Automata based verification over linearly ordered data domains*

Luc Segoufin[1] and Szymon Toruńczyk[†] [2]

1 **INRIA and ENS Cachan, LSV**
2 **University of Warsaw**

Abstract

In this paper we work over linearly ordered data domains equipped with finitely many unary predicates and constants. We consider nondeterministic automata processing words and storing finitely many variables ranging over the domain. During a transition, these automata can compare the data values of the current configuration with those of the previous configuration using the linear order, the unary predicates and the constants.

We show that emptiness for such automata is decidable, both over finite and infinite words, under reasonable computability assumptions on the linear order.

Finally, we show how our automata model can be used for verifying properties of workflow specifications in the presence of an underlying database.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases Register automata, data values, linear order

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.81

1 Introduction

System verification often requires dealing with infinite state systems. There are many sources of infinity, one of them being the presence of variables ranging over an infinite set of data values and this is the focus of this paper.

There exist several decidable models of automata and logics that explicitly manipulate data values and that can be used for verification. In order to achieve decidability there is a necessary trade-off between the permitted operations on data and the allowed recursion. For instance, many models consider only equality tests between data values [3, 10, 14], or limit the recursion or the expressive power [4, 5, 8, 9], or only apply over specific data domains [11, 7, 8, 9, 6, 1].

In terms of possible operation on data values, equality tests permit already a wide range of recursion schemes and the corresponding decidability results can be used for modeling a variety of applications. However it has been advocated in [11] that comparisons based on a linear order over the data values could be useful in many scenarios, including data centric applications built on top of a database. They propose a model for specifying "artifact centric workflows" in the presence of a database and prove that temporal properties can be verified in PSpace, if the data domain is the set of rational numbers.

In this paper, we consider automata over words which are equipped with a finite set of variables, ranging over a linearly ordered structure. Transitions of the automaton are based

* This research was funded by the ERC research project FoX under grant agreement FP7-ICT-233599.
† Thanks for funding from ESF AutoMathA.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 81–92

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

on constraints on the variables using the vocabulary of the structure. We present a method for analyzing emptiness of such automata over finite and infinite words, independently of the linearly ordered structure. We derive from it several useful decidability results. Below we describe these contributions in more details.

The setting. We consider arbitrary linearly ordered structures. Apart for a linearly ordered set, the structure may be equipped with finitely many unary predicates and constants. Over the integers, typical unary predicates might denote the set of primes or the set of numbers divisible by a fixed constant.

Automata. We present a model of automata (either for finite or infinite words) over such linearly ordered structures. A configuration of the automaton is a tuple of data values of a fixed arity. A transition constrains the values of the next configuration relative to the values of the current one by a Boolean combination of predicates in the vocabulary of the structure. The initial and accepting configurations are also specified using similar constraints.

The potential. Our main contribution is a generic toolbox for the described model of automata, which is applicable to all linearly ordered structures. It is based on the notions of *potential* and *saturation*. Intuitively, saturation[1] transforms a linearly ordered structure by inserting finitely many new constants until all intervals between two consecutive constants contain infinitely many or no points of a certain property. Once the structure is saturated, it admits a potential. One can think of the potential as of a measure of how generic a configuration is; the main property is that an automaton may choose the next configuration to be sufficiently generic, provided the previous configuration was also generic. The rest of the paper can be seen as applications of these notions.

Main results. As a first application of our toolbox, we show that over any linearly ordered structure our automata can be simulated by finite state automata. This implies that emptiness for our automata model is decidable for finite and infinite runs, if the structure satisfies reasonable complexity assumptions, typically being able to test for satisfiability of a set of constraints expressed using the predicates of the structure. This yields PSpace decision procedures for many linearly ordered structures, in particular integers and rationals.

We also present a variant of LTL, where Boolean predicates are replaced with constraints using the predicates of the structure, and which works over words extended by tuples of data values. This logic can be translated into our automata model. Combining this with the emptiness test mentioned above, we obtain decidability results concerning this logic and PSpace complexity in most important cases. As our automata are closed under intersection, this allows to test LTL properties on the runs of a given automata.

Finally, our last result shows how our toolbox can be used for dealing with automata that moreover have the possibility of consulting a finite database in order to constrain their transitions. This method works for all linearly ordered structures, giving an alternative proof of the result of [11] over rationals but also solving the case of integers, which was posed as an open problem.

Related work. Our automata model is very close to the one used in [7] over integer data values. Čerāns solved the decidability of the emptiness problem using Dickson's lemma. The obtained decision algorithm is therefore non-primitive recursive. That proof does not apply to dense linear orders and it's not clear how it can be extended to incorporate the presence of an underlying database. These issues are solved in this paper.

Our model of automata generalizes the register automata studied in [6, 14] – indeed,

[1] Our notion of saturation differs from the usual logical one in two important ways: it considers only existential types and it is constructive.

a configuration can be seen as a valuation of the registers and additionally of the current datum of the input word. Our model is more expressive as we can compare data values using a linear order and unary predicates. Our setting also generalizes the model of [1] for verifying properties of programs working on arrays. Their model allows for linear tests but is specific to integers and finite runs.

Our notion of potential uses a partitioning of the space into cells. Similar techniques were developed for timed automata or over dense linearly ordered structures. See for instance [12]. Over dense linearly ordered structures, register automata generalize the notion of timed automata in a sense explained in [13]. Our notion of potential hence generalizes these ideas even for discrete linearly ordered structures.

The work of [11] considers specification and verification of workflows over finite databases whose underlying domain is the set of rationals. In fact, over the rationals, our model of automata can be viewed as a simplification of the elaborate formalism used in [11], necessary for the specific application targeted therein. As is shown in that paper, LTL formulas can be checked in PSPACE assuming a fixed database schema, EXPSPACE otherwise. We obtain similar results but our proof also applies for other linearly ordered structures, in particular over the integers, settling an open problem raised in [11].

There exist many extensions of LTL with several kinds of constraints over various data domains. Decidable fragments can compare data values using their relative order or their value modulo some constant. The complexity of satisfiability is shown to be PSPACE-complete, see [8] for a survey. This also follows from our results.

Due to space limitations many proofs are omitted or only sketched. They will appear in the journal version of this paper. We thank the anonymous referees for the useful comments.

2 Preliminaries

Linearly ordered structures. By a *linearly ordered structure* we refer to a structure of the form $\mathcal{D} = \langle D, <, P_1, P_2, \ldots, P_l, c_1, c_2, \ldots, c_m \rangle$, where D is the domain of the structure, $<$ is a linear order on D, $P_1, \ldots, P_l$ are unary predicates denoting subsets of D and $c_1, \ldots, c_m$ are constants. Typical examples are $\langle \mathbb{Z}, <, 0, 1 \rangle$, $\langle \mathbb{Q}, <, 0 \rangle$, $\langle \{a, b\}^*, <_{lex}, P_a, P_b, \epsilon \rangle$ where $<_{lex}$ is the lexicographical order and P_a, P_b are unary predicates indicating the last letter of a word. But we also consider more elaborate linear orders such as $\langle \mathbb{Q}, <, P_{even}, P_{odd}, P_{prime} \rangle$, where the three predicates correspond to even integers, odd integers and prime integers.

Formulas and cells. We now assume a fixed linearly ordered structure $\mathcal{D}$ and dimension k. We consider k-ary relations over the domain of $\mathcal{D}$ definable by a Boolean combination of atoms built from the symbols of $\mathcal{D}$ using k variables. Each set of this form will be called a *region in* $\mathcal{D}^k$. A region that corresponds to a maximal consistent conjunction of atoms or negated atoms is called a *cell in* $\mathcal{D}^k$. Note that there are finitely many cells and any region is a disjoint union of cells. For instance, over $\mathcal{D} = \langle \mathbb{Z}, <, 0 \rangle$, $x < y$ defines a region of $\mathcal{D}^2$ which is the disjoint union of 5 cells: $x < y < 0$, $x < y = 0$, $x < 0 < y$, $x = 0 < y$ and $0 < x < y$. A tuple $\bar{x} \in D^k$ will be also called a *point in* $\mathcal{D}^k$. For each such $\bar{x}$ we denote by $\langle \bar{x} \rangle$ the unique cell in $\mathcal{D}^k$ containing $\bar{x}$.

We fix a finite alphabet A. A $\mathcal{D}$*-automaton* $\mathcal{A}$ *of dimension* k is a tuple $((\delta_a)_{a \in A}, \tau_I, \tau_F)$ described as follows. For each letter $a \in A$, δ_a is a region in $\mathcal{D}^k \times \mathcal{D}^k = \mathcal{D}^{2k}$, representing the allowed transitions of $\mathcal{A}$ when reading the letter a; τ_I and τ_F are regions in $\mathcal{D}^k$ denoting the initial and accepting configurations of $\mathcal{A}$.

The configurations of $\mathcal{A}$ are points in $\mathcal{D}^k$. Let $w = a_1 a_2 \ldots a_n$ be a finite word. A *run* ρ of $\mathcal{A}$ on w is a sequence of configurations $\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_n \in D^k$ such that $\bar{x}_0 \in \tau_I$ and

for each $i \in \{1, \ldots, n\}$, the pair $(\bar{x}_{i-1}, \bar{x}_i)$ lies in the region δ_{a_i}. The run ρ is *accepting* if the sequence terminates in a configuration $\bar{x}_n \in \tau_F$. The *language* recognized by the $\mathcal{D}$-automaton $\mathcal{A}$ is the set of words for which there is an accepting run. In Section 4 we will consider automata over infinite words, but right now we focus on finite words. As usual, we are mostly interested in determining emptiness of such automata.

▶ **Example 1.** We define a $\mathcal{D}$-automaton $\mathcal{A}$ of dimension 2, where $\mathcal{D} = \langle \mathbb{Q}, <, 0 \rangle$ or $\mathcal{D} = \langle \mathbb{N}, <, 0 \rangle$ and $A = \{a, b\}$. The regions τ_I, τ_F are both described by $x_1 \geq 0 \wedge x_2 \geq 0$. The region δ_b is the set of points (x_1, x_2, y_1, y_2) such that $y_2 = x_2 \wedge x_1 < y_1 < x_2$ and δ_a is specified by $y_2 = x_2 \wedge y_1 = 0$. The language recognized by $\mathcal{A}$ is $(b^*a)^*$.

▶ Remark. It is often convenient to equip $\mathcal{D}$-automata with states. This does not change the expressive power as states can be simulated using extra dimensions.

▶ Remark. Since one can construct Cartesian products of $\mathcal{D}$-automata, the languages they recognize are closed under union and intersection.

▶ Remark. We could also use existential formulas for defining the transitions of the automaton. This would not affect decidability nor expressiveness. However, if we allowed a logic which can define the successor relation (such as first-order logic, when working over the naturals), we would easily encode a Minsky machine into the model, resulting in an undecidable emptiness problem.

Results. In the next section we develop a framework for manipulating $\mathcal{D}$-automata based on the notions of *potential* and *saturation*. We illustrate the use of these notions by proving that for any linearly ordered structure $\mathcal{D}$ and $\mathcal{D}$-automaton $\mathcal{A}$, the language recognized by $\mathcal{A}$ is regular by exhibiting an equivalent finite state automaton, yielding:

▶ **Theorem 2.** *For any linearly ordered structure $\mathcal{D}$, $\mathcal{D}$-automata recognize precisely the class of regular languages.*

We will see in Section 3.4 that our proof is actually constructive assuming that a reasonable amount of computation can be performed on $\mathcal{D}$. Our construction brings a PSPACE complexity for the emptiness problem for all linearly ordered structures used in the literature.

An analogue of Theorem 2 for infinite words no longer holds. However, with further computational assumptions, we will extend the decidability of emptiness to the infinite setting in Section 4. In Section 5 we show how LTL formulas using constraints expressible over $\mathcal{D}$ can be translated into $\mathcal{D}$-automata. Finally in Section 6 we show how our framework can also be used to solve the case where an underlying finite database is present.

3 Finite words

3.1 Cell automata

In this section we fix a linearly ordered structure $\mathcal{D}$ and a $\mathcal{D}$-automaton $\mathcal{A}$ of dimension k, described by the tuple $((\delta_a)_{a \in A}, \tau_I, \tau_F)$. We construct a finite nondeterministic automaton $\mathcal{A}'$, called the *cell automaton*, targeted at simulating the runs of $\mathcal{A}$.

The states of $\mathcal{A}'$ are the cells of $\mathcal{D}^k$. The automaton $\mathcal{A}'$ has a transition from the state τ to the state τ' labeled by the input letter a if and only if there exist $\bar{x} \in \tau$ and $\bar{y} \in \tau'$ such that $(\bar{x}, \bar{y}) \in \delta_a$. The initial states of $\mathcal{A}'$ are those cells τ for which $\tau \subseteq \tau_I$. The accepting states are those cells τ for which $\tau \subseteq \tau_F$. The following proposition is rather immediate:

▶ **Proposition 3.** *The language recognized by $\mathcal{A}$ is included in the language recognized by $\mathcal{A}'$.*

The converse inclusion does not always hold, as shown in the following example.

▶ **Example 4.** Let $\mathcal{D} = \langle \mathbb{Z}, <, 0, 3\rangle$ and let $\mathcal{A}$ be the $\mathcal{D}$-automaton of dimension 1 described as follows. The region δ_a is the subset of $\mathcal{D}^2$ given by the condition $0 < x < y < 3$; the initial and final regions are defined by $0 < x < 3$. $\mathcal{A}$ accepts $\{\epsilon, a\}$. However, the cell automaton $\mathcal{A}'$ corresponding to $\mathcal{A}$ has a self-loop in the state corresponding to the cell $0 < x < 3$ and therefore recognizes the language a^*.

3.2 Potential

We can prove the correctness of the cell construction if $\mathcal{D}^k$ is equipped with a sort of inductive measure called the *potential* which tells, roughly, given a point $\bar{x} \in \mathcal{D}^k$, how long runs of the cell automaton can be lifted to runs of the original automaton, starting from the point $\bar{x}$. Formally, a function $E\colon \mathcal{D}^k \to \mathbb{N} \cup \{\infty\}$ is called a *potential* for $\mathcal{D}^k$ if it satisfies the following conditions.

1. *Cells have unbounded potential*: For any cell $\theta \subseteq \mathcal{D}^k$ and any number $s \geq 0$ there exists a point $\bar{x} \in \theta$ such that $E(\bar{x}) \geq s$.
2. *Transitions decrease the potential by at most 1*: Let $\tau, \tau' \subseteq \mathcal{D}^k$, $\delta \subseteq \mathcal{D}^{2k}$ be cells such that there exists $(\bar{x}_0, \bar{y}_0) \in \delta$ with $\bar{x}_0 \in \tau$ and $\bar{y}_0 \in \tau'$. Then, if $\bar{x} \in \tau$ is such that $E(\bar{x}) \geq s + 1$ for some $s \geq 0$, there exists a point $\bar{y} \in \tau'$ such that $(\bar{x}, \bar{y}) \in \delta$ and $E(\bar{y}) \geq s$.

▶ **Example 5.** If $\mathcal{D} = \langle \mathbb{Q}, <, 0, 3\rangle$, the mapping constantly equal to ∞ is a potential for $\mathcal{D}^n$, for any $n \in \mathbb{N}$. This follows easily from the fact that $\mathbb{Q}$ is a dense linear order.

On the other hand, if $\mathcal{D} = \langle \mathbb{Z}, <, 0, 3\rangle$ as in Example 4, then $\mathcal{D}^1$ cannot be equipped with a potential. Indeed, the cell $\tau :\ 0 < x < 3$ contains only two points, $\bar{x}_0 = 1$ and $\bar{y}_0 = 2$. By using the second axiom, it is not difficult to prove that both points in τ must have a potential smaller than 2, contradicting the first axiom of the potential.

However, if $\mathcal{D} = \langle \mathbb{Z}, <, 0, 1, 2, 3\rangle$, there is a potential for $\mathcal{D}^1$ which assigns to a point x the value ∞ if $x \in \{0, 1, 2, 3\}$ and the distance from x to the "critical set" $\{0, 1, 2, 3\}$ otherwise. Intuitively, a cell in $\mathcal{D}^2$, seen as a transition, either sets the value of the variable to some constant, or allows the variable to attain a value arbitrarily far from the critical set, or requires the variable to get closer by at least one to the critical set. In any case, taking a point $x \in \mathbb{Z}$ with potential at least s, we can follow any transition and guarantee to end up in a point with potential at least $s - 1$.

Finally, consider $\mathcal{D}^2$, where $\mathcal{D} = \langle \mathbb{Z}, <, 0, 1, 2, 3\rangle$. Then $\mathcal{D}^2$ can also be equipped with a potential which, roughly, to a given point (x, y) assigns a value which depends on how far the variables x, y are from the constants $0, 1, 2, 3$ and from each other. Such a potential is constructed in the next section.

In the presence of a potential, runs of the cell automaton induce corresponding runs of the original $\mathcal{D}$-automaton. This is phrased in the following proposition, whose proof is obtained from the definition by an easy induction.

▶ **Proposition 6.** *Let $\mathcal{D}^k$ be equipped with a potential E, let $\mathcal{A}$ be a $\mathcal{D}$-automaton of dimension k and let $\mathcal{A}'$ be the cell automaton corresponding to $\mathcal{A}$. Let $\tau_0 \to \tau_1 \to \cdots \to \tau_n$ be a run of the cell automaton over a word $a_1 a_2 \dots a_n$. Then, for all numbers $s \geq 0$ and for all points $\bar{x}_0 \in \tau_0$ such that $E(\bar{x}_0) \geq s + n$ there exists a sequence of points $\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n \in \mathcal{D}^k$ such that $\bar{x}_i \in \tau_i$, $E(\bar{x}_i) \geq s + i$ and $(x_{i-1}, x_i) \in \delta_{a_i}$ for all $i = 1, \dots n$.*

In the above proposition, accepting runs of $\mathcal{A}'$ clearly induce accepting runs of $\mathcal{A}$. Therefore, we immediately obtain the following.

▶ **Theorem 7.** *Assume that there is a potential E for $\mathcal{D}^k$. Then, for any $\mathcal{D}$-automaton of dimension k, the corresponding cell automaton $\mathcal{A}'$ recognizes the same language as $\mathcal{A}$.*

3.3 Constructing the potential and saturation

When $\mathcal{D}$ cannot be equipped with a potential, we show how to extend $\mathcal{D}$ to a structure which is *saturated*. For such structures, we are always able to define a potential. Altogether, this will prove Theorem 2. In Example 4 the saturation process will add the constants $1, 2$ to the structure, allowing the new cell automaton to count up to 3.

Let $\mathcal{D}$ be a linear order. A *virtual element* of $\mathcal{D}$ is a subset S of D which is downward closed (i.e. $x \in S \wedge y \leq x \implies y \in S$), but is not of the form $\{y \mid y < x\}$ or $\{y \mid y \leq x\}$ for some $x \in \mathcal{D}$. For an element $x \in D$ and a virtual element S, we will write $x < S$ if $x \in S$, and $x > S$ otherwise. Also, for two virtual elements S, S' we can write $S < S'$ if $S \subsetneq S'$. We denote by $\bar{D}$ the set of elements and virtual elements of D, linearly ordered by $<$ (known as the Dedekind completion, modulo the elements $\emptyset, D$ which are normally not considered in $\bar{D}$). Note that $\bar{D}$ has a smallest and largest element, denoted $c_{-\infty}$ and c_{∞}, respectively.

In a linearly ordered structure $\mathcal{D}$ we distinguish between two kinds of unary predicates. We denote those that correspond to virtual elements, as they play a crucial role in our proofs, as *virtual constant* and we treat them as constants. In particular we will use symbols c, d for both virtual and real constants. As an example, consider the linearly ordered structure $\langle \mathbb{Q}, <, e \rangle$ where e is a virtual constant corresponding to the set $\{x \in \mathbb{Q} \mid x < 2.718\ldots\}$.

For a point $x \in D$, we define its *type*, denoted $t(x)$ as the set of unary predicates (including virtual constants) it satisfies and constants which it is equal to.

To simplify the following definitions, we will assume that $c_{-\infty}$ and c_{∞} are (possibly virtual) constants of $\mathcal{D}$. Let $w = t_1, \ldots, t_n$ be a sequence of types. We say that a sequence of points $x_1 < \ldots < x_n$ *realizes* w if $t(x_i) = t_i$ for all i. For an interval I, we say that w is *realizable in* I if some sequence $x_1 < \ldots < x_n$ of elements of I realizes w.

We construct a function, called *quasi-potential*, $qE \colon \mathcal{D}^* \to \mathbb{N} \cup \{\infty\}$, defined on all tuples of elements of $\mathcal{D}$. It will give raise to a potential defined on $\mathcal{D}^k$, for all k. Intuitively, $\bar{x}$ has high potential if long sequences of types can be realized in between any two coordinates of $\bar{x}$ or between a coordinate of $\bar{x}$ and a constant. The precise definition is given below.

Let $\bar{x}$ be a point in $\mathcal{D}^k$. Let $\{u_1, u_2, \ldots, u_s\}$ be the union of the set of coordinates of $\bar{x}$ and of the set of constants and virtual constants of $\mathcal{D}$. We assume that $u_1 < u_2 < \ldots < u_s$. For $x \in \bar{D}$ let $\underline{c}(x)$ be the largest (virtual) constant c such that $c \leq x$ and $\overline{c}(x)$ be the smallest (virtual) constant c such that $x \leq c$. For $1 \leq i < s$, let T_i be the set of types occurring in $]\underline{c}(u_i), \overline{c}(u_{i+1})[$. For each $0 \leq i < s$, the *capacity* of the interval $]u_i, u_{i+1}[$ is the length of the shortest sequence of elements of T_i which is not realizable in $]u_i, u_{i+1}[$ or as ∞ if all such sequences are realizable. Finally, we define

$$qE(\bar{x}) = \min\{\text{capacity of }]u_i, u_{i+1}[\mid 1 \leq i < s\}.$$

In the case where $k = 0$, $\mathcal{D}^0$ contains only one element – the empty tuple ε. We say that $\mathcal{D}$ is *saturated* if $qE(\varepsilon) = \infty$. In other words, for any two consecutive (virtual) constants c, d, any sequence of types which occur between c and d is realizable between c and d. Any linearly ordered structure can be transformed into a saturated one:

▶ **Theorem 8.** *Let $\mathcal{D}$ be a linearly ordered structure. It is possible to expand $\mathcal{D}$ with a finite set of constants and virtual constants to obtain a saturated structure $\hat{\mathcal{D}}$.*

Proof. We say that a linear order $\mathcal{D}$ is *complete* if any subset of D has its supremum, i.e. a least upper bound (again, this deviates slightly from the standard notion of completeness by the least and smallest elements). Examples of complete linear orders are $\mathbb{N} \cup \{\infty\}, \mathbb{R} \cup \{-\infty, +\infty\}$ but not $\mathbb{Q} \cup \{-\infty, +\infty\}$ nor $\mathbb{R}$. The following lemma solves the case of complete linear orders.

▶ **Lemma 9.** *Let $\mathcal{D}$ be a complete linear order. Let $a < b$ be two points in D and let $t\colon]a, b[\to T$ be a function with $|T| = n < \infty$. Then there exist $a = d_0 < d_1 < \ldots < d_m = b$ such that for any interval $I =]d_i, d_{i+1}[$ and $w \in (t(I))^*$, the sequence w is realizable in I.*

Proof. We prove the claim by induction on n. If $n = |T| = 0$ there is nothing to prove. Let us assume that we have proved the proposition for all $|T| < n$. Let s denote the length of the shortest sequence $w \in T^*$ which is not realizable in the interval $]a, b[$. We do a nested induction on s. If $s = 1$, there is a $u \in T$ which does not occur in $]a, b[$, so we may reduce the set T to $T \setminus \{u\}$ with less than n elements. Let us assume that $s \geq 2$.

Let $w = t_1, \ldots, t_s \in T^*$ be the sequence of length s which is not realizable in the interval $]a, b[$. We will define an element c such that for both open intervals $]a, c[$ and $]c, b[$, there is a non-realizable sequence of length smaller than s.

If $s > 2$ then the sequence t_1, t_s is realizable in $]a, b[$, so let $x < y$ realize it. Let $c = x$. Then the sequence $t_1, t_2, \ldots, t_{s-1}$ is not realizable in the interval $]a, c[$ and the sequence $t_2, t_3, \ldots, t_s$ is not realizable in the interval $]c, b[$.

If $s = 2$, the sequence t_1, t_2 is not realizable in $]a, b[$. Let us consider the supremum c of all possible x with $t(x) = t_2$. Then, t_2 is not realizable in the interval $]c, b[$. Moreover, t_1 is not realizable in the interval $]a, c[$ — otherwise, t_1, t_2 would be realizable in $]a, b[$.

We apply the inductive assumption to $]a, c[$ and $]c, b[$ obtaining sequences $a = d_0 < d_1 < \ldots < d_m = c$ and $c = d_m < d_{m+1} \ldots < d_{m'} = b$ such that for any interval $I =]d_i, d_{i+1}[$ and $w \in (t(I))^*$, the sequence w is realizable in I. This ends the inductive proof of the lemma. ◀

Let $\mathcal{D} = \langle D, <, P_1, P_2, \ldots, P_l, c_1, c_2, \ldots, c_m \rangle$ be an arbitrary linearly ordered structure. It is well known that $\langle \bar{D}, < \rangle$ is a complete linear order. Let us consider the structure

$$\bar{\mathcal{D}} = \langle \bar{D}, <, P_0, P_1, P_2, \ldots, P_l, c_1, c_2, \ldots, c_m \rangle,$$

where P_0 is the unary predicate corresponding to the set $D \subseteq \bar{D}$. Let $t\colon \bar{D} \to T$ be the function which assigns to an element of $\bar{\mathcal{D}}$ its type, i.e. the set of unary predicates it satisfies and constants equal to it. We apply Lemma 9 to $\bar{\mathcal{D}}$ and the points $c_{-\infty} < c_\infty$. We obtain a sequence of elements $d_0 < d_1 < \ldots < d_{m'}$ such that for any interval $I =]d_i, d_{i+1}[$ and $w \in (t(I))^*$, the sequence w is realizable in I. We define $\hat{\mathcal{D}}$ as the extension of $\mathcal{D}$ by the (possibly virtual) elements $d_0, \ldots, d_{m'}$ as constants. It is easy to verify that $\hat{\mathcal{D}}$ is saturated. ◀

▶ **Example 10.** We apply the procedure of Theorem 8 to $\mathcal{D} = \langle \mathbb{Z}, <, 0, 3 \rangle$ of Example 4. The only relevant interval is $]0, 3[$. There is only one type t in $]0, 3[$ and ttt is not realized. Since tt is realized by $1 < 2$, we are in the first case and the constant 1 is added to $\mathcal{D}$. In the next step, the relevant interval is $]1, 3[$ and tt is not realized in it. Adding the supremum of all elements of type t in $]1, 3[$, i.e. the constant 2, yields a saturated structure.

To see why we might need to also add a virtual constant, consider the linearly ordered structure $\mathcal{D} = \langle \mathbb{Q}, <, 0, 3, P, Q \rangle$ where $P = \{(1-1/n)^n | n \in \mathbb{N}\}$ and $Q = \{(1+1/n)^{-n} | n \in \mathbb{N}\}$. Consider the interval $]0, 3[$. As $P \cap Q = \emptyset$ there are three types occurring in this interval: $t_1 = \{P\}$, $t_2 = \{Q\}$, $t_3 = \emptyset$. Since $t_1 t_2$ is not realized in $]0, 3[$, the procedure of Theorem 8 introduces the supremum of all elements of type t_2 in this interval, i.e. the virtual constant e. The reader can verify that the resulting structure $\hat{\mathcal{D}} = \langle \mathbb{Q}, <, 0, 3, e, P, Q \rangle$ is saturated.

▶ **Theorem 11.** *If $\mathcal{D}$ is a saturated linearly ordered structure and $k \geq 0$ then there is a potential E for $\mathcal{D}^k$, given by $E(\bar{x}) = \lfloor \log_{3^k}(qE(\bar{x})) \rfloor$.*

Proof sketch. First we show that if $\bar{x} \in \mathcal{D}^k$ has $qE(\bar{x}) \geq s$ for some large s, then we can extend it to a point $(\bar{x}, y_1)$ laying in a prescribed cell τ_1 in $\mathcal{D}^{k+1}$, and with $qE(\bar{x}, y_1) \geq s'$ for some still large s'. We need to assume that τ_1 is a "reasonable" cell – i.e. the projection of τ_1 onto the first k coordinates contains the point $\bar{x}$. The cell then τ_1 typically requires that y_1 lays in the interval between some two coordinates of $\bar{x}$, and moreover is of some type t. The idea is to choose y_1 to lie "in the middle" of the prescribed interval, obtaining a point with quasi-potential larger than a third of s. Iterating this process, we insert k coordinates, getting a point $(\bar{x}, \bar{y})$ which lays in a prescribed cell τ in $\mathcal{D}^{2k}$ with $qE(\bar{x}, \bar{y}) \geq \frac{s}{3^k}$. We conclude that $qE(\bar{y}) \geq \frac{s}{3^k}$. By taking the appropriate logarithm, we construct the actual potential. To show that qE is unbounded on all cells, we reuse the above reasoning: we start with the empty tuple ε with $qE(\varepsilon) \geq s$ for all s by saturation, and insert coordinates one after the other, preserving high qE-value, and finally obtaining a point in the desired cell. ◀

3.4 Computability and Complexity issues

We now turn to the complexity analysis. In practical applications, a base linearly ordered structure $\mathcal{D}$ is fixed and new constants come with the description of the automaton. Here, and in other sections, for a finite set $C \subseteq \mathcal{D}$, we denote by $\mathcal{D}[C]$ the structure $\mathcal{D}$ extended by the constants in C. The above motivation leads to the following decision problem, which we call *emptiness of $\mathcal{D}[C]$-automata*. **Input:** 1) A set C of elements of $\mathcal{D}$, encoded in some specified presentation 2) A description of a $\mathcal{D}[C]$-automaton $\mathcal{A}$. **Decide:** is $\mathcal{A}$ empty?

Our decision algorithm essentially works as follows: We first compute a saturated expansion $\hat{\mathcal{D}}$ of the structure $\mathcal{D}[C]$. We then compute in $\hat{\mathcal{D}}$ the cell automaton $\mathcal{A}'$ associated to $\mathcal{A}$ and check its emptiness. The correctness of this algorithm follows from the results of the previous sections. In order to decrease the complexity, we materialize neither $\hat{\mathcal{D}}$ nor $\mathcal{A}'$ but compute them on the fly. For $\hat{\mathcal{D}}$ we need to know the new (virtual) constants that were introduced together with their type. For $\mathcal{A}'$ we essentially need to know the possible types that may occur in the interval between any two successive (virtual) constants of $\hat{\mathcal{D}}$. We therefore need to assume that $\mathcal{D}$ is equipped with an algorithm giving us this information. This is formalized as follows.

A sequence $s : t_0 T_1 t_1 T_2 \cdots t_{n-1} T_n t_n$ where $t_0, t_1, \ldots, t_{n-1}, t_n$ are types and $T_1, T_2, \ldots, T_n$ are sets of types of $\mathcal{D}$ is called a *saturator*. Given two elements x and y of $\mathcal{D}$, we say that a sequence $c_0 < c_1 < \ldots < c_n$ of (possibly virtual) elements of $\mathcal{D}$ *matches s in* $[x, y]$ if $c_0 = x$, $c_n = y$, c_i is of type t_i and T_i^* are precisely the sequences of types realizable in $]c_{i-1}, c_i[$.

▶ **Example 12.** Over $\langle \mathbb{Z}, <, 0 \rangle$, the saturator $t_\emptyset T_\epsilon t_0 T_\epsilon t_\emptyset T_\epsilon t_\emptyset$, where $t_\emptyset$ is the empty type, t_0 is the type of 0 and $T_\epsilon = \{\}$, is matched in $[-1, 2]$ by the sequence $-1 < 0 < 1 < 2$.

▶ **Example 13.** Over $\langle \mathbb{Q}, <, 0, P \rangle$, where $P = \{(1 + 1/n)^n \mid n \in \mathbb{N}^+\}$, the sequence $-100 < 0 < 2 < e < 100$ matches the saturator $t_\emptyset T_\emptyset t_0 T_\emptyset t_P T_P t_\emptyset T_\emptyset t_\emptyset$ in $[-100, 100]$ where t_P is the type P, $T_P = \{t_\emptyset, t_P\}$, $T_\emptyset = \{t_\emptyset\}$, while t_0 and $t_\emptyset$ are as before.

A saturator is *realizable* in $[x, y]$ if there is a sequence which matches it in $[x, y]$. A linearly ordered structure $\mathcal{D}$ is said to be *computable* if there is an algorithm that given any two elements x, y of $\mathcal{D}$ replies the length of a saturator s realizable in $[x, y]$, and afterwards, when given a number i as input, returns the i^{th} element of the sequence s. When this algorithm works in PTime, we say that $\mathcal{D}$ is P-computable. Note that for integers or rationals, an

obvious saturator algorithm runs in PTime, even if the numbers are coded in binary. By a careful analysis of the proof of Theorem 2 we obtain the following.

▶ **Theorem 14.** *If $\mathcal{D}$ is* P*-computable (respectively, computable) the emptiness problem of $\mathcal{D}[C]$-automata is in* PSpace *(respectively, decidable).*

Sketch. Assume $\mathcal{D}$ is P-computable. Let $\mathcal{A}$ be a $\mathcal{D}[C]$-automaton. For each pair of consecutive elements in C, we invoke the saturator algorithm. Let l be maximal of the lengths of the saturators as returned by the saturator algorithm. Since l has polynomial size, each (virtual) constant in the saturated expansion $\hat{\mathcal{D}}$ of $\mathcal{D}[C]$ for that interval has a polynomial size presentation. Hence, in the end, each cell of $\hat{\mathcal{D}}$ has a polynomial size representation. Given two cells, it can be checked in polynomial time whether there is a transition in the cell automaton associated to $\mathcal{A}$. This is because $\hat{\mathcal{D}}$ is saturated, so only local consistency needs to be checked – namely, whether the type associated with each variable is indeed a possible type in the interval where this variable must be realized – and this information is also provided by the saturator algorithm. Decidability then follows by guessing on the fly the appropriate sequence of cells. ◀

4 Infinite words

We now consider infinite words. Recall that a *$\mathcal{D}$-automaton $\mathcal{A}$ of dimension k* consists of the transition regions $(\delta_a)_{a\in A}$, an initial region $\tau_I \subseteq \mathcal{D}^k$ and an accepting region $\tau_F \subseteq \mathcal{D}^k$. A *Büchi $\mathcal{D}$-automaton* is an automaton over infinite words, in which a run ρ is declared *accepting* if it visits infinitely often the region[2] τ_F.

The following example shows that Theorem 2 does not directly extend to infinite words.

▶ **Example 15.** An infinite sequence of numbers $n_1, n_2, \ldots$ induces an infinite word $b^{n_1}ab^{n_2}\ldots$ Let $\mathcal{L}$ be the language of words which are induced by bounded sequences. Then $\mathcal{L}$ is recognized by the Büchi $\mathcal{D}$-automaton $\mathcal{A}$ described in Example 1, where $\mathcal{D} = \langle \mathbb{N}, <, 0\rangle$.

The language $(b^*a)^\omega \setminus \mathcal{L}$ does not contain any ultimately periodic word. In particular, $\mathcal{L}$ cannot be ω-regular. We will see that also nonempty Büchi $\mathcal{D}$-automata must accept some ultimately periodic word, so we deduce that they are not closed under complementation.

Although Büchi $\mathcal{D}$-automata are more expressive than Büchi automata, we show that emptiness can be reduced to the finite case, under additional computability assumptions concerning $\mathcal{D}$. We need the following notions. For a type t, we say that a (virtual) element x of $\mathcal{D}$ is a *left t-limit* if for every $y \in D$ such that $y < x$, the type t occurs between y and x. A *right t-limit* is defined dually. The *ω-type* of a point $x \in \bar{D}$ is its type extended by the specification, for all types t, whether it is a left or right t-limit or none. An *ω-saturator* is a sequence of the form $t_0T_1t_1T_2\cdots T_nt_n$ where each t_i is an ω-type and each T_i is a set of ω-types. We extend the notion of *matching* to the case of ω-saturators in an obvious way. Next, we define (P-)*ω-computability* analogously to (P-)computability, where, for given $x, y \in D$ the algorithm should calculate an ω-saturator realizable in $[x, y]$ treated as a subset of $\bar{D}$. Note that the structures considered earlier are P-ω-computable.

▶ **Theorem 16.** *For a* P*-ω-computable (resp. ω-computable) linearly ordered structure $\mathcal{D}$, emptiness of Büchi $\mathcal{D}[C]$-automata is in* PSpace *(resp. decidable).*

[2] An acceptance condition requiring that ρ visits infinitely often a point $\bar{x} \in \tau_F$ yields a weaker model.

It appears that $\mathcal{D}$-automata are related with ωB-automata, a model defined in [2]. Informally, an ωB-*automaton* is a nondeterministic automaton equipped with counters which can be incremented and reset, but not tested during the run. The acceptance condition of the automaton, apart from a Büchi condition, requires that the counters remain bounded when processing the infinite input word. We call a language recognized by a $\mathcal{D}$-automaton (resp. ωB-automaton) a $\mathcal{D}$-*regular* (resp. ωB-*regular*) language.

▶ **Theorem 17.** *If $\mathcal{D} = \langle \mathbb{Q}, <, c_1, c_2, \ldots, c_m \rangle$ then the classes of $\mathcal{D}$-regular languages and ω-regular languages coincide. If $\mathcal{D} = \langle \mathbb{N}, <, c_1, c_2, \ldots, c_m \rangle$ then the classes of $\mathcal{D}$-regular languages and ωB-regular languages coincide. Moreover, all translations are effective.*

It appears that the above dichotomy is valid for all linearly ordered structures $\mathcal{D}$, depending on whether a discrete set is definable in $\mathcal{D}$ (this can be formalized). The general result will appear in the journal version of this paper.

As a conclusion from the strong complementation result of [2], we obtain:

▶ **Corollary 18.** *Let A be a finite alphabet and $\mathcal{D}$ be as in the above theorem. It is decidable whether a Boolean combination of languages accepted by $\mathcal{D}$-automata over A is empty.*

5 Temporal logic

We fix a linearly ordered structure $\mathcal{D}$. We consider a variant of LTL where each atomic predicate is replaced by a proposition comparing the current configuration with the next one. We denote this logic by LTL($\mathcal{D}$). Its syntax is given by the following grammar:

$$\begin{aligned}
\varphi &:: \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi \mid \psi \\
\psi &:: \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \mid a \mid P(\alpha) \mid \alpha = \alpha \mid \alpha < \alpha \\
\alpha &:: x \mid \mathsf{X}x \mid c
\end{aligned}$$

where $x \in \{x_1, x_2, \ldots\}$ are variables, $a \in A$ letters, $c \in \mathcal{D}$, and P unary predicates of $\mathcal{D}$.

The semantics of a formula $\varphi \in \mathrm{LTL}(\mathcal{D})$ is defined on sequences of elements in $A \times D^k$, where k is the maximal number such that x_k appears in φ. Let $w = (a_1, \bar{z}_1)(a_2, \bar{z}_2)\ldots$ be such a sequence. Let ψ be a proposition as described by the grammar above and let n be a position of w. We write $(w, n) \models \psi$ if from ψ we obtain a sentence which is valid in $\mathcal{D}$ after replacing the terms of the form x_i by the i^{th} coordinate of $\bar{z}_n$, and terms of the form $\mathsf{X}x_i$ by the i^{th} coordinate of $\bar{z}_{n+1}$. Using the classical semantics of LTL, we extend this notation to all formulas φ of LTL($\mathcal{D}$), and say that w is *accepted* by φ if $(w, 1) \models \varphi$.

The following result can be established along the same lines as in the classical translation of LTL formulas into automata. C_φ denotes the set of values which appear in the formula φ.

▶ **Theorem 19.** *For any LTL($\mathcal{D}$) formula φ there exists a $\mathcal{D}[C_\varphi]$-automaton $\mathcal{A}_\varphi$ whose runs are exactly the sequences accepted by φ.*

▶ Remark. In [8] terms of the form $\mathsf{X}^j x$ were allowed, enabling to compare the data value with one that will occur j steps later. This can be simulated by a formula of LTL($\mathcal{D}$) after adding new dimensions used to guess in advance the values that will appear in the following j steps. The consistency of the guesses can be enforced at each step by a formula of LTL($\mathcal{D}$).

▶ Remark. It is tempting to consider other temporal formalisms. One could define a variant of μ-calculus analogously to the extension LTL($\mathcal{D}$) described above, and our model of automata still can simulate such formulas. However, as noticed by [7, 11, 8] over various domains, CTL($\mathcal{D}$) is undecidable.

6 Databases

Following [11] we apply in this section the results of the previous sections in the context where a finite database is present. For this, we fix a relational schema σ and a linearly ordered structure $\mathcal{D}$. A database over σ is then, for each relation symbol of σ, a finite relation of the appropriate arity over the domain of $\mathcal{D}$.

A $(\mathcal{D}, \sigma)$-*automaton* $\mathcal{A}$ *of dimension* k is described as follows. As before, the configurations of $\mathcal{A}$ are points in the space $\mathcal{D}^k$. We assume an initial region $\tau_I \subseteq \mathcal{D}^k$ and an accepting region $\tau_F \subseteq \mathcal{D}^k$. For each $a \in A$, the transition δ_a is a set of pairs of the form (τ, φ), where τ is a region in $\mathcal{D}^k \times \mathcal{D}^k$, while φ is a propositional formula over σ with $2k$ free variables. Given a finite database M over the schema σ and two points $\bar{x}, \bar{y} \in \mathcal{D}^k$, we write $\bar{x} \xrightarrow{a}_M \bar{y}$ iff there is a pair $(\tau, \varphi) \in \delta_a$ with $(\bar{x}, \bar{y}) \in \tau$ and $M \models \varphi(\bar{x}, \bar{y})$. A *run* ρ *of* $\mathcal{A}$ *on* $w = a_1 a_2 \dots$ *over* M, is a sequence of configurations $\bar{x}_0, \bar{x}_1, \dots \in \mathcal{D}^k$ such that $\bar{x}_0 \in \tau_I$ and for each $n > 0$, $\bar{x}_{n-1} \xrightarrow{a_n}_M \bar{x}_n$. Acceptance conditions are defined as before, for finite or infinite runs.

▶ **Example 20.** We fix $\mathcal{D} = \langle \mathbb{Q}, <, 0 \rangle$ and $\sigma = \{P\}$ where P is unary. Consider the $(\mathcal{D}, \sigma)$-automaton $\mathcal{A}$ of dimension 1, where τ_I, τ_F are both described by $x = 0$. The region τ_b is the set of points (x, y) such that $x < y$ and δ_b is the pair $(\tau_b, P(y))$ while δ_a is just specified by the region $y = 0$. Hence, the length of any sequence of b's is bounded by the size of the database, so the infinite words w for which there exists a database M and a run of $\mathcal{A}$ consistent with w and M are exactly the bounded sequences of Example 15. Recall from Theorem 17 that without the underlying database $\mathcal{D}$-automata would only recognize ω-regular languages.

Our goal is to decide whether, for a given $(\mathcal{D}, \sigma)$-automaton $\mathcal{A}$, there exists a finite database M such that A has an accepting run over M.

▶ Remark. A more general setting would allow the database constraints φ to be existential queries with $2k$ free variables. This setting can be easily reduced to the one above, by having the automaton $\mathcal{A}$ guess the values for the quantified variables using extra dimensions.

It appears that adding the database does not influence the decidability results obtained in Theorem 14 and Theorem 16 for finite and infinite words, respectively. We state the result in the database setting.

▶ **Theorem 21.** *Let* $\mathcal{D}$ *be a computable (resp.* ω*-computable) linearly ordered structure. Given a* $(\mathcal{D}[C], \sigma)$*-automaton* $\mathcal{A}$*, it is decidable whether there exists a finite database* M *and a finite (resp. infinite) word* w *such that there is an accepting run of* $\mathcal{A}$ *on* w *over* M*. Moreover if* $\mathcal{D}$ *is* P*-computable (resp.* P*-*ω*-computable) then the complexity is* PSpace *for a fixed schema,* ExpSpace *otherwise.*

Sketch. We only sketch here the ideas for the case of finite words. The case of infinite runs is done by a reduction to the finite one in a way similar to the proof of Theorem 16. We temporarily treat $\mathcal{A}$ as a $\mathcal{D}[C]$-automaton, for each a merging into one all regions appearing in the transition δ_a. Let $\mathcal{A}'$ be the corresponding cell automaton, over the saturated expansion $\hat{\mathcal{D}}$ of $\mathcal{D}[C]$. A run π' of $\mathcal{A}'$ is lifted to a run π of $\mathcal{A}$ inductively, assuring that in each step we obtain a configuration with a big potential. The key observation is that the new configuration can be chosen so that it does not use values which appeared in previous configurations, unless it is explicitly required by the transition. Hence it is enough to guess the database relations for the constants $\hat{\mathcal{D}}$ (this is exponential in the maximal arity of a relation in σ, hence the complexity becomes ExpSpace unless this arity is fixed) and maintain locally, by adding states to $\mathcal{A}'$, the consistency of the database. Each time a new configuration reuses

some data values, this is enforced by the transition and hence consistency can be tested by $\mathcal{A}'$ at the time of the transition. Otherwise, as we observed, fresh data values can always be chosen and consistency with the previous constraints is immediate. ◀

Logic. The logic LTL($\mathcal{D}$) described in Section 5 can be extended to a logic LTL($\mathcal{D}, \sigma$) by adding, for each symbol E in σ of arity k, a production $\psi :: E(\alpha, \alpha, \ldots, \alpha)$, in which the right-hand side has k arguments. Atoms of the form $E(x, \mathsf{X}y)$, $F(x, y, \mathsf{X}x)$ etc. represent database queries. Just as before, the logic LTL($\mathcal{D}, \sigma$) can be transformed into $(\mathcal{D}, \sigma)$-automata, and thus can be effectively verified. We omit the details in this abstract.

7 Conclusions

We have introduced an automata model capable of storing values from a linearly ordered set, additionally equipped with constants and unary predicates. We have shown how to simulate runs of such automata by finite state automata. This translation is effective as soon as the structure has some reasonable computational properties. This provides a uniform presentation for results concerning specific data domains that were disseminated in various papers. Moreover this sometimes decreases the known complexity or solves open questions.

It would be interesting to see whether our work can be extended to other structures. We leave this for future work.

References

1 R. Alur, P. Cerný, and S. Weinstein. Algorithmic analysis of array-accessing programs. In *Computer Science Logic (CSL)*, pages 86–101, 2009.

2 M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Logic in Computer Science (LICS)*, pages 285–296, 2006.

3 M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Logic in Computer Science (LICS)*, pages 7–16, 2006.

4 A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Fundamentals of Computation Theory (FCT)*, pages 1–22, 2007.

5 A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07*, 2007.

6 P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2), 2003.

7 K. Čerāns. Deciding properties of integral relational automata. In *ICALP*, pages 35–46, 1994.

8 S. Demri. Linear-time temporal logics with Presburger constraints: An overview. *J. of Applied Non-Classical Logics*, 16(3-4):311–347, 2006.

9 S. Demri and R. Gascon. The effects of bounding syntactic resources on Presburger LTL. *Journal of Logic and Computation*, 19(6):1541–1575, 2009.

10 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

11 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *Intl. Conf. on Database Theory (ICDT)*, pages 252–267, 2009.

12 X. Du, C. R. Ramakrishnan, and S. A. Smolka. Region graphs for infinite-state systems. unpublished manuscript, 2007.

13 D. Figueira, P. Hofman, and S. Lasota. Relating timed and register automata. In *Intl. Workshop on Expressiveness in Concurrency (EXPRESS'10)*, 2010.

14 M. Kaminski and N. Francez. Finite memory automata. *Theor. Comp. Sci.*, 134(2):329–363, 1994.