said to be **(M, N)-reduced** if it has at most M data zones of size more than N and at most M siblinghoods with more than N complete pure intervals.

PROPOSITION 2. *From each puzzle P, numbers M, N can be computed such that P has a solution if and only if it has an (M, N)-reduced solution.*

The proof of Proposition 2 consists of a long sequence of steps combining cut-and-paste and counting arguments; in each step we modify an existing solution into one which is closer to being (M, N)-reduced. These steps form the most technical part of the paper. The precise statement of the propositions together with their proofs will appear in the full version of this paper. We only give some intuition here.

More precisely, we show that if there is a solution for P there is one with the following properties:

(1) At most M_1 complete pure intervals have more than N_1 nodes.

(2) At most M_2 siblinghoods contain more than N_2 complete pure intervals.

(3) There is a set $\mathcal{P}$ of at most M_3 zones such that all data paths disjoint with $\bigcup \mathcal{P}$ have at most N_3 nodes.

All the constants used above depend only on the size of the puzzle P, see Table 1 for their asymptotic values.

Const.	Value	Const.	Value
M_1	$\lvert F\rvert\lvert Q\rvert^{O(\lvert Q\rvert)}$	N_1	$O(\lvert Q\rvert^2\lvert\Sigma\rvert)$
M_2	$\lvert F\rvert\lvert Q\rvert^{O(\lvert Q\rvert)}$	N_2	$O(\lvert\Sigma\rvert\lvert Q\rvert^3)$
M_3	$\lvert F\rvert\lvert Q\rvert^{O(\lvert Q\rvert)}$	N_3	$O(\lvert\Sigma\rvert\lvert Q\rvert^2)$

Table 1: The constants used for pruning.

Once we have such a solution, one can easily see that at most

$$M = M_1 + M_2 + M_3$$

zones contain more than

$$N = (N_1 \cdot N_2)^{N_3+1}$$

nodes, thereby proving Proposition 2.

The proof of the items mentioned above is done sequentially. We first prove (1) by reducing the size of most intervals. Then we take care of (2) and reduce the size of most siblinghoods. This second step is achieved without increasing the size of any interval, thus (1) remains true. Finally we prove (3) reducing the depth of most zones. Again this is done without violating steps (1) and (2).

Each step is rather technical but follows the same pattern. We distinguish two cases. In the first case we assume that the number of *problematic* data values yielding a long interval, or a big siblinghood, or a deep zone is "huge", where "huge" is a constant that we can compute from P. In this case, we show that we can transfer part of the big interval/siblinghood/zone while changing its data value in a way that it decreases the number of problematic data values. The "huge" number guarantees that it is always possible to find a new data value which satisfies all the conditions required for being able to do such a transfer without violating the conditions of P.

In the second case, when the number of problematic data values is small, we prove (1), (2) and (3) independently for each data value. In other words we prove a *local* variant of (1), (2) and (3) where each statement is relativized to a data value d: d-pure intervals, d-parent siblinghoods, d-paths. This can be done by moving parts of the tree from one place to another without changing any data value.

The details will be found in the full version of the paper.

3.3 Satisfiability for Small Models

The final step of the proof of 1 consists of the following proposition:

PROPOSITION 3. *Given a puzzle P and numbers M, N, it is decidable whether P has an (M, N)-reduced solution.*

The proof is by reduction to the emptiness of linear constraint tree automata. We first define these automata and show that their non-emptiness problem is decidable. Afterwards we present the reduction.

A linear constraint automaton over unranked trees is obtained from a (nondeterministic) tree automaton by adding linear constraints on the number of times states appear in a run.

A **linear inequality** over variable set X is an expression of the form

$$\sum_{x\in X} k_x \cdot x \quad \geq \quad 0 \qquad k_x \in \mathbb{Z} \ .$$

A **linear constraint** over X is a boolean combination of linear inequalities. A **solution** of a linear constraint is a valuation $\nu : X \to \mathbb{N}$ satisfying it in the usual way.

A **linear constraint tree automaton (LCTA)** is a nondeterministic unranked tree automaton $\mathcal{A}$ with state space Q, together with a linear constraint over Q. The LCTA accepts a tree if the tree admits a run $\rho : V \to Q$ of $\mathcal{A}$ on t, which accepts in the usual sense, and which moreover satisfies the linear constraint wrt. its Parikh image $(|\rho^{-1}(q)|)_{q\in Q}$.

It is not hard to see that emptiness of LCTA is decidable, since context-free languages have semilinear Parikh images, and semilinear sets are closed under intersection. The following result shows that the complexity of deciding emptiness is NPTIME, by a similar proof as [21]. The latter paper shows how to compute in linear time an existential Presburger formula for the Parikh image of a context-free language described by a grammar. The proof in [21] can be directly adapted to extended context-free grammars, i.e., grammars with rules $A \to L_A$, where $L_A \subseteq \Sigma^*$ is a regular language giving the possible right-hand sides of the rule with left-hand side A.

THEOREM 2. *Emptiness of LCTA is in* NPTIME.

Note that it is important here that the linear constraints speak of states and not of letters of the input. Even over words, an automaton with linear constraints over letters in the input cannot recognize the language $\{b^m a^n b^n \mid m, n \in \mathbb{N}\}$.

The following proposition shows that when restricted to (M, N)-reduced solutions, LCTA can recognize the data erasure of puzzle solutions:

PROPOSITION 4. *Given a puzzle P over Σ and numbers M, N, one can compute an LCTA that recognizes the data erasure of (M, N)-reduced solutions of P.*

Let the puzzle $P = (L, F)$ be given, with L a regular language of unranked trees over $\Sigma \times \text{Pro}$ and F the set of accepting pairs $(D, S) \in 2^\Sigma \times 2^\Sigma$. Recall that for each accepting pair (D, S) each dog-letter of D must appear exactly once in each class of a solution of P which fulfills (D, S). Recall also that each class is partitioned into zones. Each zone of a class may contain some of the dog-letters. The set of zones containing a dog letter induces a partition of D. For each accepting pair (D, S) and each partition π of D, we call the triple (D, S, π) a **class type**. A class is of class type (D, S, π) if it fulfills (D, S) and the zones of the class partition D according to π.

We fix a tree automaton $\mathcal{A}$ with state space Q that accepts L. We also fix the following constant.

$$K = 2M + (N + 1)^2|\Sigma| + 2 \ .$$

Let ν be a function that assigns to every class type a number from $\{0, \dots, K\}$. We define L_ν to be the set of trees t' such that t' is the data erasure of some (M, N)-reduced profiled data tree t such that $\mathcal{A}$ has an accepting run ρ over t and, for every class type τ:

- If $\nu(\tau) < K$, then t has exactly $\nu(\tau)$ classes of type τ.
- If $\nu(\tau) = K$, then there are at least K classes in t of type τ.

LEMMA 3. *For every function ν, there is an LCTA $\mathcal{A}_\nu$ that recognizes L_ν. The size of $\mathcal{A}_\nu$ is exponential in the size of P.*

Proposition 4 follows immediately from this lemma, by taking a disjunction of automata over all possible functions ν.

4. 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 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*). One of the advantages of a logic-based approach to decidability is the compositionality of logic. This holds especially for $\text{FO}^2(\sim, +1)$, which is closed under all boolean operations and, as far as satisfiability is concerned even under existential set quantification.

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 inclusion constraints are decidable, even relative to structural constraints given by a regular tree language.

We first deal with regular tree languages and types.

Let $\text{EMSO}^2(\sim, +1)$ be the extension of $\text{FO}^2(\sim, +1)$ consisting of all formulas starting with a sequence of existential quantifiers over unary predicates (i.e., set variables) followed by an $\text{FO}^2(\sim, +1)$ formula. For satisfiability, it does not matter whether unary predicates are part of the signature or existentially quantified in the prefix of a formula. Therefore, from Theorem 1 we immediately get the following corollary.

COROLLARY 1. *The logic $\text{EMSO}^2(\sim, +1)$ over unranked data trees is decidable.*

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

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

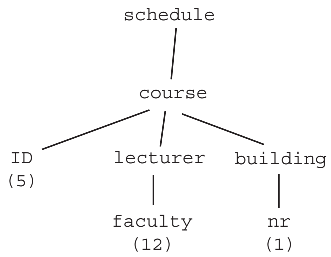


Figure 3: An XML document and its data tree encoding. In the encoding, data values are in parentheses. Data values for non-attribute nodes are not used.

run.[3] Recall from Fact 1 that every regular tree language is expressible in $\mathrm{EMSO}^2(+1)$.

A **key constraint** is an expression of the form $\tau[X] \to \tau$ 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 $|X| = |Y| = 1$.

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[4], 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.

PROPOSITION 5. *The consistency and implication problems for unary keys and unary inclusion constraints relative to a regular tree language are decidable.*

PROOF. We only consider the more general, implication problem. We encode XML documents as trees in a way which closely corresponds to the XPath data model [23], i.e., the attributes of a node v are represented by attribute nodes (labelled 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 labelled with B. An example of this encoding is presented in Figure 3.

Let A and K_1, K_2 be given, where the states of A induce the types in the constraints. By Fact 1, from A we can derive an $\mathrm{EMSO}^2(\sim, +1)$ formula of the form $\exists R_1, \ldots, R_n \varphi_A$ which (1) holds in a tree t if and only if $t \in L(A)$ and (2) such that, for each state τ of A, a position satisfies the predicate R_τ if and only if the position has type τ (uses the state τ in the unique accepting run). Thus, a unary key constraint $U : \tau[B] \to \tau$ can be expressed by the $\mathrm{FO}^2(\sim, +1)$ formula $\varphi_U =$

$$\forall x \forall y \quad \left(\begin{array}{c} (B(x) \wedge \exists y R_\tau(y) \wedge E_\downarrow(y, x)) \wedge \\ (B(y) \wedge \exists x R_\tau(x) \wedge E_\downarrow(x, y)) \wedge \\ x \sim y \end{array} \right) \to x = y \quad .$$

An inclusion constraint $U : \tau_1[B_1] \subseteq \tau_2[B_2]$ can be expressed by the $\mathrm{FO}^2(\sim, +1)$ formula $\varphi_U =$

$$\begin{array}{ll} \forall x & (B_1(x) \wedge \exists y (R_{\tau_1}(y) \wedge E_\downarrow(y, x))) \to \\ & \exists y\, (x \sim y \wedge B_2(y) \wedge \exists x (R_{\tau_2}(x) \wedge E_\downarrow(x, y))). \end{array}$$

Thus, the implication problem reduces to satisfiability of the following formula of $\mathrm{EMSO}^2(\sim, +1)$:

$$\exists R_1, \ldots, R_n (\varphi_A \wedge \bigwedge_{U \in K_1} \varphi_U \wedge \neg \bigwedge_{U \in K_2} \varphi_U).$$

$\square$

Note, that by combining key and inclusion constraints also foreign key constraints can be covered. In [2] a special case of Proposition 5 was proved: the consistency problem for unary keys and foreign keys is NP-complete relative to DTD types. The extension to XML Schema's typing system (and to any regular tree language) was left as an open question. Note also that we do not know yet the precise complexity of the implication problem, we only have the upper-bound given by the analysis of the proof of Theorem 1: 3NEXPTIME.

5. XPATH CONTAINMENT

In this section we define a fragment of XPath, which we call *LocalDataXPath,* which is captured by $\mathrm{FO}^2(\sim, +1)$, for the purpose of static analysis. More precisely, satisfiability and containment test for unary queries expressed in these fragments, possibly in the presence of integrity constraints and schemas, can be reduced to satisfiability of $\mathrm{FO}^2(\sim, +1)$.

Most fragments of XPath for which the containment problem has been studied and established to be decidable, do not allow reference to attribute values. The language LocalDataXPath allows the comparison of attribute values, but compared with Core-Data-XPath it 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 LocalDataXPath only the following axes are allowed:

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

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.

We define the syntax of LocalDataXPath next. For the purpose of this paper it is given in a simplified form as to compared with XPath.

[3]In XML Schema it is basically required that A has a unique accepting run, thus the type of each node is uniquely determined.

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

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

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

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

The following expression is also safe:

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

THEOREM 3. *Satisfiability and Containment for (unary or binary) LocalDataXPath safe expressions is decidable. This holds even relative to a schema consisting of a regular tree language and unary key and inclusion constraints.*

PROOF. (sketch) The proof is, of course, by translating the expressions into $FO^2(\sim, +1)$ formulas. We encode XML documents as in the proof of Proposition 5 (using the XPath data model) 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 $FO^2(<, +1)$ of [18].

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

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

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

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

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 $FO^2(\sim, +1)$ 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 4 to obtain the second statement of the theorem.

Containment for binary queries can be handled by having two distinguished nodes in each tree which correspond to a pair in the query result. □

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 $FO^2(\sim, <, +1)$. Unfortunately, we do not know if satisfiability of $FO^2(\sim, <, +1)$ is decidable.

6. A LOWER BOUND FOR $FO^2(\sim, <, +1)$

In this section we show that satisfiability of $FO^2(\sim, <, +1)$ on (even binary) trees is at least as hard as checking non-emptiness for vector addition tree automata. The decidability of the latter has been an open problem for many years and is, in turn, equivalent to a notorious open problem in linear logic, the decidability of MELL (Multiplicative Exponential Linear Logic) (see [9] and the references therein). Therefore proving decidability of $FO^2(\sim, <, +1)$ on trees seems to be quite challenging.

A vector addition tree automaton over binary trees is a kind of bottom-up automaton which assigns to every node, besides a state, a vector over $\mathbb{N}$. A transition has three vectors $\vec{a}, \vec{b}, \vec{c}$, three states q_0, q_1, q and a label l as parameters. It can be applied to a node v with children v_0, v_1 if

- v has label l

- v_0 has state q_0 and v_1 has state q_1
- $\vec{x} - \vec{a} \geq \vec{0}$ and $\vec{y} - \vec{b} \geq \vec{0}$, where $\vec{x}$ and $\vec{y}$ denote the vector at v_0 and v_1, respectively.

If the transition is applied then v gets state q and the vector $(\vec{x} - \vec{a}) + (\vec{y} - \vec{b}) + \vec{c}$.

More formally a **vector addition automaton** A is a tuple
$(\Sigma, k, Q, F, \delta_0, \delta)$ where Σ is a finite alphabet, k is the arity of the vectors of the automaton, Q is a finite set of states, $F \subseteq Q$ a set of accepting states, $\delta \subset \Sigma \times (Q \times \mathbb{N}^k)^2 \times Q \times \mathbb{N}^k$ is a finite (!) set of transitions and $\delta_0 \subset \Sigma \times Q \times \mathbb{N}$ is a finite set of initial transitions.

A **run** of the automaton assigns to each node a state and a k-ary vector. For a leaf of label a, the state q and the vector $\vec{n}$ have to fulfill $\delta_0(a, q, \vec{n})$ holds. For inner nodes, some transition has to be applicable, as described above.

A tree is accepted if the root carries an accepting state and the vector $\vec{0}$.

Note that the automata *cannot* test whether a component of a vector is equal to zero (otherwise the model would be immediately undecidable) and that vectors never assume negative values.

A different view of vector addition automata considers the components of the vectors as counters. In the following proof we will adapt this view to improve intuition.

THEOREM 4. *For any vector addition tree automaton A, a formula $\varphi_A \in \mathrm{FO}^2(\sim, <, +1)$ can be computed such that $L(A) \neq \emptyset$ iff φ_A has a model.*

PROOF. (sketch) Let k be the number of counters of A and Q be its set of states.

The formula φ_A uses one unary predicate P_q for each $q \in Q$ and two unary predicates I_i, D_i for each $i \in [1, k]$. The intended meaning of I_i is that counter i is increased by one while D_i means that counter i is decreased by 1. The models of φ_A are going to be trees coding possible runs of A. In such a tree a transition

$$\delta(a, q_1, (a_1, \cdots, a_k), q_2, (b_1, \cdots, b_k), q, (c_1, \cdots, c_k))$$

of A is represented by a subtree which has, for each i, in its top branch c_i symbols I_i, in the left branch a_i symbols D_i and, in the right branch b_i symbols D_i, as depicted in Figure 4. The leave conditions are handled in the same fashion. That a tree consists of such patterns can be easily described in $\mathrm{FO}^2(+1)$.

To check that all counters always have non-negative values and the value zero at the root, data values are employed. To this end, checking for all $i \in [1, k]$ that (1) all nodes labeled I_i have different data values, (2) all nodes labeled D_i have different data values, (3) for each node labeled I_i there is an ancestor labeled D_i with the same data value, (4) for each node labeled D_i there is a descendant labeled I_i with the same data value.

It is easy to express (1) - (4) in $\mathrm{FO}^2(\sim, <, +1)$. Further, it is easy to see that (1) - (4) imply that the overall number of decrements of each counter is equal to the number of increments, therefore all counters have value zero at the root. Moreover each decrement is preceded by an increment (below), therefore the value of each counter is always non-negative. (The reader should convince her- or himself that $\mathrm{FO}^2(\sim, <, +1)$ does not seem to be able to check whether at an inner node a counter has value zero.) $\square$

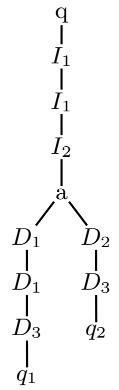


Figure 4: Coding a transition: An example with $\delta(a, q_1, (2, 0, 1), q_2, (0, 1, 1), q, (2, 1, 0))$

7. CONCLUSION

An interesting aspect of this work is to present in a unified framework decidability results that were studied separately in the past: consistency of integrity constraints and satisfiability of queries. In the future we hope to be able to also include related problems into the picture like the type inference problem [1].

Our main technical result is the decidability of $\mathrm{FO}^2(\sim, +1)$, which can be seen as a non trivial decidable fragment of XPath. A close inspection of the proof of Theorem 1 gives a upper bound of 3NEXPTIME for the decision procedure. A NEXPTIME-hardness lower bound is easy to obtain. It would be interesting to know the precise complexity of the problem.

Another obvious open question is to know whether this can be extended by allowing more features in the language. We have already mentioned the open and challenging problem of the decidability of $\mathrm{FO}^2(\sim, <, +1)$.

Maybe more doable would be to know whether $\mathrm{FO}^2(\sim, +\omega)$ is decidable. This logic can use predicates of the form $E^k_\downarrow$ and $E^k_\rightarrow$ testing whether two nodes are at distance exactly k (downwards or rightwards). This is a proper extension, since $\mathrm{FO}^2(\sim, +1)$ cannot express the fact that a position x has the same data value as its grandfather. However this feature would be useful in practice in order to be able to express tree pattern queries which do not only depend on the labels of the nodes but also how their data values compare. It would also be useful in order to express more integrity constraints, in particular some of the *relative keys* and *relative inclusion constraints* which are stated relative to a given context. Such integrity constraints were investigated in [2] in the presence of DTDs. We leave the decidability of $\mathrm{FO}^2(\sim, +\omega)$ as an open problem.

Another interesting issue would be to find an algebraic form of the considered logics. In particular, we would like to find a decidable model of tree automata that can manipulate data values and express at least all of $\mathrm{FO}^2(\sim, +1)$. Unfortunately two-way automata using registers or pebbles for comparing data values are undecidable even when using only one such register or pebble [8].

8. REFERENCES

[1] N. Alon, T. Milo, F. Neven, D. Suciu and V. Vianu. XML with Data Values: Typechecking Revisited In

Journal of Computer and System Sciences, 66(4): 688-727 (2003).

[2] M. Arenas, W. Fan and L. Libkin. Consistency of XML specifications. In *Inconsistency Tolerance*, Springer LNCS vol. 3300, 2005, pages 15-41.

[3] M. Benedikt, W. Fan, and F. Geerts. XPath Satisfiability in the Presence of DTDs. In *PODS'05*, 2005.

[4] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on words with data. LIAFA research report, available at `http://www.liafa.jussieu.fr/web9/rapportrech/description_fr.php?idrapportrech=678`

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

[6] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, W. C. Tan. Reasoning about keys for XML. In *Inf. Syst.*, 28(8): 1037-1063 (2003).

[7] J. Cristau, C. Löding, W. Thomas. Deterministic Automata on Unranked Trees. In *Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT)*, LNCS (to appear), 2005.

[8] C. David. Mots et données infinis. Master's thesis, Université Paris 7, LIAFA, 2004.

[9] P. de Groote, B. Guillaume, and S. Salvati. Vector Addition Tree Automata. In *LICS'04*, pp. 64-73, 2004.

[10] S. Demri, R. Lazic, D. Nowak. On the Freeze Quantifier in Constraint LTL: Decidability and Complexity. In *TIME'05*, 2005.

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

[12] F. Geerts and W. Fan. Satisfiability of XPath Queries with Sibling Axes. In *DBPL'05*, 2005.

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

[14] E. Grädel and M. Otto. On Logics with Two Variables. *Theor. Comp. Sci.*, 224:73-113 (1999).

[15] M. Kaminski and N. Francez. Finite memory automata. *Theor. Comp. Sci.*, 134(2):329-363 (1994).

[16] E. Kieroński and M. Otto. Small Substructures and Decidability Issues for First-Order Logic with Two Variables. In *LICS'05*, 2005.

[17] W. Martens, J. Niehren. Minimizing Tree Automata for Unranked Trees. In *10th International Symposium on Database Programming Languages*, LNCS 3774, 2005.

[18] M. Marx. First order paths in ordered trees. In *ICDT'05*, 2005.

[19] M. Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 21: 135-140 (1975).

[20] F. Neven and T. Schwentick. XPath Containment in the Presence of Disjunction, DTDs, and Variables. In *ICDT'03*, 2003.

[21] K. Neeraj Verma, H. Seidl, T. Schwentick. On the Complexity of Equational Horn Clauses. In *CADE'05*, 2005.

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

[23] XML Path Language (XPath), W3C Recommendation 16 November 1999. Available at `http://www.w3.org/TR/xpath`.