Two-Variable Logic on Words with Data*

Mikołaj Bojańczyk, Warsaw University
Anca Muscholl, LIAFA, Paris VII
Thomas Schwentick, Dortmund University
Luc Segoufin, INRIA, Paris XI
Claire David, LIAFA, Paris VII

***Abstract*— In a *data word* each position carries a label from a finite alphabet and a data value from some infinite domain. These models have been already considered in the realm of semistructured data, timed automata and extended temporal logics.**

It is shown that satisfiability for the two-variable first-order logic $\mathrm{FO}^2(\sim,<,+1)$ is decidable over finite and over infinite data words, where $\sim$ is a binary predicate testing the data value equality and $+1$,$<$ are the usual successor and order predicates. The complexity of the problem is at least as hard as Petri net reachability. Several extensions of the logic are considered, some remain decidable while some are undecidable.

I. Introduction

Finding decidable logics for models that handle data values is an important problem in several areas that need algorithmic procedures for property validation. Examples can be found in both program verification and database management. In this paper we reconsider a data model that was investigated both in verification (related to timed languages [1] and extended temporal logics [4]) and in XML reasoning [16]. As in these papers, data values are modeled by an infinite alphabet, consisting of a finite and an infinite part. The logic can address the finite part directly, while the infinite part can only be tested for equality. As a first step, this paper considers simple models: words, both finite and infinite.

Our main result is that the satisfiability problem for two variable first-order logic extended by equality tests for data values – $\mathrm{FO}^2(\sim,<,+1)$ for short – is decidable over word models. When more variables are permitted, or when a linear order on the data values is available, or when more than two equivalence relations are present, the logic becomes undecidable.

Following [1], a data word is a finite sequence of positions having each a label over some finite alphabet together with a data value over some infinite alphabet. The logic admits the equality test $x \sim y$, which is satisfied if both positions carry the same data value. Moreover, there are two navigational predicates on strings: the linear order $<$ and the successor relation $+1$. (With only two variables, the successor $+1$ cannot be defined in terms of the order $<$.) As usual we also have a unary predicate corresponding to each letter of the finite alphabet.

Perhaps surprisingly, we show that the satisfiability problem for $\mathrm{FO}^2(\sim,<,+1)$ is closely related to the well known problem of reachability for Petri nets. More precisely we show that languages formed by the projection onto the finite alphabet of word models definable by an $\mathrm{FO}^2(\sim,<,+1)$ sentence are recognized by multicounter automata (which are equivalent to Petri nets [7]). The converse is also true, modulo an erasing inverse morphism. Moreover, the correspondences are effective. We give a 2EXPTIME reduction of satisfiability for $\mathrm{FO}^2(\sim,<,+1)$ to emptiness for multicounter automata which is known to be decidable [11], [14]. For the opposite direction we provide a PTIME reduction from emptiness for multicounter automata to satisfiability for $\mathrm{FO}^2(\sim,<,+1)$. Since there is no known elementary upper bound for emptiness for multicounter automata (see e.g. [5]), finding the exact complexity of satisfiability for $\mathrm{FO}^2(\sim,<,+1)$ formulas is a hard question.

The decidability of $\mathrm{FO}^2(\sim,<,+1)$ immediately implies the decidability of $\mathrm{EMSO}^2(\sim,<,+1)$. Without data values, $\mathrm{EMSO}^2(+1)$ has the same expressive power as MSO. Here EMSO^2 stands for the closure of FO^2 under existential quantification of sets of nodes. In this sense the decidability of $\mathrm{EMSO}^2(\sim,<,+1)$ can be seen as an extension of the classical decidability result of monadic second-order logic over strings.

We also show that the satisfiability problem for $\mathrm{FO}^2(\sim,<,+1)$ remains decidable over data ω-words. In this case we no longer recognize the string projection of definable languages but we show that, again using emptiness of multicounter automata, it is decidable whether an $\mathrm{FO}^2(\sim,<,+1)$ formula is satisfied in a data ω-word whose string projection is ultimately periodic.

We then show that our decision procedure works even when the logic is extended by predicates ± 1 and $+2$, $+3$, ... Here ± 1 is a binary predicate, which relates two positions if they have the same

*Work supported by the French-German cooperation programme PROCOPE and the EU-TMR network GAMES.

data value, but all positions between them have a different data value. The $+k$ binary predicate, on the other hand, generalizes the successor predicate $+1$ to the kth successor.

The paper is organized as follows. The main result – a decision procedure for satisfiability of $\mathrm{FO}^2(\sim,<,+1)$– is presented in Section III. In the proof, a concept of data automaton is introduced. There are two steps: first we show in Section IV that each language definable in $\mathrm{FO}^2(\sim,<,+1)$ can be recognized by a data automaton; then we show in Section V how emptiness for data automata can be decided using multicounter automata. The reduction from reachability of multicounter automata to satisfiability of $\mathrm{FO}^2(\sim,<,+1)$ is shown in Section VI. In Section VII, we extend the main decidability result: first by adding new predicates, second by considering ω-words. In Section VIII, we show that the logic becomes undecidable when: a) three variables are allowed (even without the order $<$); or b) a linear order on data values is included. Finally, we conclude with a discussion of the results. Because of space limitation some proofs are missing and are available in the full version of this paper.

Related work. Automata on finite strings of data values (without labels) were introduced in [18], [9]. They used registers to store data values and data values could be compared wrt. equality. In [16] register automata and pebble automata over such words were studied. Several versions of these automata (one-way/two-way, deterministic/nondeterministic/alternating) were compared. Most of the results were negative however, i.e., most models are undecidable. Register automata have been also considered by Bouyer et al. [1] in the context of timed languages. However, the automata considered therein have a limited expressive power and cannot test data equality on arbitrary two positions (they are one-way automata). In particular the string projection of any language recognized by their automata is regular. As mentioned above, this is not the case for the logic considered in this paper.

In [4] 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 left-to-right, but can express properties that $\mathrm{FO}^2(\sim,<,+1)$ cannot.

Restricting first-order logic to its two-variable fragment is a classical idea when looking for decidability [8]. Over graphs or over any relational structures, first-order logic is undecidable, while its two-variable fragment is decidable [15]. This does not imply anything on the decidability of $\mathrm{FO}^2(\sim,<,+1)$, since the equivalence relation, the successor relation and the order $<$ cannot be axiomatized with only two variables. A recent paper generalized the result of [15] in the presence of one or two equivalence relations [10]. Again this does not apply to our context as we also have the order and the successor relation. However [10] also showed that FO^2 with *three* equivalence relations, without any other structure, is undecidable. This implies immediately that we cannot extend the decidability result to data words with more than two data parts.

In the context of XML reasoning we considered $\mathrm{FO}^2(\sim,+1)$ over unranked ordered data trees [2] and showed the decidability of the satisfiability question. In this context, the predicate $+1$ stands for two successor predicates, one for the vertical axis and one for the horizontal one. As data words are special cases of data trees, this implies the decidability of $\mathrm{FO}^2(\sim,+1)$ over words. The complexity for data trees is in 3NEXPTIME but it becomes 2NEXPTIME when restricted to data words. This should be contrasted with the complexity of satisfiability of $\mathrm{FO}^2(\sim,<,+1)$ which is not known to be elementary.

On strings over a finite alphabet (without data values), the $\mathrm{FO}^2(<,+1)$ fragment of first-order logic is very well understood. A characterization in terms of temporal logic says that it is equivalent to LTL restricted to the unary operators F, G, X and their past counterparts [6]. In terms of automata, $\mathrm{FO}^2(<)$ is equivalent to partially-ordered, two-way deterministic finite automata [17], while in terms of algebra the logic corresponds to semigroup variety DA [19]. The satisfiability question is NEXPTIME-complete in [6] (using an arbitrary number of unary predicates).

II. PRELIMINARIES

Let Σ be a finite alphabet of *labels* and D an infinite set of *data values*. A **data word** $w = w_1 \cdots w_n$ is a finite sequence over $\Sigma \times D$, i.e., each w_i is of the form (a_i, d_i) with $a_i \in \Sigma$ and $d_i \in D$. A **data language** is a set of data words, for some Σ. The idea is that the alphabet Σ is accessed directly, while data values can only be tested for equality. This amounts to considering words over Σ endowed with an equivalence relation on the set of positions. The string $\mathrm{str}(w) = a_1 \cdots a_n$ is called the **string projection** of A. The **marked string projection** $\mathrm{mstr}(w) = (a_1, b_1) \cdots (a_n, b_n) \in (\Sigma \times \{0,1\})^*$ of w adds a new coordinate, where $b_i = 1$

on all positions i with the same data value as $i-1$. For a language of data words L, we write $\text{str}(L)$ for $\{\text{str}(w) \mid w \in L\}$ and $\text{mstr}(L)$ for $\{\text{mstr}(w) \mid w \in L\}$.

A **class** is a maximal set of positions in a data word with the same data value. For a class with positions $i_1 < \cdots < i_k$ the **class string** is $a_{i_1} \cdots a_{i_k}$ and the **marked class string** is $(a_{i_1}, b_{i_1}) \cdots (a_{i_k}, b_{i_k})$, with the b_i as above.

Data words can be seen as models for first-order logic, where the carrier of the model is the set of positions in the word. Let $\text{FO}(\sim, <, +1)$ be first-order logic with the following atomic predicates: $x \sim y$, $x < y$, $x = y + 1$, and a predicate $a(x)$ for every $a \in \Sigma$. The interpretation of $a(x)$ is that the label in position x is a. The order $<$ and successor $+1$ are interpreted in the usual way. Two positions satisfy $x \sim y$ if they have the same data value. We write $L(\varphi)$ for the set of data words that satisfy the formula φ. A formula satisfied by some data word is **satisfiable**.

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

Example: We present here a formula φ such that $\text{str}(L(\varphi))$ is exactly the set of all words over $\{a, b\}$ that contain the same number of a's and b's.

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

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

  Similarly we define φ_b.
- The formula $\psi_{a,b}$ says each class with an a also contains a b:

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

  Similarly we define $\psi_{b,a}$.
- Hence, in a data word satisfying $\varphi = \varphi_a \wedge \varphi_b \wedge \psi_{a,b} \wedge \psi_{b,a}$ the numbers of a and b-labeled positions are equal.

This can be easily extended to describe data words with an equal number of a's, b's and c's, hence a language with a non-context-free string projection.

Example: For $a \in \Sigma$ the formula below is satisfied precisely by the first a which has an a in a different class on its left, i.e., the first a in the second a-class.

$$\begin{aligned} & a(x) \wedge \exists y \ (y < x \ \wedge \ a(y) \ \wedge \ x \nsim y) \wedge \\ & \quad \forall y \ (y < x \ \wedge \ a(y)) \rightarrow \\ & [x \nsim y \wedge \forall x \ \big((x < y \ \wedge \ a(x)\,) \rightarrow \ x \sim y\big)] \end{aligned}$$

Note how in this example the variable x is reused.

III. Decidability of $\text{FO}^2(\sim, <, +1)$

The main result of this paper is:

Theorem 1 *Satisfiability of* $\text{FO}^2(\sim, <, +1)$ *over data word models is decidable.*

The basic idea of the proof of Theorem 1 is to compute for each formula φ a multicounter automaton that recognizes $\text{str}(L(\varphi))$. As an intermediate step, we use a new type of finite automaton that works over data words, called a data automaton (these will be defined in Section IV). Theorem 1 follows immediately from the following three statements.

Proposition 2 Every language definable in $\text{FO}^2(\sim, <, +1)$ is recognized by an effectively obtained data automaton.

Proposition 3 From each data automaton we can compute a multicounter automaton recognizing the string projection of its recognized language.

Theorem 4 *[11], [14] Emptiness of multicounter automata is decidable.*

Proposition 2 is shown in Section IV, and Proposition 3 is shown in Section V. Regarding complexity, satisfiability of a $\text{FO}^2(\sim, <, +1)$ formula is reduced in 2EXPTIME to the emptiness of a multicounter automaton of doubly exponential size.

IV. Data automata

In this section we define *data automata*[1], a means to define data languages, and show that they can recognize all languages of data words definable in $\text{FO}^2(\sim, <, +1)$.

A **data automaton** $\mathcal{D} = (\mathcal{A}, \mathcal{B})$ consists of

- a nondeterministic letter-to-letter string transducer $\mathcal{A}$ (the **base automaton**) with input alphabet $\Sigma \times \{0, 1\}$, for some Σ and some output alphabet Γ (letter-to-letter means that each transition reads and writes exactly one symbol), and
- a nondeterministic string automaton $\mathcal{B}$ (the **class automaton**) with input alphabet Γ.

A data word $w = w_1 \cdots w_n \in (\Sigma \times D)^*$ is accepted by $\mathcal{D}$ if there is an *accepting* run of $\mathcal{A}$ on the marked string projection of w, yielding an output string $b_1 \cdots b_n$, such that, for *each class* $\{x_1, \ldots, x_k\} \subseteq \{1, \ldots, n\}$, $x_1 < \cdots < x_k$, the class automaton accepts $b_{x_1}, \ldots, b_{x_k}$.

[1]Our data automata differ from what is called data automaton in [1], which are essentially 1-way automata with one register.

As an example, consider the property: "w has at least two classes with an a". The data automaton for this property works as follows. The base automaton nondeterministically chooses two positions with an a and outputs 1 on each of them and 0 everywhere else. If there are no such two positions it rejects. The class automaton checks that each class contains at most one 1, thus verifying that the two positions were chosen from different classes. We call below *renaming* a letter-to-letter morphism, that is a morphism defined as $h : \Sigma \to \Sigma'$, where Σ, Σ' are alphabets.

Lemma 5 Languages recognized by data automata are closed under union, intersection, and renaming.

Proof
Closure under union and intersection is obtained using the usual product construction. Closure under renaming is obtained using the non-determinism of the base automaton. $\square$

The same cannot be said about negation. Indeed, if we were to have effective closure under negation, then all data languages defined by monadic second-order logic could be effectively translated into data automata, as we show that data automata recognize all of $\mathrm{EMSO}^2(\sim, <, +1)$. This would be a contradiction, since we show that emptiness of data automata is decidable, while satisfiability is undecidable for monadic second-order logic [16] (even for first-order logic, see Proposition 20).

The following lemma presents a family of data languages recognizable by data automata which will be used later in the proof.

Lemma 6 For any given regular language $L \subseteq (\Sigma \times \{0, 1\})^*$, there is a data automaton accepting all data strings w, for which each marked class string belongs to L.

Proof
The base automaton just copies its input $\Sigma \times \{0, 1\}$ and the class automaton checks membership in L. $\square$

One can also verify if *some* marked class string belongs to L: for each position i, the base automaton nondeterministically chooses to either output the input symbol a_i or a special symbol $\perp$. It accepts, if it output at least one non-$\perp$ symbol. The class automaton accepts the language $L \cup \perp^*$.

A. Reduction to data automata

The goal of this section is to prove Proposition 2, i.e. to transform a formula of $\mathrm{FO}^2(\sim, <, +1)$ into an equivalent data automaton of doubly exponential size. The transformation is done in two steps, first we rewrite the given formula into "intermediate normal form", and then we show that the normal form can be recognized by data automata. Each step gives an exponential blowup.

In the first step, we transform $\mathrm{FO}^2(\sim, <, +1)$ formulas into an intermediate normal form (we denote as *type* below any conjunction of unary predicates or their negations):

Definition 7 A formula is said to be in **intermediate normal form** if it is of the form $\exists R_1 \cdots R_m \ (\theta_1 \wedge \cdots \wedge \theta_n)$, where $\exists R_1 \cdots R_m$ quantifies over unary predicates (sets of positions) and each θ_i is of one of the following kinds:
(1) $\forall x \forall y \quad \big[\rho(x, y) \wedge \alpha(x) \wedge \beta(y) \wedge \delta(x, y)\big] \to \gamma(x, y)$
(2) $\forall x \exists y \quad \alpha(x) \to \big[\beta(y) \wedge \delta(x, y) \wedge \epsilon(x, y)\big]$
(3) $\forall x \forall y \quad \psi$
where

- α and β are types,
- $\rho(x, y) = (x \neq y + 1 \wedge y \neq x + 1 \wedge x \neq y)$,
- $\delta(x, y)$ is $x \sim y$ or $x \not\sim y$,
- $\gamma(x, y)$ is $x < y$, $x > y$, or *ff*, and
- $\epsilon(x, y)$ is $x + 1 < y$, $x + 1 = y$, $x = y$, $x = y + 1$ or $x > y + 1$,
- ψ is a quantifier free formula in DNF that doesn't use $\sim$.

Here, $x + 1 < y$ is an abbreviation for $x < y \wedge x + 1 \neq y$.

Formally speaking, a formula in intermediate normal form is a normal form of $\mathrm{EMSO}^2(\sim, <, +1)$, the extension of $\mathrm{FO}^2(\sim, <, +1)$ by existential monadic second-order predicates quantification in front of $\mathrm{FO}^2(\sim, <, +1)$ formulas. Note that as far as satisfiability is concerned $\mathrm{FO}^2(\sim, <, +1)$ and $\mathrm{EMSO}^2(\sim, <, +1)$ are equivalent.

Lemma 8 Every formula of $\mathrm{FO}^2(\sim, <, +1)$ can be effectively transformed into an equivalent formula in intermediate normal form of exponential size, with exponentially many unary predicates R_i.

Proof
The overall idea is classical: we reduce the quantifier depth to 2, then we add unary predicates that color certain distinguished positions, resp. classes containing distinguished positions. These additional colors are then used to simplify the formulas.

The formal proof proceeds in three steps.
Step 1: Scott normal form
The first step is classical for two variable logics

(see e.g. [8]). It says that each $FO^2(\sim,<,+1)$ formula φ is equivalent with respect to satisfiability to a (linear size) formula φ' in *Scott Normal Form*,

$$\forall x\forall y\ \chi \wedge \bigwedge_i \forall x\exists y\ \chi_i,$$

where χ and each χ_i are quantifier-free. The signature of φ' is an extension of the signature of φ by (linearly many) unary predicates. Furthermore, a data word satisfies φ if and only if it can be extended by additional predicates to a word satisfying φ'. In our setting, the additional unary predicates are simply relations R_i which are existentially quantified by the formula. The additional relations state which subformulas are satisfied at a given position.

Hence, φ is equivalent to a formula of the form

$$\exists R_1 \cdots \exists R_m \big(\forall x\forall y\ \chi \wedge \bigwedge_i \forall x\exists y\ \chi_i\big).$$

Step 2: Dealing with $\forall x\forall y\ \chi$.
In the second step we show that the formula $\forall x\forall y\ \chi$ can be replaced by a formula

$$\exists R_1 \cdots \exists R_m \bigwedge_i \theta_i \quad \wedge \quad \bigwedge_{i=1,2,3} \forall x\exists y\ \xi_i$$

where the ξ_i are again quantifier-free and each θ_i is of the form (1) (from Definition 7). Moreover, the number of θ_i is exponential.

To this end, we first rewrite $\forall x\forall y\ \chi$ into the following form:

$$\begin{array}{lcll}\forall x\forall y & (& y=x+1 & \rightarrow \psi_{y=x+1}(x,y)\\ & \wedge & x=y+1 & \rightarrow \psi_{x=y+1}(x,y)\\ & \wedge & x=y & \rightarrow \psi_{x=y}(x,y)\\ & \wedge & \rho(x,y) & \rightarrow \psi_\rho(x,y)\big)\end{array}$$

where the ψ formulas are quantifier-free and use only $\sim$, $<$ and the unary predicates. They have the same size as χ. Over the (linearly ordered) models considered in this paper this is logically equivalent to:

$$\begin{array}{l}\quad\forall x\forall y\ \big(\rho(x,y) \rightarrow \psi_\rho(x,y)\big)\\ \wedge\ \forall x\exists y\ \big(\neg R_{\text{last}}(x) \rightarrow (y=x+1\ \wedge \psi_{y=x+1})\big)\\ \wedge\ \forall x\exists y\ \big(\neg R_{\text{first}}(x) \rightarrow (x=y+1\ \wedge \psi_{x=y+1})\big)\\ \wedge\ \forall x\exists y\ \big((x=y\ \wedge \psi_{x=y})\big)\end{array}$$

Here, we assume that R_{first} and R_{last} are two extra predicates marking the first and last position of the word, respectively. They can easily be enforced by a formula of the form (3). The last three conjuncts give rise to ξ_1, ξ_2, ξ_3 and we are left with the first conjunct $\forall x\forall y\ \big(\rho(x,y) \rightarrow \psi_\rho(x,y)\big)$. We turn ψ_ρ into CNF (with an exponential blow-up) and rewrite it as

$$\bigwedge_{\alpha,\beta} (\alpha(x) \wedge \beta(y)) \rightarrow \psi(x,y)$$

where α, β are types occurring in ψ_ρ, and $\psi(x,y)$ is a quantifier-free formula using only $<$ and $\sim$. Finally, we rewrite the formula $\psi(x,y)$ into the form:

$$\psi(x,y) = (x \sim y \rightarrow \gamma_1(x,y)) \wedge (x \nsim y \rightarrow \gamma_2(x,y))$$

where $\gamma_1(x,y)$ and $\gamma_2(x,y)$ are $x<y$, $x>y$, *ff* or *tt*.

Step 3: Dealing with $\bigwedge_i \forall x\exists y\ \chi_i$.

In the last step, we show that each formula $\forall x\exists y\ \chi$ can be rewritten into an equivalent formula $\exists R'_1 \cdots \exists R'_n \bigwedge_i \theta_i$ with θ_i of type (2) or (3) in Definition 7. Moreover, the size of θ_i and the number n of additional predicates are both polynomial.

First, χ can be written as a disjunction (of exponential size)

$$\bigvee_j \big(\alpha_j(x) \rightarrow \beta_j(y) \wedge \delta_j(x,y) \wedge \epsilon_j(x,y)\big),$$

where the $\alpha_j, \beta_j, \delta_j, \epsilon_j$ are of corresponding forms as in (2). It only remains to eliminate the disjunction. To this end, we add for each conjunct above a new unary predicate $R_{\chi,j}$ with the intended meaning that $R_{\chi,j}$ holds at a position x if there is a y such that $\alpha_j(x) \wedge \beta_j(y) \wedge \delta_j(x,y) \wedge \epsilon_j(x,y)$ holds. Formally, we rewrite each $\forall x\exists y\ \chi$ as

$$\begin{array}{l}\exists R_{\chi,1}\ R_{\chi,2} \cdots (\forall x \quad \bigvee_i R_{\chi,i}(x))\ \wedge\\ \bigwedge_j \forall x\exists y\ \big(\alpha_j(x) \wedge R_{\chi,j}(x) \rightarrow\\ \qquad\qquad (\beta_j(y) \wedge \delta_j(x,y) \wedge \epsilon_j(x,y))\big)\ .\end{array}$$

By putting together the obtained formulas we get a formula in intermediate normal form. $\square$

We are now ready for the second step, where the intermediate normal form is transformed into a data automaton, thus completing the proof of Proposition 2. Since data automata are closed under renaming and intersection, it suffices to consider just the conjuncts θ in Definition 7. Note that the Cartesian product gives an exponential blowup.

Lemma 9 Every conjunct θ of a formula in intermediate normal form can be recognized by a data automaton with constantly many states.

Proof
The formula θ may be in one of the three forms (1), (2) or (3) from Definition 7. The case of formula

of the form (3) is no problem as it can easily be checked using the base automaton. We consider first the case of (1):

$$\forall x \forall y \ \big[(\rho(x,y) \ \wedge \ \alpha(x) \ \wedge \ \beta(y) \ \wedge \ \delta(x,y)) \ \rightarrow \gamma(x,y)\big]$$

The proof uses the data automaton for marking a fixed number of classes, say k for some constant k. We explain this technique first. The base automaton uses the output alphabet $\Gamma_k = \{\bot, (1,1), \ldots, (k,1), (1,0), \ldots, (k,0)\}$. It guesses, for each position i an output symbol $b_i \in \Gamma_k$. It makes sure that, for each j, at most once the symbol $(j,1)$ is chosen. The class automaton accepts then all strings of the form $\bot^*$ and $(l,1)(l,0)^*$, for some l. In this way, it is ensured that, for each class, always the output symbol $\bot$ is chosen or the first output symbol is $(l,1)$ and all others are $(l,0)$, for some l. As each $(j,1)$ is used at most once, it can not happen that two classes share the same $(j,0)$ (and $(j,1)$) symbols. Thus, the base automaton can assume, for each position, to which of the $\leq k$ classes it belongs.

It remains to perform a case analysis on the formulas δ, γ and ϵ.

In the case where $\delta(x,y)$ is $x \sim y$ the formula θ gives a regular condition that must be satisfied by each class. Thanks to Lemma 6, we can use that class automaton to verify θ.

If $\delta(x,y)$ is $x \not\sim y$ there are three subcases.

- $\gamma(x,y) = ff$ means that the data string may not have an α- and a β-position which are not adjacent (or identical) and in different classes. It is easy to see that the formula evaluates to false if both α and β appear in the string and there are at least 4 classes with an α or at least 4 classes with a β.
  Thus, it is sufficient that the base automaton selects (at most) 6 classes, using the technique explained above, and tests that (a) neither α nor β occur outside these 6 classes and (b) within the 6 classes θ holds.
- $\gamma(x,y) = x < y$. In this case only two classes are involved.
  Let Lst_α be the *position* of the rightmost α in the string and Llst_α be the position of the rightmost α that is in a different class than Lst_α. Using this notation, θ holds if and only if w (a) has no β up to position $\mathrm{Llst}_\alpha - 2$; and (b) the β-positions between $\mathrm{Llst}_\alpha - 1$ and $\mathrm{Lst}_\alpha - 2$ are in the same class as Lst_α.
  Thus, the base automaton simply guesses the two classes containing Lst_α and Llst_α and tests that (a) and (b) hold.
- The case where $\gamma(x,y)$ is $x > y$ is analogous.

It remains to consider formulas θ of type (2):

$$\forall x \exists y \quad \alpha(x) \rightarrow \ \big[\beta(y) \ \wedge \ \delta(x,y) \ \wedge \ \epsilon(x,y)\big]$$

As before, the difficult case is when $\delta(x,y)$ is $x \not\sim y$. If $\epsilon(x,y)$ is one of $x+1=y$, $x=y$, $x=y+1$ then θ can be verified by the base automaton. Otherwise $\epsilon(x,y)$ is $x+1<y$ or $x>y+1$. We describe the case of $x+1<y$, the other being analogous.

In this case, θ expresses that each α-position needs a β-position in a different class to its right (but not as its right neighbor). Since every α-position before $\mathrm{Llst}_\beta - 2$ is guaranteed to have such a β-witness in a different class, it suffices to require the following properties: a) from position $\mathrm{Lst}_\beta - 1$ on, the data word contains no α; and b) all α's between $\mathrm{Llst}_\beta - 1$ and $\mathrm{Lst}_\beta - 2$ are not in the same class as Lst_β. This involves checking 2 classes and can be handled analogously as the cases above.

As for the size, it is easy to check that the base and class automata have a number of states bounded by a constant. The number of transitions is bounded by the number of types (if we allow also type negations in the transitions, the number of transitions is also bounded by a constant). $\square$

We would like to note that the converse of Proposition 2 does not hold. There are two reasons for this. First, a data automaton can verify arbitrary regular properties of classes, which cannot be done with first-order logic. For instance, no $\mathrm{FO}^2(\sim,<,+1)$ formula captures the language: "each class is of even length". This problem can be solved by adding a prefix of monadic second-order existential quantification. However, even with such a prefix, it is difficult to write a formula that describes accepting runs of data automata. The problem is that describing runs of the class automata requires comparing successive positions in the same class, which need not be successive positions in the word. That is why we consider a new predicate ± 1, called the **class successor**, which is true for two successive positions in a same class of the data word. The following result easily follows from Proposition 2 and the obvious extension of its proof to include $\mathrm{EMSO}^2(\sim,<,+1,\pm 1)$:

Proposition 10 A language is recognized by a data automaton iff it is definable in $\mathrm{EMSO}^2(\sim,<,+1,\pm 1)$.

Proof
It is easy to extend the proof of Proposition 2 to the

logic $\mathrm{EMSO}^2(\sim,<,+1,\pm1)$. The other direction follows immediately from the classical simulation of automata in $\mathrm{EMSO}^2(+1)$. $\square$

By translating a formula into a data automaton and then back again into a formula, we can obtain a certain normal form for $\mathrm{EMSO}^2(\sim,<,+1,\pm1)$. In the normal form, the formulas verify the correctness of transitions in an accepting run. In particular, the order $<$ is not used.

Remark: Using the same idea of the proof of Proposition 2 one could show the following normal form for $\mathrm{EMSO}^2(\sim,<,+1)$. Each formula of $\mathrm{EMSO}^2(\sim,<,+1)$ is equivalent to one where the FO part is a Boolean combination of simple formulas of the form (where α and β are types):

(a) θ is does not use $\sim$ (i.e., a $\mathrm{FO}^2(<,+1)$ formula).
(b) Each class contains at most one occurrence of α.
(c) In each class, all occurrences of α occur strictly before all occurrences of β.
(d) In each class with at least one occurrence of α, there must be a β, too.
(e) If x is not in the same class at its successor then it is of type α.

V. Recognizing the Projection by Multicounter Automata

In this section, we show that the string projection of a language recognized by a data automaton can be recognized by a multicounter automaton.

We first introduce multicounter automata. An ϵ-free **multicounter automaton** is a finite automaton extended by a finite set $C = \{1,\ldots,n\}$ of counters. It can be described as a tuple $(Q,\Sigma,C,\delta,q_I,F)$. The set of states Q, finite alphabet Σ, initial state $q_I \in Q$ and final states $F \subseteq Q$ are as in a usual finite automaton. The transition relation δ is more involved – it is a finite subset of $Q \times \Sigma \times (\mathit{dec}^*(i)\,\mathit{inc}^*(i))_{i\in C} \times Q$.

The idea is that in each step, the automaton can change its state and modify the counters, by incrementing or decrementing them, according to the current state and the current letter on the input. In a step, the automaton can apply to each counter $i \in C$ a sequence of decrements, followed by a sequence of increments. Whenever it tries to decrement a counter of value zero the computation stops. Besides this, the transition of a multicounter automaton does not depend on the value of the counters. In particular, it cannot test whether a counter is exactly zero. Nevertheless, by decrementing a counter k times and incrementing it again afterward it can test whether the value of that counter is at least k.

A **configuration** of such an automaton is a tuple $c = (q,(c_i)_{i\in C}) \in Q \times \mathbb{N}^n$, where q is the current state and c_i is the value of the counter i. A transition

$$(q,a,(\mathit{dec}^{k_i}(i)\mathit{inc}^{l_i}(i))_{i\in C},q') \in \delta$$

can be applied if the current state is q, the current letter is a and for every counter $i \in C$, the value c_i is at least k_i. The successor configuration is $d = (q',(c(i)-k_i+l_i)_{i\in C})$. A **run** over a word w is a sequence of configurations that is consistent with the transition function δ. The acceptance condition is given by a subset R of the counters C and the final states. A run is **accepting** if it starts in the state q_I with all counters empty and ends in a configuration where all counters in R are empty and the state is final.

The key idea in the reduction from data automata to multicounter automata, is that acceptance can be expressed using the Shuffle(L) operation defined below.

Definition 11 A word $v \in (\Sigma \times \{0,1\})^*$ is a **marked shuffle** of n words $u_1,\ldots,u_n$ if its positions can be colored with n colors so that we have:

1) for every i, the positions colored with color i – read from left to right – give the word u_i,
2) a position of v is labeled by a symbol from $\Sigma \times \{1\}$ iff its predecessor position has the same color (neighborhood condition).

We write Shuffle(L) for the set of marked shuffles of words from a language $L \subseteq \Sigma^*$.

Proposition 12 Let $\mathcal{D} = (\mathcal{A},\mathcal{B})$ be a data automaton. The string projection str$(L(\mathcal{D}))$ is recognized by a multicounter automaton of size $O(|\mathcal{A}||\mathcal{B}|)$.

Proof
By definition of data automata, a word

$$v = (a_1,m_1)\cdots(a_n,m_n) \in (\Sigma \times \{0,1\})^*$$

belongs to the marked string projection mstr$(L(\mathcal{D}))$ if and only if it there is an accepting run of the base automaton on v with output $b_1\cdots b_n$ such that $(b_1,m_1)\cdots(b_n,m_n)$ is a marked shuffle of the language accepted by the class automaton. We will show in Proposition 13 that the set of such words $(b_1,m_1)\cdots(b_n,m_n)$ is recognized by a multicounter automaton $\mathcal{M}$ of same size as $\mathcal{B}$.

Thus, it is sufficient to compose $\mathcal{M}$ with a nondeterministic transducer which, on input $a_1\cdots a_n$, outputs a string $(a_1, m_1) \cdots (a_n, m_n)$, where each $m_i \in \{0,1\}$ is guessed independently, and which simulates $\mathcal{A}$. We obtain a multicounter automaton which on input $a_1 \cdots a_n$ constructs a string $(a_1, m_1) \cdots (a_n, m_n)$ which is read by $\mathcal{A}$ and whose output in turn is the input for $\mathcal{M}$. $\square$

Thus it remains to prove the following proposition which is an adaptation of Lemma (IV.6) in [7], where the result is shown for the usual shuffle operation (i.e., without marking explicitly the positions where the coloring changes).

Proposition 13 If $L \subseteq \Sigma^*$ is regular then Shuffle(L) is recognized by a multicounter automaton of size bounded by the size of an NFA recognizing L.

VI. Lower bounds

In this section we show that satisfiability for $\mathrm{FO}^2(\sim,<,+1)$ is at least as hard as non-emptiness of multicounter automata. The best lower bound known for the latter problem is ExpSpace [12] and no elementary upper bound is known.

Theorem 14 *Emptiness of multicounter automata can be reduced in* PTime *to the satisfiability problem of* $\mathrm{FO}^2(\sim,<,+1)$.

Proof sketch
Without loss of generality we assume that the multicounter has a one-letter input alphabet, no states (also known as vector addition system) and it accepts when *all* counters are empty. This can be done by adding one counter per states. Given a multicounter automaton $\mathcal{A}$, we construct a $\mathrm{FO}^2(\sim,<,+1)$ formula whose models are exactly (the encoding of) the accepting runs of the automaton.

Let $C = \{1, \ldots, n\}$ be the counters of the automaton, and let δ be its transition relation. We define Σ to be $D_1, I_1, \ldots, D_n, I_n$, and \$. An occurrence of D_i (I_i) codes a decrement (an increment) of counter i. The idea is to use data values to make sure that each decrement matches a previous increment. We encode a transition $t = (dec(i)^{k_i} inc(i)^{l_i})_{i \in C} \in \delta$ by a data word $\mathrm{enc}(t) = D_1^{k_1} I_1^{l_1} \cdots D_n^{k_n} I_n^{l_n}$ \$ where each occurrence of I_i have a new data value while each occurrence of D_i has the data value of the matching increment of the counter.

We can now check in $\mathrm{FO}^2(\sim,<,+1)$ that the string projection belongs to $\{\mathrm{enc}(t) \mid t \in \delta\}^*$. This is expressible in $\mathrm{FO}^2(<,+1)$ by a formula whose size is polynomial in δ. Then we enforce by a $\mathrm{FO}^2(\sim,<,+1)$ formula that each class string is either \$ or $I_k D_k$ for some k.

$\square$

VII. Decidable extensions

A. More successors

It is often useful to be able to express in the logic the existence of a given pattern in the string, e.g. the existence of two positions x and y such that x and y are in the same class and the substring between x and y is *abc*. This does not seem to be expressible in $\mathrm{FO}^2(\sim,<,+1)$. This kind of properties becomes immediately expressible in the presence of the predicates $+k$ for any $k \in \mathbb{N}$, where $x = y + k$ has the obvious meaning. We denote by $\mathrm{FO}^2(\sim,<,+\omega)$ the logic extending $\mathrm{FO}^2(\sim,<,+1)$ with all predicates $+k$. It turns out that this does not affect the decidability of the logic. The proof follows the lines of the of the proof for $\mathrm{FO}^2(\sim,<,+1)$ and will appear in the full version of the paper.

Theorem 15 *Satisfiability of* $\mathrm{FO}^2(\sim,<,+\omega)$ *is decidable.*

B. Infinite words

Another extension which is useful in the context of verification is the case of data ω-words, i.e., infinite length data strings. In this section we show the following result.

Theorem 16 *It is decidable whether a sentence of* $\mathrm{FO}^2(\sim,<,+1)$ *has a data ω-word model.*

The proof is along very similar lines as that of Theorem 1 but slightly departs as it does not reason about the string projection $\mathrm{str}(L(\varphi))$. Instead, the basic idea is to show that each satisfiable formula $\varphi \in \mathrm{FO}^2(\sim,<,+1)$ has a *simple* model of a given shape and that it is decidable whether a formula has a simple model. A data ω-word x is call **simple** if $\mathrm{mstr}(x)$ is of the form $u \cdot v^\omega$ for some finite words u and v over Σ.

As an intermediate step we use data ω-automata which are defined in analogy to data automata. We only mention the differences here. A **data ω-automaton** $(\mathcal{A}, \mathcal{B}_f, \mathcal{B}_i)$ consists of (1) a **base automaton** $\mathcal{A}$ which is a Büchi letter-to-letter transducer with output over some alphabet Γ, (2) a **finitary class automaton** $\mathcal{B}_f$ which is a finite string automaton over Γ and 3) an **infinitary class automaton** $\mathcal{B}_i$, which is another Büchi automaton over Γ. A data ω-word w is accepted if the base automaton has an accepting run over the marked string projection of w with output $b_1b_2\cdots$ such that for every finite class $i_1 < \cdots < i_k$, $b_{i_1}\cdots b_{i_k}$ is accepted by $\mathcal{B}_f$; similarly, for every infinite class $i_1 < i_2 < \cdots$, the ω-string $b_{i_1}b_{i_2}\cdots$ is accepted by $\mathcal{B}_i$.

Theorem 16 follows immediately from the following propositions.

Proposition 17 Every data ω-language definable in $\mathrm{FO}^2(\sim,<,+1)$ is recognized by some data ω-automaton.

Proof sketch
The proof follows exactly the lines of the proof of Proposition 2. Actually, the reduction to intermediate normal form is literally identical, as the proof of Lemma 8 does not assume finiteness. For the transformation of the intermediate normal form into a data ω-automaton, it suffices again to consider just the conjuncts θ since data ω-automata are closed under renaming, union and intersection.

Thus, it only remains to show that every conjunct θ of a formula in intermediate normal form can be recognized by a data ω-automaton.

This statement can be shown by a similar case analysis as in the proof of Lemma 9. □

Proposition 18 If a data ω-automaton accepts any ω-string it also accepts some simple ω-string.

Proof sketch
Let w be a (data) ω-string accepted by a data ω-automaton $(\mathcal{A}, \mathcal{B}_f, \mathcal{B}_i)$. Let $r = (r_{\mathcal{A}}, r_{\mathcal{B}_f}, r_{\mathcal{B}_i})$ be an accepting run for w that we view as functions from position to states. We call a position in w which is in a different class than its successor a **border position**. If w contains only finitely many border positions we can find u and v analogously as for classical (non-data) ω-automata. We only have to make sure that u contains all border positions. We thus assume in the following that there are infinitely many border positions in w.

A class c **overlaps** a position x if c contains positions $y < x$ and $z > x$. We say the class is q**-open** at x if $r_{\mathcal{B}_i}$ or $r_{\mathcal{B}_f}$ assigns the state q to the last position of c occurring before x. For a border position x of w, and for each state q, let m_q denote the number of q-open classes of w overlapping x.

The construction of u and v is based on the fact that there exist two border positions $x < x'$ of w such that:

(1) $r_{\mathcal{A}}(x) = r_{\mathcal{A}}(x')$ and $r_{\mathcal{A}}(y) \in F$, for some y, $x < y \le x'$.
(2) For each $q \in \mathcal{B}_f$ and each q-open class c of w overlapping x there is a position $x < y_c \le x'$ of c with $r_{\mathcal{A}}(y_c)$ accepting for $\mathcal{B}_f$.
(3) For each $q \in \mathcal{B}_i$ and each q-open class c of w overlapping x there is a position y_c, $x < y_c \le x'$ with $r_{\mathcal{B}}(y_c)$ accepting for $\mathcal{B}_i$.
(4) for each q, $m_q(x) = m_q(x') = 0$ or $0 < m_q(x) \le m_q(x')$.

Let w_u and w_v the data subwords of w from positions 1 to x and from $x+1$ to x', respectively. Let $u = \mathrm{mstr}((w_u))$ and $v = \mathrm{mstr}((w_v))$. The proof is completed by showing how to choose the data values for uv^ω in order to get the desired data ω-word. □

Proposition 19 It is decidable whether a data ω-automaton accepts some simple ω-string.

Proof sketch
We construct a multicounter automaton which tests, for a string uv whether it can be marked and extended by data values and a (partial) run such that conditions (1) - (4) above are satisfied with x the last position of u and x' the last position of v. If this automaton accepts some string uv we can conclude that uv^ω is the string projection of an ω-string accepted by E. Otherwise, it can be shown that E accepts no data string at all, in particular it accepts no simple string. □

VIII. Undecidable extensions

In this section we show that many immediate extensions yields undecidability. In the context of XML, nodes in the document may have several different attributes which are accessed via the query languages. Equality tests between node attributes could be simulated using several equivalence relations. For instance checking that the nodes x and y agree on attribute a could be written as $x \sim_a y$. However, very recently Kieroński and Otto [10] showed that two-variable logic with 3 equivalence relations and some unary relations is undecidable.

Extending the model by allowing more variables, even three, also gives undecidability.

Proposition 20 Satisfiability of $\mathrm{FO}^3(\sim,+1)$ is undecidable.

Note that this implies the undecidability of $\mathrm{FO}^3(\sim,<)$, since the relation $+1$ is definable from $<$ if three variables are allowed.

Proof sketch
We reduce Post's Correspondence Problem (PCP) to the satisfiability of $\mathrm{FO}^3(\sim,+1)$. An instance of

PCP consists of a finite number of pairs (u_i, v_i) of words from Σ^* and the question is whether there exists a non-empty, finite sequence of indexes $i_0, \ldots, i_n$ such that $u_{i_0} u_{i_1} \cdots u_{i_n} = v_{i_0} v_{i_1} \cdots v_{i_n}$.

Given an instance I of PCP, let $\Sigma' = \Sigma \cup \overline{\Sigma}$ be the alphabet consisting of two disjoint copies of Σ.

Consider a solution $i_0, \ldots, i_n$ such that $w = u_{i_0} u_{i_1} \cdots u_{i_n} = v_{i_0} v_{i_1} \cdots v_{i_n}$. We encode w by a data word $\hat{w} \in (\Sigma' \times D)^*$ satisfying the following:

- The string projection $\mathrm{str}(\hat{w})$ is $u_{i_0}\overline{v_{i_0}} \cdots u_{i_n}\overline{v_{i_n}}$. In particular, the sequence of letters from Σ is w and the sequence of letters from $\overline{\Sigma}$ is $\overline{w}$.
- Each data value appears exactly twice, once associated with a letter of Σ and once associated with the same letter in $\overline{\Sigma}$. Moreover, if a data value of $\hat{w}$ occurs at position i within w then its second occurrence must be at the same position i within $\overline{w}$.

It is possible to construct a formula φ of $\mathrm{FO}^3(\sim, +1)$ such that w is a solution of I iff $\hat{w}$ is a model of φ. □

Another possible extension is to suppose that there is a linear order on the data values and to include in the logic an extra binary predicate $\prec$ such that $x \prec y$ if the data value of x is smaller than the one of y. Unfortunately this yields undecidability even for FO^2.

Proposition 21 Satisfiability of $\mathrm{FO}^2(\sim, \prec, +1, <)$ is undecidable.

IX. Discussion

We have shown that satisfiability of $\mathrm{FO}^2(\sim, <, +1)$ over data words is decidable. Actually we have shown that the stronger logic $\mathrm{EMSO}^2(\sim, <, +1, \pm 1)$ is decidable over such models.

In the absence of data values, $\mathrm{FO}^2(+1, <)$ has several equivalent characterizations, for instance it corresponds to the fragment of LTL that uses only unary temporal predicates. Still in the absence of data values, $\mathrm{EMSO}^2(+1, <)$ has the same expressive power as MSO. In a sense the decidability of $\mathrm{EMSO}^2(\sim, <, +1)$ can be seen as an extension of classical decidability result of MSO over strings.

An interesting side result is the connection between $\mathrm{FO}^2(\sim, <, +1)$ and multicounter automata (and therefore Petri nets). Indeed, if we project out the data values, the languages defined by $\mathrm{FO}^2(\sim, <, +1)$ formulas are recognized by multicounter automata. The converse is also true modulo an erasing inverse morphism. It would be interesting to understand better the connection between the two formalisms. Because of the connection with Petri nets pinpointing the complexity of satisfiability is likely to be difficult.

Our reduction from the decidability of $\mathrm{FO}^2(\sim, <, +1)$ to emptiness multicounter automata, is 2NExpTime. We do not know whether this is optimal or not.

When only one of the two predicates $+1$ and $<$ is present we can show that the decision problem is elementary. It is NExpTime-complete for $\mathrm{FO}^2(\sim, <)$ and inside 2NExpTime for $\mathrm{FO}^2(\sim, +1)$. In [2] we studied in more details the logic $\mathrm{FO}^2(\sim, +1)$ and proved that it is decidable over unranked ordered trees. We inferred from this result many interesting consequences for XML reasoning. Whether $\mathrm{FO}^2(\sim, <, +1)$ is decidable over trees is still an open question which was shown in [2] to be at least as hard as checking emptiness of multicounter automata over trees which is stated as an open question in [3].

References

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

[2] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-Variable Logic on Data Trees and XML Reasoning. To appear in PODS'06.

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

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

[5] J. Esparza and M. Nielsen. Decidability Issues for Petri Nets - a survey. *Elektronische Informationsverarbeitung und Kybernetik*, 30(3): 143-160 (1994).

[6] 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).

[7] J. L. Gischer. Shuffle Languages, Petri Nets, and Context-Sensitive Grammars. *Commun. ACM*, 24(9):597-605 (1981).

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

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

[10] E. Kieroński and M. Otto. Small Substructures and Decidability Issues for First-Order Logic with Two Variables. Preprint. 2005.

[11] S.R. Kosaraju. Decidability of reachability in vector addition systems. In *STOC'84*, pp. 267-281. 1984.

[12] R.J. Lipton. The reachability problem requires exponential space. Dep. of Comp.Sci., Research report 62, Yale University, 1976.

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

[14] E. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. of Comp.*, 13:441-459 (1984).

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

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

[17] Th. Schwentick, D. Thérien, and H. Vollmer. Partially-Ordered Two-Way Automata: A New Characterization of DA. In *Developments in Language Theory (DLT'01)*, pp. 239-250, 2001.

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

[19] D. Thérien and Th. Wilke. Over Words, Two Variables Are as Powerful as One Quantifier Alternation. In *STOC'98*, pp. 234-240, 1998.