

A Pumping-Like Lemma for Languages over Infinite Alphabets

Yoav Danieli  

The Henry and Marilyn Taub Faculty of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel

Abstract

We prove a kind of a pumping lemma for languages accepted by one-register alternating finite-memory automata. As a corollary, we obtain that the set of lengths of words in such languages is semi-linear.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases infinite alphabets, pumping lemma, alternation, semi-linearity

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.29

Related Version *Full Version:* <https://arxiv.org/abs/2512.23403>

1 Introduction

Finite-memory automata [18, 19] generalize classical Rabin-Scott finite-state automata [29] to *infinite* alphabets. By equipping the automata with a finite set of registers (called in [18, 19] *windows*), which store letters from the infinite alphabet during computation, and restricting the power of the automaton to comparing the input letters with register contents and copying input letters to registers, the automaton retains a fixed finite-memory of input letters. Consequently, the languages accepted by finite-memory automata possess properties similar to regular languages and are, thus, termed *quasi-regular* languages.

An important facet of finite-memory automata is a certain *indistinguishability* view of the infinite alphabet embedded in the modus operandi of the automaton. The language accepted by an automaton is invariant under permutations of the infinite alphabet. Thus, the actual symbols occurring in the input are of no real significance. Only the initial and repetition patterns matter.

Automata over infinite alphabets have gained increasing importance in computer science as they provide formal models for analyzing systems that operate over unbounded data domains. Such models are essential for specifying and verifying properties of XML documents, database queries, and programs with variables over infinite domains. The ability to reason about infinite alphabets while maintaining decidability of key properties makes these models particularly valuable for the formal verification of data-aware systems.

Over the years, many models of automata over infinite alphabets have been proposed (see surveys in [32, 20, 6]), though most of these models are incomparable. In the absence of a definitive model for managing infinite alphabets, evaluating a formalism requires consideration of its desirable properties, including: expressive power, closure and regular properties, decidability and complexity of classical problems, and applicability of the model.

Finite-memory automata provide a structured way to reason about such systems by focusing on patterns and repetitions of data values rather than their specific identities. This perspective is particularly useful in applications where the exact values of data are less important than their relative behavior over time. Note that quasi-regular languages are a particular case of nominal sets (also known as sets with atoms or Fraenkel-Mostowski sets) [3], and finite-memory automata themselves are expressively equivalent to nominal automata (also known as orbit-finite automata) [4].



© Yoav Danieli;

licensed under Creative Commons License CC-BY 4.0

43rd International Symposium on Theoretical Aspects of Computer Science (STACS 2026).

Editors: Meena Mahajan, Florin Manea, Annabelle McIver, and Nguyễn Kim Thăng

Article No. 29; pp. 29:1–29:19



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



However, the non-emptiness problem for finite-memory automata is NP-complete [30] and the universality problem is undecidable for automata with two (or more) registers [27].

Nevertheless, when restricted to a single register, the emptiness problem is decidable for the *alternating* variant of finite-memory automata [9, 12, 15], albeit with non-primitive recursive complexity [9]. Since alternation extends the power of nondeterministic automata by introducing existential and universal states, effectively allowing the automaton to explore multiple computation paths in parallel [5], the universality and containment problems are decidable for these automata as well.

The restriction of finite-memory automata to a single register (while still allowing a finite set of constant symbols) is of significance: it preserves a Parikh-like theorem [17], the existence of a deterministic equivalent automaton is decidable [7], and, for ordered alphabets, the intersection of nondeterministic and co-nondeterministic one-register languages lies within the deterministic class [23].

This paper deals with alternating finite-memory automata with one register. Such automata are tightly related to the linear and branching temporal logics with the freeze quantifier. In particular, the satisfiability problem for formulas in linear temporal logic with forward operators and one freeze quantifier is decidable, see [9]. In addition, these models are particularly robust and possess many regular properties, being closed under Boolean operations and subsuming various other automata models, such as top-view pebble automata and weak two-pebble automata [33]. Recently, it was shown in [8] that the boundness problem for these models is also decidable, by proving an extension lemma for sufficiently long words. Also, there is a strong connection to alternating timed automata with one clock, such that complexity and decidability results transfer back and forth between these models [12].

A fundamental property of classical finite-state automata is their regularity, which is often characterized using the pumping lemma. The pumping lemma serves as a crucial tool for proving that certain languages are not regular by identifying structural constraints on words within the language.

For general many-register finite-memory automata, it is well known that the classical pumping lemma does not necessarily hold, see [19, Example 3]. Therefore, in any suitable pumping lemma for languages over infinite alphabets, the pumped patterns are not, necessarily, identical, but, instead, are equivalent modulo some permutation of the infinite alphabet. A pumping lemma of this kind has been proven for finite-memory automata in [26], though it does not involve the action of permutations on repeated subwords and only establishes the existence of such pumped patterns.

The main result of this paper is a kind of pumping lemma for languages accepted by one-register alternating finite-memory automata. As a corollary, we obtain that the set of lengths of words in such languages is semi-linear.

This paper is organized as follows. In the next section, we recall the definition of finite-memory automata from [19] and, in Section 3, we prove a pumping lemma for these automata. Sections 4 and 5, respectively, contain the definition of alternating finite-memory automata and the statement of the pumping lemma for these automata, that is the main result of our paper. In Section 6, we present some applications of the pumping lemma, including the semi-linearity result. Due to space constraints, the full proof of the pumping lemma is omitted and can be found in the full version of this article. We provide the proof overview and the main methods in Section 7. Finally, Section 8 contains some remarks concerning the complexity of computing the pumping lemma parameter and possible extensions for ordered alphabets and timed automata.

2 Finite-memory automata

Throughout this paper, we employ the following conventions.

- Σ is a fixed infinite alphabet.
- Symbols in Σ are denoted by σ, θ , etc., sometimes indexed or primed.
- Bold low-case Greek letters $\boldsymbol{\sigma}, \boldsymbol{\theta}$, etc., also sometimes indexed or primed, denote words over Σ .
- Symbols that occur in a word denoted by a boldface letter are always denoted by the same *non-boldface* letter with an appropriate subscript. For example, the letters that occur in $\boldsymbol{\sigma}'$ are denoted by σ'_i .
- For word $\boldsymbol{\sigma} = \sigma_1\sigma_2\cdots\sigma_n \in \Sigma^*$, we denote by $[\boldsymbol{\sigma}]$ the set of letters occurring in $\boldsymbol{\sigma}$:

$$[\boldsymbol{\sigma}] = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$$

and call this set the *contents* of $\boldsymbol{\sigma}$.

- For a subset Φ of Σ and a positive integer r , we denote by $\Phi^{r\neq}$ the subset of Φ^r whose elements do not contain repeated letters:

$$\Phi^{r\neq} = \{\sigma_1 \dots \sigma_r \in \Phi^r : \text{for all } i \text{ and } j, \text{ such that } i \neq j, \sigma_i \neq \sigma_j\}.$$

- The length of a word $\boldsymbol{\sigma} \in \Sigma^*$ is denoted by $|\boldsymbol{\sigma}|$ and the cardinality of a finite set X is denoted by $\|X\|$.

Next, we recall the definition of finite-memory automata from [18, 19] in which a fixed finite number of distinct letters can be stored in the automaton memory during a computation.

► **Definition 1** (Cf. [19, Definition 1]). *An r -register finite-memory automaton (over Σ) is a system $\mathbf{A} = \langle S, s_0, F, \Delta, \boldsymbol{\theta}_0, \mu_\Delta, \mu_=: \mu_{\neq\text{skip}}, \mu_{\neq\text{replace}} \rangle$ whose components are as follows.*

- $S, s_0 \in S$, and $F \subseteq S$ are the finite set of states, the initial state and the set of accepting states, accordingly.
- $\Delta \subset \Sigma$ is a finite set of distinguished letters (constants).
- $\boldsymbol{\theta}_0 \in (\Sigma \setminus \Delta)^{r\neq}$ is the initial register assignment.¹
- $\mu_\Delta : S \times \Delta \rightarrow 2^S$, $\mu_=: S \times \{1, 2, \dots, r\} \rightarrow 2^S$, $\mu_{\neq\text{skip}} : S \rightarrow 2^S$, and $\mu_{\neq\text{replace}} : S \rightarrow 2^{S \times \{1, 2, \dots, r\}}$ are the transition functions.

The intuitive meaning of these functions is as follows. Let \mathbf{A} read a letter σ being in state s with a letter θ_i stored in the i th register, $i = 1, 2, \dots, r$.

- If $\sigma \in \Delta$, then \mathbf{A} enters a state from $\mu_\Delta(s, \sigma)$, with the same letters in the registers.
- If for some $i = 1, 2, \dots, r$, $\sigma = \theta_i$, then \mathbf{A} enters a state from $\mu_=(s, i)$, with the same letters in the registers.
- If $\sigma \notin \Delta \cup \{\theta_1, \theta_2, \dots, \theta_r\}$, then, either \mathbf{A} enters a state from $\mu_{\neq\text{skip}}(s)$, with the same letters in the registers, or for some $(s', i) \in \mu_{\neq\text{replace}}(s)$, \mathbf{A} enters s' and replaces, in the i th register, θ_i with σ , leaving all other registers intact.

A *configuration* of \mathbf{A} is a pair in $S \times (\Sigma \setminus \Delta)^{r\neq}$, where the state component of the pair is the current state and the letters' components are the letters stored in the registers. The configuration $(s_0, \boldsymbol{\theta}_0)$ is the *initial* configuration, and the configurations with the first component in F are *accepting* (or *final*) configurations.

¹ We require that the memory does not repeat letters, as retaining a letter more than once is unnecessary. Moreover, distinguished symbols from Δ cannot be stored in the registers, as they are recognized without requiring allocated space.

29:4 Pumping Lemma

For configurations (s, θ) and (s', θ') , where $\theta = \theta_1\theta_2 \cdots \theta_r$ and $\theta' = \theta'_1\theta'_2 \cdots \theta'_r$, we write $(s, \theta) \xrightarrow{\sigma} (s', \theta')$, if one of the following holds.

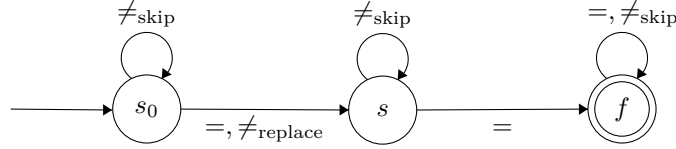
- $\sigma \in \Delta$, $s' \in \mu_{\Delta}(s, \sigma)$, and $\theta' = \theta$.
- For some $i = 1, 2, \dots, r$, $\sigma = \theta_i$, $s' \in \mu_{=}(s, i)$, and $\theta' = \theta$.
- $\sigma \notin \Delta \cup [\theta]$ and either $s' \in \mu_{\neq \text{skip}}(s)$, and $\theta' = \theta$, or for some $i = 1, 2, \dots, r$, $(s', i) \in \mu_{\neq \text{replace}}(s)$, and, for $j = 1, 2, \dots, r$,

$$\theta'_j = \begin{cases} \theta_j & \text{if } j \neq i. \\ \sigma & \text{if } j = i. \end{cases}$$

A run of \mathbf{A} on a word $\sigma = \sigma_1\sigma_2 \cdots \sigma_n$ is a sequence of configurations c_0, c_1, \dots, c_n such that for all $i = 1, 2, \dots, n$, $c_{i-1} \xrightarrow{\sigma_i} c_i$. In such a case, we shall also write $c_0 \xrightarrow{\sigma} c_n$.

A run is *accepting*, if it starts from the *initial* configuration (s_0, θ_0) and ends in a final configuration. A word is accepted by \mathbf{A} , if there exists an accepting run of \mathbf{A} on it. The language of all accepted words is denoted by $L(\mathbf{A})$.

► **Example 2** ([19, Example 1]). Consider a one-register finite-memory automaton \mathbf{A} , a self-explanatory diagram of which is shown in Figure 1.²



■ **Figure 1** The graph representation of \mathbf{A} .

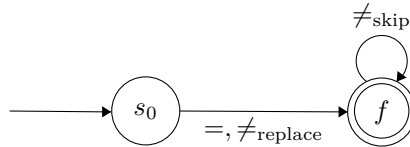
It is easy to see that the language of \mathbf{A} consists of all words over Σ in which some letter appears more than once:

$$L(\mathbf{A}) = \{\sigma_1\sigma_2 \cdots \sigma_n : \text{there exist } 1 \leq i < j \leq n \text{ such that } \sigma_i = \sigma_j\}.$$

► **Example 3.** The language

$$L_{\text{first}} = \{\sigma_1\sigma_2 \cdots \sigma_n : \text{for all } 1 < i \leq n, \sigma_1 \neq \sigma_i\}$$

is accepted by the automaton \mathbf{B} in Figure 2.



■ **Figure 2** The graph representation of \mathbf{B} .

The pumping lemmas in this paper involve the following definitions.

► **Definition 4.** The order of a permutation $\alpha : \Sigma \rightarrow \Sigma$ is the smallest positive integer k such that α^k is the identity permutation: $\alpha^k = \text{id}$,³ if no such k exists, the order of α is infinite. Moreover, we say that α is a Δ -permutation if it is invariant on Δ (i.e., for all $\delta \in \Delta$, $\alpha(\delta) = \delta$).

² Since $r = 1$, the register component of the transition functions is omitted.

³ As usual, the product of two permutations is their composition and powers defined recursively by $\alpha^k = \alpha \circ \alpha^{k-1}$.

► **Definition 5.** Let α be a permutation of Σ . We extend α to configurations by $\alpha(s, \theta) = (s, \alpha(\theta))$ and then to finite sets of configurations by $\alpha(C) = \{\alpha(c) : c \in C\}$.

► **Proposition 6** (Invariance of FMA). Let α be a Δ -permutation of Σ , $\sigma \in \Sigma^*$, and let c and c' be configurations of \mathbf{A} such that $c \xrightarrow{\sigma} c'$. Then $\alpha(c) \xrightarrow{\alpha(\sigma)} \alpha(c')$.

The proof of Proposition 6 is similar to that of [19, Lemma 1] and is omitted.

3 Pumping lemma for finite-memory automata

It has been shown in [19, Example 3] that the pumping lemma for regular languages [29, Lemma 8] does not hold for the quasi-regular ones. A variant of the pumping lemma for quasi-regular languages was later published in [26], although it had previously been obtained independently by two groups [22].⁴ The pumping lemma presented below refines these results in which

- the pumped patterns are generated by a fixed finite-ordered permutation, and
- an explicit upper bound is given on the order of that permutation.

► **Theorem 7.** Let $\mathbf{A} = \langle S, s_0, F, \Delta, \theta_0, \mu_\Delta, \mu_-, \mu_{\neq \text{skip}}, \mu_{\neq \text{replace}} \rangle$ be an r -register finite-memory automaton. Then for every word $\sigma \in L(\mathbf{A})$ and every subword χ of σ , $\sigma = \psi\chi\omega$, $|\chi| > \|S\|$, there exists a decomposition $\chi = \tau\nu\varphi$ and a Δ -permutation α of order at most $(2r)!$ such that

- $0 < |\nu| \leq \|S\|$,
- for all $k = 1, 2, \dots$,

$$\psi\tau\nu\alpha(\nu)\alpha^2(\nu)\cdots\alpha^k(\nu)\alpha^k(\varphi\omega) \in L(\mathbf{A}), \quad (1)$$

- and

$$\psi\tau\alpha^{-1}(\varphi\omega) \in L(\mathbf{A}). \quad (2)$$

Even though the classical pumping lemma does not hold for quasi-regular languages, as shows Corollary 8 below, some sufficiently long words in a quasi-regular language possess periodicity properties.

► **Corollary 8.** If L is an unbounded quasi-regular language, then there are words σ', σ'' , and σ''' such that for all $k = 0, 1, \dots$,

$$\sigma'\sigma''^k\sigma''' \in L.$$

Proof. Let $\sigma \in L$ be a sufficiently long word. In the notation in the statement of Theorem 7, $\sigma' = \psi\tau\nu$, $\sigma'' = \alpha(\nu)\alpha^2(\nu)\cdots\alpha^i(\nu)$, and $\sigma''' = \alpha^i(\varphi\omega) = \varphi\omega$, where i is the order of α . ◀

4 Alternating finite-memory automata

In this section, we recall the definition of alternating finite-memory automata and state some of its basic properties. We restrict ourselves to one-register automata with which we deal in this paper.

⁴ The author in [22] also posed the question of whether their pumping lemma extends to alternating automata. However, it does not apply directly to this setting; see the discussion in Section 5.1.

► **Definition 9.** An alternating one-register finite-memory automaton (over Σ) is a system $\mathbf{A} = \langle S, s_0, F, \Delta, \theta_0, \mu_\Delta, \mu_-, \mu_\neq \rangle$ whose components are like in Definition 1 except that the transition functions are as follows:

- $\mu_\Delta : S \times \Delta \rightarrow 2^{2^S}$,
- $\mu_- : S \rightarrow 2^{2^S}$, and
- $\mu_\neq : S \rightarrow 2^{(2^S)^2}$.

The intuitive meaning of these functions is as follows. Let \mathbf{A} read a letter σ being in state s with a letter θ stored in the register.

- If $\sigma \in \Delta$, then, for some $Q \in \mu_\Delta(s, \sigma)$, the computation splits to the computations from all states in Q with θ in the register.
- If $\sigma = \theta$, then, for some $Q \in \mu_-(s)$, the computation splits to the computations from all states in Q with the same θ in the register.
- If $\sigma \notin \Delta \cup \{\theta\}$, then, for some $(Q', Q'') \in \mu_\neq(s)$, the computation splits to the computations from all states in Q' with θ in the register and the computations from all states in Q'' with the current input letter σ in the register.⁵

Like in the case of (non-alternating) finite-memory automata, a *configuration* of \mathbf{A} is a pair in $S \times (\Sigma \setminus \Delta)$.

The transition functions μ_Δ , μ_- , and μ_\neq induce the following transition function μ^c on a configuration (s, θ) and an input letter σ , as follows,

- If $\sigma \in \Delta$, then $\mu^c((s, \theta), \sigma) = \{Q \times \{\theta\} : Q \in \mu_\Delta(s, \sigma)\}$.
- If $\sigma = \theta$, then $\mu^c((s, \theta), \sigma) = \{Q \times \{\theta\} : Q \in \mu_-(s)\}$.
- If $\sigma \notin \Delta \cup \{\theta\}$, then $\mu^c((s, \theta), \sigma) = \{Q' \times \{\theta\} \cup Q'' \times \{\sigma\} : (Q', Q'') \in \mu_\neq(s)\}$.

► **Remark 10.** Note that $\mu^c((s, \theta), \sigma)$ is a finite set of finite sets of configurations.

Next, we extend μ^c to finite sets of configurations in the standard “alternating” manner. Let $C = \{c_1, c_2, \dots, c_k\}$ be a set of configurations, $\sigma \in \Sigma$, and let $\mu^c(c_i, \sigma) = \{C_{i,1}, C_{i,2}, \dots, C_{i,m_i}\}$, $i = 1, 2, \dots, k$. Then $\mu^c(C, \sigma)$ consists of all finite sets of configurations of the form $\bigcup_{i=1}^k C_{i,j_i}$, $j_i = 1, 2, \dots, m_i$. That is, $C' \in \mu^c(C, \sigma)$, if there is a set of finite sets of configurations $\{C_{1,j_1}, C_{2,j_2}, \dots, C_{k,j_k}\}$, $C_{i,j_i} \in \mu^c(c_i, \sigma)$, $i = 1, 2, \dots, k$, such that $C' = \bigcup_{i=1}^k C_{i,j_i}$. Like in the case of (non-alternating) finite-memory automata, we write $C \xrightarrow{\sigma} C'$ for $C' \in \mu^c(C, \sigma)$.

Now, we can define the notion of a *run* of \mathbf{A} . Let $\sigma = \sigma_1 \sigma_2 \dots \sigma_n \in \Sigma^*$. A *run* of \mathbf{A} on σ is a sequence of finite sets of configurations C_0, C_1, \dots, C_n , where

- $C_0 = \{(s_0, \theta_0)\}$ and
- $C_i \xrightarrow{\sigma_{i+1}} C_{i+1}$, $i = 0, 1, \dots, n-1$.

That is $C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} C_2 \xrightarrow{\sigma_3} \dots \xrightarrow{\sigma_n} C_n$ and, like in the case of (non-alternating) finite-memory automata, we also write $C_0 \xrightarrow{\sigma} C_n$. This run is *accepting*, if $C_n \subseteq F \times \Sigma$, i.e., C_n is a set of *accepting* configurations, and the automaton *accepts* a word $\sigma \in \Sigma^*$, if there is an accepting run of \mathbf{A} on σ . The language $L(\mathbf{A})$ consists of all words accepted by \mathbf{A} .

⁵ That is, the components Q' and Q'' of pair (Q', Q'') correspond to the values of the transition functions $\mu_{\neq_{\text{skip}}}$ and $\mu_{\neq_{\text{replace}}}$ from Definition 1, respectively.

► **Example 11.** The language $L_{\text{diff}} = \{\sigma_1\sigma_2\cdots\sigma_n \in \Sigma^* : \text{for all } i, j \text{ if } i \neq j, \text{ then } \sigma_i \neq \sigma_j\}$, that is the complement of the quasi-regular language L from Example 2, is not quasi-regular, because it is an unbounded language not containing words with periodic patterns, in contradiction with Corollary 8 (see also [19, Example 5]). However, it is easy to see that L_{diff} is accepted by a one-register alternating finite-memory automaton.

► **Example 12.** Let $L_{\subseteq} = \{\psi\#\omega : \psi, \omega \in L_{\text{diff}} \text{ and } [\psi] \subseteq [\omega] \subset \Sigma \setminus \{\#\}\}$. This language is accepted by an alternating one-register finite-memory automaton. The automaton verifies that the letter $\#$ appears exactly once in the input, that the prefix before $\#$ and the suffix after $\#$ do not contain repeated letters, and that every letter in the prefix also appears in the suffix.

► **Example 13.** The language $L_{\text{last}} = \{\sigma_1\sigma_2\cdots\sigma_n : \text{for all } 1 \leq i < n, \sigma_i \neq \sigma_n\}$, that is the reversal of L_{first} from Example 3, is accepted by rejecting all the words which contain a letter equal to the last one.

► **Proposition 14.** *Let α be a Δ -permutation of Σ . Then $C \in \mu^c(c, \sigma)$ implies $\alpha(C) \in \mu^c(\alpha(c), \alpha(\sigma))$.*

The proof of Proposition 14 is similar to that of [19, Lemma 1] and is omitted. As a corollary, every run of the automata is mapped by permutations to an “equivalent” run.

► **Corollary 15.** *Let C, C' be finite sets of configurations, $\sigma \in \Sigma^*$ be such that $C \xrightarrow{\sigma} C'$ and let α be a Δ -permutation of Σ . Then $\alpha(C) \xrightarrow{\alpha(\sigma)} \alpha(C')$.*

5 Pumping lemma for alternating finite-memory automata

Our pumping lemma for alternating finite-memory automata is as follows.

► **Theorem 16.** *Let \mathbf{A} be a one-register alternating finite-memory automaton. There is a computable constant $N_{\mathbf{A}}$ such that, for every word $\sigma \in L(\mathbf{A})$ that is longer than $N_{\mathbf{A}}$, there exists a decomposition $\sigma = \tau\nu\varphi$ of σ and a Δ -permutation α of Σ such that*

- $|\nu\varphi| \leq N_{\mathbf{A}}$,
- $|\nu| > 0$, and
- for all $k = 1, 2, \dots$,

$$\tau\nu\alpha(\nu)\alpha^2(\nu)\cdots\alpha^k(\nu)\alpha^k(\varphi) \in L(\mathbf{A}). \quad (3)$$

5.1 Theorem 16 vs. Theorem 7

Note that the statement of Theorem 16 is weaker than that of Theorem 7. This is for the following three reasons, see the examples below which illustrate the limitations of the classical approach for languages over infinite alphabets.

- The alphabet permutation is not necessarily of a finite order;
- pumping is not possible in every sufficiently long pattern of σ ; and
- “shrinking” the word as in (2) is not always possible.⁶

First, we demonstrate the impossibility of pumping certain patterns as they are, without any modification.

⁶ Actually, shrinking is possible in the prefix (and not the suffix!) of any sufficiently long word, that is by Lemmas 14 and 16 in [15].

► **Example 17.** The language L_{diff} from Example 11 is unbounded, but no word in it can be pumped in the classical sense, as this would imply a letter repetition, violating the language's defining property. For the same reason, it follows from Corollary 8 that no word in L_{diff} can be pumped by any finite order permutation of Σ .

The following (and a bit longer) example illustrates the necessity of both the permutation and the restriction to a bounded suffix. General pumping with permutations in an arbitrary pattern, like in Theorem 7, is not always possible.

► **Example 18.** The language L_{\subseteq} , from Example 12, does not satisfy the pumping lemma with permutations applied arbitrarily within the word, which can be shown as follows.

For $\psi = \psi_1\psi_2 \cdots \psi_m \in L_{\text{diff}}$, the word $\sigma = \psi\#\psi^R$, where $\psi^R = \psi_m\psi_{m-1} \cdots \psi_1$ is the reversal of ψ , belongs to L_{\subseteq} .

We claim that if a subword can be repeatedly pumped by means of some permutation, it must be contained within the suffix $\#\psi^R$ of σ .

Assume to the contrary that there is a decomposition $\sigma = \psi\#\psi^R = \tau\nu\varphi$ with $|\nu| > 0$ and a Δ -permutation α such that $\sigma_1 = \tau\nu\alpha(\nu)\alpha(\varphi) \in L_{\subseteq}$ and ν is not a subword of the suffix $\#\psi^R$ of σ .

We distinguish between the cases of $\alpha(\#) = \#$ and $\alpha(\#) \neq \#$.

In the former case, ν is a subword of the prefix ψ of σ , because, otherwise, in contradiction with the definition of L_{\subseteq} , $\#$ would appear twice in $\tau\nu\alpha(\nu)\alpha(\varphi)$. Thus, the prefix of $\tau\nu\alpha(\nu)\alpha(\varphi)$ up to $\#$ is longer than ψ , which, again, contradicts the definition of L_{\subseteq} .

In the latter case, the suffix $\alpha(\nu)\alpha(\varphi)$ of σ_1 contains the pattern $\alpha(\psi_m)\alpha(\#)\alpha(\psi_m)$ in which two occurrences of $\alpha(\psi_m)$ are not separated by $\#$. This, however, contradicts the definition of L_{\subseteq} .

The last example in this section shows that the pumping lemma for alternating finite-memory automata cannot be strengthened with (2).

► **Example 19.** For σ from Example 18, the pumping can be applied only in the suffix $\#\psi^R$ of σ and the suffix of $\tau\alpha^{-1}(\varphi)$ beginning at $\#$ is shorter than ψ .

6 Some applications of Theorem 16

As the section title suggests, this section contains a number of applications of Theorem 16.

6.1 Disproving membership

In this section, we apply Theorem 16 to show that a language is not accepted by a one-register alternating finite-memory automaton. Namely, we use the pumping lemma to show three negative closure results on the class of languages accepted by these automata. These negative results are not unexpected, because runs of alternating automata from universal states are independent.⁷

► **Example 20** (Cf. Example 12). The language

$$L_{\supseteq} = \{\psi\#\omega : \psi, \omega \in L_{\text{diff}}, [\psi] \supseteq [\omega], \text{ and } [\psi], [\omega] \subset \Sigma \setminus \{\#\}\}, \quad (4)$$

where L_{diff} is the language from Example 11, is not accepted by a one-register alternating finite-memory automaton.

⁷ For the same reason, the proof of Theorem 16 is rather involved.

For the proof, assume to the contrary that, for some one-register alternating finite-memory automaton \mathbf{A} , $L_{\supseteq} = L(\mathbf{A})$.

Let $\psi \in L_{\text{diff}}$ be such that $[\psi] \subset \Sigma \setminus \{\#\}$ and $|\psi| > N_{\mathbf{A}}$. By the definition of L_{\supseteq} , it contains $\sigma = \psi\#\psi$. Let $\psi\#\psi = \tau\nu\varphi$, $\tau = \tau'\#\tau''$ be the decomposition of σ and α be the Δ -permutation provided by Theorem 16. Then, by that theorem,

$$\sigma' = \tau'\#\tau''\nu\alpha(\nu)\alpha(\varphi) \in L_{\supseteq},$$

notice the following length comparison between the prefix and suffix of the word σ' ,

$$|\tau''\nu\alpha(\nu)\alpha(\varphi)| = |\tau''\nu\alpha(\varphi)| + |\alpha(\nu)| = |\psi| + |\nu| > |\psi| \geq |\tau'|. \quad (5)$$

However, since $\sigma' \in L_{\supseteq}$, the prefix and the suffix of σ' contain distinct letters and they satisfy $[\tau''\nu\alpha(\nu)\alpha(\varphi)] \subseteq [\tau']$, this contradicts (5).

Since L_{\supseteq} is the reversal of the language L_{\subseteq} from Example 12, the class of languages accepted by one-register alternating finite-memory automata is not closed under reversal.⁸

► **Example 21.** The concatenation $L_{\text{co}} = L_{\text{last}} \cdot L_{\text{first}}$ of the languages from Examples 3 and 13 is not accepted by a one-register alternating finite-memory automaton.

For the proof, consider the language

$$L_{2\text{diff}} = \{\psi\#\#\omega : \psi, \omega \in L_{\text{diff}} \text{ and } [\psi], [\omega] \subseteq \Sigma \setminus \{\#\}\},$$

that is accepted by a one-register alternating finite-memory automaton similar to that in Example 12.

Were L_{co} accepted by a one-register alternating finite-memory automaton, the language

$$L_{=} = L_{2\text{diff}} \cap \overline{L_{\text{co}}}$$

would be accepted as well, because the class of the languages accepted by such automata is closed under boolean operations.

Since $\overline{L_{\text{co}}}$ consists of all words $\sigma_1 \cdots \sigma_m$ such that for all $i = 1, 2, \dots, m$ (cf. [28]) either $\sigma_{i-1} \in [\sigma_1 \cdots \sigma_{i-2}]$ or $\sigma_i \in [\sigma_{i+1} \cdots \sigma_m]$,

$$L_{=} = \{\psi\#\#\omega : \psi, \omega \in L_{\text{diff}}, [\psi], [\omega] \subset \Sigma \setminus \{\#\}, \text{ and } [\psi] = [\omega]\}. \quad (6)$$

The inclusion \subseteq of (6) is immediate and, for the converse inclusion, let

$$\psi\#\#\omega = \sigma_1 \cdots \sigma_m \in L_{=}$$

and let $\sigma \in [\psi]$. Then, $\sigma = \sigma_i$, for some $i = 1, 2, \dots, |\psi|$. Since $\psi \in L_{\text{diff}}$, $\sigma_{i-1} \notin [\sigma_1 \cdots \sigma_{i-2}]$, implying $\sigma_i \in [\sigma_{i+1} \cdots \sigma_m]$, which, together with $\psi \in L_{\text{diff}}$ and $\sigma \neq \#$, in turn, implies $\sigma_i \in [\omega]$. Therefore, $[\psi] \subseteq [\omega]$. The case of $\sigma \in [\omega]$ is similar to the above and is omitted. It follows that $|\psi| = |\omega|$ and, like in the previous example, one can show that $L_{=}$ is not recognizable by a one-register alternating finite-memory automaton.

Thus, the class of languages accepted by one-register alternating finite-memory automata is not closed under concatenation.

⁸ Actually, the non-closure under reversal has been already established in [3, Exercise 40]. However, the proof there is rather involved and relies on simulating runs of Minsky machines.

29:10 Pumping Lemma

► **Example 22.** Consider the language

$$L_{\text{first}\$\text{last}} = \{\psi\$\omega : \psi \in L_{\text{first}}, \omega \in L_{\text{last}}, \text{ and } [\psi], [\omega] \subseteq \Sigma \setminus \{\$\}\},$$

delimiting the language L_{co} from Example 21. This delimiting language is accepted by a one-register alternating finite-memory automaton because of its specified delimiter. However, the language $L_{\text{Kleene}} = (L_{\text{first}\$\text{last}})^*$ is not accepted by a one-register alternating finite-memory automaton.

For the proof, consider the language $L_{\&\$\$\&} = \{\&\$\sigma\$\& : \sigma \in \Sigma^*\}$ that is accepted by a one-register finite-memory automata (no registers are really required). The intersection of L_{Kleene} with $L_{\&\$\$\&}$ is $\{\&\$\}\cdot L_{\text{co}}\cdot\{\$\&\}$.

Were L_{Kleene} accepted by a one-register alternating finite-memory automaton, the language L_{co} would be also accepted by such an automaton, which is not possible by Example 21.

Thus, the class of languages accepted by one-register alternating finite-memory automata is not closed under the Kleene star.

6.2 Semi-linearity

In this section, we show that the set of lengths of words accepted by a one-register alternating finite-memory automaton is semi-linear and can be effectively described. The proof is based on the pumping lemma and a reduction theorem from [15]. The argument is general and is not limited to languages over infinite alphabets: it can be applied to any computational model that possesses similar properties of pumping and periodicity.

► **Definition 23.** A set of nonnegative integers is *linear* if it is of the form $\{a + ib : i \in \mathbb{N}\}$, for some nonnegative integers a and b . A set of nonnegative integers is *semi-linear* if it is a finite union of linear sets and a language L is called *semi-linear* if the set

$$|L| = \{|\sigma| : \sigma \in L\}$$

is *semilinear*.

► **Remark 24.** Every linear set $T = \{a + ib : i \in \mathbb{N}\}$ is either of cardinality one (if $b = 0$) or is of positive natural density $\frac{1}{b}$, otherwise. Thus, every semi-linear set is either finite or has positive natural density.

► **Theorem 25.** Let \mathbf{A} be a one-register alternating finite-memory automaton. Then, the set $|L(\mathbf{A})|$ is semi-linear and can be described effectively.

► **Lemma 26.** For every $\sigma \in L(\mathbf{A})$ with $|\sigma| \geq N_{\mathbf{A}}$, where $N_{\mathbf{A}}$ is the constant provided by Theorem 16, there is a word $\sigma' \in L(\mathbf{A})$ such that $|\sigma'| = |\sigma| + N_{\mathbf{A}}!$.

Proof. Let $\sigma \in L(\mathbf{A})$ be such that $|\sigma| \geq N_{\mathbf{A}}$. By Theorem 16, there is a decomposition $\sigma = \tau v \varphi$, $|\varphi| \leq N_{\mathbf{A}}$ and Δ -permutation α such that for every $k = 1, 2, \dots$,

$$\sigma_k = \tau v \alpha(v) \alpha^2(v) \cdots \alpha^k(v) \alpha^k(\varphi) \in L(\mathbf{A}).$$

Note that $|\sigma_k| = |\sigma| + k|\varphi|$ and $0 < |\varphi| \leq |\varphi| \leq N_{\mathbf{A}}$. Then, $\frac{N_{\mathbf{A}}!}{|\varphi|}$ is a positive integer and, for $t = \frac{N_{\mathbf{A}}!}{|\varphi|}$, $|\sigma_t| = |\sigma| + N_{\mathbf{A}}!$. ◀

Proof of Theorem 25. For $i = 0, 1, \dots, N_{\mathbf{A}}! - 1$, let the language L_i be defined by

$$L_i = L(\mathbf{A}) \cap \Sigma^{N_{\mathbf{A}}!+i} (\Sigma^{N_{\mathbf{A}}!})^*$$

and let

$$L_{N_{\mathbf{A}}!} = L(\mathbf{A}) \cap \bigcup_{i=0}^{N_{\mathbf{A}}!-1} \Sigma^i.$$

Each such language is accepted by a one-register alternating finite-memory automata resulting from \mathbf{A} in adding to it new states for counting the input word length (and the corresponding transitions, of course).

Obviously,

$$L(\mathbf{A}) = \bigcup_{i=0}^{N_{\mathbf{A}}!} L_i.$$

Since $L_{N_{\mathbf{A}}!}$ is finite, for the proof of Theorem 25 it suffices to show that, for each $i = 0, 1, \dots, N_{\mathbf{A}} - 1$, the language L_i is linear. This is immediate, if L_i is empty.

So, assume that $L_i \neq \emptyset$ and let σ_i be a word in L_i of the minimum length. Then, it follows from Lemma 26, by a straightforward induction, that

$$|L_i| = \{|\sigma_i| + jN_{\mathbf{A}}! : j \in \mathbb{N}\},$$

Finally, by [15, Theorem 1], the emptiness of L_i , $i = 0, 1, \dots, N_{\mathbf{A}} - 1$, is decidable and, for a non-empty L_i , $|\sigma_i| = i + N_{\mathbf{A}}!(k + 1)$, where k is the minimum integer for which $L(\mathbf{A}) \cap \Sigma^{i+N_{\mathbf{A}}!(k+1)}$ is non-empty. \blacktriangleleft

The following corollary to Theorem 25 is immediate.

► **Corollary 27** (Cf. [8, Theorem 1]). *For a one-register alternating finite-memory automaton \mathbf{A} , it is decidable whether $|L(\mathbf{A})|$ is bounded.*⁹

A non-semilinear languages accepted by a two-register alternating finite-memory automaton

Consider the following language,

$$L_{\#} = \{\sigma_1\#\sigma_1\sigma_2\#\sigma_1\sigma_2\sigma_3\#\dots\#\sigma_1\sigma_2\dots\sigma_n\# \in \Sigma^* : \sigma_1\sigma_2\dots\sigma_n \in L_{\text{diff}} \text{ and } \# \notin [\sigma_1\dots\sigma_n]\}.$$

It can be readily seen that $L_{\#}$ is the intersection of the languages L_i , $i = 1, 2, \dots, 7$, defined below.

- L_1 consists of all words whose first letter is not $\#$.
- L_2 consists of all words whose last letter is $\#$.
- L_3 consists of all words whose second letter is $\#$.
- L_4 consists of all words in which the letter following every non-last $\#$ is the same as the first letter.
- L_5 consists of all words in which no letter appears more than once between two consecutive $\#$ s.

⁹ This is an infinite alphabet counterpart of *finiteness* of languages over a finite alphabet.

29:12 Pumping Lemma

- L_6 consists of all words in which two letters which are consecutive once, are always consecutive.
- L_7 consists of all words in which the next appearance of every letter followed by a non-last $\#$ is followed by $\sigma\#$ for some $\sigma \in \Sigma$.

Obviously, languages L_1, L_2 , and L_3 are accepted by a one-register (deterministic) finite-memory automata.

Language L_4 is accepted by a one-register (deterministic) finite-memory automaton that “remembers” the first letter of the input word and verifies that, after every $\#$, the next letter matches the content of the register.

The complement of L_5 is accepted by a one-register finite-memory automaton that “guesses” and remembers a letter and accepts, if it appears again before $\#$.

The complement of L_6 is accepted by a two-register finite-memory automaton that nondeterministically selects two consecutive letters, remembers them, and accepts, if at one of the future appearances of the first letter, it is followed by a letter different from the second one.

Finally, the complement of L_7 is accepted by a one-register finite-memory automaton that nondeterministically selects a letter, verifies that it is followed by $\#$, and accepts, if the next appearance of the selected letter is followed by $\#$ or is followed by two non- $\#$.

Overall, since, for any r , the class of languages accepted by r -register alternating finite-memory automata is closed under Boolean operations, $L_\#$ is accepted by a two-register alternating finite-memory automaton (with the distinguished letter $\#$). In fact, this language is even lower on the hierarchy, it is simply co-nondeterministic (universal).

However, the set of lengths of words in $L_\#$ is

$$|L_\#| = \left\{ \frac{n^2 + 3n}{2} : n \in \mathbb{N} \right\}.$$

This is an infinite set of natural density zero. Thus, by Remark 24, it is not semi-linear.

In particular, Theorem 16 does not extend onto general alternating finite-memory automata.

7 Proof road-map of Theorem 16

In this section, we provide a high-level overview of the proof strategy, highlighting the key techniques and the main challenges involved.

The core difference in proving a pumping lemma for languages over infinite alphabets stems from the unbounded variability of symbols. Unlike classical finite-alphabet automata, we cannot rely on repetitions of identical configurations to enable pumping.

Suppose, ideally, that a run contains two sets of configurations where the latter is a subset of the former. In this case, the subword between them could be pumped directly. However, such inclusion is rare, especially as transitions may introduce fresh symbols into the configurations. This obstacle motivates the need for a more flexible framework for comparing configurations.

The key idea, originally introduced in [19, Appendix A], is to shift from reasoning about concrete sets of configurations to a more structured representation: nonnegative integer vectors. Specifically, we map finite sets of configurations to vectors in $\mathbb{N}^{2^{\|\Sigma\|}-1}$, where each coordinate corresponds to a nonempty set of states and counts how many distinct symbols are currently associated with that set.

As a result, we obtain a sequence of $(2^{\|S\|} - 1)$ -dimensional vectors v_0, v_1, \dots and seek a pair of indices $i < j$ such that v_j is component-wise less than or equal to v_i . If such a pair exists, we can find a permutation that maps the symbols of the configuration set related to v_j to those of the configuration set related to v_i , while maintaining the correspondence between the respective sets of states they are associated with. Then, the corresponding pattern can be pumped iteratively under that permutation.

Here another (major) problem arises: in an arbitrary sequence v_0, v_1, \dots of natural-valued vectors, such pair v_i, v_j does not necessarily exist. Therefore, techniques of forward analysis of well-structured transition systems do not apply [13, 14, 2]. This motivates a fundamental shift in perspective. Instead of analyzing the computation forward, we analyze the computation *backward*, from the final set of configurations toward the initial one.

Reversing the computation introduces a new complication: the number of symbols in the final configuration depends on the input length and, therefore, is unbounded. To address this, we restrict the sets of configurations to only those involving symbols in the suffix of the input. This pruning retains all relevant information for acceptance while ensuring that each configuration involves only symbols drawn from a bounded domain. As a result, the associated vector sequence becomes linearly bounded in its size.

This boundedness is crucial: it places the vector sequence within a well-quasi-ordered space, which guarantees the existence of such a pair of comparable vectors, corresponding to a “pumpable” structure in the original computation.

Our backward analysis technique refines the approach in [8], where configuration sets were truncated with an integer cap. Here, instead we restrict the configuration sets to a finite domain, the set of symbols drawn from the suffix of the word.

The proof of Theorem 16 itself is composed of finding sufficient conditions for a pattern to be pumped. The sufficient condition we need is established step by step. First, we show that pumping is possible, if we already have an appropriate Δ -permutation of Σ and then, we show that such a permutation exists, if the automaton run meets certain requirements. Finally, we prove that any sufficiently long run meets this requirement, thereby completing the proof of Theorem 16. In the following sections we state the main lemmas associated with each of these three steps.

Notation

Following [15], we introduce the notation below. For a finite set C of configurations of \mathbf{A} , we denote by $\Sigma(C)$ the following subset of $\Sigma \setminus \Delta$.

$$\Sigma(C) = \{\sigma \in \Sigma \setminus \Delta : \text{for some } s \in S, (s, \sigma) \in C\}. \quad (7)$$

Consider the relation \equiv_C on $\Sigma(C)$ such that $\sigma \equiv_C \sigma'$ if and only if the following holds.

- For each $s \in S$, $(s, \sigma) \in C$ if and only if $(s, \sigma') \in C$.

Obviously, \equiv_C is an equivalence relation. The equivalence classes of \equiv_C can be described as follows. Let $\sigma \in \Sigma \setminus \Delta$ and let the subset $S^\sigma(C)$ of S be defined by

$$S^\sigma(C) = \{s : (s, \sigma) \in C\}.$$

Then $\sigma \equiv_C \sigma'$ if and only if $S^\sigma(C) = S^{\sigma'}(C)$.

For a subset Σ' of Σ , we define the set of states $S^{\Sigma'}(C)$

$$S^{\Sigma'}(C) = \bigcup_{\sigma \in \Sigma'} S^\sigma(C) \left(= \bigcup_{\sigma \in \Sigma' \cap \Sigma(C)} S^\sigma(C) \right). \quad (8)$$

29:14 Pumping Lemma

Next, let Q be a nonempty subset of S and let the subset $\Sigma^Q(C)$ of $\Sigma(C)$ be defined by

$$\Sigma^Q(C) = \{\sigma : S^\sigma(C) = Q\}. \quad (9)$$

It follows from (7) and (9) that

$$\Sigma(C) = \bigcup_{Q \subseteq 2^S \setminus \{\emptyset\}} \Sigma^Q(C) \quad (10)$$

and the equivalence classes of \equiv_C are in one-to-one correspondence with those subsets Q of S for which $\Sigma^Q(C)$ is nonempty.

7.1 Partial order on sets of configuration

The first step of the proof involves the definition below.

► **Definition 28.** Let C_1, C_2 be finite sets of configurations, $\Sigma_1, \Sigma_2 \subseteq \Sigma$ (not necessarily finite), and let α be a Δ -permutation of Σ . We write

$$C_1, \Sigma_1 \preceq_\alpha C_2, \Sigma_2, \quad (11)$$

if

- (i) $\Sigma_1 \subseteq \alpha(\Sigma_2)$;
- (ii) $\alpha(\Sigma_2) \cap \Sigma(C_1) \subseteq \Sigma_1$;
- (iii) for all $\sigma \in \Sigma_1$, $S^\sigma(C_1) \subseteq S^{\alpha^{-1}(\sigma)}(C_2)$; and
- (iv) $S^{\Sigma \setminus \Sigma_1}(C_1) \subseteq S^{\Sigma \setminus \Sigma_2}(C_2)$.

The intuition behind this definition arises from the need to embed C_1 , associated with Σ_1 , into C_2 associated with Σ_2 . This embedding is mediated by the Δ -permutation α , that plays a central role in the pumping construction described in the proof overview.

Even though, the actual embedding is performed via α^{-1} , we state the relation in the terms of α , because α is the permutation that will act on the pattern in pumping.

Intuitively, clause (i) ensures that the embedding is well-defined, while clause (ii) prevents “overshooting” the intended domain. Clause (iii) guarantees that the structural information is preserved under the symbol transformation, and clause (iv) ensures consistency of symbols outside the specified domains.

► **Remark 29.** By clauses (iii) and (iv) of the above definition, (11) implies $S^\Sigma(C_1) \subseteq S^\Sigma(C_2)$. In particular, if C_2 is a set of accepting configurations, then so is C_1 .

If there is an accepting run containing two configuration sets such that the previous including the subsequent (with respect to the domains of the symbols in the suffix, by means of Definition 28), then we have a straightforward pumping.

► **Lemma 30.** Let $\sigma = \sigma_1\sigma_2 \cdots \sigma_m \in L(\mathbf{A})$, $C = C_0, C_1, \dots, C_m$ be an accepting run of \mathbf{A} on σ and α be a Δ -permutation and $\Sigma' \subseteq \Sigma$. Let $\sigma = \tau\mathbf{v}\varphi$, $|\mathbf{v}| > 0$, be a decomposition of σ such that,

$$C_{|\tau\mathbf{v}|}, \alpha(\Sigma') \preceq_\alpha C_{|\tau|}, \Sigma', \quad (12)$$

and

$$\alpha(\Sigma'), [\mathbf{v}\varphi] \subseteq \Sigma'. \quad (13)$$

Then, (3) is satisfied for all $k = 1, 2, \dots$

7.2 Permutation construction

The lemma below provides a sufficient condition for the prerequisites of Lemma 30.

► **Lemma 31.** *Let C_1 and C_2 be finite sets of configurations and let Σ_1 and Σ_2 be finite subsets of Σ such that*

$$\Sigma_1 \subseteq \Sigma_2, \quad (14)$$

for all nonempty subsets Q of S ,

$$\|\Sigma^Q(C_1) \cap \Sigma_1\| \leq \|\Sigma^Q(C_2) \cap \Sigma_2\|, \quad (15)$$

and

$$S^{\Sigma \setminus \Sigma_1}(C_1) = S^{\Sigma \setminus \Sigma_2}(C_2). \quad (16)$$

Then, there exists a Δ -permutation α and a subset $\Sigma' \subseteq \Sigma$ such that

$$C_1, \alpha(\Sigma') \preceq_\alpha C_2, \Sigma', \quad (17)$$

and

$$\alpha(\Sigma'), \Sigma_2 \subseteq \Sigma'. \quad (18)$$

7.3 Trace reversal sequences

In order to complete the proof, we shall need the notation below.

Let $\sigma = \sigma_1 \sigma_2 \cdots \sigma_m$ and let $\mathbf{C} = C_0, C_1, \dots, C_m$ be a run of \mathbf{A} on σ :

$$C_0 \xrightarrow{\sigma_1} C_1 \xrightarrow{\sigma_2} C_2 \xrightarrow{\sigma_3} \cdots \xrightarrow{\sigma_m} C_m. \quad (19)$$

and let the sequence $\Sigma_0, \Sigma_1, \dots, \Sigma_m$ of subsets of $[\sigma]$ be defined by

$$\Sigma_i = \begin{cases} \emptyset, & \text{if } i = 0 \\ [\sigma_{m-i+1} \sigma_{m-i+2} \cdots \sigma_m], & \text{otherwise} \end{cases}.$$

It follows that $\|\Sigma_i\| \leq i$ and, if $i < j$, then $\Sigma_i \subseteq \Sigma_j$.

For $i = 0, 1, \dots, m$, let

$$S_{\sigma, \mathbf{C}, i} = S^{\Sigma \setminus \Sigma_i}(C_{m-i}),$$

and let the function $f_{\sigma, \mathbf{C}, i} : 2^S \setminus \{\emptyset\} \rightarrow \{0, 1, \dots, i\}$ be defined by

$$f_{\sigma, \mathbf{C}, i}(Q) = \|\Sigma^Q(C_{m-i}) \cap \Sigma_i\|.$$

► **Definition 32.** *The sequence of pairs*

$$(S_{\sigma, \mathbf{C}, m}, f_{\sigma, \mathbf{C}, m}), (S_{\sigma, \mathbf{C}, m-1}, f_{\sigma, \mathbf{C}, m-1}), \dots, (S_{\sigma, \mathbf{C}, 0}, f_{\sigma, \mathbf{C}, 0})$$

corresponds to computation (19) and is called the trace of that computation. Respectively, the sequence of pairs

$$(S_{\sigma, \mathbf{C}, 0}, f_{\sigma, \mathbf{C}, 0}), (S_{\sigma, \mathbf{C}, 1}, f_{\sigma, \mathbf{C}, 1}), \dots, (S_{\sigma, \mathbf{C}, m}, f_{\sigma, \mathbf{C}, m}) \quad (20)$$

is called the computation trace reversal of (19).

29:16 Pumping Lemma

In the notation above, Lemmas 30 and 31 imply the following sufficient condition.

► **Corollary 33.** *Let $\sigma = \sigma_1\sigma_2\cdots\sigma_m \in L(\mathbf{A})$ and let $\mathbf{C} = C_0, C_1, \dots, C_m$ be an accepting run of \mathbf{A} on σ . If there are $i < j \leq m$ such that*

$$S_{\sigma, \mathbf{C}, i} = S_{\sigma, \mathbf{C}, j}, \quad (21)$$

and

$$f_{\sigma, \mathbf{C}, i} \leq f_{\sigma, \mathbf{C}, j}. \quad (22)$$

Then, there is Δ -permutation α such that (3) is satisfied for all $k = 1, 2, \dots$. Where the decomposition $\sigma = \tau\mathbf{v}\varphi$, is $\tau = \sigma_1 \cdots \sigma_{m-j}$, $\mathbf{v} = \sigma_{m-j+1} \cdots \sigma_{m-i}$ and $\varphi = \sigma_{m-i+1} \cdots \sigma_m$.

Now Theorem 16 follows from Corollary 33 and the theorem below.

► **Theorem 34.** *There is a computable constant $N_{\mathbf{A}}$ such that every computation trace reversal sequence of length greater than $N_{\mathbf{A}}$ contains a pair $i < j < N_{\mathbf{A}}$ satisfying (21) and (22).*

Theorem 34 is obtained via well-quasi-order theory, for the set $\mathbb{N}^{2^{\|\mathbf{S}\|}-1} \times 2^S$ with the product order of \leq for $\mathbb{N}^{2^{\|\mathbf{S}\|}-1}$ and $=$ for 2^S . Indeed, trace resembling sequences are bounded by a linear function and the proofs follow by Dickson's Lemma [10] and KM-tree introduced in [21], see [11, Section 7] for extended discussion and related complexity.¹⁰

8 Concluding remarks

In this paper, we introduced and proved a pumping-like lemma for languages over infinite alphabets, specifically those recognized by one-register alternating finite-memory automata. As a key corollary, we established that the set of word lengths in such languages is semilinear.

The central contribution lies not only in the results themselves but also in the underlying technique, a novel shift in perspective for analyzing transition systems that are not well-structured in the forward direction but can be controlled in reverse. This reverse-analysis allows us to recover a form of well-quasi-ordering by restricting attention to suffix-dependent symbol domains, enabling a form of pumping that respects the structural symmetries of infinite alphabets.

Complexity of computing $N_{\mathbf{A}}$. Most algorithms for computational models over unbounded domains (such as finite-memory automata, Petri nets, vector addition systems with states) have extremely high complexity, often beyond primitive-recursive bounds. These complexities are typically classified within the fast-growing hierarchy \mathfrak{F} [25].

In our case, for a fixed number of states, computing $N_{\mathbf{A}}$ is in \mathfrak{F}_k , where $k = 4^{\|\mathbf{S}\|} - 1$, see Footnote 10 and the main result in [11].

Furthermore, it can be readily seen that the constant $N_{\mathbf{A}}$ is at least as large as the constant N from [15, Lemma 18] (where the ordered set is only $\mathbb{N}^{2^{\|\mathbf{S}\|}-1}$). This constant N is used to decide the emptiness of one-register alternating finite-memory automata. The latter is known to be at least as hard as the emptiness of incrementing counter automata [9] which are complete for \mathfrak{F}_ω [31]. In particular, the function $\mathbf{A} \mapsto N_{\mathbf{A}}$ is not primitive recursive.

¹⁰In the notation of [11], $N_{\mathbf{A}} \leq L_{r, \tau} = L_{\tau'}$, where $r = 2^{\|\mathbf{S}\|} + 1$, $\tau = 2^{\|\mathbf{S}\|} - 1$ and $\tau' = r \times \tau = 4^{\|\mathbf{S}\|} - 1$.

Ordered alphabets. A natural question is whether our results can be extended to any orbit-finite alphabet [4]. For example, alphabets with some order relation. In that setting, one might consider replacing the notion of a Δ -permutation with that of an order-preserving permutation.

Here, the main technical challenge lies in the construction of an “infinite” order-preserving permutation that generates infinitely many ordered copies of names. This is generally not possible for arbitrary orders.

Consider, for example, the alphabet $\Sigma = (\mathbb{N}, \leq)$ and the language

$$L_{ord} = \{\sigma_1\sigma_2 \cdots \sigma_n : \text{for all } i < j, \sigma_i > \sigma_j\}$$

that is accepted by a one-register alternating automaton with order comparisons. This language is unbounded, yet it contains no pumpable patterns: for any positive integer N , the word $N \cdot (N - 1) \cdots 1$ cannot be pumped while preserving the strict decreasing order.

Nevertheless, if the underlying order is dense and total, then it is possible to construct an order-preserving permutation. In that setting, Higman’s lemma [16] can be applied to obtain a respective counterpart of Theorem 16.

Timed automata. Another intriguing and immediate question is whether a similar pumping lemma can be formulated for one-clock alternating timed automata [1, 24]. Extending our results to timed domains is nontrivial due to two complications in the translation between timed automata and finite-memory automata, as explored in [12].

First, the translation assumes an ordered alphabet, specifically (\mathbb{Q}, \leq) , that is dense and total, and, therefore, compatible with the techniques discussed above.

The second, and more significant, challenge is that the translation is not linear. Thus, it remains unclear whether languages accepted by a one-clock alternating timed automata have semilinear lengths.

References

- 1 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.
- 2 Michael Blondin, Alain Finkel, and Jean Goubault-Larrecq. Forward analysis for wsts, part III: karp-miller trees. *Log. Methods Comput. Sci.*, 16(2), 2020. doi:10.23638/LMCS-16(2:13)2020.
- 3 Mikołaj Bojanczyk. Slightly infinite sets. *A draft of a book available at <https://www.mimuw.edu.pl/~bojan/paper/atom-book>*, 2019.
- 4 Mikołaj Bojanczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3):1–44, 2014. doi:10.2168/LMCS-10(3:4)2014.
- 5 A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.
- 6 Taolue Chen, Fu Song, and Zhilin Wu. Formal reasoning on infinite data values: An ongoing quest. In Jonathan P. Bowen, Zhiming Liu, and Zili Zhang, editors, *Engineering Trustworthy Software Systems - Second International School, SETSS 2016, Chongqing, China, March 28 - April 2, 2016, Tutorial Lectures*, volume 10215 of *Lecture Notes in Computer Science*, pages 195–257, 2016. doi:10.1007/978-3-319-56841-6_6.
- 7 Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Determinisability of register and timed automata. *Logical Methods in Computer Science*, 18:Paper No. 9, 37, 2022. doi:10.46298/LMCS-18(2:9)2022.
- 8 Yoav Danieli and Michael Kaminski. Bounded languages over infinite alphabets. In Klaus Meer, Alexander Rabinovich, Elena Ravve, and Andrés Villaveces, editors, *Model Theory, Computer Science, and Graph Polynomials: Festschrift in Honor of Johann A. Makowsky*, pages 193–213. Springer Nature Switzerland, Cham, 2025. doi:10.1007/978-3-031-86319-6_15.

- 9 S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10, 2009. Article 16.
- 10 L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35:413–422, 1913.
- 11 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson’s lemma. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science*, pages 269–278. IEEE, 2011. doi:10.1109/LICS.2011.39.
- 12 Diego Figueira, Piotr Hofman, and Sławomir Lasota. Relating timed and register automata. *Mathematical Structures in Computer Science*, 26:993–1021, 2016. doi:10.1017/S0960129514000322.
- 13 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for wsts, part II: complete WSTS. *Log. Methods Comput. Sci.*, 8(3), 2012. doi:10.2168/LMCS-8(3:28)2012.
- 14 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for wsts, part I: completions. *Math. Struct. Comput. Sci.*, 30(7):752–832, 2020. doi:10.1017/S0960129520000195.
- 15 D. Genkin, M. Kaminski, and L. Peterfreund. A note on the emptiness problem for alternating finite-memory automata. *Theoretical Computer Science*, 526:97–107, 2014. doi:10.1016/J.TCS.2014.01.020.
- 16 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1):326–336, 1952.
- 17 Piotr Hofman, Marta Jucepczuk, Sławomir Lasota, and Mohnish Pattathurajan. Parikh’s theorem for infinite alphabets. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470626.
- 18 M. Kaminski and N. Francez. Finite-memory automata. In *Proceedings of the 31th Annual IEEE Symposium on Foundations of Computer Science*, pages 683–688, Los Alamitos, CA, 1990. IEEE Computer Society Press.
- 19 M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134:329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 20 Ahmet Kara. *Logics on data words: Expressivity, satisfiability, model checking*. PhD thesis, Technical University of Dortmund, Germany, 2016. URL: <https://hdl.handle.net/2003/35216>.
- 21 R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.
- 22 B. Klin. A pumping lemma for automata with atoms. <https://atoms.mimuw.edu.pl/?p=43/>, 26.03.2014. [accessed 08.08.2025].
- 23 Bartek Klin, Sławomir Lasota, and Szymon Toruńczyk. Nondeterministic and co-nondeterministic implies deterministic, for data languages. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021*, volume 12650 of *Lecture Notes in Computer Science*, pages 365–384. Springer, 2021. doi:10.1007/978-3-030-71995-1_19.
- 24 Sławomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Log.*, 9(2):10:1–10:27, 2008. doi:10.1145/1342991.1342994.
- 25 M. Löb and S. Wainer. Hierarchies of number-theoretic functions I, II: a correction. *Archive for Mathematical Logic*, 14(3-4):198–199, 1971.
- 26 Rindo Nakanishi, Yoshiaki Takata, and Hiroyuki Seki. Pumping lemmas for languages expressed by computational models with registers. *IEICE Transactions on Information and Systems*, 106:284–293, 2023. doi:10.1587/TRANSINF.2022FCP0004.
- 27 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5:403–435, 2004. doi:10.1145/1013560.1013562.
- 28 Alexander Okhotin. The dual of concatenation. *Theoretical Computer Science*, 345(2-3):425–447, 2005. doi:10.1016/J.TCS.2005.07.019.
- 29 M. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research Development*, 3:114–125, 1959. doi:10.1147/RD.32.0114.

- 30 H. Sakamoto and D. Ikeda. Intractability of decision problems for finite-memory automata. *Theoretical Computer Science*, 231:297–308, 2000. doi:10.1016/S0304-3975(99)00105-X.
- 31 Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *International Symposium on Mathematical Foundations of Computer Science*, 2010. URL: <https://api.semanticscholar.org/CorpusID:14086694>.
- 32 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006. doi:10.1007/11874683_3.
- 33 T. Tan. On pebble automata for data languages with decidable emptiness problem. *Journal of Computer and System Sciences*, 76:778–791, 2010. doi:10.1016/J.JCSS.2010.03.004.