Submitted to the Symposium on Theoretical Aspects of Computer Science
www.stacs-conf.org

AUTOMATA BASED VERIFICATION OVER LINEARLY ORDERED DATA DOMAINS

LUC SEGOUFIN [1] AND SZYMON TORUŃCZYK [2]

[1] INRIA and ENS Cachan

[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.

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 [BMS$^+$06, DL09, KF94], or limit the recursion or the expressive power [BHJS07, BJS07, Dem06, DG09], or only apply over specific data domains [DHPV09, Čerāns94, Dem06, DG09, BPT03, ACW09].

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 [DHPV09] 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.

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

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

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 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. In this model, the automaton passes from one configuration to another while processing an input word. A configuration of the automaton is a tuple of data values of a fixed arity. A transition constrains the values of the current configuration relative to the values of the previous configuration, using 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

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

proof of the result of [DHPV09] 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 [Čerāns94] over integer data values. Čerāns solved the decidability of the emptiness problem using Dixon'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 also generalizes the register automata studied in [BPT03, KF94] as registers can be simulated by adding extra dimensions in the arity of the tuples of the configurations. 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 [ACW09] 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 where developed for timed automata or over dense linearly ordered structures. See for instance [DRS07]. Over dense linearly ordered structures, register automata generalize the notion of timed automata in a sense explained in [FHL10]. Our notion of potential hence generalizes these ideas even for discrete linearly ordered structures.

The work of [DHPV09] 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 [DHPV09], 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 [DHPV09].

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 [Dem06] for a survey. This also follows from our results.

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 $w = a_1 a_2 \dots a_n$ be a finite word. A *run* ρ of $\mathcal{A}$ on w is a sequence of configurations $\bar{x}_0, \bar{x}_1, \dots, \bar{x}_n \in D^k$ such that $\bar{x}_0 \in \tau_I$ and for each $i \in \{1, \dots, n\}$, 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.

Example 2.1. 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 2.2. 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 2.3. Since one can construct Cartesian products of $\mathcal{D}$-automata, the languages they recognize are closed under union and intersection.

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 2.4. *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 2.4 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 from the definitions:

Proposition 3.1. *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 3.2. 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, 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 3.3. If $\mathcal{D} = \langle \mathbb{Q}, <, 0, 3 \rangle$, the mapping constantly equal to ∞ is a potential for $\mathcal{D}^2$. Yet, if $\mathcal{D} = \langle \mathbb{Z}, <, 0, 3 \rangle$ as in Example 3.2, it can be verified that $\mathcal{D}^2$ cannot be equipped with a potential. However, if $\mathcal{D} = \langle \mathbb{Z}, <, 0, 1, 2, 3 \rangle$, there is a potential for $\mathcal{D}^2$ which, roughly, assigns to a pair (x_1, x_2) a number depending on how far x_1, x_2 are from each other and from the constants $0, 1, 2, 3$. 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 by an easy induction from the definition.

Proposition 3.4. *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 3.5. *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 2.4. In Example 3.2 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[2] of elements and virtual elements of D, linearly ordered by $<$. 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 3.6. *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*[3] if any subset of D has its supremum, i.e. a least upper bound. 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.

[2]Known as the Dedekind completion, modulo the elements $\emptyset, D$ which are normally not considered in $\bar{D}$

[3]Again, this deviates slightly from the standard notion of completeness by the least and smallest elements

Lemma 3.7. *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 $t \in T$ which does not occur in $]a,b[$, so we may reduce the set T to $T \setminus \{t\}$ 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 3.7 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 3.8. We apply the procedure of Theorem 3.6 to $\mathcal{D} = \langle \mathbb{Z}, <, 0, 3 \rangle$ of Example 3.2. 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 3.6 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 3.9. *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:*

$$E(\bar{x}) = \lfloor \log_{3^k}(qE(\bar{x})) \rfloor.$$

The detailed proof is relegated to Appendix A. Below we give the main idea.

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 1. 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 2. 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. A careful analysis of the proof of Theorem 2.4 gives:

Theorem 3.10. *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 hence only local consistency needs to be checked: Whether the type associated to 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[4] τ_F.

The following example shows that Theorem 2.4 does not directly extend to infinite words.

Example 4.1. 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 2.1, 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 4.2. *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).*

It appears that $\mathcal{D}$-automata are related with ωB-automata, a model defined in [BC06]. 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

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

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 4.3. *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.*

The proof of the theorem is in Appendix C. It appears that the above dichotomy is valid for any 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 [BC06], we obtain:

Corollary 4.4. *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:

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

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 semantic 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 semantic 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 5.1. *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 5.2. In [Dem06] terms of the form $\mathsf{X}^j x$ where also allowed in the formulas, enabling to compare the current data value with one that will occur j steps later. This can be simulated by a formula of LTL($\mathcal{D}$) after multiplying the dimension by j and guessing in advance the value that will appear in the next j steps. The consistency of the guesses can be enforced at each step by a formula of LTL($\mathcal{D}$).

Remark 5.3. 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 [Čerāns94, DHPV09, Dem06] over various domains, CTL($\mathcal{D}$) is undecidable.

6. Databases

Following [DHPV09] 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 6.1. 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 $y = 0$. As the length of any sequence of b's must be bounded by the size of the underlying finite database, 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 4.1. Recall from Theorem 4.3 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 6.2. 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 3.10 and Theorem 4.2 for finite and infinite words, respectively. We state the theorem in the database setting. The proof can be found in Appendices D and E.

Theorem 6.3. *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 4.2. 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 become 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 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

[ACW09] R. Alur, P. Cerný, and S. Weinstein. Algorithmic analysis of array-accessing programs. In *Computer Science Logic (CSL)*, pages 86–101, 2009.

[BC06] M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Logic in Computer Science (LICS)*, pages 285–296, 2006.

[BHJS07] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Fundamentals of Computation Theory (FCT)*, pages 1–22, 2007.

[BJS07] A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07*, 2007.

[BMS+06] 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.

[BPT03] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2), 2003.

[Čerāns94] K. Čerāns. Deciding properties of integral relational automata. In *ICALP*, pages 35–46, 1994.

[Dem06] S. Demri. Linear-time temporal logics with Presburger constraints: An overview. *J. of Applied Non-Classical Logics*, 16(3-4):311–347, 2006.

[DG09] S. Demri and R. Gascon. The effects of bounding syntactic resources on Presburger LTL. *Journal of Logic and Computation*, 19(6):1541–1575, 2009.

[DHPV09] 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.

[DL09] S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

[DRS07] X. Du, C. R. Ramakrishnan, and S. A. Smolka. Region graphs for infinite-state systems. unpublished manuscript, 2007.

[FHL10] D. Figueira, P. Hofman, and S. Lasota. Relating timed and register automata. In *Intl. Workshop on Expressiveness in Concurrency (EXPRESS'10)*, 2010.

[KF94] M. Kaminski and N. Francez. Finite memory automata. *Theor. Comp. Sci.*, 134(2):329–363, 1994.

Appendix A. Proof of Theorem 3.9

Let $k, l \geq 0$ and let $\pi\colon \mathcal{D}^{k+l} \to \mathcal{D}^k$ denote the projection onto the first k coordinates. The following two lemmas are the main ingredients of the proof of the theorem.

Lemma A.1. *For all $\bar{x} \in \mathcal{D}^{k+l}$ we have $qE(\pi(\bar{x})) \geq qE(\bar{x})$.*

Proof. Fix $\bar{x} \in \mathcal{D}^{k+l}$ and assume that $qE(\pi(\bar{x})) = s$.

Recall the definition of capacity, from $\pi(\bar{x})$ we obtain a sequence $u_1 < u_2 < \ldots < u_n$ of (virtual) constants or coordinates of $\pi(\bar{x})$ and a set T_i of types for all $i < n$ used in the definition of capacity for each interval $]u_i, u_{i+1}[$. Similarly from $\bar{x}$ we obtain a sequence $v_1 < v_2 < \ldots < v_m$ of (virtual) constants or coordinates of $\bar{x}$ and a set U_i of types for all $i < m$ used in the definition of capacity of each interval $]v_i, v_{i+1}[$.

By hypothesis we have an interval $I =]u_i, u_{i+1}[$ and a sequence $t_1, \ldots, t_s$ of types in T_i which is not realizable in I. There must be an interval $J =]v_j, v_{j+1}[$ which is contained in I. Then the sequence $t_1, \ldots, t_s$ is also not realizable in J. As $T_i \subseteq U_j$ we have $qE(\bar{x}) \leq s$. ∎

Lemma A.2. *Let θ, τ be cells in $\mathcal{D}^{k+l}$ and $\mathcal{D}^k$, respectively, such that $\pi(\theta) \subseteq \tau$ and let $s > 0$ be a number.*

If $\bar{x} \in \tau$ is such that $qE(\bar{x}) > 3^l \cdot s$ then there exists an element $\bar{y} \in \theta$ of the form $\bar{y} = (\bar{x}, x', x'', \ldots, x^{(l)})$ such that $qE(\bar{y}) > s$.

Proof. Let us consider the case $l = 1$. Let us suppose the converse of the statement of the lemma, i.e. for all x' such that $(\bar{x}, x') \in \theta$ we have $qE(\bar{y}) \leq s$. We will show that in this case, $qE(\bar{x}) \leq 3s$, contradicting the assumption.

Recall the definition of capacity, from $\pi(\bar{x})$ we obtain a sequence $u_1 < u_2 < \ldots < u_n$ of (virtual) constants or coordinates of $\pi(\bar{x})$ and a set T_i of types for all $i < n$ used in the definition of capacity for each interval $]u_i, u_{i+1}[$.

The cell θ requires the point x' to be located in some interval $I =]u_i, u_{i+1}[$ of $\bar{x}$ and also to be of some type t. Inserting x' in that interval creates two new intervals $]u_i, x'[$ and $]x', u_{i+1}[$. Let $\mathrm{left}(x')$ denote the set of sequences of length s not realizable in $]u_i, x'[$. Similarly, let $\mathrm{right}(x')$ denote the set of sequences of length s not realizable in $]x', u_{i+1}[$. By assumption, for any $x' \in I$ of type t, at least one of the sets $\mathrm{left}(x'), \mathrm{right}(x')$ is nonempty.

Assume that both $x' < x''$ are of type t and lie in I. If $w \in \mathrm{left}(x'')$ then $w \in \mathrm{left(x')}$. Therefore, $\mathrm{left}(x')$ is monotonically decreasing with x' of type t. Let w_L be a sequence which lies in the intersection of all nonempty sets of the form $\mathrm{left}(x')$ where x' ranges through the elements of I of type t (if all of these sets $\mathrm{left}(x')$ are empty, let w_L be the empty sequence). Similarly, $\mathrm{right}(x')$ is monotonically increasing with x', and let w_R be defined analogously to w_L.

We claim that the concatenation w of w_L, t and w_R is not realizable in I. Assume the contrary, i.e. there is a sequence $x_1 < x_2 < \ldots < x_s < x_{s+1} < \ldots < x_{2s+1}$ of elements of I which realizes w. If $\mathrm{left}(x_{s+1})$ is nonempty then it contains w_L. In particular, w_L is not realizable in $]u_i, x_{s+1}[$. This is a contradiction, since $x_1 < \ldots < x_s$ is such a realization. Thus, $\mathrm{left}(x_{s+1})$ is empty. Similarly, $\mathrm{right}(x_{s+1})$ must be empty. This contradicts the assumption that for any point x' of type t which lies in I, either $\mathrm{left}(x')$ or $\mathrm{right}(x')$ is nonempty. Thus, the sequence w is not realizable in I, showing that $qE(\bar{x}) \leq 2s + 1 \leq 3s$. This proves the lemma in the case $l = 1$.

For $l > 1$ we apply inductively the result obtained above, in each step extending $\bar{x}$ with extra coordinates $x', x'', x''', \ldots$, according to the restrictions prescribed by θ. In step i, we obtain a point $\bar{y} = (\bar{x}, x', x'', \ldots x^{(i)})$ in $\mathcal{D}^{k+i}$ with $qE(\bar{y}) > 3^{l-i}$ and repeat the reasoning if

$i < l$. It is straightforward to verify that after l steps we obtain a point $\bar{y} \in \mathcal{D}^{k+l}$ with the desired property. ∎

We now turn to the proof of the theorem. First, we prove that qE is not bounded when restricted to cells. Let $\theta \subseteq \mathcal{D}^l$ be a cell. We consider the mapping π which maps every element of $\mathcal{D}^l$ to the element $\varepsilon \in \mathcal{D}^0$. Note that $qE(\varepsilon) = \infty$ since $\mathcal{D}$ is saturated. In particular, for any number $s > 0$, $qE(\varepsilon) > 3^l \cdot s$. We apply Lemma A.2 with $k = 0$, obtaining an element $\bar{y} \in \theta$ with $qE(\bar{y}) > s$.

Next we show that transitions decrease the potential at most 3^k times. Let $\tau \subseteq \mathcal{D}^{2k}$, $s > 0$ and $\bar{x} \in \pi_1(\bar{\tau})$ be such that $qE(\bar{x}) > 3^k \cdot s$. We apply Lemma A.2 to the mapping $\pi \colon \mathcal{D}^{2k} \to \mathcal{D}^k$, finding an element $\bar{y} \in \tau$ with $qE(\bar{y}) > s$. We define $\bar{z} = \pi_2(\bar{y})$. Then, by Lemma A.1 we have that $qE(\bar{z}) > s$. From this it follows that the function $E(\bar{x}) = \lfloor \log_{3^k}(qE(\bar{x})) \rfloor$ is a potential for $\mathcal{D}^k$, ending the proof of the theorem. ∎

Appendix B. Proof of Theorem 4.2

Let $\mathcal{A}$ be a $\mathcal{D}$-automaton of dimension k. Given a point $\bar{x}$ of $\mathcal{D}^k$, we denote by $\bar{x}[j]$ it's j^{th}-coordinate.

Let $\bar{\mathcal{D}}$ be the completion of $\mathcal{D}$. Let $\bar{\mathcal{D}}^{\pm}$ denote $\bar{\mathcal{D}}$ extended by the unary predicates P_t^+ and P_t^- marking those elements which are left, respectively right, t-limits, for all types t appearing in $\bar{\mathcal{D}}$. Let $\hat{\mathcal{D}}$ be obtained from saturating $\bar{\mathcal{D}}^{\pm}$. Note that (P-)ω-computability of $\mathcal{D}$ is equivalent to (P-)computability of $\hat{\mathcal{D}}$. Let π_1 denote the projection from $\hat{\mathcal{D}}^{3k}$ onto the first k coordinates.

Let τ be a cell in $\hat{\mathcal{D}}^{3k}$. We say the run ρ of $\mathcal{A}$ is *τ-cyclic* if there exists a factorization $v_0 v_1 v_2 \dots$ of w and a vector $\bar{l} = (l_1, \dots, l_k)$ of elements of $\bar{D}$ such that, with $\bar{x}_i$ denoting the configuration in ρ after reading the prefix $v_0 v_1 v_2 \dots v_{i-1}$,

i) $(\bar{x}_i, \bar{x}_{i+1}, \bar{l}) \in \tau$ for $i = 1, 2, 3, \dots$,
ii) for every $j \in \{1, \dots, k\}$, the sequence $\bar{x}_1[j], \bar{x}_2[j], \bar{x}_3[j], \dots$ is either strictly increasing with supremum $\bar{l}[j]$ or strictly decreasing with infimum $\bar{l}[j]$ or constantly equal to $\bar{l}[j]$.

The proof of the following lemma is a standard Ramsey argument.

Lemma B.1. *Let $\mathcal{A}$ be a Büchi $\mathcal{D}$-automaton of dimension k, admitting an accepting run ρ. Then ρ is τ-cyclic for some cell τ in $\hat{\mathcal{D}}^3$ such that $\pi_1(\tau) \subseteq \tau_F$.*

Let π, λ be two finite runs of $\mathcal{A}$. If $I \subseteq D$ is an interval and $j \in \{1, \dots, k\}$, we denote by $\lambda[j] \cap I$ the set of j^{th} coordinates of the elements of λ which lie in I. Let $\bar{T}$ be a k-tuple $(T_1, \dots, T_k)$ where each T_j is a set of types of $\mathcal{D}$.

Let τ be a cell in $\hat{\mathcal{D}}^{3k}$. We say that π, λ is a *lasso compatible with* $(\tau, \bar{T})$ if for some $\bar{x}_0 \in \tau_I$,$(\bar{x}, \bar{y}, \bar{l}) \in \tau$,

(1) π starts in $\bar{x}_0$ and ends in $\bar{x}$
(2) λ starts in $\bar{x}$ and ends in $\bar{y}$
(3) $\langle \bar{x} \rangle = \langle \bar{y} \rangle$
(4) for each $1 \leq j \leq k$, T_j is precisely the set of types of the elements $\lambda[j] \cap [\bar{x}[j], \bar{y}[j]]$
(5) for each $1 \leq j \leq k$ and $t \in T_j$.
  – if $\bar{x}[j] < \bar{y}[j]$ then $\bar{l}[j]$ is a left t-limit and $\bar{y}[j] < \bar{l}[j]$,
  – if $\bar{x}[j] > \bar{y}[j]$ then $\bar{l}[j]$ is a right t-limit and $\bar{y}[j] > \bar{l}[j]$,

– if $\bar{x}[j] = \bar{y}[j]$ then $\bar{l}[j]$ has type t and $\bar{y}[j] = \bar{l}[j]$.

(6) for each $1 \leq i, j \leq k$, if $\bar{x}[i] \leq \bar{x}[j]$, then $\bar{l}[i] \leq \bar{l}[j]$.

Lemma B.2. *Let τ be a cell in $\hat{\mathcal{D}}^{3k}$ and $\bar{T}$ be a k-tuple of set of types of $\mathcal{D}$. It is decidable whether $\mathcal{A}$ has a lasso compatible with $(\tau, \bar{T})$.*

Proof. We can first easily verify that the properties 3, 5, 6 in the definition of the lasso are consistent with the cell τ. The remaining properties 1, 2, 4 can be tested by a finite $\hat{\mathcal{D}}$-automaton $\mathcal{B}$ of dimension $5k$ with states, described below (recall that states can be simulated with one extra dimension). The first $4k$ coordinates are used to guess $\bar{x}_0, \bar{x}, \bar{y}, \bar{l}$. The last k coordinates are used to simulate $\mathcal{A}$. The initial configuration is a point $(\bar{x}_0, \bar{x}, \bar{y}, \bar{l}, \bar{z})$ such that $\bar{x}_0 \in \tau_I$, $(\bar{x}, \bar{y}, \bar{l}) \in \tau$ and $\bar{z} = \bar{x}_0$.

The run of the automaton consists of two stages: finding the run π and finding the run λ. The automaton knows which stage it is performing thanks to a Boolean flag encoded in the state of the automaton. In each stage, $\mathcal{B}$ simulates $\mathcal{A}$ on the $\bar{z}$ coordinate. The first stage is ended once $\bar{z} = \bar{x}$. The second stage is ended once $\bar{z} = \bar{y}$. This way, we ensure the properties 1 and 2. To ensure the property 4, during the second stage, whenever $\bar{x}[j] \leq \bar{z}[j] \leq \bar{y}[j]$, the automaton adds the type of $\bar{z}[j]$ to a set S_j, which is also stored in the state of the automaton. The accepting state of the automaton requires that $S_j = T_j$ for all j. It is clear that $\mathcal{B}$ accepts if and only if there exists a lasso compatible with $(\tau, \bar{T})$. ∎

The following Proposition ends the proof of Theorem 4.2 as the number of τ and $\bar{T}$ that needs to be considered is finite (exponential in k and doubly exponential in the number of unary predicates).

Proposition B.3. *$\mathcal{A}$ has an accepting run if and only if there is a cell τ in $\hat{\mathcal{D}}^{3k}$ and a k-tuple $\bar{T}$ of sets of types of $\mathcal{D}$, such that $\pi_1(\tau)$ is contained in τ_F and $\mathcal{A}$ has a lasso compatible with $(\tau, \bar{T})$.*

Proof. One direction follows from Lemma B.1. For the converse, let us say that λ is a *loop* compatible with $(\tau, \bar{T})$ if it satisfies the properties 2,3,4,5,6 of the definition of the lasso. In the following, we show that if π, λ is a lasso compatible with τ, then the loop λ can be iterated, in some sense, obtaining an infinite accepting run of $\mathcal{A}$.

Let $\bar{x}_0, \bar{x}, \bar{y}, \bar{l}$ be the points of $\mathcal{D}^k$ witnessing the fact that π, λ is lasso compatible with $(\tau, \bar{T})$.

We construct by induction a sequence $\bar{x}_1, \bar{x}_2, \ldots$ such that $\mathcal{A}$ can go from $\bar{x}_i$ to $\bar{x}_{i+1}$ following a path of transitions equivalent to λ (in the sense that they induce the same paths in the cell automaton). This will conclude the proof of the proposition. We set $\bar{x}_1$ to $\bar{x}$ and $\bar{x}_2$ to $\bar{y}$. Assume now that we have constructed our sequence up to $\bar{x}_i$. We show how to simulate the run λ starting from $\bar{x}_i$. Let n be the length of λ and let $\bar{x} = \bar{z}_0, \bar{z}_1, \ldots, \bar{z}_n = \bar{y}$ be the sequence of configurations on λ. We say that a coordinate j of $\bar{z}_l$ is *safe* if it does not occur between $\bar{x}[j]$ and $\bar{l}[j]$. For each j, let $U_{l,j}$ be the set of unsafe coordinates of $\bar{z}_l$.

We construct by induction vectors $\bar{z}'_l$ for $0 \leq l \leq n$ such that

(1) $\bar{z}'_0 = \bar{x}_i$,

(2) $\bar{z}'_l$ is in the same cell as $\bar{z}_l$ for all $l \leq n$,

(3) $qE(\bar{x}_i, \bar{z}'_l) \geq qE(\bar{x}, \bar{z}_l)$ for all $l \leq n$.

(4) $\bar{z}'_l[j] = \bar{z}_l[j]$ if j is safe of $\bar{z}_l$,

Assume we have constructed $\bar{z}'_l$. We show how to construct $\bar{z}'_{l+1}$. For all the safe coordinates, we have no choice for $\bar{z}'_{l+1}$ in order to satisfy the last item. We aim at choosing

a sequence $\bar{z}'_{l+1}[U_{l+1,j}]$ between $\bar{x}_i[j]$ and $\bar{l}[j]$ such that

$$\langle \bar{z}'_l[U_{l,j}], \bar{z}'_{l+1,j}[U_{l+1,j}]\rangle = \langle \bar{z}_l[U_{l,j}], \bar{z}_{l+1}[U_{l+1,j}]\rangle \tag{1}$$

$$qE(\bar{z}'_{l+1}[U_{l+1,j}]) \geq qE(\bar{z}_{l+1}[U_{l+1,j}]) \tag{2}$$

Notice that all the types involved are in T_j and recall that $\bar{l}[j]$ is a T_j-limit. The existence of the desired $\bar{z}'_{l+1,j}[U_{l+1,j}]$ is then a simple consequence of the third item of our induction hypothesis and the fact that the capacity between any element of $z'_l[U_{l,j}]$ and $\bar{l}[j]$ is infinite as $\bar{l}[j]$ is a T_j-limit.

We claim that the resulting vector has the desired property. The second item follows from the fact that $\langle \bar{z}'_{l+1,j}[U_{l+1,j}]\rangle = \langle \bar{z}_{l+1}[U_{l+1,j}]\rangle$ for all j because of (1) and that the remaining coordinates are in the appropriate intervals. The third item follows from (2) and the fact that all remaining intervals are actually increasing in capacity. The first and last item are by construction. ∎

Appendix C. Proof of Theorem 4.3

In this section, we present the proof of Theorem 4.3. The theorem is split into two separate propositions stated below. Actually, some of the results below apply to more general linearly ordered structures, but the whole proof of the classification of $\mathcal{D}$-regular languages is complete only in the case $\mathcal{D} = \langle \mathbb{N}, <, c_0, \cdots, c_m\rangle$ and $\mathcal{D} = \langle \mathbb{Q}, <, c_0, \cdots, c_m\rangle$.

Proposition C.1. *Let $\mathcal{D}$ be a linearly ordered structure. Any ω-regular language can be recognized by a $\mathcal{D}$-automaton. If $\mathcal{D}^1$ has a cell α which is order-isomorphic to $\langle \mathbb{N}, <\rangle$, then any ωB-regular language can be recognized by a $\mathcal{D}$-automaton.*

Proof. It is clear that for any linearly ordered structure $\mathcal{D}$, any ω-regular language can be recognized by a $\mathcal{D}$-automaton.

For the remaining part of the proposition, assume that α is a cell in $\mathcal{D}^1$ which is order-isomorphic to $\langle \mathbb{N}, <\rangle$. We will we encode a given ωB-automaton $\mathcal{B}$ into an equivalent $\mathcal{D}$-automaton $\mathcal{A}$. To achieve that, we simulate each counter c of $\mathcal{B}$ by a variable x_c in $\mathcal{A}$ and introduce two extra variables, x_0 and x_B, representing the number zero and the guessed bound, which, during the run, will have fixed, but arbitrary values in α. An operation $c := c + 1$ of $\mathcal{B}$ is simulated by the operation $x'_c > x_c$, where x'_c represents the value of the variable x_c in the next step, and a resets are simulated by assignments to x_0. The initial region of $\mathcal{A}$ requires that $x_0, x_1, x_2, \ldots, x_n, x_B \in \alpha$. To assert that a counter c remains bounded, we require that in each transition of $\mathcal{A}$, $x_c \leq x_B$. It is easy to see that runs of $\mathcal{A}$ correspond to the runs of $\mathcal{B}$ and that we can define the acceptance condition of $\mathcal{A}$ so that the accepting runs also correspond to each other. ∎

Proposition C.2. *If $\mathcal{D} = \langle \mathbb{Q}, <, c_1, \ldots, c_m\rangle$ then any $\mathcal{D}$-automaton can be simulated by an equivalent Büchi automaton. If $\mathcal{D} = \langle \mathbb{N}, <, c_1, \ldots, c_m\rangle$ then any $\mathcal{D}$-automaton can be simulated by an equivalent ωB-automaton.*

Assume that $\mathcal{A}$ is an n-dimensional $\mathcal{D}$-automaton recognizing a language L over the alphabet A. We may extend $\mathcal{D}$ into a saturated structure as described in Theorem 3.6, as it only increases the power of $\mathcal{D}$-automata. Below we assume that $\mathcal{D}$ is the extended, saturated linearly ordered structure. In particular, $\mathcal{D}$ contains a smallest and largest element.

We will construct an automaton $\mathcal{B}$ which recognizes L. In the case of $\mathbb{Q}$ (which we will call the dense case), the resulting automaton will be a Büchi automaton, and in the case of $\mathbb{Q}$ (which we will call the discrete case) it will be an ωB-automaton.

By increasing the number of variables and states (the new states of $\mathcal{A}$ will encode an assignment of the original variables of $\mathcal{A}$ to the linearly ordered variables $x_1 < x_2 < \ldots < x_n$) in $\mathcal{A}$ we may assume that:

a) There is a strict order of the variables of $\mathcal{A}$, which doesn't change during the run, i.e. in each step, $x_1 < x_2 < \ldots < x_n$

b) Each variable x_i has a prescribed infinite cell α_i in $\mathcal{D}^1$ which it occupies (a finite cell is always a singleton containing a constant and we may assume that variables are always different than the constants)

Assume that the constants in $\mathcal{D}$ are $c_0 < c_1 < \ldots < c_l$. Observe that there is no interaction between variables which are in two disjoint intervals (c_i, c_{i+1}) and (c_j, c_{j+1}). Therefore, $\mathcal{A}$ is equivalent to a product of automata $\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_l$, such that for $i = 1, \ldots, l$, the $\mathcal{D}$-automaton $\mathcal{A}_i$ satisfies the properties $a), b)$ and moreover all its variables lay in the interval (c_{i-1}, c_i). Let $\mathcal{D}_i$ denote the linearly ordered structure $\mathcal{D}$ restricted to the interval $[c_{i-1}, c_i]$. This structure has only two constants, corresponding to its smallest and largest elements.

It therefore suffices to prove the theorem in the case where $\mathcal{D}$ has only two constants, $c_\perp < c_\top$, representing the smallest and largest elements. We call such linearly ordered structures *simple*. In the dense case, the simple linearly ordered structures are isomorphic to $\langle \mathbb{Q} \cup \{-\infty, +\infty\}, -\infty, +\infty \rangle$ and in the discrete case, they are isomorphic to $\langle \mathbb{N} \cup \{+\infty\}, 0, +\infty \rangle$.

So assume that $\mathcal{D}$ is a simple linearly ordered structure and that the automaton $\mathcal{A}$ has variables $x_1, x_2, \ldots, x_n$ which satisfy the properties $a), b)$. Let τ denote the cell in $\mathcal{D}^n$ defined by the inequalities $c_\perp < x_1 < x_2 < \ldots < x_n < c_\top$. Let C denote the set of cells δ in $\mathcal{D}^{2n}$ such that there exist tuples $\bar{x}, \bar{y} \in \tau$ with $(\bar{x}, \bar{y}) \in \delta$.

Consider a sequence $\bar{x}_0, \bar{x}_1, \bar{x}_2, \ldots$ of elements of τ. Such a sequence induces a sequence of cells $\delta_1, \delta_2, \ldots$, by letting $\delta_i = \langle \bar{x}_{i-1}, \bar{x}_i \rangle$ for $i = 1, 2, \ldots$. We call $\delta_1, \delta_2, \ldots$ an *admissible* sequence if it is induced by a sequence of elements of τ as described above.

Lemma C.3. *Let $\mathcal{D}$ be a simple linearly ordered structure. Then the set of admissible sequences is an ωB-regular language in the discrete case or an ω-regular language in the dense case.*

First we show how to conclude Proposition C.2 from the above lemma. Let Q denote the set of states of $\mathcal{A}$. Under our assumptions, the transition relation of $\mathcal{A}$ can be viewed as a mapping $\delta \colon Q \times A \times Q \to \mathcal{P}(C)$, where $\sigma \in \delta(q, a, q')$ iff $\mathcal{A}$ for any $(\bar{x}, \bar{y}) \in \sigma$, $\mathcal{A}$ allows a transition from the configuration $(q, \bar{x})$ to $(q', \bar{y})$.

A run ρ of $\mathcal{A}$ can be viewed as a word over the alphabet $A \times \tau \times Q$, where the first component describes the input letter, the second component describes the valuation of the variables and the last component describes the state of the automaton $\mathcal{A}$ during the run ρ.

It is easy to see that a word $a_1, a_2, \ldots$ is accepted by $\mathcal{A}$ if and only if there exists a sequence $\langle \rho \rangle = (q_1, a_1, \delta_1), (q_2, a_2, \delta_2), \ldots$ over the alphabet $Q \times A \times C$ such that:

- $\delta_i \in \delta(q_i, a_i, q_{i+1})$ for each $i = 0, 1, 2, \ldots$
- $\langle \rho \rangle$ satisfies the acceptance condition of $\mathcal{A}$ (which can be translated into a Büchi condition over $\langle \rho \rangle$)
- The sequence $\delta_1, \delta_2, \ldots$ is admissible.

Since the first two items can be verified by a Büchi automaton and the last one can be verified by a Büchi/ωB-automaton, we conclude that there is a Büchi/ωB-automaton recognizing precisely the language accepted by $\mathcal{A}$.

C.1. Proof of Lemma C.3.

In this section we will use a different nomenclature than in the rest of the paper, which is more suited to analyzing linear and partial orders.

All partial and linear orders in this section will be assumed to contain the largest and smallest elements, denoted $\top$ and $\bot$, respectively. If $x \leq y$ are elements of a partially ordered set U, then we denote by $[x, y]$ the set $\{z \in U : (x \leq z) \wedge (z \leq y)\}$.

Let U, U' be two partial orders. We will say that a mapping $f\colon U' \to U$ is *monotone* (resp. *strictly monotone*) if whenever $x < y$ then $f(x) \leq f(y)$ (resp. $f(x) < f(y)$). We say that f is $\top\bot$*-preserving* if $f(\bot) = \bot$, $f(\top) = \top$. Note that a strictly monotone map might not be one-to-one, since it can merge incomparable elements. We call a strictly monotone, $\top\bot$-preserving map an *embedding*. We say that U' is *embeddable* in U if there exists an embedding $f\colon U' \to U$.

Figure 1 should guide the reader in an easier understanding of the following definitions.

We fix a finite linear order $\mathcal{P}$, called the *pattern*. Let its elements be $\bot, p_1, p_2, \ldots, p_n, \top$, in increasing order. The *slots* in $\mathcal{P}$ are the elements:

$$\bot, (\bot, p_1), p_1, (p_1, p_2), p_2, (p_2, p_3), \ldots, (p_{n-1}, p_n), p_n, (p_n, \top), \top.$$

The slots are linearly ordered according to the order of appearance in the above list. We denote the set of slots by $slots(\mathcal{P})$. Clearly, $\mathcal{P} \subseteq slots(\mathcal{P})$.

A *diagram* is a monotone, $\top\bot$-preserving map $d\colon \mathcal{P} \to slots(\mathcal{P})$, such that no two elements are mapped to the same element of $\mathcal{P} \subseteq slots(\mathcal{P})$. Obviously, there are only finitely many diagrams.

Remark C.4. Diagrams over $\mathcal{P}$ are equivalent to cells in C, using the nomenclature of the previous section. Indeed, a diagram d corresponds to a cell in $\mathcal{D}^{2n}$ described by

$$\begin{gathered} c_\bot < x_1 < x_2 < \ldots < x_n < c_\top \\ c_\bot < y_1 < y_2 < \ldots < y_n < c_\top \\ y_i = x_j \quad \text{if} \quad d(p_i) = p_j \\ x_j < y_i < x_{j+1} \quad \text{if} \quad d(p_i) = (p_j, p_{j+1}) \end{gathered}$$

The cell above is clearly an element of the set C. Moreover, any cell $\delta \in C$ corresponds to precisely one diagram over $\mathcal{P}$.

Consider now a finite or infinite sequence of diagrams $\Delta = d_1, d_2, \ldots$ (see Figure C.1). This sequence induces a partially ordered set $\mathcal{P}_\Delta$ described as follows:

- The elements of $\mathcal{P}_\Delta$ are maximal (which cannot be expanded in any direction), finite or infinite, sequences $p_i, p_{i+1}, p_{i+2}, \ldots$ of elements of $slots(\mathcal{P})$ such that $d_j(p_{j+1}) = p_j$ for $j \geq i$. Such sequences are called *threads*, denoted π, ρ, etc. Note that only first element of thread might be an element of $slots(\mathcal{P}) \setminus \mathcal{P}$, since otherwise $d_j(p_{j+1})$ is undefined. If π is a thread and $j \in \mathbb{N}$, the item with index j in π is denoted π_j (if it is defined). For an element $x \in \mathcal{P}$ and $i \in \mathbb{N}$, let $\rho_{x,i}$ denote the unique thread ρ in $\mathcal{P}_\Delta$ such that $\rho_i = x$. Note that there is a thread (denoted $\bot$) which is constantly equal to $\bot$ and a thread (denoted $\top$) which is constantly equal to $\top$

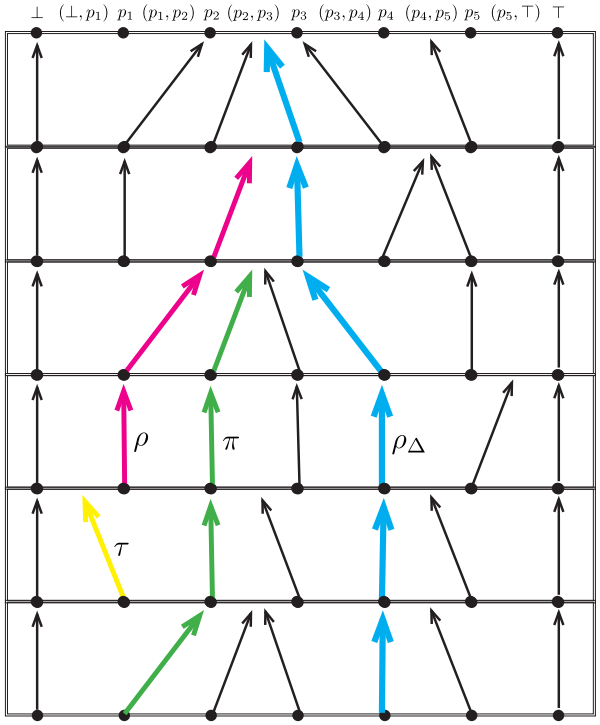

Figure 1: A sequence Δ of diagrams. Four threads are outlined: $\tau < \rho < \pi < \rho_\Delta$

- For two distinct threads π, ρ, we write that $\pi <_0 \rho$ if there is a $j \in \mathbb{N}$ such that π_j and ρ_j are both defined and $\pi_j < \rho_j$. Note that that for any thread $\pi \in \mathcal{P}_\Delta$, $\bot <_0 \pi$ and $\pi <_0 \top$.
- The relation $<_0$ is not necessarily transitive, so we will consider its transitive closure instead, denoted $<$. It is not difficult to check that it defines a partial order on $\mathcal{P}_\Delta$.

We say that Δ is *embeddable* in U if $\mathcal{P}_\Delta$ is. Note that any finite sequence Δ is embeddable in U – it suffices to extend the partial order on $\mathcal{P}_\Delta$ into a linear order and notice that any finite linear order embeds into any infinite linear order. Our aim is to characterize those infinite sequences of diagrams which are embeddable in U.

Proposition C.5. *All sequences Δ of diagrams over a finite pattern $\mathcal{P}$ are embeddable in $\langle \mathbb{Q} \cup \{-\infty, +\infty\}, <, -\infty, +\infty \rangle$. The set of sequences Δ of diagrams over a finite pattern $\mathcal{P}$ which are embeddable in $\langle \mathbb{N} \cup \{\infty\}, <, 0, \infty \rangle$ can be recognized by an ωB-automaton.*

Remark C.6. The above proposition implies Lemma C.3. Indeed, as we noted in Remark C.4, sequences of elements of C correspond to sequences of diagrams over $\mathcal{P}$ and below

we notice that admissible sequences of elements of C correspond to embeddable sequences of diagrams.

An admissible sequence Δ in C^ω is induced by some sequence of values $\bar{x}_0, \bar{x}_1, \ldots \in \mathcal{D}^n$. This sequence of values defines the embedding of $\mathcal{P}_\Delta$ into U, which maps the thread $\rho_{p_i,n}$ to the i^{th} element of $\bar{x}_n$. That this mapping is well defined (doesn't depend on the chosen element of the thread) follows from the assumption that the sequence $\bar{x}_0, \bar{x}_1, \ldots$ induces the sequence of cells Δ.

Conversely, an embedding $f\colon \mathcal{P}_\Delta \to U$ defines a sequence $\bar{x}_0, \bar{x}_1, \ldots$, by setting the i^{th} element of $\bar{x}_n$ to $f(\rho_{p_i,n})$. The resulting sequence induces the sequence of cells Δ.

Now we proceed to the proof of Proposition C.5. The case of $\mathbb{Q}$ is easy, since any countable partially ordered set U' can be embedded in $\mathbb{Q}$. To notice that, enumerate the elements of U' in a sequence, and define the embedding f inductively for all the elements of U'. Since $\mathbb{Q}$ is dense, the induction can proceed indefinitely.

The rest of this section is devoted to the proof of the discrete case. Hence, in what follows, we assume that $U = \langle \mathbb{N} \cup \{\infty\}, <, 0, \infty \rangle$.

Let Δ be an infinite sequence of diagrams. Note that there can be only finitely many threads in $\mathcal{P}_\Delta$ which are infinite. This is because for any n, $\pi \mapsto \pi_n$ is an injective mapping from infinite threads to the finite set $\mathcal{P}$. Moreover, any two infinite threads π, ρ are comparable with respect to the relation $<$. Let ρ_Δ denote the largest infinite thread in $\mathcal{P}_\Delta$ which is smaller than $\top$.

We denote the partially ordered set $[\bot, \rho_\Delta]$ by $\mathcal{C}_\Delta$ and the the partially ordered set $[\rho_\Delta, \top]$ by $\mathcal{R}_\Delta$. Clearly, $\mathcal{C}_\Delta \leq \mathcal{R}_\Delta$. Note that there might be some elements in $\mathcal{P}_\Delta \backslash (\mathcal{C}_\Delta \cup \mathcal{R}_\Delta)$, i.e. threads which are incomparable with ρ_Δ, but all these threads have length bounded by the first position of ρ_Δ.

In what follows, a *chain* is a sequence of comparable elements.

Lemma C.7. *Let Δ be an infinite sequence of patterns over $\mathcal{P}$ and $U = \langle \mathbb{N} \cup \{\infty\}, <, 0, \infty \rangle$. Then Δ is embeddable in U iff the following items hold:*

- *there exists a number $B \in \mathbb{N}$ such that the chains in $\mathcal{C}_\Delta$ have length bounded by B*
- *there exist no infinite descdending chains in $\mathcal{R}_\Delta$*

Proof. First, we prove the left-to-right implication. Let f be an embedding of $\mathcal{P}_\Delta$ into U. Under this embedding, $\mathcal{C}_\Delta$ is mapped into an interval $[0, B]$ with $B < \infty$, since $0 = f(\bot)$ and $B = f(\rho_\Delta) < f(\top) = \infty$. Then $\mathcal{R}_\Delta$ is mapped to $[B, \infty]$. Moreover, f maps chains to chains, and the interval $[0, B]$ has no chains longer than B and the interval $[B, \infty]$ has no infinite descending chains. This finishes the proof of the first implication.

Now we will prove the right-to-left implication. We will first embed $\mathcal{C}_\Delta$ into the finite interval $[0, B]$. For $x \in \mathcal{C}_\Delta$, let $f(x)$ denote the length of the longest descending sequence $x_1 > x_2 > x_3 \ldots$ with $x_1 = x$. Since all chains have length bounded by B, we have that $1 \leq f(x) \leq B$. Moreover, if $x > x'$ in $\mathcal{C}_\Delta$ then $f(x) \geq f(x') + 1$, since any descending chain starting in x' can be extended to a descending chain starting in x which is one element longer. Therefore, $f\colon \mathcal{C}_\Delta \to [0, B]$ is an embedding.

In a similar fashion, we can embed $\mathcal{R}_\Delta$ in $[0, \infty]$. Indeed, we can consider the function f defined just as above, but over the domain $\mathcal{R}_\Delta$. This time, however, it is not obvious that it is well defined – even though there is no infinite descending chain, a priori, there could be arbitrarily long finite descending chains.

We therefore use another definition of f. For a given $x \in \mathcal{R}_\Delta \setminus \{\top\}$, consider the tree t_x whose set of vertices is $[\rho_\Delta, x]$, and y' is a child of y iff $y' <_0 y$. In this tree, each element has a finite degree – this is a consequence of the definition of $\mathcal{R}_\Delta$, since there is no infinite thread in $\mathcal{R}_\Delta$ except for $\top$ and ρ_Δ, and ρ_Δ is a minimal element. If the tree t_x contains arbitrarily long paths, from König's lemma we conclude that it must contain an infinite path. This contradicts the assumption that $\mathcal{R}_\Delta$ has no infinite descending chain. Therefore, t_x has finite height, and we define $f(x)$ as this height. By extending to $\top$ by $f(\top) = \infty$, we obtain an embedding $f \colon \mathcal{R}_\Delta \to [0, \infty]$.

Let $f_\mathcal{C} \colon \mathcal{C}_\Delta \to [0, B]$ and $f_\mathcal{R} \colon \mathcal{R}_\Delta \to [0, \infty]$ denote the above defined embeddings. Recall that the set $\mathcal{P}_\Delta \setminus (\mathcal{C}_\Delta \cup \mathcal{R}_\Delta)$ is finite, and therefore there is some embedding f_0 which maps it to some finite interval $[1, B']$.

Altogether, $f_\mathcal{C}, f_\mathcal{R}$ and f_0 can be merged into one embedding f of $\mathcal{P}_\Delta$ into $[0, \infty]$, defined as follows.

$$f(x) = \begin{cases} f_\mathcal{C}(x) & \text{if } x \leq \rho \\ f_\mathcal{R}(x) + B & \text{if } x > \rho \\ f_0(x) & \text{if } x \text{ and } \rho \text{ are incomparable} \end{cases}$$

■

Now we are ready to sketch the proof of Proposition C.5.

Sketch. We construct a nondeterministic ωB-automaton $\mathcal{B}$ which, for a given infinite input sequence of diagrams Δ:

(1) nondeterministically guesses the emergence of the maximal infinite thread ρ_Δ and continues tracing it
(2) computes in its counters the lengths of the increasing and decreasing chains contained in $\mathcal{C}_\Delta$
(3) verifies that $\mathcal{R}_\Delta$ contains no infinite descending sequence

It can be easily seen that the item (3) above is equivalent to the second condition stated in Lemma C.7. It can be verified by using a Büchi condition. According to item (2), the first condition of Lemma C.7 is equivalent to $\mathcal{B}$ having bounded counters.

Therefore, the ωB-automaton $\mathcal{B}$ accepts precisely those sequences Δ which are embeddable in $\langle \mathbb{N} \cup \{\infty\}, <, 0, \infty \rangle$. ■

Appendix D. Proof of Theorem 6.3

The proof is a reduction to Theorem 3.10.

Fix a $(\mathcal{D}, \sigma)$-automaton $\mathcal{A}$ of dimension k.

From Theorem 3.6 we can assume without loss of generality that $\mathcal{D}$ is saturated.

From $\mathcal{A}$ we construct an $\mathcal{D}$-automaton $\mathcal{B}$ such that there is a finite database M and a finite word w accepted by $\mathcal{A}$ over M iff $\mathcal{B}$ accepts w. This will complete the proof of the theorem by Theorem 3.10.

In order to simplify the presentation of the construction, $\mathcal{B}$ will contain states but we have seen already that states could be simulated using constants. Hence the transitions of $\mathcal{B}$ are of the form (q, a, τ, q') stating that when in state q, in configuration $\bar{x}$, and reading letter a, the automata may move to state q' and configuration $\bar{y}$ assuming $(\bar{x}, \bar{y}) \in \tau$. A run of $\mathcal{B}$ is then accepting if the final configuration in τ_F and $\mathcal{B}$ in a set $q \in F$ where F is the set of accepting states of $\mathcal{B}$.

The idea of the construction of $\mathcal{B}$ is that it simulates $\mathcal{A}$ in order to maintain the right configuration tuple of $\mathcal{D}^k$ while using its states to store finitely many consistency conditions ensuring the existence of a finite database M making the whole run valid for $\mathcal{A}$. It turns out that for saturated $\mathcal{D}$, it is roughly enough to remember the tuples formed with the constants of $\mathcal{D}$ that must or must not be part of the database together with the positions of the current configuration that are equal to a constant of $\mathcal{D}$.

Let C be the constants of $\mathcal{D}$ and $\sigma(C)$ be the set of atoms or negated atoms that can be built from the relation symbols in σ and the constants of $\mathcal{D}$. Note that the cardinality of $\sigma(C)$ is $O(|\sigma||C|^r)$ where r is the maximal arity of a relation in σ.

The set Q of states of $\mathcal{B}$ is $2^{\sigma(C)} \times \{C, \perp\}^k \times \Psi$, where Ψ is the set of formulas occurring in the transition function of $\mathcal{A}$. The reader should interpret a state $q \in Q$ as, for the first part, a set of atoms or negated atoms that must be valid in M, for the middle part, a function which, for each $i \leq k$, assigns $c \in C$ if the element x_i of the current configuration $\bar{x}$ is equal to c or, $\perp$ if x_i does not belong to C and, for the last part, the database query from the transition of $\mathcal{A}$ that is being simulated. Note that the size of Q is bounded by $2^{|\sigma||C|^r}|C|^k|\mathcal{A}|$, in particular it can be doubly exponential in the size of $\mathcal{A}$ if r is not fixed, a problem that was already present in [DHPV09].

The transitions of $\mathcal{B}$ are defined as follows. If $(\tau, a, \varphi) \in \delta$ then the tuple (p, a, τ, q) is a transition of $\mathcal{B}$ for any state $p = (S, f, \varphi)$ and $q = (T, g, \phi)$ such that:

(1) For all $i \leq k$, if either $y_i = c \in \tau$ or $\exists j \ f(j) = c \wedge x_j = y_i \in \tau$ then $g(i) = c$ else $g(i) = \perp$.
(2) T is S augmented with all the atoms $R(c_{i_1}, \ldots, c_{i_r})$, or its negation, such that $R(z_{i_1}, \ldots, z_{i_r})$ is an atom of φ and $f(i_j) = c_{i_j}$ if $z_{i_j} = x_{i_j}$ and $g(i_j) = c_{i_j}$ if $z_{i_j} = y_{i_j}$.

The initial region of $\mathcal{B}$ is the one of $\mathcal{A}$. The accepting region of $\mathcal{B}$ are the pairs (q, τ_F) where $q = (S, g, \phi)$ with S consistent and τ_F is the accepting region of $\mathcal{A}$.

The following lemma, showing that the construction is complete, is obvious by construction.

Lemma D.1. *If $\mathcal{A}$ accepts a word w over a database M then $\mathcal{B}$ accepts w.*

We now show that the construction is sound.

Lemma D.2. *If $\mathcal{B}$ accepts a word w then there exists a database M such that $\mathcal{A}$ accepts the word w over M.*

Proof. Let ρ be an accepting run of $\mathcal{B}$ for w. Let n be the length of w and assume that ρ ends in an accepting state $q = (S, g, \phi)$ in a configuration inside the accepting cell τ_F. We construct a database M such that $\mathcal{A}$ has an accepting run ρ' for w on M. We construct ρ' such that whenever ρ perform a transition (p, a, τ, p') with $p = (S, f, \varphi)$, then ρ' performs a transition (τ, a, φ).

This is done by induction on w. Assume $w = w'a$. Let τ be the cell of the configuration reached by ρ after reading w' and $q' = (S', g', \varphi')$ be the corresponding state. Because S is consistent and by construction $S' \subseteq S$, S' is also consistent. Hence, by induction we have a finite database M' such that $\mathcal{A}$ has a run ρ' ending in τ for w' over M'.

Let X be C plus all the elements of $\mathcal{D}$ occurring in the configurations of ρ'. The size of X is bounded by $k^{n-1} + |C|$. Let (q', a, τ, q) be the last transition taken by ρ.

Let $\bar{x}$ be the configuration reached at the end of ρ'. By Proposition 3.4 we can assume without loss of generality that the potential of $\bar{x}$ is greater than $(k^{n-1} + |C|)(k + 1)$.

It is now sufficient to show that from $\bar{x}$ we can choose $\bar{y} \in \tau_F$ and an extension M of M' such that the transition (q', a, τ, q) can be performed.

This can be done because of the choice of the potential of $\bar{x}$, all the intervals are big enough so that the new elements that needs to be introduced can be chosen distinct from the ones already present in the domain of M'. Hence by taking M as M' plus the new tuples necessary to make ϕ true, we are done. ∎

This concludes the proof of the theorem. ∎

Appendix E. Proof of Theorem 6.3

The proof is similar to the proof of Theorem 4.2. From $\mathcal{A}$ we construct an $(\mathcal{D}, \sigma)$-automaton $\hat{\mathcal{A}}$ which encodes the runs of $\mathcal{A}$. As for Theorem 4.2, $\hat{\mathcal{A}}$ will only simulate runs of $\mathcal{A}$ of a special form.

Recall Lemma B.1. We will need to extend the notion of a τ-cyclic run ρ to include the database M. On top of the conditions $i), ii)$ of the definition of a τ-cyclic run, we add the following extra requirements:

iii) There is a set T such that the set of transitions taken by $\mathcal{A}$ between $\bar{x}_i$ and $\bar{x}_{i+1}$ is exactly T.

iv) For every strictly monotone sequence as in condition $ii)$, there is no element of M which lies between the first element of the sequence and its limit.

As for Lemma B.1, the proof of this normal form follows from a simple Ramsey argument.

The construction of $\hat{\mathcal{B}}$ then roughly follows the same lines as for the proof of Theorem 4.2. We only briefly recall the main steps of the construction here, the reader is referred to the proof of Theorem 4.2 for the missing details. We fix a cell τ of $\hat{\mathcal{D}}^{3k}$.

As in the proof of Lemma B.2, the automaton $\mathcal{B}$ has dimension $5k$. The behavior of $\mathcal{B}$ is the same as the one of the automaton constructed in the proof of Lemma B.2 plus some additional conditions concerning the database accesses, in the second stage of the run of $\mathcal{B}$. There, the automaton $\mathcal{B}$ tests for the existence of a run from $(\bar{x}_0, \bar{x}, \bar{y}, \bar{l}, \bar{x})$ to $(\bar{x}_0, \bar{x}, \bar{y}, \bar{l}, \bar{y})$ such that $\mathcal{A}$ has a run from $\bar{x}$ to $\bar{y}$ and such that $(\bar{x}, \bar{y}, \bar{l}) \in \tau$. Moreover, $\mathcal{B}$ enforces that whenever $x_j < y_j$ is a constraint of τ for some $j \leq k$, then, during the simulation of $\mathcal{A}$ (performed on the last k coordinates of $\hat{\mathcal{D}}^{5k}$), no element occurring between the j^{th} coordinate of $\bar{x}$ and the corresponding coordinate of $\bar{l}$ occurs in the database, i.e. is never used in a positive atom of some formula accessing the database. We do similarly for decreasing sequences.

From the Ramsey argument above we deduce that whenever $\mathcal{A}$ has an infinite accepting run over some finite database M then $\mathcal{B}$ has a finite accepting run over the finite database M. The converse is shown in the same way than for the proof of Proposition B.3.

Given a finite run $\hat{\pi}$ of $\mathcal{B}$ over a finite database M, we construct in the same way as in the proof of Proposition B.3 an infinite sequence $\bar{x}_1, \bar{x}_2, \ldots$ of tuples such that $\mathcal{A}$ can go from $\bar{x}_i$ to $\bar{x}_{i+1}$ following a path of transitions equivalent to λ (in the sense that they induce the same paths in the cell automaton).

We show by induction that $\mathcal{A}$ can actually go from $\bar{x}_i$ to $\bar{x}_{i+1}$ via $\hat{\pi}$ over M.

To see this we simply reuse the data values occurring in $\hat{\pi}$ whenever this is possible. This is not possible if we have $x_j < y_j$ (or $x_j > y_j$) in τ and the data value is between the j^{th}

coordinate of $\bar{x}$ and its limit. But in this case we know that this data value cannot appear in the database. In any other cases the data value coming from $\hat{\pi}$ can always be reused. ∎