# On the Satisfiability of Two-Variable Logic over Data Words

Claire David, Leonid Libkin, and Tony Tan

School of Informatics, University of Edinburgh

**Abstract.** Data trees and data words have been studied extensively in connection with XML reasoning. These are trees or words that, in addition to labels from a finite alphabet, carry labels from an infinite alphabet (data). While in general logics such as MSO or FO are undecidable for such extensions, decidablity results for their fragments have been obtained recently, most notably for the two-variable fragments of FO and existential MSO. The proofs, however, are very long and nontrivial, and some of them come with no complexity guarantees. Here we give a much simplified proof of the decidability of two-variable logics for data words with the successor and data-equality predicates. In addition, the new proof provides several new fragments of lower complexity. The proof mixes database-inspired constraints with encodings in Presburger arithmetic.

## 1 Introduction

The classical theory of automata and formal languages deals primarily with finite alphabets. Nonetheless, there are several models of formal languages, regular or context free, that permit an infinite alphabet, e.g., [3, 6, 14, 17]. Most of the models, however, lack the usual nice decidability properties of automata over finite alphabets, unless strongs restrictions are imposed.

Recently the subject of languages over infinite alphabets received much attention due to its connection with the problems of reasoning about XML [2, 4, 5, 8, 9]. The structure of XML documents is usually modeled by labeled unranked trees [15, 21, 25], and thus standard automata techniques can be used to reason about the structure of XML documents. However, XML documents carry *data*, which is typically modeled as labeling nodes by letters from a different, infinite alphabet.

Thus, one needs to look for *decidable formalisms* in the presence of a second, infinite alphabet. Such formalisms are hard to come by, and tend to be of very high complexity. Nonetheless, some significant progress has been made recently [4]. Namely, it was shown that the restriction of first-order logic to its two-variable fragment, $\mathrm{FO}^2$, remains decidable over trees with labels coming from an infinite alphabet (we refer to them as *data trees*). This is the best possible restriction in terms of the number of variables: the three-variable fragment $\mathrm{FO}^3$ is undecidable [4].

The result is true even if the sentence is preceded by a sequence of existential monadic second-order quantifiers, i.e., for logic $\exists\text{MSO}^2$. The 30-page long proof of this decidability result, however, was very nontrivial relying on a complicated automaton model and over 20 combinatorial reduction lemmas, and gave no insight into the complexity of fragments nor the possibility of extending it to more expressive logics.

However, we do want to have tools to reason about languages over infinite alphabets, so there is a search for easier tools. One direction is to look for restrictions, both in terms of structures and logics [2, 10]. As for restrictions, the most natural idea appears to be to look for tools over *words*, rather than trees. This has been done in [5, 8], which provided decidable formalisms for *data words*, i.e., words labeled by both a finite and an infinite alphabet. In fact, [5] showed that the two-variable logic is decidable. Specifically, it showed that in the presence of both ordering and successor relations on the domain, the logic $\exists\text{MSO}^2$ is decidable; no upper bound on the complexity is known, however. The proof, however, is again highly nontrivial and not particularly modular.

Our *main goal* is to give a much simpler and completely self-contained proof of the decidability of satisfiability of the two-variable logic over data words. We do it for the case when the successor relation is available on the domain (as with the successor relation, the logic $\exists\text{MSO}^2$ can already define all regular languages). The proof avoids most of the complicated combinatorics of the proofs of [4, 5], instead relying on two key ideas: reducing the problem to resoning about some specific constraints (similar to those used in database theory [1]), and using tools based on Presburger arithmetic to reason about those.

**Organization** In Section 2 we give the key definitions of data words and logics on them and state the main result. In Section 3 we provide additional definitions and notations. In Section 4 we provide the machinery needed for the main proof. In Section 5 we present the proof of the decidability result. In conclusion, we analyse the complexity of the our decision procedures.

## 2 Data words and the main result

### 2.1 Data words

Let $\Sigma$ be a finite alphabet and $\mathfrak{D}$ be an infinite set of data values. To be concrete, we assume that $\mathfrak{D}$ contains $\mathbb{N}$, t he set of natural numbers. A data word is simply an element of $(\Sigma \times \mathfrak{D})^*$. We usually write $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$ for data words, where $a_i \in \Sigma$ and $d_i \in \mathfrak{D}$. We define the $\Sigma$ projection of $w$ as $\mathsf{Proj}(w) = a_1 \cdots a_n$.

An $a$-position is a position labeled with the symbol $a$. The set of data values found in $a$-positions of a data word $w$ is denoted by $V_w(a)$, while the number of $a$-positions in $w$ is denoted by $\#_w(a)$.

The following notion is used throughout the paper. For a set $S \subseteq \Sigma$,

$$[S]_w = \bigcap_{a \in S} V_w(a) \cap \bigcap_{b \notin S} \overline{V_w(b)}.$$

That is, $[S]_w$ is the set of data values that are found in $a$-positions for all $a \in S$ but are not found in any $b$-position for $b \notin S$. Note that the sets $[S]_w$'s are disjoint, and that $V_w(a) = \bigcup_{a \in S} [S]_w$ for each $a \in \Sigma$.

We say that a data word is *locally different*, if every position has a different data value than its left- and right-neighbors.

## 2.2 Logics over data words

For the purpose of logical definability, we view data words of length $n$ as structures

$$w = \langle \{1, \ldots, n\}, +1, \{a(\cdot)\}_{a \in \Sigma}, \sim \rangle, \tag{1}$$

where $\{1, \ldots, n\}$ is the domain of positions, $+1$ is the successor relation (i.e., $+1(i, j)$ iff $i + 1 = j$), the $a(\cdot)$'s are the labeling predicates, and $i \sim j$ holds iff positions $i$ and $j$ have the same data value.

We let FO stand for first-order logic, MSO for monadic second-order logic (which extends FO with quantification over sets of positions), and $\exists$MSO for existential monadic second order logic, i.e., sentences of the form $\exists X_1 \ldots \exists X_m\ \psi$, where $\psi$ is an FO formula over the vocabulary extended with the unary predicates $X_1, \ldots, X_m$. We let $\mathrm{FO}^2$ stand for FO with two variables, i.e., the set of FO formulae that only use two variables $x$ and $y$. The set of all sentences of the form $\exists X_1 \ldots \exists X_m\ \psi$, where $\psi$ is an $\mathrm{FO}^2$ formula is denoted by $\exists\mathrm{MSO}^2$.

To emphasize that we are talking about a logic over data words we write $(+1, \sim)$ after the logic: e.g., $\mathrm{FO}^2(+1, \sim)$ and $\exists\mathrm{MSO}^2(\sim, +1)$. Note that $\exists\mathrm{MSO}^2(+1)$ is equivalent in expressive power to MSO over the usual (not data) words, i.e., it defines precisely the regular languages [24].

It was shown in [5] that $\exists\mathrm{MSO}^2(+1, <, \sim)$ is decidable over data words. In terms of complexity, the satisfiability of this logic is shown to be at least as hard as reachability in Petri nets. Without the $+1$ relation, the complexity drops to NEXPTIME-complete; however, without $+1$ the logic is not sufficiently expressive to capture regular relations on the data-free part of the word.

Our main goal is to give a transparent and self-contained proof of the following:

**Theorem 1.** *The satisfiability problem is decidable for* $\exists\mathrm{MSO}^2(\sim, +1)$ *over data words. Moreover, the complexity of the decision procedure is elementary.*

The result itself can already be infered from the decidability proof of the logic with local navigation over data trees given in [4], which yields a 4-exponential complexity bound. However this proof does not give any hints in understanding the difficulty of the problem : it is a 30-page proof long, and goes via more than a dozen combinatorial reduction lemmas. Our proof yields a 5-exponential bound.

Neither of these bounds are of course even remotely practical. The primary goal of these results is to delineate the boundary of decidability, so that later we could search for efficient subclasses of decidable classes of formulae. And for such a search, it is crucial to have simple and well-defined tools for proving decidability; providing such tools is precisely our goal here.

Indeed a few fragments of lower complexity are already provided here. Furthermore, in [9] a fragment whose satisfiability is decidable in NP is obtained. [1] With our proof we gain some insight on how the complexity "moves up continuously" from NP to 5 exponential. Such insight is lacking in [4].

## 3 Additional notations

### 3.1 Disjunctive constraints for data words

We consider two types of constraints on data words, which are slight generalizations of keys and inclusion constraints used in relational databases [1]. They are defined as the following logical sentences.

1. A *disjunctive key* constraint (dk) is a sentence of the form:
$$\forall x \, \forall y \, \big( \big( \bigvee_{a \in \Sigma'} a(x) \wedge \bigvee_{a \in \Sigma'} a(y) \wedge x \sim y \big) \rightarrow x = y \big),$$
where $\Sigma' \subseteq \Sigma$. We denote such sentence by $V(\Sigma') \mapsto \Sigma'$.
2. A *disjunctive inclusion constraint* (dic) is as sentence of the form:
$$\forall x \, \exists y \, \big( \bigvee_{a \in \Sigma_1} a(x) \rightarrow \bigvee_{b \in \Sigma_2} b(y) \wedge x \sim y \big),$$
where $\Sigma_1, \Sigma_2 \subseteq \Sigma$. We denote such sentence by $V(\Sigma_1) \subseteq V(\Sigma_2)$.

For a set $\mathcal{C}$ of dk's and dic's, the data word $w$ satisfies $\mathcal{C}$, written as $w \models \mathcal{C}$, if $w$ satisfies all sentences in $\mathcal{C}$.

In [9] the constraints considered are when the cardinalities $|\Sigma'|$, $|\Sigma_1|$, $|\Sigma_2|$ are all one, which are simply known as key and inclusion constraint.

### 3.2 Existential Presburger formulae

Atomic Presburger formulae are of the form: $x_1 + x_2 + \cdots x_n \leq y_1 + \cdots + y_m$, or $x_1 + \cdots x_n \leq K$, or $x_1 + \cdots x_n \geq K$, for some constant $K \in \mathbb{N}$. *Existential Presburger formulae* are Presburger formulae of the form $\exists \bar{x} \; \varphi$, where $\varphi$ is a Boolean combination of atomic Presburger formulae.

We shall be using Presburger formulae defining Parikh images of words. Let $\Sigma = \{a_1, \ldots, a_k\}$, and let $v \in \Sigma^*$. By $\mathsf{Parikh}(v)$ we mean the *Parikh image* of $v$, i.e., $(\#_v(a_1), \ldots, \#_v(a_k))$, i.e., $k$-tuple of integers $(n_1, \ldots, n_k)$ so that $n_i$ is the number of occurrences of $a_i$ in $v$.

With alphabet letters, we associate variables $x_{a_1}, \ldots, x_{a_k}$. Given a Presburger formula $\varphi(x_{a_1}, \ldots, x_{a_k})$, we say that a word $v \in \Sigma^*$ satisfies it, written as $v \models \varphi(x_{a_1}, \ldots, x_{a_k})$ if and only if $\varphi(\mathsf{Parikh}(v))$ holds. It is well-known that for every regular language $L$, one can construct an existential Presburger formula $\varphi_L(x_{a_1}, \ldots, x_{a_k})$ so that a word $v$ satisfies it iff it belongs to $L$ [20]; moreover, the formula can be constructed in polynomial time [23].

[1] This fragment is context free languages with the constraints on the data values of the forms: $\forall x \forall y \; a(x) \wedge a(y) \wedge x \sim y \rightarrow x = y$, and $\forall x \exists y \; a(x) \rightarrow b(y) \wedge x \sim y$.

## 3.3 Presburger automata

A *Presburger automaton* is a pair $(\mathcal{A}, \varphi)$, where $\mathcal{A}$ is a finite state automaton and $\varphi$ is a existential Presburger formula. A word $w$ is accepted by $(\mathcal{A}, \varphi)$, denoted by $\mathcal{L}(\mathcal{A}, \varphi)$ if $w \in \mathcal{L}(\mathcal{A})$ (the language of $\mathcal{A}$) and $\varphi(\mathsf{Parikh}(w))$ holds.

**Theorem 2.** *[23] The emptiness problem for presburger automata is decidable in* NP*.*

## 3.4 Profile automata for data words

Given a data word $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$, the *profile word* of $w$, denoted by $\mathsf{Profile}(w)$, is the word

$$\mathsf{Profile}(w) = (a_1, L_1, R_1), \ldots, (a_n, L_n, R_n) \in (\Sigma \times \{*, \top, \bot\} \times \{*, \top, \bot\})^*$$

such that for each position $i = 1, \ldots, n$, the values of $L_i$ and $R_i$ are either $\top$, or $\bot$, or $*$. If $L_i = \top$ and $i > 1$, it means that the position on the left, $i - 1$, has the same data value as position $i$; otherwise $L_i = \bot$. If $i = 1$ (i.e., there is no position on the left), then $L_i = *$. The meaning of the $R_i$'s is similar with respect to positions on the right of $i$.

**Definition 1.** *A* profile automaton $\mathcal{A}$ *is a finite state automaton over the alphabet* $\Sigma \times \{*, \top, \bot\} \times \{*, \top, \bot\}$*. It defines a set* $\mathcal{L}_{data}(\mathcal{A})$ *of data words as follows:* $w \in \mathcal{L}_{data}(\mathcal{A})$ *if and only if* $\mathcal{A}$ *accepts* $\mathsf{Profile}(w)$ *in the standard sense.*

A profile automaton $\mathcal{A}$ and a set $\mathcal{C}$ of disjunctive constraints define a set of data words as follows.

$$\mathcal{L}(\mathcal{A}, \mathcal{C}) = \{w \mid w \in \mathcal{L}_{data}(\mathcal{A}) \text{ and } w \models \mathcal{C}\}.$$

## 3.5 A normal form for $\exists\mathrm{MSO}^2(\sim, +1)$

Decidability proofs for two-variable logics typically follow this pattern: first, in an easy step, a syntact normal form is established; then the hard part is combinatorial, where decidability is proved for that normal form (by establishing the finite-model property, or by automata techniques, for example).

A normal form for $\exists\mathrm{MSO}^2(\sim, +1)$ was already given in [4], and we shall use it with just a small modification. In [4] it was shown that every $\exists\mathrm{MSO}^2(\sim, +1)$ formula over data words is equivalent to a formula

$$\exists X_1 \ldots \exists X_k (\chi \wedge \bigwedge_i \varphi_i \wedge \bigwedge_j \psi_j)$$

where

1. $\chi$ describes the behavior of a profile automaton (i.e., it can be viewed as an $\mathrm{FO}^2(+1)$ formula over the extended alphabet $\Sigma \times \{*, \top, \bot\} \times \{*, \top, \bot\}$);

2. each $\varphi_i$ is of the form $\forall x \forall y (\alpha(x) \wedge \alpha(y) \wedge x \sim y \rightarrow x = y)$, where $\alpha$ is a conjunction of labeling predicates, $X_k$'s, and their negations; and
3. each $\psi_j$ is of the form $\forall x \exists y\ \alpha(x) \rightarrow (x \sim y \wedge \alpha'(y))$, with $\alpha$, $\alpha'$ as in item 2.

The number of the unary predicates $X$'s is single exponential in the size of the original input sentence.

If we extend the alphabet to $\Sigma \times 2^k$ so that each label also specifies the family of the $X_i$'s the node belongs to, then formulae in items 2 and 3 can be encoded by disjunctive constraints: formulae in item 2 become dk's $V(\Sigma') \mapsto \Sigma'$, and formulae in item 3 become dic's $V(\Sigma_1) \subseteq V(\Sigma_2)$, where $\Sigma', \Sigma_1, \Sigma_2 \subseteq \Sigma \times 2^k$.

Indeed, consider, for example, the constraint $\forall x \forall y (\alpha(x) \wedge \alpha(y) \wedge x \sim y \rightarrow x = y)$. Let $\Sigma'$ be the set of all symbols $(a, \bar{b}) \in \Sigma \times 2^k$ consistent with $\alpha$. That is, $a$ is the labeling symbol used in $\alpha$ (if $\alpha$ uses one) or an arbitrary letter (if $\alpha$ does not use a labeling predicate), and the Boolean vector $\bar{b}$ has 1 in positions of the $X_i$s used positively in $\alpha$ and 0 in positions of $X_j$'s used negatively in $\alpha$. Then the original constraint is equivalent to $V(\Sigma') \mapsto \Sigma'$. The transformation of type-2 constraints into dic's is the same. The details of this straightforward construction can be found in the Appendix.

Hence, [4] and the above, imply the following. Let SAT-PROFILE be the problem:

| PROBLEM: | SAT-PROFILE |
|---|---|
| INPUT: | a profile automaton $\mathcal{A}$ and |
| | a collection $\mathcal{C}$ of disjunctive constraints |
| QUESTION: | is there a data word $w \in \mathcal{L}_{data}(\mathcal{A})$ such that $w \models \mathcal{C}$? |

Then:

**Lemma 1.** *Given an* $\exists\mathrm{MSO}^2(\sim, +1)$ *sentence* $\varphi$*, one can construct, in triple exponential time, an instance* $(\mathcal{A}, \mathcal{C})$ *of* SAT-PROFILE *over a new alphabet* $\Sigma$ *so that* SAT-PROFILE$(\mathcal{A}, \mathcal{C})$ *returns true iff* $\varphi$ *is satisfiable. However, the size of* $(\mathcal{A}, \mathcal{C})$ *and* $\Sigma$ *is double exponential in the size of* $\varphi$*.*

Thus, our main goal now is to prove:

**Theorem 3.** SAT-PROFILE *is decidable with elementary complexity.*

The main result, Theorem 1, is an immediate consequence of Theorem 3 and Lemma 1.

## 4 Some preliminary results

**Proposition 1.** *For every data word* $w$*, the following holds.*

1. $w \models V(\Sigma') \mapsto \Sigma'$ *if and only if* $\#_w(a) = |V_w(a)|$ *for each* $a \in \Sigma'$ *and* $[S]_w = \emptyset$*, whenever* $|S \cap \Sigma'| \geq 2$.
2. $w \models V(\Sigma_1) \subseteq V(\Sigma_2)$ *if and only if* $[S]_w = \emptyset$*, for all* $S$ *such that* $S \cap \Sigma_1 \neq \emptyset$ *and* $S \cap \Sigma_2 = \emptyset$.

*Proof.* Part 1 is trivial. For part 2, note that $\bigcup_{a\in\Sigma_1} V_w(a) \subseteq \bigcup_{b\in\Sigma_2} V_w(b)$ if and only if $\left(\bigcup_{a\in\Sigma_1} V_w(a)\right) \cap \bigcap_{b\in\Sigma_2} \overline{V_w(b)} = \emptyset$, which, of course, is equivalent to $[S]_w = \emptyset$, whenever $S \cap \Sigma_1 \neq \emptyset$ and $S \cap \Sigma_2 = \emptyset$. □

**Lemma 2.** *For every set $\mathcal{C}$ of disjunctive constraints, one can construct, in single-exponential time, a Presburger formula $\varphi_{\mathcal{C}}(x_{a_1}, \ldots, x_{a_k})$ such that for every data word $w$, we have $w \models \mathcal{C}$ if and only if $\varphi_{\mathcal{C}}(\mathsf{Parikh}(\mathsf{Proj}(w)))$ holds.*

*Proof.* Let $S_1, \ldots, S_m$ be the enumeration of non-empty subsets of $\Sigma$, where $m = 2^{|\Sigma|} - 1$. The formula $\varphi_{\mathcal{C}}$ is of the form $\exists z_{S_1} \cdots \exists z_{S_m}\ \psi$, where $\psi$ is the conjunction of the following quantifier-free formulas:

P1. $x_a \geq \sum_{S\ni a} z_S$, for every $a \in \Sigma$;
P2. if $V(\Sigma') \mapsto \Sigma' \in \mathcal{C}$, we have the conjunction:

$$\bigwedge_{|S\cap\Sigma'|\geq 2} z_S = 0 \qquad \wedge \qquad \bigwedge_{a\in\Sigma'} x_a = \sum_{a\in S} z_S$$

P3. if $V(\Sigma_1) \subseteq \Sigma_2 \in \mathcal{C}$, we have the conjunction:

$$\bigwedge_{S\cap\Sigma_1\neq\emptyset \text{ and } S\cap\Sigma_2=\emptyset} z_S = 0$$

We claim that for every data word $w$, $w \models \mathcal{C}$ if and only if $\varphi_{\mathcal{C}}(\mathsf{Parikh}(\mathsf{Proj}(w)))$ holds.

Let $w$ be a data word such that $w \models \mathcal{C}$. We need to show that $\varphi_{\mathcal{C}}(\mathsf{Parikh}(\mathsf{Proj}(w)))$ holds. As witnesses for $z_S$, for each $S \subseteq \Sigma$, we pick $z_S = |[S]_w|$. Now we need to show that all the conjuctions P1–P3 above are satisfied. P1 is definitely satisfied, as for each $a \in \Sigma$, $\sum_{S\ni a} z_S = |V_w(a)| \leq \#_w(a)$. P2 and P3 follow from Proposition 1.

- If $w \models V(\Sigma') \mapsto \Sigma'$, then $\#_w(a) = |V_w(a)|$ for each $a \in \Sigma$ and $[S]_w = \emptyset$, whenever $|S \cap \Sigma'| \geq 2$. So, P2 is automatically satisfied.
- If $w \models V(\Sigma_1) \subseteq V(\Sigma_2)$, then $[S]_w = \emptyset$, for all $S$ such that $S \cap \Sigma_1 \neq \emptyset$ and $S \cap \Sigma_2 = \emptyset$. Obviously then P3 is satisfied.

Now suppose that $v$ is a word such that $\varphi_{\mathcal{C}}(\mathsf{Parikh}(v))$ holds. We can assign data values to $v$ such that the resulting data word $w$ satisfies every constraints in $\mathcal{C}$. Let $z_S = m_S$ be some witnesses of that $\varphi_{\mathcal{C}}(\mathsf{Parikh}(v))$ holds. Let $K = \sum_S m_S$. We are going to assign the data values $\{1, \ldots, K\}$ to $v$ as follows. Define a function

$$\xi : \{1, \ldots, K\} \to 2^{\Sigma} - \{\emptyset\},$$

such that $|\xi^{-1}(S)| = m_S$. We then assign the $a$-positions in $v$ with the data values $\bigcup_{a\in S} \xi^{-1}(S)$, for each $a \in \Sigma$, resulting in a data word $w$. Such assignment is possible since $\sum_{a\in S} |\xi^{-1}(S)| = \sum_{a\in S} m_S \leq \#_v(a)$. By definition of the function $\xi$, we obtain that $[S]_w = \xi^{-1}(S)$. That $w \models \mathcal{C}$ follows immediately from Proposition 1. □

Lemma 2 immediately implies the decidability of a slightly simpler version of SAT-PROFILE. Consider the following problem:

| PROBLEM: | SAT-AUTOMATON |
|---|---|
| INPUT: | a finite state automaton $\mathcal{A}$ and a collection $\mathcal{C}$ of disjunctive constraints |
| QUESTION: | is there a data word $w$ such that $\mathsf{Proj}(w) \in \mathcal{L}(\mathcal{A})$ and $w \models \mathcal{C}$? |

By Lemma 2, we can construct in exponential time a Presburger formula $\varphi_{\mathcal{C}}$ of exponential size such that for all data words $w$ we have, $w \models \mathcal{C}$ if and only if $\varphi_{\mathcal{C}}(\mathsf{Parikh}(\mathsf{Proj}(w)))$. Combining it with Theorem 2, we immediately obtain the decidability of the above problem:

**Corollary 1.** SAT-AUTOMATON *is decidable with elementary complexity.*

The following lemma is crucial in our proof of Theorem 3.

**Lemma 3.** *Let $v$ be a word over $\Sigma$. Suppose that for each $a \in \Sigma$, we are given a set $V_a$ of data values such that*

- *if $V_a = \emptyset$, then $\#_v(a) = 0$; and*
- *$\#_v(a) \geq |V_a| \geq |\Sigma| + 3$ otherwise.*

*Then we can assign a data value to each position in $v$ such that the resulting data word $w$ is locally different and for each $a \in \Sigma$, $V_a = V_w(a)$.*

*Proof.* Let $v = a_1 \cdots a_n$. First we assign data values in the following manner: *Let $a \in \Sigma$. Assign each of the data values from $V_a$ in $|V_a|$ number of $a$-positions in $v$. One position gets one data value.* Since $\#_a(v) \geq |V_a|$, such assignment is possible, and moreover, if $\#_a(v) > |V_a|$, then some $a$-positions are without data values. We do this for each $a \in \Sigma$.

Let $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$ be the resulting data word, where we write $d_i = \sharp$ to denote that position $i$ is still without data value. In the data word $w$, for each $a \in \Sigma$, we already have $V_w(a) = V_a$.

However, by assigning data values just like that, the data word $w$ may not be locally different. There may exists $i \in \{1, \ldots, n-1\}$ such that $d_i = d_{i+1}$ and $d_i, d_{i+1} \neq \sharp$. We call such a position a *conflict position*. Now, we show that we can always rearrange the data values in $w$ such that the resulting data word has no conflict positions. Suppose position $i$ is a conflict position labeled $a$. Since there are only $|\Sigma|$ symbols, the data value $d_i$ can only occur at most $|\Sigma|$ times in $w$. Since $|V_a| \geq |\Sigma| + 3 > |\Sigma|$, there exists a position $j$ such that

- $a_j = a$ and $d_j \neq \sharp$;
- $d_{j-1}, d_{j+1} \neq d_i$.

Now there are $\geq |\Sigma| + 3 - |\Sigma| = 3$ such positions. From all such positions, pick one position $j$ whose data value $d_j \neq d_{i-1}, d_{i+1}$. We can then swap the data values $d_i$ and $d_j$, resulting in less number of conflict positions inside $w$. We can repeat this process until there is no more conflict positions inside $w$.

The final step is to assign data values for the positions in $w$ which do not have data value. This is easy. Since for each $a \in \Sigma$, $|V_a| \geq |\Sigma|+3 \geq 3$, if the data value $d_i = \sharp$, then we can choose one data value from $V_{a_i}$ which is different from its left- and right-neighbors. This still ensures that we get a a locally different data word at the end. This completes the proof. $\Box$

## 5 Proof of Theorem 3

For the sake presentation, we divide it into a few subsections. In Subsection 5.1, we present our algorithm for deciding SAT-PROFILE over locally different data words. Then, we explain how our algorithm can be extended to the general case in Subsection 5.2.

### 5.1 Satisfiability over locally different data words

In this subsection we give elementary algorithm to decide the problem SAT-LOCALLY-DIFFERENT define below. This problem is still a more restricted version of SAT-PROFILE, but more general than SAT-AUTOMATON.

| PROBLEM: | SAT-LOCALLY-DIFFERENT |
|---|---|
| INPUT: | a finite state automaton $\mathcal{A}$ and |
| | a collection $\mathcal{C}$ of disjunctive constraints |
| QUESTION: | is there a locally different data word $w$ such that |
| | $\mathsf{Proj}(w) \in \mathcal{L}(\mathcal{A})$ and $w \models \mathcal{C}$? |

We further divide the proof for satisfiability SAT-LOCALLY-DIFFERENT into two cases:

– First, we show how to decide SAT-LOCALLY-DIFFERENT over data words with "many" data values.
– Second, we settle SAT-LOCALLY-DIFFERENT in the general case.

We say that a data word $w$ has *"many" data values* if for all $S \subseteq \Sigma$, the cardinality $|[S]_w|$ is either 0 or $\geq |\Sigma|+3$. Notice that if a data word $w$ has many data values, then either $|V_w(a)| = 0$ or $|V_w(a)| \geq |\Sigma| + 3$ for all $a \in \Sigma$.

**The case of data words with many data values.** By Lemma 2, we can construct a Presburger formula $\varphi_{\mathcal{C}}$ such that for every data word $w$,

$$w \models \mathcal{C} \text{ if and only if } \varphi_{\mathcal{C}}(\mathsf{Parikh}(\mathsf{Proj}(w))) \text{ holds.}$$

So, for every data word $w$, $w \in \mathcal{L}(\mathcal{A}, \mathcal{C})$ if and only if $\mathsf{Proj}(w) \in \mathcal{L}(\mathcal{A}, \varphi_{\mathcal{C}})$. Recall that the formula $\varphi_{\mathcal{C}}$ is of the form: $\exists z_{S_1} \cdots \exists z_{S_m}\ \psi_{\mathcal{C}}$, where $S_1, \ldots, S_m$ is the enumeration of non-empty subsets of $\Sigma$ and the intention of each $z_{S_i}$ is to represent $|[S_i]_w|$ for data words $w$ for which $\varphi_{\mathcal{C}}(\mathsf{Parikh}(\mathsf{Proj}(w)))$ holds.

The idea is as follows: given a set $\mathcal{F} \subseteq 2^{\Sigma} - \{\emptyset\}$, we can decide the existence of a locally different data word $w \in \mathcal{L}(\mathcal{A}, \mathcal{C})$ such that $|[S]_w| = 0$, if $S \in \mathcal{F}$ and $|[S]_w| \geq |\Sigma| + 3$, if $S \notin \mathcal{F}$.

Now, to decide the existence of a locally different data word with many data values in $\mathcal{L}(\mathcal{A}, \mathcal{C})$, we do the following.

1. Guess a set $\mathcal{F} \subseteq 2^{\Sigma} - \{\emptyset\}$.
2. Construct the formula $\varphi_{\mathcal{C}}$ from $\mathcal{C}$ according to Lemma 2.
   Let $\varphi_{\mathcal{C}}$ be in the form of $\exists z_{S_1} \cdots \exists z_{S_m}\ \psi_{\mathcal{C}}$.
3. Define the formula $\varphi_{\mathcal{C},\mathcal{F}}$ as:

$$\exists z_{S_1} \cdots \exists z_{S_m} \left( \psi_{\mathcal{C}} \wedge \bigwedge_{S_i \in \mathcal{F}} z_{S_i} = 0 \wedge \bigwedge_{S_i \notin \mathcal{F}} z_{S_i} \geq |\Sigma| + 3 \right)$$

4. Test the emptiness of $\mathcal{L}(\mathcal{A}, \varphi_{\mathcal{C},\mathcal{F}})$.

To show that such algorithm is correct, we claim the following.

**Claim 4** *For every word $v \in \Sigma^*$, $v \in \mathcal{L}(\mathcal{A}, \varphi_{\mathcal{C},\mathcal{F}})$ for some $\mathcal{F} \subseteq 2^{\Sigma}$ if and only if there exists a locally different data word $w \in \mathcal{L}(\mathcal{A}, \mathcal{C})$ with many data values such that $\mathsf{Proj}(w) = v$.*

*Proof.* If $v \in \mathcal{L}(\mathcal{A}, \varphi_{\mathcal{C},\mathcal{F}})$ for some $\mathcal{F}$, then there exist witnesses $z_{S_i} = m_{S_i}$ such that $\varphi_{\mathcal{C},\mathcal{F}}(\mathsf{Parikh}(v))$ holds. By the construction of $\varphi_{\mathcal{C},\mathcal{F}}$, we have $m_{S_i} = 0$, if $S_i \in \mathcal{F}$ and $m_{S_i} \geq |\Sigma| + 3$, if $S_i \notin \mathcal{F}$. As in the proof of Lemma 2, we can assign data values to each position of $v$, resulting in a data word $w$ such that for each $S \subseteq \Sigma$, $|[S]_w| = m_S$ which is either $\geq |\Sigma| + 3$ or 0. This means that $|V_w(a)|$ is either $\geq |\Sigma| + 3$ or 0. (If $|V_w(a)| = 0$, it means that the symbol $a$ does not appear in $v$.) By Theorem 3, we can rearrange the data values in $w$ to obtain a locally different data word. This data word is in $\mathcal{L}(\mathcal{A}, \mathcal{C})$.

The converse is straightforward. if $w \in \mathcal{L}(\mathcal{A}, \mathcal{C})$ has many data values, then $v = \mathsf{Proj}(w)$ immediately satisfies $\varphi_{\mathcal{C},\mathcal{F}}$, where $\mathcal{F} = \{S_i \mid |[S_i]_w| = 0\}$. Thus, $v \in \mathcal{L}(\mathcal{A}, \varphi_{\mathcal{C},\mathcal{F}})$. □

**The general case of** SAT-LOCALLY-DIFFERENT**.** The algorithm is more or less the same as above. The only extra care needed is to consider the case if there exists a locally different data word $w \in \mathcal{L}(\mathcal{A}, \mathcal{C})$ such that $|[S]_w| \leq |\Sigma| + 2$, for some $S \subseteq \Sigma$.

As before, the idea is to decide, given a set $\mathcal{F} \subseteq 2^{\Sigma} - \{\emptyset\}$, whether there exists a locally different data word $w \in \mathcal{L}(\mathcal{A}, \mathcal{C})$ such that $[S]_w \leq |\Sigma| + 2$, if $S \in \mathcal{F}$ and $\geq |\Sigma| + 3$, otherwise. Again, we reduce the problem to the emptiness of a Presburger automaton.

The main difference is the way to deal with the $S \in \mathcal{F}$, as $S \in \mathcal{F}$ does not always imply that $[S]_w$ is empty but only that its cardinality is bounded by the constant $|\Sigma| + 2$. For all these $S \in \mathcal{F}$, we can assume that the data $[S]_w$ consists of constants, since $|[S]_w| \leq |\Sigma| + 2$. We denote such sets of constants by $\Gamma_S$, for all $S \in \mathcal{F}$. Then we embed those constants into the finite alphabet $\Sigma$ and extend the automaton $\mathcal{A}$ to handle the constraints on those constants.

The details of the algorithm are as follows. It consists of four main steps.

**1. The guessing of the set $\mathcal{F}$ and the constants $\Gamma_S$.**
  *a)* Guess a set $\mathcal{F} \subseteq 2^{\Sigma} - \{\emptyset\}$.
  *b)* For each $S \in \mathcal{F}$, guess an integer $m_S \leq |\Sigma| + 2$ according to the following rule.
    – If $V(\Sigma') \mapsto \Sigma' \in \mathcal{C}$, then $m_S = 0$, if $|S \cap \Sigma'| \geq 2$.
    – If $V(\Sigma_1) \subseteq V(\Sigma_2) \in \mathcal{C}$, then $m_S = 0$, if $S \cap \Sigma_1 \neq \emptyset$ and $S \cap \Sigma_2 = \emptyset$.
  *c)* For each $S \in \mathcal{F}$, fix a set $\Gamma_S = \{\alpha_1^S, \ldots, \alpha_{m_S}^S\}$ of constants such that $\Gamma_S$'s are disjoint, and $\Gamma_S \cap \mathbb{N} = \emptyset$. Let $\Gamma_{\mathcal{F}} = \bigcup_{S \in \mathcal{F}} \Gamma_S$.

**2. Embedding the constants of $\Gamma_S$'s into $\mathcal{A}$.**
Construct a finite state automaton $\mathcal{A}'$ (from the automaton $\mathcal{A}$) over the alphabet $\Sigma \cup \Sigma \times \Gamma_{\mathcal{F}}$ as follows. $\mathcal{A}'$ accepts the word $v = b_1 \cdots b_n$ over $\Sigma \cup \Sigma \times \Gamma_{\mathcal{F}}$ if and only if the following holds.
  – A symbol $(a, d) \in \Sigma \times \Gamma_{\mathcal{F}}$ can appear in $v$ if and only if $a \in S$ and $d \in \Gamma_S$.
  – Let $u = a_1 \cdots a_n$ be a word over $\Sigma$ such that

$$a_i = \begin{cases} b_i & \text{if } b_i \in \Sigma, \\ c & \text{if } b_i = (c, d) \in \Sigma \times \Gamma_{\mathcal{F}} \end{cases}$$

  Then, $u \in \mathcal{L}(\mathcal{A})$.
  – For $i = 1, \ldots, n-1$, if $b_i = (a_i, d_i) \in \Sigma \times \Gamma_{\mathcal{F}}$ and $b_{i+1} = (a_{i+1}, d_{i+1}) \in \Gamma_{\mathcal{F}}$, then $d_i \neq d_{i+1}$.
  – If $V(\Sigma') \mapsto \Sigma'$ is in $\mathcal{C}$, then for each $a \in \Sigma'$ and $\alpha \in \Gamma_S$, where $a \in S$, the symbol $(a, \alpha^S)$ appears exactly once in $v$.

Note that $\mathcal{A}'$ is defined from $\mathcal{A}$ with the parameters: $\mathcal{F}$ and $\{\Gamma_S \mid S \in \mathcal{F}\}$. The construction is straightforward and can be found in Appendix.

**3. Constructing the Presburger formula for $\mathcal{C}$.**
  *a)* Construct the formula $\varphi_{\mathcal{C}}$ from $\mathcal{C}$ according to Lemma 2.
  Let $\varphi_{\mathcal{C}}$ be in the form of $\exists z_{S_1} \cdots \exists z_{S_m}\ \psi_{\mathcal{C}}$.
  *b)* Denote by $\varphi_{\mathcal{C},\mathcal{F}}$ the formula:

$$\exists z_{S_1} \cdots \exists z_{S_m} \left( \psi_{\mathcal{C}} \wedge \bigwedge_{S_i \in \mathcal{F}} z_{S_i} = 0 \wedge \bigwedge_{S_i \notin \mathcal{F}} z_{S_i} \geq |\Sigma| + 3 \right)$$

**4. The decision step.** Test the emptiness of $\mathcal{L}(\mathcal{A}', \varphi_{\mathcal{C},\mathcal{F}})$.

The correctness of the algorithm follows from Claim 5 below.

**Claim 5** *There exists a locally different data word $w \in \mathcal{L}(\mathcal{A}, \mathcal{C})$ if and only if there exist a set $\mathcal{F} \subseteq 2^{\Sigma}$ and $\{\Gamma_S \mid S \in \mathcal{F}$ and $|\Gamma_S| \leq |\Sigma| + 2\}$ such that $\mathcal{L}(\mathcal{A}', \varphi_{\mathcal{C},\mathcal{F}}) \neq \emptyset$, where $\mathcal{A}'$ and $\varphi_{\mathcal{C},\mathcal{F}}$ are as defined in our algorithm and the constants $\Gamma_S$'s respect Step 1.b) above.*

*Proof.* We prove "only if" part first. Let $w \in \mathcal{L}(\mathcal{A}, \mathcal{C})$ be a locally different data word. The set $\mathcal{F}$ is defined as follows.

– $S \in \mathcal{F}$, if the cardinality $|[S]_w| \leq |\Sigma| + 2$.
– $S \notin \mathcal{F}$, if the cardinality $|[S]_w| \geq |\Sigma| + 3$.

Without loss of generality, we assume that $[S]_w = \Gamma_S$, for $S \in \mathcal{F}$. Let $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$. We construct the word $v = b_1 \cdots b_n$ as follows. For each $i = 1, \ldots, n$, $b_i = (a_i, d_i)$, if $d_i$ is in some $\Gamma_S$, otherwise $b_i = a_i$.

The rest of data values are in $[S]_w$, for some $S \notin \mathcal{F}$. So, $z_S = |[S]_w| \geq |\Sigma| + 3$ serves as witnesses for $S \notin \mathcal{F}$, and $z_S = 0$, for $S \in \mathcal{F}$. Thus, $v \in \mathcal{L}(\mathcal{A}', \varphi_{\mathcal{C},\mathcal{F}})$.

Now we prove the "only if" part. Suppose there exist some $\Gamma_S$'s and a word $v$ over the alphabet $\Sigma \cup (\Sigma \times \bigcup_{S \in \mathcal{S}} \Gamma_S)$ such that $v \in \mathcal{L}(\mathcal{A}', \varphi_{\mathcal{C},\mathcal{F}})$.

Let $v = b_1 \cdots b_n$. If $b_i = (a, \alpha) \in \Sigma \times \Gamma_S$, then we simply view $\alpha$ as the data value in that position. For the other positions, where $b_i = a \in \Sigma$, we assign the data values as before in Lemma 2.

Let $z_S = m_S$ be the witnesses that $v \in \mathcal{L}(\mathcal{A}', \varphi_{\mathcal{C},\mathcal{F}})$ holds. Let $K = \sum_S m_S$. Define the following function:

$$\xi : \{1, \ldots, K\} \to 2^{\Sigma} - \{\emptyset\},$$

where $|\xi^{-1}(S)| = m_S$.

For each $a \in \Sigma$, we assign the $a$-positions in $v$ with the data values from $\bigcup_{a \in S} \xi^{-1}(S)$. If necessary, we can apply Theorem 3 to obtain a locally different data word. The data values from $\Gamma_S$ does not prevent us from applying Theorem 3, since $\Gamma_{\mathcal{F}} \cap \{1, \ldots, K\} = \emptyset$. □

## 5.2 Satisfiability over general data words

Now we extend our idea above to prove Theorem 3. For that we need some auxiliary terms. Let $w = \binom{a_1}{d_1} \cdots \binom{a_n}{d_n}$ be a data word over $\Sigma$. A *zone* is a maximal interval $[i, j]$ with the same data values, i.e. $d_i = d_{i+1} = \cdots = d_j$ and $d_{i-1} \neq d_i$ (if $i > 1$) and $d_j \neq d_{j+1}$ (if $j < n$). Obviously each two consecutive zones have different data values. The zone $[i, j]$ is called an $S$-zone, if $S$ is the set of labels occuring in the zone.

The *zonal partition* of $w$ is a sequence $(k_1, \ldots, k_l)$, where $1 \leq k_1 < k_2 < \cdots < k_l \leq n$ such that $[1, k_1], [k_1 + 1, k_2], \ldots, [k_l + 1, n]$ are the zones in $w$. Let the zone $[1, k_1]$ be an $S_1$-zone, $[k_1 + 1, k_2]$ an $S_2$-zone, $[k_2 + 1..k_3]$ an $S_3$-zone, and so on. The *zonal word* of $w$ is a data word over $\Sigma \cup 2^{\Sigma}$ defined as follows.

$$\mathsf{Zonal}(w) = a_1 \cdots a_{k_1} \binom{S_1}{d_{k_1}} a_{k_1+1} \cdots a_{k_2} \binom{S_2}{d_{k_2}} \cdots a_{k_l+1} \cdots a_n \binom{S_l}{d_n}.$$

That is, the zonal word of a data word is a word in which each zone is succeeded by a label $S \in 2^{\Sigma}$, if the zone is an $S$-zone.

Moreover, it is sufficient to assume that only the positions labeled with symbols from $2^{\Sigma}$ carry data values, i.e., data values of their respective zones. Since two consecutive zones have different data values, two consecutive positions (in $\mathsf{Zonal}(w)$) labeled with symbols from $2^{\Sigma}$ also have different data values.

Furthermore, if $w$ is a data word over $\Sigma$, then for each $a \in \Sigma$,

$$V_w(a) = \bigcup_{a \in S} V_{\mathsf{Zonal}(w)}(S).$$

Proposition 2 below shows that disjunctive constraints for data words over the alphabet $\Sigma$ can be converted into disjunctive constraints for the zonal data words over the alphabet $\Sigma \cup 2^{\Sigma}$.

**Proposition 2.** *For every data word $w$ over $\Sigma$, the following holds.*

- *For $\Sigma' \subseteq \Sigma$, $w \models V(\Sigma') \mapsto \Sigma'$ if and only if*
  *K1.* $\mathsf{Zonal}(w) \models V(\mathcal{Q}) \mapsto \mathcal{Q}$, *where* $\mathcal{Q} = \{S \mid S \cap \Sigma' \neq \emptyset\}$;
  *K2. in* $\mathsf{Zonal}(w)$ *every zone contains at most one symbol from* $\Sigma'$.
  *(By a zone in* $\mathsf{Zonal}(w)$*, we mean a maximal interval in which every positions are labeled with symbols from* $\Sigma$*.)*
- *For $\Sigma_1, \Sigma_2 \subseteq \Sigma$, $w \models V(\Sigma_1) \subseteq V(\Sigma_2)$ if and only if* $\mathsf{Zonal}(w) \models V(\mathcal{Q}_1) \subseteq V(\mathcal{Q}_2)$, *where* $\mathcal{Q}_1 = \{S \mid S \cap \Sigma_1 \neq \emptyset\}$ *and* $\mathcal{Q}_2 = \{S \mid S \cap \Sigma_2 \neq \emptyset\}$.

Now, given a profile automaton $\mathcal{A}$ over the alphabet $\Sigma$, we can construct effectively an automaton $\mathcal{A}^{\text{zonal}}$ such that for all data word $w$,

$$\mathsf{Profile}(w) \in \mathcal{L}(\mathcal{A}) \text{ if and only if } \mathsf{Proj}(\mathsf{Zonal}(w)) \in \mathcal{L}(\mathcal{A}^{\text{zonal}}).$$

Such an automaton $\mathcal{A}^{\text{zonal}}$ is called a *zonal automaton* of $\mathcal{A}$. Moreover, if the dk $V(\Sigma') \mapsto \Sigma' \in \mathcal{C}$, we can impose the condition $K2$ in Proposition 2 inside the automaton $\mathcal{A}^{\text{zonal}}$.

This together with Proposition 2 imply that the instance $(\mathcal{A}, \mathcal{C})$ of SAT-profile can be reduced to an instance of the following problem.

| Problem: | SAT-locally-different-for-zonal-words |
|---|---|
| Input: | • a zonal automaton $\mathcal{A}^{\text{zonal}}$ |
| | • a collection $\mathcal{C}^{\text{zonal}}$ of disjunctive constraints over the alphabet $2^{\Sigma}$ |
| Question: | is there a zonal word $w$ such that |
| | • $\mathsf{Proj}(w) \in \mathcal{L}(\mathcal{A}^{\text{zonal}})$ and $w \models \mathcal{C}^{\text{zonal}}$ and |
| | • in which two consecutive positions labeled with symbols from $2^{\Sigma}$ have different data values? |

The proof in the previous subsection can then be easily adapted to SAT-locally-different-for-zonal-words. The details can be found in the Appendix.

## 6 Analysis of the complexity

As a conclusion, we provide the complexity of our algorithms.

| | |
|---|---|
| SAT-AUTOMATON | : NExpTime<br>NP(if the alphabet $\Sigma$ is fixed) |
| SAT-LOCALLY-DIFFERENT | : NExpTime<br>NP(if the alphabet $\Sigma$ is fixed) |
| SAT-PROFILE | : 2-NExpTime<br>NP(if the alphabet $\Sigma$ is fixed) |

In our algorithms, all three problems are reduced to the emptiness problem for Presburger automata which is decidable in NP(Theorem 2).

In SAT-AUTOMATON the exponential blow-up occurs when reducing the dk's and dic's in $\mathcal{C}$ to the existential Presburger formula $\varphi_{\mathcal{C}}$ (Lemma 2). This formula $\varphi_{\mathcal{C}}$ has exponentially many variables $z_S$, for every $S \subseteq \Sigma$. Of course, if the alphabet $\Sigma$ is fixed, then the reduction is polynomial, hence, the NP-membership for SAT-AUTOMATON. It is the same complexity for SAT-LOCALLY-DIFFERENT.

For SAT-PROFILE the additional exponential blow-up occurs when translating the dk's and dic's over the alphabet $\Sigma$ to the dk's and dic's over the alphabet $2^{\Sigma}$. Now combining this with Lemma 1, we obtain the 4-NExpTime upper bound for the satisfaction of $\exists\text{MSO}^2(\sim, +1)$.

## References

1. S. Abiteboul, R. Hull, V. Vianu. *Foundations of Databases*, Addison Wesley, 1995.
2. H. Björklund, M. Bojanczyk. Bounded depth data trees. In *ICALP'07*, pages 862–874.
3. L. Boasson. Some applications of CFL's over infinte alphabets. *Theoretical Computer Science, LNCS vol. 104*, 1981, pages 146–151.
4. M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM* 56(3): (2009).
5. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on words with data. In *LICS'06*, pages 7-16.
6. P. Bouyer, A. Petit, D. Thérien. An algebraic characterization of data and timed languages. *CONCUR'01*, pages 248–261.
7. S. Dal-Zilio, D. Lugiez, C. Meyssonnier. A logic you can count on. In *POPL 2004*, pages 135–146.
8. S. Demri, R. Lazic. LTL with the freeze quantifier and register automata. *ACM TOCL* 10(3): (2009).
9. W. Fan, L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM* 49(3): 368–406 (2002).
10. D. Figueira. Satisfiability of downward XPath with data equality tests. In *PODS'09*, pages 197–206.
11. S. Ginsburg and E.H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific J. Math.*, 16 (1966), 285–296.
12. E. Grädel, Ph. Kolaitis, M. Vardi. On the decision problem for two-variable first-order logic. *BSL*, 3(1):53–69 (1997).
13. M. Jurdzinski, R. Lazic. Alternation-free modal mu-calculus for data trees. In *LICS'07*, pages 131–140.
14. M. Kaminski, T. Tan. Tree automata over infinite alphabets. In *Pillars of Computer Science*, 2008, pages 386–423.

15. L. Libkin. Logics for unranked trees: an overview. In *ICALP'05*, pages 35-50.
16. F. Neven. Automata, logic, and XML. In *CSL 2002*, pages 2–26.
17. F. Neven, Th. Schwentick, V. Vianu. Finite state machines for strings over infinite alphabets. *ACM TOCL* 5(3): (2004), 403–435.
18. M. Otto. Two variable first-order logic over ordered domains. *J. Symb. Log.* 66(2): 685-702 (2001).
19. C. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.
20. R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.
21. Th. Schwentick. Automata for XML – a survey. *JCSS* 73 (2007), 289–315.
22. H. Seidl, Th. Schwentick, A. Muscholl. Numerical document queries. In *PODS 2003*, 155–166.
23. H. Seidl, Th. Schwentick, A. Muscholl, P. Habermehl. Counting in trees for free. In *ICALP 2004*, pages 1136–1149.
24. W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages, Vol. 3*, Springer, 1997, pages 389–455.
25. V. Vianu. A web Odyssey: from Codd to XML. In *PODS'01*, pages 1–15.