Reachability and Bounded Emptiness Problems of Constraint Automata with Prefix, Suffix and Infix

Jakub Michaliszyn
University of Wrocław, Poland

Jan Otop
University of Wrocław, Poland

Piotr Wieczorek
University of Wrocław, Poland

Abstract

We study constraint automata, which are finite-state automata over infinite alphabets consisting of tuples of words. A constraint automaton can compare the words of the consecutive tuples using Boolean combinations of the relations prefix, suffix, infix and equality.

First, we show that the reachability problem of such automata is PSPACE-complete. Second, we study automata over infinite sequences with Büchi conditions. We show that the problem: given a constraint automaton, is there a bound B and a sequence of tuples of words of length bounded by B, which is accepted by the automaton, is also PSPACE-complete. These results contribute towards solving the long-standing open problem of the decidability of the emptiness problem for constraint automata, in which the words can have arbitrary lengths.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases constraint automata, emptiness problem

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2023.3

Funding This work was supported by the National Science Centre (NCN), Poland under grant 2020/39/B/ST6/00521.

1 Introduction

Logics and automata over *data values*, i.e., values from an *infinite* domain, have applications in formal verification, automated reasoning, databases and others [7, 6, 4, 10, 2, 1, 19, 22].

A notable example of a formalism for data values is *constraint linear temporal logic (CLTL)* [7, 6, 4]. CLTL formulas are defined w.r.t. a relational structure, e.g. $(\mathbb{N}, =)$ or $(\mathbb{Z}, <)$; the variables in formulas range over the domain of the structure. *Constraints* are atomic formulas defined using variables and symbols from the structure. CLTL formulas combine such constraints with LTL modalities and Boolean connectives, i.e., CLTL is LTL in which propositional variables are replaced with constraints. Every CLTL formula defines a *data language*, which is a set of infinite sequences of data values.

Constraint automata is the automata-based formalism accompanying CLTL [5, 11, 7]. Again, they are parameterized by a structure, which can have an infinite domain. A constraint automaton is a Büchi automaton, in which transitions are labeled with constraints; an automaton can take a transition only if the current and the next data values satisfy the constraint labeling this transition. The satisfiability problem for CLTL can be reduced to the non-emptiness of constraint automata as in the LTL case. Furthermore, constraint automata can use the equality constraints to store a data value for future use, which makes them closely related to *register automata* [14, 22, 8].

There are several results for CLTL and constraint automata for specific structures (see survey [9]) but also for some unified classes, like linear orders [26, 16, 4, 3, 10].

We are interested in data values being strings (A^*) over a finite alphabet A. The structures, considered in the context of strings, may be equipped with basic relations on strings like *prefix*, *suffix* or *subsequence*. Motivations come from the fact that although temporal logics with constraints over integers can be helpful in formal analysis of programs with counters, in order to analyse pushdown systems we need constraints over strings and the prefix relation. The most typical of prefix and suffix usage is for queues: adding to a queue (represented as a word) corresponds to stating that the old word is a prefix of the new one, whereas removing from a queue corresponds to stating that the new word is a suffix of the old one, so both prefix and suffix are important (and infix can be expressed as a combination of prefix and suffix). The reachability problem for queue automata (a PDA with a queue instead of a stack) is undecidable, therefore analyzing systems with a queue is a challenging task.

There has been a large body of work on various problems involving strings especially for multiple variants of first-order logic (FO) [15, 12, 18]. These results have implications for CLTL and constraint automata as pointed out in [23]. The undecidability of the satisfiability problem for CLTL over structure $\mathbb{A} = (A^*, \leq_{\mathrm{sub}}, (= w)_{w\in\Sigma^*})$, where A is a finite alphabet and $\leq_{\mathrm{sub}}$ is the subsequence order, follows from undecidability of the satisfiability problem for Σ_1-fragment of FO logic over $\mathbb{A}$ [12].

The satisfiability problem for CLTL over $(A^*, \leq_{\mathrm{p}}, =, (= w)_{w\in\Sigma^*})$, where $\leq_{\mathrm{p}}$ is prefix order, is PSPACE-complete [6]. Note that words with the prefix order alone form the structure isomorphic to an infinite tree with descendant/ancestor relations. However, it was shown in [3] that the known unified technique, involving the "existence of homomorphisms is decidable"-property, for satisfiability results of branching-time logics (like CTL* or ECTL*) [4] with integer constraints cannot be used to resolve the satisfiability status of temporal logics with constraints over trees. This in turn shows the difficulty of the result from [6]. In the automata approach the emptiness problem for constraint automata is PSPACE-complete when the relation is the infinitely branching infinite order tree [16]. Because of the symmetry, the same complexity follows in the case when prefix order is replaced with suffix order.

Once the satisfiability for CLTL with the prefix (or the suffix) order alone is answered, a natural question arises: what happens if we study CLTL over the structure A^* equipped with both of them? This question has been asked by Demri and Deters [6]. Having both prefix and suffix allows for checking properties depending on both ends of strings like in Example 2 provided at the end of Section 2.2. In this work we also explicitly include the infix relation as well as a form of negation for each of the three relations (i.e., *incomparable w.r.t. prefix* or *suffix* or *infix* respectively). Although infix alone is definable with prefix and suffix using an additional variable, it is not the case for its negation. It is an important part of our results.

Peteler and Quaas [23] studied the emptiness problem for constraint automata over the prefix and the suffix orders. They exemplified that FO logic with the prefix order alone is decidable [25] while FO logic with the prefix order and the suffix order is undecidable (this follows from the undecidability result for the FO theory for the substring (infix) orders [17], and the fact that the substring order is FO-definable using prefix and suffix). On the positive side, as noted in [6], the Σ_1-fragment of FO logic is decidable for finite strings over a finite alphabet. The proof uses an algorithm based on the word equation approach [24, 21, 13].

In the same paper, Peteler and Quaas proved that it is decidable in NL when the automaton uses only a single variable that ranges over finite strings. The strings can be over a finite or countably infinite alphabet. Their proof proceeds by reduction to reachability queries on the finite graph underlying the automaton. They show that their technique works in the presence of equality tests with the empty string (similar to a zero test in one-counter automata). The result implies PSpace-completeness of the satisfiability problem for `CLTL` over a single variable. In contrast, in our work we study the emptiness problem for constraint automata that can use many variables over finite strings, but the string lengths are bounded.

Our contribution

We study constraint automata with the *prefix*, *infix*, *suffix* and equality ($=$) relations. Our crucial technical contribution is Lemma 4 in Section 3, which provides a model-theoretic characterisation of the satisfiability of a maximal set of constraints expressed using the prefix, suffix and infix relations (without constants) over some (countable, possibly infinite) set of variables. It says that such a set of constraints is satisfiable if variables with the prefix (resp., the suffix) relation form a forest, and variables with the infix relation form a finitary partial order, i.e., every variable has finitely many predecessors.

We employ this lemma to show that the reachability problem of constraint automata is PSpace-complete. To do so, we first introduce *type-tracking* automata, which store in the state the type of the processed input tuple, i.e., the information on the relation between all words in the input tuple. In such automata every path corresponds to some (pre)run over some sequence. For a constraint automaton, the type-tracking automaton may have exponential size, but it can be constructed on-the-fly and there is no need to store it in memory. Therefore, the reachability problem for unrestricted constraint automata can be solved in PSpace. We also prove the matching lower bound.

For constraint automata over infinite sequences with Büchi conditions, we define the *bounded emptiness problem* as follows: given a constraint automaton, is there a bound B such that some sequence accepted by the automaton and all words in all tuples of that sequence have length bounded by B? We show that this problem is PSpace-complete as well. To do so, we first prove that if there is a bounded sequence accepted by the automaton, there is an ultimately-periodic sequence accepted by the automaton. Next, we establish a condition for a cycle in such an ultimately-periodic sequence. Finally, we show how to check the existence of such a cycle in polynomial space. The corresponding lower bound can be obtained from the lower bound for the reachability problem.

2 Preliminaries

2.1 Relations and constraints

We assume Σ to be a finite *alphabet*, whose elements are *letters*. A *word* is a finite sequence of letters; ϵ denotes the empty word. For words w, v, the word wv is the concatenation of w and v. An n-tuple is a tuple consisting of n words. An n-sequence is a sequence of n-tuples. This is illustrated in Figure 1.

We say that a word w is

- a (strict) *prefix* of v, denoted as $w \sqsubset_P v$ if there is a non-empty word t such that $wt = v$;
- a (strict) *suffix* of v, denoted as $w \sqsubset_S v$, if there is a non-empty word t such that $tw = v$;
- a (strict) *infix* of v, denoted as $w \sqsubset_I v$ if there are words t, t', at least one of them non-empty, such that $twt' = v$.

We say that two words v, w are *incomparable with respect to the prefix order*, denoted as $v \perp_P w$ if none of the following holds: $v = w$, $w \sqsubset_P v$, $v \sqsubset_P w$. The $v \perp_S w$ and $v \perp_I w$ relations for suffix and infix are defined in a similar way.

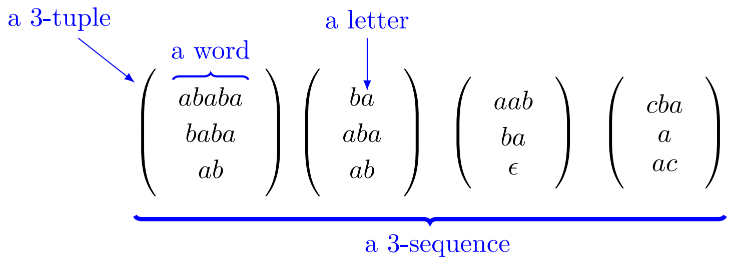

Figure 1 An example of a 3-sequence.

An n-*constraint* (over $\{x_1, \ldots, x_n\}$) is a set consisting of atoms of the form $x \oplus y$, where $x, y \in \{x_1, \ldots, x_n\}$ are variables and $\oplus \in \{<_P, <_S, <_I, \perp_P, \perp_S, \perp_I, =\}$ is a relation symbol. Given an n-tuple of words $\vec{w} = (w_1, \ldots, w_n)$, we define the interpretation $\mathrm{I}_{\vec{w}}$ such that for each variable x_i we have $\mathrm{I}_{\vec{w}}(x_i) = w_i$, and for relational symbols we have $I(<_P) = \sqsubset_P$, $\mathrm{I}_{\vec{w}}(<_S) = \sqsubset_S$, $\mathrm{I}_{\vec{w}}(<_I) = \sqsubset_I$, $\mathrm{I}_{\vec{w}}(\perp_P) = \perp_P$, $\mathrm{I}_{\vec{w}}(\perp_S) = \perp_S$, $\mathrm{I}_{\vec{w}}(\perp_I) = \perp_I$, and $\mathrm{I}_{\vec{w}}(=)$ is $=$. An n-constraint γ is *satisfied* by $\vec{w} = (w_1, \ldots, w_n)$, denoted as $\vec{w} \models \gamma$, if for every atom $x \oplus y$ from γ the expression over words $I_{\vec{w}}(x \oplus y)$ is true. An n-constraint γ is *satisfiable* if there exists an n-tuple of words $\vec{w}$ satisfying γ.

▶ **Example 1.** Consider the 6-constraint $\gamma = \{x_3 = x'_3, x_1 \perp_P x'_1, x'_1 <_S x_1\}$ over variables $x_1, x_2, x_3, x'_1, x'_2, x'_3$. Let w_1, w_2, w_3, w_4 be the 3-sequence presented in Figure 1. We will write (w_i, w_j) to denote a 6-tuple of words containing first the words of w_i, and then the words of w_j.

Then, $(w_1, w_2) \models \gamma$ holds, because $ab = ab, ababa \perp_P ba$ and $ba <_S ababa$. On the other hand, $(w_2, w_3) \models \gamma$ does not hold because $ab \neq \epsilon$. ◀

An ordered set $(X, <)$ is *finitary* if for every $t \in X$, the set $\{s \in X \mid s < t\}$ is finite. An ordered set $(X, <)$ is a (finitary) *tree* if for every $t \in X$, the set $\{s \in X \mid s < t\}$ is finite and totally ordered w.r.t. $<$. A (finitary) *forest* is a disjoint union of trees.

2.2 Constraint automata and their semantics

A (non-deterministic) n-*constraint automaton* $\mathcal{A} = (Q, Q_0, Q_F, \delta)$ is an automaton which processes tuples of finite words, i.e., $(\Sigma^*)^n$. Such an automaton consists of:

- A finite set of states Q and its subsets: initial states Q_0 and final states Q_F.
- A transition relation $\delta \subseteq Q \times \Gamma \times Q$, where Γ is the set of all satisfiable $2n$-constraints over $\{x_1, \ldots, x_n, x'_1, \ldots, x'_n\}$.

The *size* of an n-constraint automaton is the number of elements of Q and δ.

The semantics of n-constraint automata is defined over n-sequences. Intuitively, an automaton starts in an initial state with the first n-tuple, and then changes the state according to the following n-tuples and the transition relation. In transitions, unprimed variables $x_1, \ldots, x_n$ are interpreted as words at the origin and the primed variables are interpreted as words at the destination. The formal description follows below.

We will consider two semantics of constraint automata: over finite and infinite sequences. Let $\mathcal{A} = (Q, Q_0, Q_F, \delta)$ be an n-constraint automaton.

Constraint automata over finite sequences. A *partial run* of $\mathcal{A}$ over a finite sequence $w_0, \ldots, w_m$, with $m \geq 1$, is a sequence of states $q_0, \ldots, q_m$ where for each $i < m$ there is γ such that $(q_i, \gamma, q_{i+1}) \in \delta$ and $(w_i, w_{i+1}) \models \gamma$. A *run* is a special case of a partial run that starts in an initial state, i.e., $q_0 \in Q_0$. A run is *accepting* if the last state $q_m \in Q_F$.

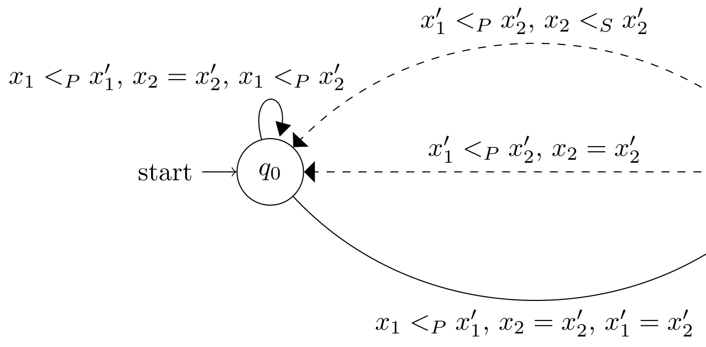
$x'_1 <_P x'_2,\ x_2 <_S x'_2$
$x_1 <_P x'_1,\ x_2 = x'_2,\ x_1 <_P x'_2$
start
q_0
$x'_1 <_P x'_2,\ x_2 = x'_2$
q_a
$x_1 <_P x'_1,\ x_2 = x'_2,\ x'_1 = x'_2$

Figure 2 An example of a 2-constraint automaton.

Constraint automata over infinite sequences. A *partial run* of $\mathcal{A}$ over an infinite sequence $w_0, w_1, \ldots$ is an infinite sequence of states $q_0, q_1, \ldots$ such that every finite prefix is a partial run over the corresponding prefix of $w_0, w_1, \ldots$. As before, a partial run is a *run* if it starts in an initial state. An infinite run is *accepting* if there is a state $q \in Q_F$ such that for infinitely many j we have $q_j = q$ (Büchi condition).

In both cases, we say that the automaton accepts a sequence s if there is an accepting run on it. The *language* of an automaton is the set of sequences it accepts. Two automata are equivalent if their languages are the same.

▶ **Example 2.** Consider the 2-constraint automaton depicted at Figure 2 without the dashed edges. This automaton, considered over finite sequences, accepts sequences $(w_1, v_1) \ldots (w_s, v_s)$ such that $w_s = v_s$, for all i, j we have $v_i = v_j$, for all $i < s$ we have $w_i <_P v_i$, $w_i <_P w_{i+1}$. This can model a process that starts from some list of tasks to accomplish (in a specific order) v and maintains the current list of finished tasks as w. The automaton accepts once the list is completed.

The automaton depicted in Figure 2 with the dashed edges can be considered on infinite sequences. In this case, the procedure described above is repeated infinitely often: once a current list is completed, the process starts over with a new list; in this example, the new list is the previous list possibly extended with new tasks at the beginning. The "new list" can be empty or non-empty, as no constraints check whether a word is empty. Note also how we have used nondeterminism to express the disjunction $x_2 = x'_2$ or $x_2 <_S x'_2$ during the return.

2.3 Decision problems

The *emptiness* problem for constraint automata over finite (resp., infinite) sequences is the question: given a n-constraint automaton, is there a finite (resp., an infinite) n-sequence accepted by this automaton?

For finite sequences, we consider a (slightly) more general question, that of *reachability*: given an n-constraint automaton $\mathcal{A}$ and its two states s, t, is there a finite n-sequence and a partial run over that sequence starting in s and ending in t? Even though the reachability problem and the emptiness problem over finite sequences are mutually reducible, the reachability problem is often more convenient to apply.

We say that an infinite n-sequence σ is bounded if there is a bound B, such that in every n-tuple $\sigma[i]$, words have length bounded by B. Note that finite and ultimately periodic sequences are bounded. The *bounded emptiness* problem is as follows: given a n-constraint automaton $\mathcal{A}$, is there an infinite bounded n-sequence accepted by $\mathcal{A}$?

3 Constraints and their Satisfaction

In this section, we study satisfiability of n-constraints. First, we give a characterization of satisfiability of constraints based on the shape of constraints (Lemma 4), which gives insight into the expressive power of constraints. While n-constraints are over finitely many variables, the characterization of Lemma 4 holds for countable sets of variables and hence it can have applications beyond this paper.

Next, we show a Craig-interpolation type lemma, which states that for two maximal satisfiable constraints, if they are consistent on the common variables, then their union is satisfiable (Lemma 6). We use Lemma 6 to derive a local-to-global principle, which states that local consistency implies global consistency for constraints resulting from runs of constraint automata (Theorem 10).

3.1 Satisfiability of maximal constraints

Let γ be a constraint over a set of variables $V = \{x_1, x_2, \ldots\}$.

▶ **Definition 3.** *We say that γ is* maximal *if for every pair of different variables x, y, either $x = y$ is in γ or for every $\rho \in \{P, S, I\}$ we have one of the following $x <_\rho y$, $y <_\rho x$ or $x \perp_\rho x'$ belongs to γ.*

Consider a maximal constraint γ. First, observe that we can eliminate equality constraints easily. Let E be the least equivalence relation on V containing all pairs (x, x') such that $x = x'$ occurs in γ. For each equivalence class C of E we pick the least i such that $x_i \in C$ and substitute all $y \in C$ with x_i. Let γ' be the resulting constraint, which we call the *equality-reduced* γ. If γ' has a conflicting pair of constraints (e.g. $y <_P y'$ and $y \perp_P y'$), then γ' is unsatisfiable as well as γ. Otherwise, if there is no such pair then γ' is a maximal constraint and it is satisfiable if and only if γ is.

Assume, without loss of generality, that γ is maximal and without equality constraints. We study three graphs over V: (V, P_γ), (V, S_γ), and (V, I_γ), which are obtained from γ by stating atomic constraints from γ as edges, i.e., we define $P_\gamma, S_\gamma, I_\gamma$ over V^2 such that for all $x, x' \in V$ we have $xP_\gamma x'$ (resp., $xS_\gamma x'$ or $xI_\gamma x'$) if and only if $x <_P x' \in \gamma$ (resp., $x <_S x' \in \gamma$ or $x <_I x' \in \gamma$).

Assume that γ is satisfiable, and it is satisfied by (possibly infinite) $\vec{w}$. Since γ is maximal and has no equality constraints, words in $\vec{w}$ are pairwise distinct. First, observe that graphs (V, P_γ) and (V, S_γ) are forests (union of disjoint trees). Indeed, a set of (pairwise distinct) words ordered by the prefix relation $\sqsubset_P$ is a forest, and hence (V, P_γ) is a forest. Similarly, (V, S_γ) is a forest as well. Second, observe that (V, I_γ) is an ordered set such that every element has finitely many predecessors, i.e., for $v \in V$ the set $A_v = \{u \in V \mid uI_\gamma v\}$ is finite. Indeed, $w \sqsubset_I w'$ implies that $|w| < |w'|$ and hence there are no infinite descending chains. Finally, I_γ contains P_γ and S_γ as every prefix (resp., suffix) is an infix as well.

Interestingly, these properties are in fact sufficient for γ to be satisfiable.

▶ **Lemma 4.** *Let γ be a maximal constraint without equality over the set of variables V. Then, γ is satisfiable if and only if (V, P_γ) and (V, S_γ) are forests, (V, I_γ) is a finitary ordered set, and I_γ contains P_γ and S_γ.*

We sketch the proof of the remaining implication, that the above conditions imply satisfiability of γ. We construct an assignment satisfying γ as follows. We first consider a possibly infinite set $\Gamma = \{a_v \mid v \in V\}$ as the alphabet; we reduce the obtained assignment later.

For all minimal elements v in (V, I_γ), we assign the letter a_v to v. Then, inductively, for an I_γ-minimal unassigned v, we proceed as follows. Our assumption is that for every pair of different variables x, y such that $xI_\gamma v$ and $yI_\gamma v$ all the constraints in γ involving both x and y are satisfied.

Since (V, P_γ) is a forest, either v has a unique P_γ-predecessor u_P, to which a word w_P is assigned, or v is P_γ-minimal and we put $w_P = \epsilon$. Similarly, either v has a unique S_γ-predecessor u_S, to which a word w_S is assigned, or v is S_γ-minimal and we put $w_S = \epsilon$. Finally, let A be the set of all u such that $uI_\gamma v$. Since (V, I_γ) is a well partial order, the set A is finite. Let $u[1], \ldots, u[k]$ be all words in A and $w' = w_{u[1]} \ldots w_{u[k]}$ be the concatenation of the words that have already been assigned to $u[1], \ldots, u[k]$. Then, we assign with v the word $w_v = w_P a_v w' a_v w_S$. Note that this is the first time the letter a_v is used.

Observe that for every $u \in V$, if $u <_P v \in \gamma$, then u has already been assigned with a word. Indeed, $uP_\gamma v$ and due to $P_\gamma \subseteq I_\gamma$ we have $uI_\gamma v$, and hence u has already been considered. It follows that all constraints $u <_P v \in \gamma$ are satisfied by the assignment. Similarly, all constraints $u <_S v \in \gamma$ and $u <_I v \in \gamma$ are satisfied by the assignment.

Now, observe that exactly these positive constraints are satisfied. First, consider, for an already assigned u that is different from v, the constraint $u <_I v \notin \gamma$. Recall that w_v is the word assigned to v and let w_u be the word assigned to u. We show that $w_u \perp_I w_v$ and hence the constraint $u \perp_I v$, which has to belong to γ due to maximality, is satisfied. Indeed, observe that I_γ contains P_γ and S_γ, and w_P, w' and w_S contain only letters a_x such that $xI_\gamma v$. Since $u <_I v \notin \gamma$ implies $uI_\gamma v$ does not hold, we get that w_v does not contain a_u. Moreover, w_u does not contain a_v and hence $w_u \perp_I w_v$.

Second, consider $u <_P v \notin \gamma$. We show $w_u \perp_P w_v$. If $uI_\gamma v$ does not hold, then $w_u \perp_I w_v$ and in particular $w_u \perp_P w_v$. Therefore, $uI_\gamma v$ holds. Assume towards contradiction $w_u \sqsubset_P w_v$. We know that w_u does not contain a_v because u was assigned before v. Therefore, if $w_u \sqsubset_P w_v$ then w_u is either equal to or is a prefix of w_v. In this case, w_v has to be non-empty. Recall also that it is the word w_x assigned to x, the P_γ-predecessor of v. Note, however that w_u is not equal to $w_P = w_x$ as all the words in the constructed substitution are different. Moreover, if $w_u \sqsubset_P w_x$ then due to the induction hypothesis and because of $P_\gamma \subseteq I_\gamma$ we have $u <_P x \in \gamma$. Therefore $u <_P v \in \gamma$, a contradiction. Thus, $w_u \perp_P w_v$ and $u \perp_P v$ is satisfied.

Similarly, if $u <_S v \notin \gamma$, then $u \perp_S v$ is satisfied. As a consequence, the constructed substitution over $\Gamma = \{a_v \mid v \in V\}$ satisfies γ.

Finally, we can transform the variable assignment over the infinite alphabet Γ to a satisfying assignment over any Σ with at least two letters. We take two distinct $b, c \in \Sigma$ and enumerate $a_1, a_2, \ldots$ the set Γ. Next, we apply to each $a_i \in \Gamma$ in the assignment the transformation $a_i \mapsto bc^i b$. One can easily check, that this transformation preserves prefixes, suffixes and infixes, and hence it is an assignment over Σ satisfying γ.

Observe that having a finite maximal constraint γ, we can eliminate equality in polynomial time, and then check the conditions of Lemma 4 in polynomial time as well. As a consequence we have:

▶ **Lemma 5.** *The satisfiability problem for maximal constraints can be solved in polynomial time.*

3.2 Joining constraints

We now prove the second crucial lemma that says that whenever we have two maximal satisfiable sets of constraints, if the sets agree on the constraints regarding the common variables, then the union of these sets is satisfiable.

▶ **Lemma 6.** *Let γ_1, γ_2 be maximal satisfiable constraints over variables X_1, X_2 respectively. If γ_1 and γ_2 restricted to $X_1 \cap X_2$ coincide, then $\gamma_1 \cup \gamma_2$ is satisfiable.*

First, observe that it suffices to show the lemma in the special case of $X_1 = X \cup \{x\}$ and $X_2 = X \cup \{z\}$, i.e., X_1 and X_2 differ in two variables.

▶ **Lemma 7.** *Let γ_1, γ_2 be maximal satisfiable constraints over variables $X \cup \{x\}, X \cup \{z\}$ respectively. If γ_1 and γ_2 restricted to X coincide, then $\gamma_1 \cup \gamma_2$ is satisfiable.*

Proof. The proof strategy is to define a maximal γ over $V = X \cup \{x, z\}$ such that $\gamma_1 \cup \gamma_2 \subset \gamma$, (V, P_γ) and (V, S_γ) are forests, (V, I_γ) is a finitary ordered set, and I_γ contains P_γ and S_γ. Then, Lemma 4 delivers the satisfiability of γ, and therefore $\gamma_1 \cup \gamma_2$. Since γ_1 and γ_2 are maximal, we only need to define the relation between x and z in γ.

First, we check whether some of the relations follow from transitivity. More precisely, we define γ^T as the least constraint that subsumes γ_1 and γ_2 and such that all the relations among $\{<_P, <_S, <_I\}$ are transitively closed in γ^T.

We can show that the relations $P_{\gamma^T}, S_{\gamma^T}$ and I_{γ^T} in γ^T are partial orders. The reflexivity holds trivially and transitivity follows from the definition. To see that the relations are antisymmetric, observe that the transitive closure only defines relations between x and z, as γ_1 and γ_2 are maximal and satisfiable and hence transitively closed. we discuss the case of I_{γ^T} here, the others are analogous. Assume towards contradiction that γ^T contains $x <_I z$ and $z <_I x$; then there are $v, v' \in X$ such that in $\gamma_1 \cup \gamma_2$ we have $x <_I v$, $v <_I z$ and $z <_I v', v' <_I x$. However, since γ_1 is maximal, $<_I$ is transitive and hence $v' <_I v$ is in γ_1. Similarly, $v <_I v'$ is in γ_2. Since γ_1 and γ_2 restricted to X coincide, we have both $v <_I v'$ and $v' <_I v$ belong to γ_1 and γ_2 and hence they are not satisfiable.

It is possible that $(X \cup \{x, z\}, P_{\gamma^T})$ is not a forest. This happens when both x and z are prefixes of some variable y, but the prefix order between x and z is not set. We fix this order as follows. If a constraint determines that $x <_I z$ or $z <_I x$, then we set $x <_P z$ or $z <_P x$ accordingly. Otherwise, we set the order in an arbitrary way. The same reasoning applies to the suffix order.

More precisely, we construct the set γ as an extension of γ^T in the following way. For each $R \in \{P, S\}$, if there is $y \in X$ such that $x \leq_R y$ and $z \leq_R y$, then we add to γ:

- $x \leq_R z$ if $x \leq_I z \in \gamma^T$
- $z \leq_R x$ and $z \leq_I x$ otherwise.

If there is no y such that $x \leq_R y$ and $z \leq_R y$, and the R-relation between x and z is not defined, we set x and z to be R incomparable in γ. This concludes the construction of γ.

This construction guarantees that I_γ contains P_γ and S_γ, and P_γ, S_γ, I_γ are partial orders. The last step ensures that (V, P_γ) and (V, S_γ) are forests. To see that (V, I_γ) is finitary in γ, observe that (V, I_γ) was finitary is γ_1 and γ_2. Thus, every variable in $X \cup \{x\}$ has in γ finitely many predecessors (at most one more than it has in γ_2), and the same holds for $X \cup \{z\}$. ◀

Lemma 6 follows from the above lemma using inductive reasoning.

Lemma 6 shows that a finite union of finite satisfiable constraints is satisfiable. This does not translate to the infinite union case; the following example shows a constraint that can be repeated any number of times, but not infinitely many times.

▶ **Example 8.** Consider a single-state 1-constraint automaton $\mathcal{A}$ with the state q that is both initial and accepting. The only transition is $(q, \{x_1' <_P x_1\}, q)$. This automaton can accept sequences of arbitrary length; it also has an infinite path $(q, q, q, \dots)$, but it does not accept any infinite sequence. This is because there is no infinite sequence of finite words such that each consecutive word is a prefix of the previous one.

3.3 Stratified constraints and their satisfaction

In this section, we connect general constraints with constraints that are derived from partial runs of n-constraint automata. One of the key differences is a natural structure of constraints resulting from partial runs; such constraints are local, which is captured by the following definition.

Consider a (partial) run π of an n-constraint automaton over some finite sequence σ and let $\gamma_1, \ldots, \gamma_k$ be the sequence of constraints along transitions of this run. First, for each j we relabel variables in γ_j in a way such that x_i becomes x_i^{j-1} and x_i' becomes x_i^j. We denote the resulting constraint by γ_j'. Second, the constraints $\gamma_1', \ldots, \gamma_k'$ can be extended to maximal satisfiable constraints $\gamma_1^t, \ldots, \gamma_k^t$, which agree on the common variables; it suffices to check the relations in the sequence σ. Consider γ_π to be the union of constraints $\gamma_1^t, \ldots, \gamma_k^t$. The constraints in γ_π are local, which is captured by the following definition of stratified constraints; there are constraints only between variables corresponding to the successive positions.

▶ **Definition 9.** *Given a natural number n and $k \in \mathbb{N}$, a* stratified (n,k)-constraint γ *is a constraint over the set of variables of the form x_i^j, where $i \in \{1, \ldots, n\}$ and $0 \leq j < k$, such that all the atoms $x_i^j \oplus x_{i'}^{j'}$ are such that $j - j' \in \{-1, 0, 1\}$. The* j-th layer *of a stratified (n,k)-constraint is the set of variables $x_1^j, \ldots, x_n^j$.*

The constraint γ_π is a stratified (n,k)-constraint. As it results from a (partial) run (the run π), it is satisfiable. However, satisfiability of γ_π follows also from a general principle.

We show a local-to-global principle, which states that for stratified constraints local consistency (subconstraints γ_j^t are satisfiable and agree on common variables) implies global consistency (i.e., γ_π is satisfiable.)

▶ **Theorem 10.** *Let γ be a stratified (n,k)-constraint such that $n, k \in \mathbb{N}$ and for every $0 \leq j < k-1$ the constraint γ restricted to layers $j, j+1$ is maximal and satisfiable. Then, the constraint γ is satisfiable.*

The proof of Theorem 10 follows by induction from Lemma 6. Consider a stratified $(n, k+1)$-constraint γ and let $\hat{\gamma}_1$ be the constraint obtained from γ by dropping the last layer. Since $\hat{\gamma}_1$ is a stratified (n,k)-constraint, assume that it is consistent. Let $\hat{\gamma}_2$ be the $(n,2)$-constraint consisting of the last two layers: k-th and $(k+1)$-th. Note that $\hat{\gamma}_2$ is maximal and satisfiable and the intersection of $\hat{\gamma}_1$ and $\hat{\gamma}_1$ is the k-th layer. Therefore, by Lemma 6 the constraint γ is satisfiable.

4 Reachability via type-tracking automata

We introduce a special type of n-constraint automata, called *type-tracking automata*, which keep track of the *type* (intuitively: what relations hold between the words of the tuple) of the current tuple in the states. While in n-constraint automata a path in the automaton, considered a labeled graph, may not correspond to a partial run, which involves satisfaction of constraints along the path, in type-tracking automata every finite path corresponds to a partial run over some sequence. This property is key in solving the reachability problem for n-constraint automata.

The *type* of an n-tuple $\vec{w}$ is the set of all non-trivial atomic n-constraints over $\{x_1, \ldots, x_n\}$ satisfied for $\vec{w}$. Observe that a constraint γ is a type if and only if it is maximal and satisfiable.

In the above, non-trivial atomic constraints are the constraints of the form $x \oplus y$ where x and y are different variables. For example, the type of the first 3-tuple of Figure 1 is the set containing the following atoms:

- $x_1 \perp_P x_2, x_2 \perp_P x_1, x_2 <_S x_1, x_2 <_I x_1$
- $x_3 <_P x_1, x_1 \perp_S x_3, x_3 \perp_S x_1, x_3 <_I x_1$
- $x_2 \perp_P x_3, x_3 \perp_P x_2, x_2 \perp_S x_3, x_3 \perp_S x_2, x_2 <_I x_3$

Let $\mathbb{T}$ be the set of all types (of some n-tuples). The set $\mathbb{T}$ is exponentially bounded in n and their members have size polynomial in n.

We now introduce a special version of n-constraint automata whose states carry information regarding the type of current n-tuple.

▶ **Definition 11.** *For an n-constraint automaton $\mathcal{A} = \langle Q, Q_0, Q_F, \delta\rangle$, the* type-tracking n-constraint automaton *$\mathcal{A}^{FT}$ resulting from $\mathcal{A}$ is the n-constraint automaton $\langle Q', Q'_0, Q'_F, \delta'\rangle$ such that:*

- *the states of $\mathcal{A}^{FT}$ are the pairs of a state of $\mathcal{A}$ and a type: $Q' = Q \times \mathbb{T}$, $Q'_0 = Q_0 \times \mathbb{T}$ and $Q'_F = Q_F \times \mathbb{T}$,*
- *δ' is the set of tuples $\langle (q_1, \gamma_1), \gamma', (q_2, \gamma_2)\rangle$, such that for some $\langle q_1, \gamma, q_2\rangle \in \delta$, the constraint γ' is a maximal consistent constraint that contains γ_1, γ_2 and γ.*

We say that a partial run $(q_0, \gamma_0), (q_1, \gamma_1), \ldots$ (finite or infinite) of a type-tracking n-constraint automaton $\mathcal{A}^{FT}$ over a sequence $\sigma = w_0, w_1, \ldots$ is *consistent* if for every $0 \le i \le |\sigma| - 1$ we have γ_i is the type of $\sigma[i]$. Observe that every partial run of an n-constraint automaton has the corresponding *consistent* run in the type-tracking automaton.

The main advantage of type-tracking n-constraint automata is that every path in a type-tracking automaton corresponds to some partial run, which is not the case for n-constraint automata in general.

▶ **Lemma 12.** *Let $\mathcal{A}$ be an n-constraint automaton and $\mathcal{A}^{FT}$ be its the type-tracking n-constraint automaton.*

1. *Every (finite or infinite) partial run in $\mathcal{A}$ over a sequence σ has a (unique) corresponding consistent partial run of $\mathcal{A}^{FT}$ over σ.*
2. *Every finite path in $\mathcal{A}^{FT}$ corresponds to a partial run of $\mathcal{A}^{FT}$ over some sequence σ.*

Proof. Property 1 follows from augmenting states of the partial runs with the types of the corresponding tuples of the given sequence. To see 2, observe that a path π of length k in the type-tracking automaton $\mathcal{A}^{FT}$ yields a stratified (n, k)-constraint such that any two successive layers are maximal and satisfiable. Thus, by Theorem 10 it is satisfiable and hence there is a sequence σ such that $\mathcal{A}^{FT}$ over σ has a consistent partial run corresponding to π. ◀

Type-tracking n-constraint automata are typically exponentially larger than their n-constraint counterparts. For example, consider a single state n-constraint automaton accepting all the sequences. Any corresponding type-tracking n-constraint automaton has to have at least as many states as there are types, so exponentially many.

The type-tracking n-constraint automata need not be explicitly stored. We can compute the states of type-tracking n-constraint automata *on the fly*. To do so, we employ the following result:

▶ **Lemma 13.** *For a given n-constraint automaton $\mathcal{A}$, the following problems can be solved in polynomial time.*

1. *Given (s, t), check whether (s, t) is a state of $\mathcal{A}^{FT}$.*
2. *Given (s_1, t_1), (s_2, t_2) and γ, check whether $((s_1, t_1), \gamma, (s_2, t_2))$ is a transition of $\mathcal{A}^{FT}$.*

Proof. To check (s,t) whether it is a state of $\mathcal{A}^{FT}$ it suffices to check whether s is a state of $\mathcal{A}$ and t is a type. The latter can be done in polynomial time as follows. Checking maximality is straightforward and for maximal constraints checking satisfiability can be done in polynomial time (Lemma 5).

Solving 2 amounts to checking maximality, satisfiability and containment, which can be done in polynomial time. ◀

Lemma 13 implies that graph-reachability in $\mathcal{A}^{FT}$ can be solved in polynomial space in $|\mathcal{A}|$.

▶ **Lemma 14.** *The problem: given an n-constraint automaton, its states q_1, q_2 and two types γ_1, γ_2, decide whether (q_2, γ_2) is path-reachable from (q_1, γ_1) in the type-tracking n-constraint automaton resulting from $\mathcal{A}$, is in PSPACE.*

We now show the upper bound for the reachability problem.

▶ **Theorem 15.** *The reachability problem for n-constraint automata is in PSPACE.*

This theorem follows from Lemma 12 and Lemma 14. To check the reachability from q_1 to q_2, the algorithm non-deterministically picks (recall that Savitch's Theorem proves that PSPACE=NPSPACE) two types γ_1, γ_2 and employs Lemma 14 to check if there is a path in the type-tracking automaton. By Lemma 12, such γ_1, γ_2 and a path exist if and only if there is a path from q_1 to q_2.

We show PSPACE-hardness of reachability in n-constraint automata. For $n > 0$, we say that a propositional formula ϕ over $2n$ variables *represents* a directed graph $G = (V, E)$, if V is the set of binary sequences of length n, and for all vertices $\vec{x}, \vec{y}$, we have $E(\vec{x}, \vec{y})$ if and only if $\phi(\vec{x}, \vec{y})$ is satisfied. The *reachability problem in succinct graphs* is defined as follows: given $n > 0$, a formula ϕ over $2n$ variables and two binary sequences $\vec{s}, \vec{t}$ of length n, decide whether $\vec{t}$ is reachable from $\vec{s}$ in the graph represented by φ. This problem is known to be PSPACE-complete [20].

We say that a propositional formula ϕ is in an *extended-DNF* if it is a disjunction of conjunctions of *literals* of the following three forms: p_i, $\neg p_i$ or $p_i \Leftrightarrow p_j$. Note that in the standard DNF, the equivalence is not allowed. It turns out that extended-DNF formulas are enough to make the reachability problem in succinct graphs PSPACE-complete.

▶ **Lemma 16.** *The reachability problem in graphs given by formulae in extended-DNF is PSPACE-complete.*

The proof follows from the fact that extended-DNF are sufficient to express property of being the successor configuration of polynomial-space Turing machine. The main idea is that the two consecutive configurations of a Turing machine differ only on a head position, a state of the Turing machine, and at most one tape cell; this can be expressed using a disjunction of polynomially many formulas. The remaining part of the configuration is the same, and this can be expressed in the conjunctions using $\Leftrightarrow$.

We now prove the lower bound for the n-constraint automata.

▶ **Theorem 17.** *The reachability problem for n-constraint automata is PSPACE-hard.*

Proof. Observe that formulae in extended-DNF can be encoded in an $(n+2)$-constraint automaton, and hence the reachability problem in succinct graphs reduces to the reachability problem in constraint automata with three states: q_0, q, q_F. To do so, we designate the last two elements w_{n+1}, w_{n+2} in each tuple to be different words (e.g. stating in the constraints $x_{n+1} \perp_I x_{n+2}$), which do not change along the run, and encode *true* and *false* in the propositional sense. Then, literals p_i, $\neg p_i$ and $p_i \Leftrightarrow p_j$ are respectively translated to constraints $x_i = x_{n+1}$, $x_i = x_{n+2}$, and $x_i = x_j$. The conjunction of literals can be stated in the constraints, and the disjunction can be encoded with non-determinism of the $(n+2)$-constraint automaton, i.e., the disjunction $d_1 \vee \ldots \vee d_k$ is translated to k transitions from q to itself each with the constraints resulting from d_i, which is a conjunction of literals. Finally, we set the first transition from q_0 to q to set the initial vertex in the reachability problem in graphs given by a propositional formula and one outgoing transition from q to q_F, which is possible only with the valuation of variables corresponding to the final vertex in the instance of the problem. ◀

As a direct consequence of Theorems 15 and 17 we have:

▶ **Corollary 18.** *The reachability problem for n-constraint automata is PSPACE-complete.*

5 Checking emptiness over bounded words

In this section, we consider constraint automata over bounded sequences over finite alphabets. We establish PSPACE-completeness of the emptiness problem for constraint automata restricted to bounded sequences. Notice that Example 8 shows that there is no straightforward counterpart of Lemma 12 for infinite runs.

We show that we can focus on ultimately periodic runs over an ultimately periodic sequences.

▶ **Lemma 19.** *An n-constraint automaton has an accepting run over some bounded infinite sequence if and only if it has an accepting ultimately periodic run over an ultimately periodic sequence.*

Proof. Observe that having a bound B, the set of B bounded words is finite and hence an n-constraint automaton $\mathcal{A}$ over B-bounded sequences can be considered as a Büchi-automaton. Therefore, if $\mathcal{A}$ accepts a B-bounded sequence, then it accepts an ultimately periodic B-bounded sequence. Clearly, every ultimately periodic sequence is bounded from some B. As a consequence, we can focus on ultimately periodic words. ◀

To decide whether there exists an ultimately periodic sequence $\sigma_0\sigma_1^\omega$ it suffices to decide the existence of an appropriate σ_0 and σ_1 almost independently. First, it is convenient to work with the type-tracking n-constraint automaton $\mathcal{A}^{FT}$ for $\mathcal{A}$, as every finite path there is realizable. Furthermore, there is a simple condition for a cycle, which can be iterated indefinitely. We discuss it in the following section.

5.1 Finding a cycle

The cycle c of $\mathcal{A}^{FT}$ defines a stratified (n,k)-constraint γ, which is satisfiable, i.e., it is a partial run over some sequence σ_1. We say that the cycle c is *iterable* if and only if it contains an accepting state and γ extended with constraints $x_1^0 = x_1^k, \ldots, x_n^0 = x_n^k$ (equality between corresponding variables in the first and the last layer) is still satisfiable. Observe, that if c is an iterable cycle, then c^ω is a partial run over σ_1, i.e., it is realizable.

▶ **Lemma 20.** *Let $\mathcal{A}$ be an n-constraint automaton and $\mathcal{A}^{FT}$ be its type-tracking n-constraint automaton. The automaton $\mathcal{A}$ has an accepting ultimately periodic run over an ultimately periodic sequence if and only if there exists a state s in $\mathcal{A}^{FT}$ such that*

- *(s,γ) is reachable from the initial state, and*
- *there exists an iterable cycle c from (s,γ) to itself.*

Observe that having a state (s, γ) in $\mathcal{A}^{FT}$, which can be non-deterministically picked, the first condition can be verified in PSPACE due to Theorem 15. For the second condition, we can employ the following non-deterministic procedure, which works in polynomial space in $|\mathcal{A}|$, as follows.

▶ **Lemma 21.** *Given an n-constraint automaton $\mathcal{A}$ and a state s of its type-tracking n-constraint automaton $\mathcal{A}^{FT}$, one can decide in PSPACE whether there is an iterable cycle in $\mathcal{A}^{FT}$ from s to itself.*

Proof. Our non-deterministic algorithm is similar to the standard on-the-fly reachability checking, but it requires additional information regarding the traversed path to ensure that the computed cycle is iterable. In particular, it ensures that an accepting state has been visited and verifies the relations between the initial and the final configuration, that may depend on the path.

The algorithm stores five objects: the initial state (s, γ_1), the current state (t, γ_2), number of steps k, the maximal constraint γ containing γ_1, γ_2 over the variables from γ_1 and γ_2, and a boolean value *Acc* stating whether an accepting state has been observed. We start with the state (s, γ_1) and initially $(s, \gamma_1) = (t, \gamma_2)$, $k = 0$, γ being the constraint describing two copies of γ_1 and the equality constraints between the corresponding variables, and *Acc* being true if s is accepting. Then, we compute the next value so that the following invariant holds:

inv (t, γ_2) is reachable from (s, γ_1) over some sequence of length k consistent with γ, i.e., there is a sequence σ of length k such that (a) there is a partial run (visiting an accepting state if *Acc* it true) over σ from (s, γ_1) to (s, γ_2), and (b) the constraint γ is consistent with the relations over $\sigma[1]$ and $\sigma[k]$.

We discuss how to maintain the invariant (inv). Assume that $(s, \gamma_1), (t, \gamma_2)$, k, γ and *Acc* are correct. Now, suppose that t' is some successor of t in $\mathcal{A}^{FT}$ and γ^+ is any maximal consistent constraint over variables from s, t, t'. The projection of γ^+ on the variables of s and t' satisfies the invariant. To see this, we apply Lemma 6 to $\hat{\gamma}_1$ being the constraint corresponding to the sequence σ and a partial run from s to t, and $\hat{\gamma}_2$ being γ^+. Both sets are consistent and they agree over the common variables, therefore their union is satisfiable and the satisfying sequence has length $k + 1$.

As a consequence, it suffices to execute this non-deterministic procedure until it reaches the state with $(s, \gamma_1), (s, \gamma_1)$, $k > 0$, $Acc = true$ and γ containing the equality constraint for the corresponding variables, in which case it accepts, or it works indefinitely, but it can be stopped after k exceeds the number of states of $\mathcal{A}^{FT}$ times the number of possible transitions, which is exponential in n. ◀

5.2 Solving the reachability problem

We can now conclude that solving reachability can be done in polynomial space.

▶ **Theorem 22.** *Checking whether there is a bounded infinite sequence accepted by a given constraint automaton $\mathcal{A}$ can be done in polynomial space in $|\mathcal{A}|$.*

The (non-deterministic) algorithm guesses a state s and a type γ such that (s, γ) is reachable from the initial state, and there exists an iterable cycle c from (s, γ) to itself. Verifying both properties was shown to be decidable in polynomial space. Since NPSPACE= PSPACE, the same can be done deterministically in polynomial space.

The matching lower bound follows from a straightforward reduction from the reachability problem.

▶ **Theorem 23.** *Checking whether there is a bounded infinite sequence accepted by a given constraint automaton is PSPACE-complete.*

6 Conclusions

We have shown that the reachability problem and the non-emptiness over bounded sequences problem are PSpace-complete for constraint automata. The proof works for constraint automata with the prefix, the suffix, the infix and the equality relations. The presented hardness proof requires only equality and negated equality, which can be expressed having non-strict order (prefix, infix or suffix). We believe that it can be adapted to the case of two relations: strict prefix and negated strict prefix (resp., suffix). This shows that the complexity follows from the number of variables.

The remaining open question is whether the (unrestricted) emptiness problem for constraint automata over infinite words is decidable. Lemma 4 gives us some insight. An important step towards this result would be to determine whether every non-empty constraint automaton has an accepting run (over some sequence) that is ultimately periodic. We discuss here an example demonstrating that it is not as straightforward as it may initially appear.

Consider a 3-constraint automaton $\mathcal{A}$ with a single state and a single transition. This transition is a conjunction of the following atoms:

- $x_1 = x_1'$
- $x_2 <_I x_2'$
- $x_3 \perp_I x_2$
- $x_3 <_I x_1$
- $x_3 <_I x_2'$

At first glance, it seems that the language of this automaton should be non-empty: x_1 is always the same, x_2 always increases, and x_3 is defined based on variables x_1 and x_2. To illustrate the issue, consider the following sequence:

$$\begin{pmatrix} abcde \\ x \\ a \end{pmatrix}\begin{pmatrix} abcde \\ ax \\ bc \end{pmatrix}\begin{pmatrix} abcde \\ axbc \\ bcd \end{pmatrix}\begin{pmatrix} abcde \\ axbcd \\ bcde \end{pmatrix}$$

Observe that this sequence is accepted by $\mathcal{A}$, but it cannot be extended in a way that maintains the acceptance. This is because the next value of x_2 must include all the infixes of x_1, which is contradictory with the conditions stating that x_3 is an infix of x_1 not contained in x_2. It can be easily checked that $\mathcal{A}$ does not accept any infinite sequence.

This example can be extended (by adding a lot of constraints to the only transition) in such a way that the only transition is maximal. In this case, there exist arbitrarily long sequences accepted by the automaton, where the types of all tuples and relations between all pairs of tuples in the same order are the same. Moreover, it can be done in a way that the lengths of x_2 and x_3 always increase (and x_1 remains unchanged). Despite this, there is no infinite sequence accepted by this automaton. This shows that formulating a pumping-lemma-esque argument in this context is elusive.

References

1 Mikołaj Bojańczyk. Atom book, September 2019. URL: `https://www.mimuw.edu.pl/~bojan/upload/main-10.pdf`.

2 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.

3 Claudia Carapelle, Shiguang Feng, Alexander Kartzow, and Markus Lohrey. Satisfiability of ECTL$_*$ with local tree constraints. *Theory Comput. Syst.*, 61(2):689–720, 2017. doi:10.1007/s00224-016-9724-y.

4 Claudia Carapelle, Alexander Kartzow, and Markus Lohrey. Satisfiability of ECTL$_*$ with constraints. *J.omput. Syst. Sci.*, 82(5):826–855, 2016. doi:10.1016/j.jcss.2016.02.002.

5 Karlis Cerans. Deciding properties of integral relational automata. In *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, volume 820, pages 35–46. Springer, 1994. doi:10.1007/3-540-58201-0_56.

6 Stéphane Demri and Morgan Deters. Temporal logics on strings with prefix relation. *J. Log. Comput.*, 26(3):989–1017, 2016. doi:10.1093/logcom/exv028.

7 Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint LTL. *Inf. Comput.*, 205(3):380–415, 2007. doi:10.1016/j.ic.2006.09.006.

8 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.

9 Stéphane Demri and Karin Quaas. Concrete domains in logics: a survey. *ACM SIGLOG News*, 8(3):6–29, 2021. doi:10.1145/3477986.3477988.

10 Stéphane Demri and Karin Quaas. Constraint automata on infinite data trees: From CTL(Z)/CTL*(Z) to decision procedures, 2023. arXiv:2302.05327.

11 Régis Gascon. An automata-based approach for CTL$*$ with constraints. In *Joint Proceedings of the 8th, 9th, and 10th International Workshops on Verification of Infinite-State Systems, INFINITY 2006 / 2007 / 2008*, volume 239 of *Electronic Notes in Theoretical Computer Science*, pages 193–211. Elsevier, 2009. doi:10.1016/j.entcs.2009.05.040.

12 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005141.

13 Artur Jeż. Recompression: A simple and powerful technique for word equations. *J. ACM*, 63(1):4:1–4:51, 2016. doi:10.1145/2743014.

14 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

15 Prateek Karandikar and Philippe Schnoebelen. Decidability in the logic of subsequences and supersequences. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015*, volume 45 of *LIPIcs*, pages 84–97, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.84.

16 Alexander Kartzow and Thomas Weidner. Model checking constraint LTL over trees, 2015. arXiv:1504.06105.

17 Dietrich Kuske. Theories of orders on the set of words. *RAIRO Theor. Informatics Appl.*, 40(1):53–74, 2006. doi:10.1051/ita:2005039.

18 Dietrich Kuske and Georg Zetzsche. Languages ordered by the subword order. In *Foundations of Software Science and Computation Structures – 22nd International Conference, FOSSACS 2019, Held as Part of ETAPS 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 348–364. Springer, 2019. doi:10.1007/978-3-030-17127-8_20.

19 Leonid Libkin, Wim Martens, and Domagoj Vrgoc. Querying graphs with data. *J. ACM*, 63(2):14:1–14:53, 2016. doi:10.1145/2850413.

20 Antonio Lozano and José L Balcázar. The complexity of graph problems for succinctly represented graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 277–286. Springer, 1989.

21 G. S. Makanin. The problem of solvability of equations in a free semigroup. *Mathematics of The Ussr-sbornik*, 32:129–198, 1977.

22 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.

23 Dominik Peteler and Karin Quaas. Deciding emptiness for constraint automata on strings with the prefix and suffix order. In *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022*, volume 241 of *LIPIcs*, pages 76:1–76:15, 2022. `doi:10.4230/LIPIcs.MFCS.2022.76`.

24 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004. `doi:10.1145/990308.990312`.

25 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

26 Luc Segoufin and Szymon Toruńczyk. Automata based verification over linearly ordered data domains. In *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011*, volume 9 of *LIPIcs*, pages 81–92. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. `doi:10.4230/LIPIcs.STACS.2011.81`.