Theoretical Computer Science 526 (2014) 97–107

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

A note on the emptiness problem for alternating finite-memory automata

Daniel Genkin *, Michael Kaminski, Liat Peterfreund

Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel

ARTICLE INFO

Article history:
Received 4 March 2013
Received in revised form 7 January 2014
Accepted 15 January 2014
Communicated by D. Perrin

Keywords:
Infinite alphabets
Alternating finite-memory automata
Alternating finite-memory tree automata
Emptiness problem

ABSTRACT

We present alternative relatively simple and self-contained proofs of decidability of the emptiness problems for one-register alternating finite-memory automata and one-register alternating finite-memory tree automata.

1. Introduction

Finite-memory automata [5,6] are a generalization of the classical Rabin–Scott finite-state automata [12] to *infinite* alphabets. In addition to a finite set of "proper" states, finite-memory automata are equipped with a finite set of registers which in any stage of a computation (automaton's run) contain a symbol from the infinite alphabet. By restricting the power of the automaton to comparing the input symbol with the contents of the registers and copying the input symbol to a register, without the ability to perform *any* operations, the automaton is only able to "remember" a finite set of input symbols. Thus, the languages accepted by finite-memory automata possess many of the desirable properties of regular languages and therefore, are referred to as *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 automorphisms of the infinite alphabet. Thus, the actual symbols occurring in the input are of no real significance. Only the initial and repetition patterns matter. This follows from the nature of the restriction to copying and comparison (reminiscent of *term-unification*). If a new symbol (i.e., one not in any register) is copied and later successfully compared, any other new symbol, appearing in the same position, would cause the *same* transitions.

This paper deals with the *alternating* versions of finite-memory automata [10,11] and finite-memory tree automata [4]. These automata are tightly related to the linear and branching temporal logics with the freeze quantifier, see [2,3] and [4], respectively. Whereas the emptiness problem for two-register alternating finite-memory automata is undecidable, see [11, Theorem 5.1], it was shown in [2,3] and [4], respectively, that the emptiness problem for both one-register alternating finite-memory automata and one-register alternating finite-memory tree automata is decidable.[1] The proofs in [2–4] are based on reduction to decidability of the empriness problem for *counter automata with incrementing errors* and are very

* Corresponding author.

[1] Decidability of the former problem implies decidability of the emptiness problem for *top-view* 2-pebble automata, see [13,14].

0304-3975/$ – see front matter
http://dx.doi.org/10.1016/j.tcs.2014.01.020

involved.[2] In this paper we present alternative relatively simple and self-contained proofs of decidability of the emptiness problems for one-register alternating finite-memory automata and for one-register alternating finite-memory tree automata.

Theorem 1. *The emptiness problem for one-register alternating finite-memory automata is decidable.*

Theorem 2. *The emptiness problem for one-register alternating finite-memory tree automata is decidable.*

The proofs of these theorems are modifications of the decidability of the universality problem in [6, Appendix A] that is, actually, the emptiness problem for *universal* alternating finite-memory automata, whose behavior is deterministic. For the proofs of the above theorems we extend the construction in [6] to a nondeterministic behavior of general alternating finite-memory automata and alternating finite-memory tree automata. Namely, we show that, if the language of a one-register alternating finite-memory automaton (respectively, a one-register alternating finite-memory tree automaton) is nonempty, then it contains a "short" word (respectively, a "small" tree) whose length (respectively, size) is computable. Also, similarly to the construction in [6, Appendix A], our proofs extend to a proof of decidability of inclusions of a quasi-regular language in the language of a one-register alternating finite-memory automaton and a tree automaton language in the language of a one-register alternating finite-memory tree automaton.

The rest of this paper is organized as follows. In the next section we recall the definition of one-register alternating finite-memory automata and prove its basic properties needed for the proof of Theorem 1. The proof of the theorem is presented in Section 3. In Section 4, we recall the definition of one-register alternating finite-memory tree automata. Finally, Section 5 contains the proof of Theorem 2.

2. Alternating finite-memory automata

In this section we recall the definition of alternating finite-memory automata and prove its basic properties needed for the proof of Theorem 1. We restrict ourselves to one-register automata with which we deal in this paper. Like in the classical finite alphabet case, these automata have a finite set of proper states in which the "real computation" is done. In addition, the automaton is equipped with a register storing a symbol from the infinite alphabet. We refer to the register as the "*window*," see [5,6].

Throughout this paper we use the following conventions.

- Σ is a fixed infinite alphabet.
- Symbols in Σ are denoted by σ (sometimes indexed or primed), τ, or δ.
- Bold low-case Greek letters $\boldsymbol{\sigma}$, $\boldsymbol{\sigma}'$, and $\boldsymbol{\sigma}''$ denote words over Σ.
- Symbols which occur in a word denoted by a boldface letter are always denoted by the same *non-boldface* letter with an appropriate subscript. For example, symbols which occur in $\boldsymbol{\sigma}'$ are denoted by σ'_i.

Definition 3. (Cf. [6, Definition 1].) A *one-window alternating finite-memory automaton* (over Σ) is a system $\boldsymbol{A} = \langle S, s_0, F, \Delta, \delta, \mu_\Delta, \mu_=, \mu_{\neq}\rangle$ whose components are as follows.

- S is a finite set of *states*.
- $s_0 \in S$ is the *initial state*.
- $F \subseteq S$ is the set of *accepting* states.
- $\Delta \subset \Sigma$ is a finite set of *distinguished* symbols.
- $\delta \in \Sigma \setminus \Delta$ is the *initial window assignment*.
- $\mu_\Delta : S \times \Delta \to 2^{2^S}$, $\mu_= : S \to 2^{2^S}$, and $\mu_{\neq} : S \to 2^{(2^S)^2}$ are the *transition* functions.
  The intuitive meaning of these functions is as follows. Let $\boldsymbol{A}$ read a symbol σ being in state s with a symbol τ stored in the window.
  - 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 τ in the window.
  - If $\sigma = \tau$, then, for some $Q \in \mu_=(s)$, the computation splits to the computations from all states in Q with the same τ in the window.
  - If $\sigma \notin \Delta \cup \{\tau\}$, then, for some $(Q', Q'') \in \mu_{\neq}(s)$, the computation splits to the computations from all states in Q' with τ in the window and the computations from all states in Q'' with the current input symbol σ in the window.

In accordance with the above intuitive meaning of the transition functions, an actual state of $\boldsymbol{A}$ is an element of $S \times (\Sigma \setminus \Delta)$, where the state component of the pair is the current state and the symbol component is the symbol stored in

[2] The converse reduction shows that the emptiness problem for one-register alternating finite-memory automata is not primitive recursive, see [3, Theorem 5.2].

the window. Thus $\boldsymbol{A}$ has infinitely many actual states, which are pairs (s,σ), where $s \in S$ and $\sigma \in \Sigma \setminus \Delta$. These are called *configurations* of $\boldsymbol{A}$.

The transition functions μ_Δ, $\mu_=$, and $\mu_\neq$ induce the following transition function $\mu^{\boldsymbol{c}} : (S \times (\Sigma \setminus \Delta)) \times \Sigma \to 2^{2^{(S\times(\Sigma\setminus\Delta))}}$.

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

Remark 4. Note that $\mu^{\boldsymbol{c}}((s,\tau),\sigma)$ is a finite set of finite sets of configurations.

Next, we extend $\mu^{\boldsymbol{c}}$ to finite sets of configurations in the standard "alternating" manner. Let $C = \{c_1, c_2, \ldots, c_k\}$ be a set of configurations, $\sigma \in \Sigma$, and let $\mu^{\boldsymbol{c}}(c_i,\sigma) = \{C_{i,1}, C_{i,2}, \ldots, C_{i,m_i}\}$, $i = 1,2,\ldots,k$. Then $\mu^{\boldsymbol{c}}(C,\sigma)$ consists of all finite sets of configurations of the form $\bigcup_{i=1}^{k} C_{i,j_i}$, $j_i = 1,2,\ldots,m_i$. That is, $C' \in \mu^{\boldsymbol{c}}(C,\sigma)$, if there is a sequence of finite sets of configurations $C_{1,j_1}, C_{2,j_2}, \ldots, C_{k,j_k}$, $C_{i,j_i} \in \mu^{\boldsymbol{c}}(c_i,\sigma)$, $i = 1,2,\ldots,k$, such that $C' = \bigcup_{i=1}^{k} C_{i,j_i}$.

Now, we can define the notion of a *run* of $\boldsymbol{A}$. Let C be a finite set of configurations and let $\boldsymbol{\sigma} = \sigma_1\sigma_2\cdots\sigma_n \in \Sigma^*$. A *$C$-run* (or just a run, if C is clear from the context) of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$ is a sequence of finite sets of configurations $C_0, C_1, \ldots, C_n$, where

- $C_0 = C$ and
- $C_{i+1} \in \mu^{\boldsymbol{c}}(C_i, \sigma_{i+1})$, $i = 0,1,\ldots,n-1$.

This run is *accepting*, if $C_n \subseteq F \times (\Sigma \setminus \Delta)$ and $\boldsymbol{A}$ *C-accepts* a word $\boldsymbol{\sigma} \in \Sigma^*$, if there is an accepting C-run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$.
The set of all words C-accepted by $\boldsymbol{A}$ is denoted by $L(\boldsymbol{A}_C)$.
For a one-element set of configurations $\{c\}$, we write just $\boldsymbol{A}_c$ for $\boldsymbol{A}_{\{c\}}$. It immediately follows from the definition that

$$L(\boldsymbol{A}_C) = \bigcap_{c\in C} L(\boldsymbol{A}_c). \tag{1}$$

Finally, we define the language $L(\boldsymbol{A})$ of $\boldsymbol{A}$ as the language $L(\boldsymbol{A}_{(s_0,\delta)})$.
Proposition 5 below immediately follows from (1).

Proposition 5. *(Cf. [6, Lemma A.2].) Let C and C' be finite sets of configurations such that $C \subseteq C'$. Then $L(\boldsymbol{A}_{C'}) \subseteq L(\boldsymbol{A}_C)$.*

The proof of Theorem 1 is based on the following closure property of accepted languages. To state it we need Definitions 6 and 7 below.

Definition 6. Let α be an automorphism of Σ. We extend α to configurations by $\alpha(s,\sigma) = (s,\alpha(\sigma))$ and then to finite sets of configurations by

$$\alpha(C) = \big\{\alpha(c) \colon\ c \in C\big\}.$$

Definition 7. Automorphisms of Σ which are invariant on Δ are called *Δ-automorphisms*.

Proposition 8. *(Cf. [6, Lemma A.4].) Let $\boldsymbol{A}$ be a one-window alternating finite-memory automaton, $\boldsymbol{\sigma} = \sigma_1\sigma_2\cdots\sigma_n \in \Sigma^*$, C be a finite set of configurations of $\boldsymbol{A}$, $C_0, C_1, \ldots, C_n$ be a C-run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$, and let α be a Δ-automorphism of Σ. Then $\alpha(C_0), \alpha(C_1), \ldots, \alpha(C_n)$ is an $\alpha(C)$-run of $\boldsymbol{A}$ on $\alpha(\boldsymbol{\sigma})$.*

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

Corollary 9. *(Cf. [6, Lemma A.4].) Let C be a finite set of configurations of $\boldsymbol{A}$ and let α be a Δ-automorphism of Σ. Then $\alpha(L(\boldsymbol{A}_C)) = L(\boldsymbol{A}_{\alpha(C)})$.*

Corollary 9 shows that emptiness is invariant under Δ-automorphisms. For this reason, in [2,3] symbols of the infinite alphabet Σ are equivalently replaced with the sets of positions which they occupy in the input word.

3. Proof of Theorem 1

Let $\boldsymbol{A} = \langle S, s_0, F, \Delta, \delta, \mu_\Delta, \mu_=, \mu_{\neq}\rangle$ be a one-window alternating finite-memory automaton. To decide whether $L(\boldsymbol{A}) = \emptyset$, we shall compute a positive integer N (that depends on $\boldsymbol{A}$) such that $L(\boldsymbol{A})$ is nonempty if and only if it contains a word of length less than N.[3] Then the decision procedure is as follows.

Decision procedure

Assume that we have computed the above positive integer N. Let $\tau_1, \tau_2, \ldots, \tau_{N-1}$ be pairwise distinct symbols different from the symbols appearing in the description of $\boldsymbol{A}$, i.e., $\tau_i \notin \Delta \cup \{\delta\}$, and let $\Sigma' = \{\tau_i\}_{i=1,\ldots,N-1} \cup \Delta \cup \{\delta\}$. Then $L(\boldsymbol{A})$ is nonempty if and only if it contains a word over Σ' of length less than N. Indeed, the "if" direction is trivial and, for the proof of the "only if" direction, let $\boldsymbol{\sigma} = \sigma_1, \sigma_2, \ldots, \sigma_n \in L(\boldsymbol{A})$, $n < N$. Since $n < N$, there is a Δ-automorphism α of Σ such that $\alpha(\delta) = \delta$ and $\alpha(\boldsymbol{\sigma})$ is a word over alphabet Σ'. Then, by Corollary 9 with $C = \{(s_0, \delta)\}$ and $\alpha(\delta) = \delta$, $\alpha(\boldsymbol{\sigma}) \in L(\boldsymbol{A})$ as well.

Therefore, $L(\boldsymbol{A}) \neq \emptyset$ if and only if $L(\boldsymbol{A}) \cap \Sigma'^* \neq \emptyset$. The alphabet Σ' is finite and, similarly to the proof of [6, Proposition 1], one can construct a finite alternating automaton that accepts $L(\boldsymbol{A}) \cap \Sigma'^*$. By [1, Theorem 5.2], this language is regular and our problem is reduced to decidability of the emptiness problem for regular languages.[4]

So, it remains to compute the above upper bound N on the minimum length words in $L(\boldsymbol{A})$. The algorithm for computing N is based on an analysis of finite sets of configurations of $\boldsymbol{A}$. Namely, we establish an invariant of a finite set of configurations (under Δ-automorphisms of Σ) that will be used for computing a positive integer N defined in the beginning of this section.

For a finite set C of configurations of $\boldsymbol{A}$, we denote by Σ_C the following subset of $\Sigma \setminus \Delta$

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

That is, Σ_C consists of all elements of $\Sigma \setminus \Delta$ which occur in configurations from C.

Consider the relation $\equiv_C$ on Σ_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 $(s, \sigma') \in C$.

It immediately follows from the definition that $\equiv_C$ is an equivalence relation. The equivalence classes of $\equiv_C$ can be described as follows. Let $\sigma \in \Sigma$ and let the subset S_C^σ of S be defined by

$$S_C^\sigma = \{s\colon (s, \sigma) \in C\}.$$

Then $\sigma \equiv_C \sigma'$ if and only if $S_C^\sigma = S_C^{\sigma'}$.

Next, let P be a nonempty subset of S and let the subset Σ_C^P of Σ_C be defined by

$$\Sigma_C^P = \{\sigma\colon S_C^\sigma = P\}.$$

Then

$$\Sigma_C = \bigcup_{P \subseteq 2^S \setminus \{\emptyset\}} \Sigma_C^P$$

and the equivalence classes of $\equiv_C$ are in one-to-one correspondence with those subsets P of S for which Σ_C^P is nonempty.

In what follows we shall deal with the sets of functions $(2^S)^\Delta$ and $\mathbb{N}^{2^S \setminus \{\emptyset\}}$ with the following partial orders $\leqslant_\Delta$ and $\leqslant_{\bar{\Delta}}$, respectively.[5]

- For $f, f' \in (2^S)^\Delta$, $f \leqslant_\Delta f'$ if and only if for all $\sigma \in \Delta$, $f(\sigma) \subseteq f'(\sigma)$, and
- for $f, f' \in \mathbb{N}^{2^S \setminus \{\emptyset\}}$, $f \leqslant_{\bar{\Delta}} f'$ if and only if for all $P \in 2^S \setminus \{\emptyset\}$, $f(P) \leqslant f'(P)$.

The partial order $\leqslant$ on the product $(2^S)^\Delta \times \mathbb{N}^{2^S \setminus \{\emptyset\}}$ is the product of $\leqslant_\Delta$ and $\leqslant_{\bar{\Delta}}$. That is,

- for $(f, g), (f', g') \in (2^S)^\Delta \times \mathbb{N}^{2^S \setminus \{\emptyset\}}$, $(f, g) \leqslant (f', g')$ if and only if $f \leqslant_\Delta f'$ and $g \leqslant_{\bar{\Delta}} g'$.

[3] Of course, the "if" direction is immediate.

[4] Alternatively, the set of all words over Σ' of length less than N is finite and for each such word we can check whether it is accepted by $\boldsymbol{A}$.

[5] As usual, X^Y denotes the set of all functions from Y to X, and $\mathbb{N}$ is the set of the nonnegative integers.

Now, with each finite set of configurations C we associate the pair of functions f_C,

$$f_C = \left(f_C^{\Delta}, f_C^{\bar{\Delta}}\right) \in \left(2^S\right)^{\Delta} \times \mathbb{N}^{2^S \setminus \{\emptyset\}},$$

that is defined as follows.

- For $\sigma \in \Delta$, $f_C^{\Delta}(\sigma) = S_C^{\sigma}$, and
- for $P \subseteq 2^S \setminus \{\emptyset\}$, $f_C^{\bar{\Delta}}(P) = |\Sigma_C^P|$.[6]

Remark 10. Like in [6, Appendix A], the elements of $(2^S)^{\Delta} \times \mathbb{N}^{2^S \setminus \{\emptyset\}}$ can be thought of as $(|\Delta| + 2^{|S|} - 1)$-dimensional vectors whose first $|\Delta|$ components correspond to the symbols in Δ and the last $2^{|S|} - 1$ components correspond to the nonempty subsets of S.

The notion of the associated pairs of functions reflects the structure of a configuration up to a Δ-automorphism of Σ.

Proposition 11. *Let C be a finite set of configurations and let α be a Δ-automorphism of Σ. Then $f_C = f_{\alpha(C)}$.*

Proof. We have to show that $f_C^{\Delta} = f_{\alpha(C)}^{\Delta}$ and that $f_C^{\bar{\Delta}} = f_{\alpha(C)}^{\bar{\Delta}}$.

The former equality is immediate, because, for $\sigma \in \Delta$, $\alpha(\sigma) = \sigma$. The latter equality is also immediate, because, for $P \subseteq 2^S \setminus \{\emptyset\}$, $\alpha(\Sigma_C^P) = \Sigma_{\alpha(C)}^P$, implying

$$\left|\Sigma_C^P\right| = \left|\alpha\left(\Sigma_C^P\right)\right| = \left|\Sigma_{\alpha(C)}^P\right|. \qquad \square$$

Proposition 12. *Let C and C' be finite sets of configurations. Then $f_C \leqslant f_{C'}$ if and only if there is a Δ-automorphism α of Σ such that for all $P \in 2^S \setminus \{\emptyset\}$, $\alpha(\Sigma_C^P) \subseteq \Sigma_{C'}^P$.*

Proof. The proof of the "if" part of the lemma is immediate, because Δ-automorphisms do not affect the first component f_C^{Δ} of f_C and for all $P \in 2^S \setminus \{\emptyset\}$, $\alpha(\Sigma_C^P) \subseteq \Sigma_{C'}^P$ implies $|\Sigma_C^P| \leqslant |\Sigma_{C'}^P|$.

The proof of the "only if" direction is equally easy. Since $f_C^{\bar{\Delta}} \leqslant f_{C'}^{\bar{\Delta}}$, by definition, for all $P \in 2^S \setminus \{\emptyset\}$, the cardinality of Σ_C^P does not exceed the cardinality of $\Sigma_{C'}^P$. Therefore, there is an embedding $e : \Sigma_C \to \Sigma_{C'}$ such that for all $P \in 2^S \setminus \{\emptyset\}$, $e(\Sigma_C^P) \subseteq \Sigma_{C'}^P$. Since e is one-to-one and Σ is infinite, e extends to a Δ-automorphism of Σ. $\square$

Corollary 13. *Let C and C' be finite sets of configurations such that $f_C \leqslant f_{C'}$. Then there is a Δ-automorphism α of Σ such that $\alpha(C) \subseteq C'$.*

Proof. Let $(s, \sigma) \in C$. Then, by the definition of S_C^{σ}, $s \in S_C^{\sigma}$.

Assume $\sigma \in \Delta$. Then, by the definition of the order $\leqslant$,

$$S_C^{\sigma} = f_C^{\Delta}(\sigma) \subseteq f_{C'}^{\Delta}(\sigma) = S_{C'}^{\sigma}.$$

Therefore, $s \in S_{C'}^{\sigma}$ and, by the definition of $S_{C'}^{\sigma}$, $(s, \sigma) \in C'$. Thus, the containment $(s, \alpha(\sigma)) \in C'$ follows from $\alpha(\sigma) = \sigma$.

Assume $\sigma \in \Sigma \setminus \Delta$. Again, by the definition of S_C^{σ}, $s \in S_C^{\sigma}$ and, by the definition of $\Sigma_C^{S_C^{\sigma}}$, $\sigma \in \Sigma_C^{S_C^{\sigma}}$.

By Proposition 12, there is a Δ-automorphism α of Σ such that for all $P \in 2^S \setminus \{\emptyset\}$, $\alpha(\Sigma_C^P) \subseteq \Sigma_{C'}^P$. Therefore, $\alpha(\sigma) \in \Sigma_{C'}^{S_C^{\sigma}}$ and the containment $(s, \alpha(\sigma)) \in C'$ follows from $s \in S_C^{\sigma}$. $\square$

There is the following relationship between non-emptiness of languages defined by finite sets of configurations and the partial order on the associated pairs of functions.

Lemma 14. *Let C and C' be finite sets of configurations such that $f_C \leqslant f_{C'}$. If $L(\boldsymbol{A}_{C'})$ contains a word of length n, then $L(\boldsymbol{A}_C)$ also contains a word of length n.*

Proof. Let $\boldsymbol{\sigma} \in L(\boldsymbol{A}_{C'})$. Since $f_C \leqslant f_{C'}$, by Corollary 13, there is an automorphism α of Σ such that $\alpha(C) \subseteq C'$. Therefore, by Proposition 5, $\boldsymbol{\sigma} \in L(\boldsymbol{A}_{\alpha(C)})$ and, by Corollary 9, $\alpha^{-1}(\boldsymbol{\sigma}) \in L(\boldsymbol{A}_C)$. $\square$

Definition 15. A sequence $f_0, f_1, \ldots, f_N$ of elements of $(2^S)^{\Delta} \times \mathbb{N}^{2^S \setminus \{\emptyset\}}$ is called *reducible* if there are i and j, $i < j$, such that $f_i \leqslant f_j$.

[6] As usual, $|X|$ is the number of elements of X.

Lemma 16. *(Cf. [6, Lemma A.3].) Let a positive integer N be such that for every word $\boldsymbol{\sigma} \in \Sigma^N$ and for every run $C_0, C_1, \ldots, C_N$ of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$, the associated sequence $f_{C_0}, f_{C_1}, \ldots, f_{C_N}$ is reducible. Then, $L(\boldsymbol{A}_{C_0})$ is nonempty if and only if it contains a word shorter than N.*

Proof. The "if" part of the lemma is immediate. For the proof of the "only if" part, assume that $L(\boldsymbol{A}) \neq \emptyset$ and let $\boldsymbol{\sigma} = \sigma_1 \cdots \sigma_n$ be a word of the minimum length belonging to $L(\boldsymbol{A})$. We contend that $n < N$.

To prove our contention we assume to the contrary that $n \geqslant N$ and let $C_0, C_1, \ldots, C_n$ be an accepting C_0-run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$. Then $C_0, C_1, \ldots, C_N$ is a C_0-run of $\boldsymbol{A}$ on $\sigma_1 \cdots \sigma_N$. Let i and j, $i < j$, be such that $f_{C_i} \leqslant f_{C_j}$. Since, by definition, $\sigma_{j+1} \cdots \sigma_n \in L(\boldsymbol{A}_{C_j})$, by Lemma 14, there exists a word $\boldsymbol{\sigma}'$ of length $n - j$ such that $\boldsymbol{\sigma}' \in L(\boldsymbol{A}_{C_i})$.

Since $C_0, C_1, \ldots, C_i$ is a C_0-run of $\boldsymbol{A}$ on $\sigma_1\sigma_2 \cdots \sigma_i$, the word $\boldsymbol{\sigma}'' = \sigma_1 \cdots \sigma_i\boldsymbol{\sigma}'$ is in $L(\boldsymbol{A}_{C_0})$. The length of $\boldsymbol{\sigma}''$ is $n - j + i < n$, in contradiction with the minimality assumption on the length n of $\boldsymbol{\sigma}$. □

It remains to compute the constant N from Lemma 16. For this we need the following notation.

- For a positive integer n, T_n denotes the set of all sequences of pairs of functions $f_{C_0}, f_{C_1}, \ldots, f_{C_n}$, $n \geqslant 0$, where $C_0, C_1, \ldots, C_n$ is an $\{(s_0, \delta)\}$-run of $\boldsymbol{A}$ on a word of length n; and
- $T = \bigcup_{n=0}^{\infty} T_n$.

Lemma 17. *(Cf. [6, Lemma A.5].) For all positive integers n, the set T_n is computable.*

Proof. The proof is similar to that of [6, Lemma A.5]. Let $\tau_1, \tau_2, \ldots, \tau_n$ be pairwise distinct symbols different from the symbols appearing in the description of $\boldsymbol{A}$ and let $\Sigma' = \{\tau_i\}_{i=1,\ldots,n} \cup \Delta \cup \{\delta\}$. Let $f_{C_0}, f_{C_1}, \ldots, f_{C_n}$ be an element of T_n that results from the $\{(s_0, \delta)\}$-run $C_0, C_1, \ldots, C_n$ of $\boldsymbol{A}$ on $\boldsymbol{\sigma} = \sigma_1\sigma_2 \cdots \sigma_n \in \Sigma^n$. There exists a Δ-automorphism α of Σ such that $\alpha(\delta) = \delta$ and $\alpha(\boldsymbol{\sigma})$ is a word over the (finite) alphabet Σ'. By Proposition 11, $f_{C_i} = f_{\alpha(C_i)}$, $i = 0, 1, \ldots, n$. Therefore, by Proposition 8, the sequence $f_{C_0}, f_{C_1}, \ldots, f_{C_n}$ can also be obtained from the run $\alpha(C_0), \alpha(C_1), \ldots, \alpha(C_n)$ of $\boldsymbol{A}$ on $\alpha(\boldsymbol{\sigma})$.

Thus, for computing T_n, we may restrict ourselves to the runs of $\boldsymbol{A}$ on the words from Σ'^n. The latter set is finite and the set of all sequences associated with runs of $\boldsymbol{A}$ on a given word is computable. □

Lemma 18. *(Cf. [6, Lemma A.5].) A positive integer N satisfying the prerequisites of Lemma 16 is computable.*

Proof. The proof is similar to that of [6, Lemma A.5]. For each positive integer N, applying Lemma 17, we can check whether for each $\boldsymbol{\sigma} \in \Sigma^N$, all sequences associated with runs of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$ are reducible. Therefore, if a positive integer N satisfying the prerequisites of Lemma 16 exists, it can be found by checking all sequences of length 1, then all sequences of length 2, etc. (This process must eventually terminate when we arrive at the right N.)

To complete the proof we have to show that there indeed always exists a positive integer N satisfying the prerequisites of Lemma 16. For this, we impose on the set of sequences T the following (unranked) tree structure.

- Sequence $f_{C'_0}, f_{C'_1}, \ldots, f_{C'_{n'}}$ is a successor of sequence $f_{C_0}, f_{C_1}, \ldots, f_{C_n}$ if and only if $n' = n + 1$ and $f_{C'_m} = f_{C_m}$, for all $m = 0, 1, \ldots, n$.

It follows from the proof of Lemma 17 that the tree T is of a finite branching degree and that for any two nodes of T it is decidable whether one is a successor of the other.[7]

For the proof of the lemma, assume to the contrary that there is no such N. That is, for each $N = 1, 2, \ldots$ there exists an irreducible node $f_{C_{N,0}}, f_{C_{N,1}}, \ldots, f_{C_{N,N}}$ in the tree T.

Since the number of such nodes is infinite and the tree T is of a finite branching degree, by a straightforward modification of König's Infinity Lemma [9], there exists an infinite path in T such that for each node $\boldsymbol{f}$ on it there is an irreducible sequence $f_{C_{N,0}}, f_{C_{N,1}}, \ldots, f_{C_{N,N}}$ for which

$$\boldsymbol{f} \preccurlyeq f_{C_{N,0}}, f_{C_{N,1}}, \ldots, f_{C_{N,N}}.^{8}$$

The sequence of the nodes on the path is an infinite sequence of all prefixes of some infinite sequence of pairs of functions $f_0, f_1, \ldots, f_m, \ldots$. In addition, each f_m, $m = 0, 1, \ldots$, being a prefix of some irreducible sequence $f_{C_{N,0}}, f_{C_{N,1}}, \ldots, f_{C_{N,N}}$, is also irreducible. That is, the infinite sequence $f_0, f_1, \ldots, f_m, \ldots$ contains no two pairs of functions f_i and f_j, $i < j$, such that $f_i \leqslant f_j$. However, by Remark 10, this contradicts [8, Lemma 4.1] stating that exactly the opposite holds. □

Now we can implement the decision procedure presented in the beginning of this section.

[7] In this paper, we identify a tree with the set of its nodes, because the successor relation will be always clear from the context.

[8] Here and hereafter $\prec$ and $\preccurlyeq$ denote the transitive and the reflexive transitive closures, respectively, of the successor relation in a tree.

4. Alternating finite-memory tree automata

In this section we recall the definitions of alternating finite-memory *tree* automata. Again, we restrict ourselves to one-window automata with which we deal in this paper. Like in the word case, these automata have a finite set of proper states in which the "real computation" is done and are equipped with a window storing a symbol from the infinite alphabet.

We use the following conventions and notation.

- The set of all non-leaf nodes of a tree T is denoted by $\boldsymbol{nl}(T)$ and the set of all leaf nodes of T is denoted by $\boldsymbol{l}(T)$.
- For a tree T and a node v of T, we denote by T_v the subtree of T rooted at v.
- By a "binary tree" we mean a finite full ranked binary tree.[9]
- A Σ-tree is a labeling of all non-leaf nodes of a binary tree with symbols from Σ.
- In the rest of this paper, we do not distinguish between isomorphic trees.

Definition 19. (Cf. [7, Definition 1].) A (top-down) *one-window alternating finite-memory tree automaton* is a system $\boldsymbol{A} = \langle S, s_0, F, \Delta, \delta, \mu_\Delta, \mu_=, \mu_{\neq}\rangle$ whose components are as follows.

- S is a finite set of *states*.
- $s_0 \in S$ is the *initial state*.
- $F \subseteq S$ is the set of *accepting* states.
- $\Delta \subset \Sigma$ is a finite set of *distinguished* symbols.
- $\delta \in \Sigma \setminus \Delta$ is the *initial window assignment*.
- $\mu_\Delta : S \times \Delta \to 2^{(2^S)^2}$, $\mu_= : S \to 2^{(2^S)^2}$, and $\mu_{\neq} : S \to 2^{(2^S)^4}$ are the *transition* functions.
  The intuitive meaning of these functions is as follows. Let $\boldsymbol{A}$ read symbol σ being in state s with a symbol τ stored in the window.
  - If $\sigma \in \Delta$, then, for some $(Q_0, Q_1) \in \mu_\Delta(s, \sigma)$, the computation splits to the computations from the states in Q_0 (respectively, in Q_1) from the "left" (respectively, "right") child of the node, all with τ in the window.
  - If $\sigma = \tau$, then, for some $(Q_0, Q_1) \in \mu_=(s)$, the computation splits to the computations from the states in Q_0 (respectively, in Q_1) from the "left" (respectively, "right") child of the node, all with the same τ in the window.
  - If $\sigma \notin \Delta \cup \{\tau\}$, then, for some $((Q_0', Q_0''), (Q_1', Q_1'')) \in \mu_{\neq}(s)$, the computation splits to the computations from the states in Q_0' (respectively, in Q_1') with τ in the window and the computations from the states in Q_0'' (respectively, in Q_1'') with the current input symbol σ in the window from the "left" (respectively, "right") child of the node.

Like in the case of (word) alternating finite-memory automata, an actual state of $\boldsymbol{A}$ is a configuration from $S \times (\Sigma \setminus \Delta)$. The transition functions μ_Δ, $\mu_=$, and $\mu_{\neq}$ induce the following function $\mu^{\boldsymbol{c}} : (S \times (\Sigma \setminus \Delta)) \times \Sigma \to 2^{2^{(S \times (\Sigma \setminus \Delta))^2}}$.

- If $\sigma \in \Delta$, then
$$\mu^{\boldsymbol{c}}\big((s, \tau), \sigma\big) = \big\{\big(Q_0 \times \{\tau\}, Q_1 \times \{\tau\}\big) \colon (Q_0, Q_1) \in \mu_\Delta(s, \sigma)\big\}.$$
- If $\sigma = \tau$, then
$$\mu^{\boldsymbol{c}}\big((s, \tau), \sigma\big) = \big\{\big(Q_0 \times \{\tau\}, Q_1 \times \{\tau\}\big) \colon (Q_0, Q_1) \in \mu_=(s)\big\}.$$
- If $\sigma \notin \Delta \cup \{\tau\}$, then
$$\mu^{\boldsymbol{c}}\big((s, \tau), \sigma\big) = \big\{\big(\big(Q_0' \times \{\tau\} \cup Q_0'' \times \{\sigma\}\big), \big(Q_1' \times \{\tau\} \cup Q_1'' \times \{\sigma\}\big)\big) \colon \big((Q_0', Q_0''), (Q_1', Q_1'')\big) \in \mu_{\neq}(s)\big\}.$$

Remark 20. Note that, like in the case of (word) alternating finite-memory automata, $\mu^{\boldsymbol{c}}((s, \tau), \sigma)$ is a finite set consisting of pairs of finite sets of configurations, cf. Remark 4.

Next, we extend $\mu^{\boldsymbol{c}}$ to sets of configurations in the standard "tree-alternation" manner. Let $C = \{c_1, c_2, \ldots, c_k\}$ be a set of configurations, $\sigma \in \Sigma$, and let
$$\mu^{\boldsymbol{c}}(c_i, \sigma) = \big\{(C_{0,i,1}, C_{1,i,1}), (C_{0,i,2}, C_{1,i,2}), \ldots, (C_{0,i,m_i}, C_{1,i,m_i})\big\}, \quad i = 1, 2, \ldots, k.$$
Then $\mu^{\boldsymbol{c}}(C, \sigma)$ consists of all pairs of finite sets of configurations of the form
$$\left(\bigcup_{i=1}^{k} C_{0,i,j_i}, \bigcup_{i=1}^{k} C_{1,i,j_i}\right),$$

[9] A binary tree is *full* if each its non-leaf node is of degree two.

$j_i = 1, 2, \ldots, m_i$. That is, $(C_0, C_1) \in \mu^{\mathbf{c}}(C, \sigma)$, if there is a sequence of pairs of finite sets of configurations $(C_{0,1,j_1}, C_{1,1,j_1})$, $(C_{0,2,j_2}, C_{1,2,j_2}), \ldots, (C_{0,k,j_k}, C_{1,k,j_k})$, $(C_{0,i,j_i}, C_{1,i,j_i}) \in \mu^{\mathbf{c}}(c_i, \sigma)$, $i = 1, 2, \ldots, k$, such that

$$C_0 = \bigcup_{i=1}^{k} C_{0,i,j_i}$$

and

$$C_1 = \bigcup_{i=1}^{k} C_{1,i,j_i}.$$

Now, we can define the notion of a *run* of $\boldsymbol{A}$. Let C be a finite set of configurations, T be a binary tree, and let $\boldsymbol{\sigma} : \boldsymbol{nl}(T) \to \Sigma$ be a Σ-tree. A C-run $\boldsymbol{r}$ of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$ is a labeling of T with finite sets of configurations,

$$\boldsymbol{r} : T \to 2^{S \times (\Sigma \setminus \Delta)},$$

such that

- the root of T is labeled C and
- for every non-leaf node v of T, $(\boldsymbol{r}(v_0), \boldsymbol{r}(v_1)) \in \mu^{\mathbf{c}}(\boldsymbol{r}(v), \boldsymbol{\sigma}(v))$, where v_0 and v_1 are the left and the right child of v, respectively.

This run is *accepting*, if

$$\bigcup_{v \in \boldsymbol{l}(T)} \boldsymbol{r}(v) \subseteq F \times (\Sigma \setminus \Delta)$$

and $\boldsymbol{A}$ *C-accepts* a Σ-tree $\boldsymbol{\sigma}$, if there is an accepting C-run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$.

The set of all Σ-trees C-accepted by $\boldsymbol{A}$ is denoted by $L(\boldsymbol{A}_C)$. Then, like in the case of (word) alternating finite-memory automata,

$$L(\boldsymbol{A}_C) = \bigcap_{c \in C} L(\boldsymbol{A}_c). \tag{2}$$

Finally, we define the language $L(\boldsymbol{A})$ of $\boldsymbol{A}$ as the language $L(\boldsymbol{A}_{(s_0,\delta)})$.

Proposition 21 below immediately follows from (2).

Proposition 21. *(Cf. Proposition 5.) Let C and C' be finite sets of configurations such that $C \subseteq C'$. Then $\boldsymbol{\sigma} \in L(\boldsymbol{A}_{C'})$ implies $\boldsymbol{\sigma} \in L(\boldsymbol{A}_C)$.*

The proof of Theorem 2 is based on the following closure property of accepted languages.

Proposition 22. *(Cf. Proposition 8.) Let $\boldsymbol{A}$ be a one-window alternating finite-memory tree automaton, $\boldsymbol{\sigma} : \boldsymbol{nl}(T) \to \Sigma$ be a Σ-tree, C be a finite set of configurations of $\boldsymbol{A}$, $\boldsymbol{r} : T \to 2^{S \times (\Sigma \setminus \Delta)}$ be a C-run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$, and let α be a Δ-automorphism of Σ. Then $\alpha(\boldsymbol{r})$ is an $\alpha(C)$-run of $\boldsymbol{A}$ on $\alpha(\boldsymbol{\sigma})$.*[10]

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

Corollary 23. *(Cf. Corollary 9.) Let C be a finite set of configurations of $\boldsymbol{A}$ and let α be a Δ-automorphism of Σ. Then $\boldsymbol{\sigma} \in L(\boldsymbol{A}_C)$ if and only if $\alpha(\boldsymbol{\sigma}) \in L(\boldsymbol{A}_{\alpha(C)})$.*

5. Proof of Theorem 2

Let $\boldsymbol{A} = \langle S, s_0, F, \Delta, \delta, \mu_\Delta, \mu_=, \mu_{\neq} \rangle$ be a one-window alternating finite-memory tree automaton. To decide whether $L(\boldsymbol{A}) = \emptyset$, we shall compute a positive integer N (that depends on $\boldsymbol{A}$) such that $L(\boldsymbol{A})$ is nonempty if and only if it contains a Σ-tree with less than N nodes.[11] Then the decision procedure is as follows.

[10] As usual, $\alpha(\boldsymbol{r})$ is defined by $(\alpha(\boldsymbol{r}))(v) = \alpha(\boldsymbol{r}(v))$.

[11] Of course, the "if" direction is immediate.

Decision procedure

Assume that we have computed the above positive integer N. Let $\tau_1, \tau_2, \ldots, \tau_{N-1}$ be pairwise distinct symbols different from the symbols appearing in the description of $\boldsymbol{A}$ and let $\Sigma' = \{\tau_i\}_{i=1,\ldots,N-1} \cup \Delta \cup \{\delta\}$. Then $L(\boldsymbol{A})$ is nonempty if and only if it contains a Σ'-tree with less than N nodes. Indeed, the "if" direction is trivial and, for the proof of the "only if" direction let $\boldsymbol{\sigma} : \boldsymbol{nl}(T) \to \Sigma$ be a Σ-tree in $L(\boldsymbol{A})$ with less than N nodes. Then there is a Δ-automorphism α of Σ such that $\alpha(\delta) = \delta$ and $\alpha(\boldsymbol{\sigma})$ is a Σ'-tree. Then, by Corollary 23 with $C = \{(s_0, \delta)\}$ and $\alpha(\delta) = \delta$, $\alpha(\boldsymbol{\sigma}) \in L(\boldsymbol{A})$ as well.

Therefore, $L(\boldsymbol{A}) \neq \emptyset$ if and only if $L(\boldsymbol{A}) \cap \Sigma'^* \neq \emptyset$. The alphabet Σ' is finite, implying that the set of all Σ'-trees with less than N nodes is also finite, and for each such Σ'-tree we can check whether it is accepted by $\boldsymbol{A}$.

Now, like in the proof of Theorem 1, it remains to compute the above upper bound N on the minimum number of nodes of Σ-trees in $L(\boldsymbol{A})$. The algorithm for computing N is based on an analysis of finite sets of configurations of $\boldsymbol{A}$. It is similar to the algorithm in Section 3. That is, we establish an invariant of a finite set of $\boldsymbol{A}$-configurations (under Δ-automorphisms of Σ) that will be used for computing a positive integer N defined in the beginning of this section.

We proceed with a series of auxiliary results similar to those in Section 3.

Lemma 24. *(Cf. Lemma 14.) Let C and C' be finite sets of configurations such that $f_C \leqslant f_{C'}$. If $L(\boldsymbol{A}_{C'})$ contains a Σ-tree over a binary tree T, then $L(\boldsymbol{A}_C)$ also contains a Σ-tree over T.*

Proof. Let $\boldsymbol{\sigma} \in L(\boldsymbol{A}_{C'})$. By Corollary 13, there is a Δ-automorphism α of Σ such that $\alpha(C) \subseteq C'$. Therefore, by Proposition 21, $\boldsymbol{\sigma} \in L(\boldsymbol{A}_{\alpha(C)})$ and, by Corollary 23, $\alpha^{-1}(\boldsymbol{\sigma}) \in L(\boldsymbol{A}_C)$. □

Definition 25. (Cf. Definition 15.) Let T be a binary tree. A labeling $\boldsymbol{f} : T \to (2^S)^\Delta \times \mathbb{N}^{2^S \setminus \{\emptyset\}}$ of the nodes of T with pairs of functions from $(2^S)^\Delta \times \mathbb{N}^{2^S \setminus \{\emptyset\}}$ is called *reducible* if there are two nodes $v, v' \in T$, $v \prec v'$, such that $f(v) \leqslant f(v')$.

Lemma 26. *(Cf. Lemma 16.) Let a positive integer N be such that for every N-node binary tree T, every Σ-tree $\boldsymbol{\sigma} : \boldsymbol{nl}(T) \to \Sigma$, and all runs $\boldsymbol{r} : T \to 2^{S \times (\Sigma \setminus \Delta)}$ of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$, the labeling $\boldsymbol{f} : T \to (2^S)^\Delta \times \mathbb{N}^{2^S \setminus \{\emptyset\}}$ defined by $\boldsymbol{f}(v) = f_{\boldsymbol{r}(v)}$, $v \in T$, is reducible. Then, $L(\boldsymbol{A})$ is nonempty if and only if it contains a Σ-tree with less than N nodes.*

Proof. The "if" part of the lemma is immediate. For the proof of the "only if" part, assume that $L(\boldsymbol{A}) \neq \emptyset$ and let $\boldsymbol{\sigma} : \boldsymbol{nl}(U) \to \Sigma$ be a Σ-tree with the minimum number n of nodes belonging to $L(\boldsymbol{A})$. We contend that $n < N$.

To prove our contention, we assume to the contrary that $n \geqslant N$ and let $\boldsymbol{r} : U \to 2^{S \times (\Sigma \setminus \Delta)}$ be an accepting run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$. Let T be an N-node *upper* binary subtree of U, i.e., the root of U is also the root of T,[12] and let $\boldsymbol{\sigma}|_{\boldsymbol{nl}(T)} : \boldsymbol{nl}(T) \to \Sigma$ be the restriction of $\boldsymbol{\sigma}$ to $\boldsymbol{nl}(T)$. Then the restriction $\boldsymbol{r}|_T$ of $\boldsymbol{r}$ to T is a run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}|_{\boldsymbol{nl}(T)}$.

Let v and v', $v \prec v'$, be nodes of T such that $f_{\boldsymbol{r}|_T(v)} \leqslant f_{\boldsymbol{r}|_T(v')}$. Then

$$f_{\boldsymbol{r}(v)} \leqslant f_{\boldsymbol{r}(v')}, \tag{3}$$

because, by the definition of restriction, $\boldsymbol{r}|_T(v) = \boldsymbol{r}(v)$ and $\boldsymbol{r}|_T(v') = \boldsymbol{r}(v')$.

Let $\boldsymbol{\sigma}_{v'} : \boldsymbol{nl}(U_{v'}) \to \Sigma$ be the restriction of $\boldsymbol{\sigma}$ to $\boldsymbol{nl}(U_{v'})$. Then $\boldsymbol{\sigma}_{v'} \in L(\boldsymbol{A}_{\boldsymbol{r}(v')})$, because the restriction of $\boldsymbol{r}$ to $U_{v'}$ is an accepting $\boldsymbol{r}(v')$-run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}_{v'}$. By (3) and Lemma 24, there exists a Σ-tree $\boldsymbol{\sigma}' : U_{v'} \to \Sigma$ in $L(\boldsymbol{A}_{\boldsymbol{r}(v)})$ and let $\boldsymbol{r}'$ be an accepting $\boldsymbol{r}(v)$-run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}'$.

Let U'' be the binary tree obtained from U by replacing U_v with $U_{v'}$:

$$U'' = (U \setminus U_v) \cup U_{v'}$$

with the induced parent-child relation, see Fig. 1, and let $\boldsymbol{\sigma}'' : \boldsymbol{nl}(U'') \to \Sigma$ be defined by

$$\boldsymbol{\sigma}''(u) = \begin{cases} \boldsymbol{\sigma}(u), & \text{if } u \in U \setminus U_v \\ \boldsymbol{\sigma}'(u), & \text{if } u \in U_{v'}. \end{cases}$$

Then $\boldsymbol{r}'' : U'' \to \Sigma$ defined by

$$\boldsymbol{r}''(u) = \begin{cases} \boldsymbol{r}(u), & \text{if } u \in U \setminus U_v \\ \boldsymbol{r}'(u), & \text{if } u \in U_{v'} \end{cases}$$

is an accepting run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}''$. That is, $\boldsymbol{\sigma}'' \in L(\boldsymbol{A})$. The number of nodes in U'' is

$$n - |U_v| + |U_{v'}| < n,$$

in contradiction with the minimality assumption on the number n of nodes in U. □

[12] Recall that, in Computer Science, trees grow downward, see Fig. 1.

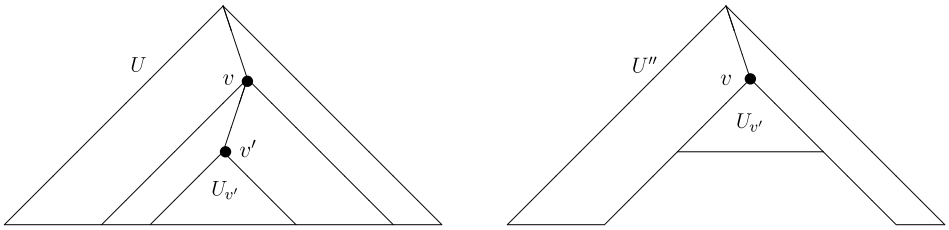

Fig. 1. Replacing U_v with $U_{v'}$.

It remains to compute the constant N from Lemma 26. For this we need the following notation.

- For a positive integer n, $\boldsymbol{T}_n$ denotes the set of all n-node binary trees labeled with pairs of functions from $(2^S)^\Delta \times \mathbb{N}^{2^S\setminus\{\emptyset\}}$ in the following manner.
  Let T be an n-node binary tree, $\boldsymbol{\sigma} : \boldsymbol{nl}(T) \to \Sigma$, and let $\boldsymbol{r} : T \to 2^{S\times(\Sigma\setminus\Delta)}$ be an $\{(s_0,\delta)\}$-run of $\boldsymbol{A}$ on $\boldsymbol{\sigma}$. Then the label of a node $v \in T$ is $f_{\boldsymbol{r}(v)}$.
- $\boldsymbol{T} = \bigcup_{n=0}^{\infty} \boldsymbol{T}_n$.

Lemma 27. *(Cf. Lemma 17.) For all positive integers n, the set $\boldsymbol{T}_n$ is computable.*

Proof. Let $\tau_1, \tau_2, \ldots, \tau_n$ be pairwise distinct symbols different from the symbols appearing in the description of $\boldsymbol{A}$ and let $\Sigma' = \{\tau_i\}_{i=1,\ldots,n} \cup \Delta \cup \{\delta\}$. Let T be an n-node binary tree and let $\boldsymbol{f} : T \to (2^S)^\Delta \times \mathbb{N}^{2^S\setminus\{\emptyset\}}$ be an element of $\boldsymbol{T}_n$ that results from the $\{(s_0,\delta)\}$-run $\boldsymbol{r} : T \to 2^{S\times\Sigma}$ of $\boldsymbol{A}$ on $\boldsymbol{\sigma} : \boldsymbol{nl}(T) \to \Sigma$. There exists a Δ-automorphism α of Σ such that $\alpha(\delta) = \delta$ and $\alpha(\boldsymbol{\sigma})$ is a word over the (finite) alphabet Σ'. Therefore, by Propositions 22 and 11, the labeling $\boldsymbol{f}$ can also be obtained from the run $\alpha(\boldsymbol{r})$ on $\alpha(\boldsymbol{\sigma})$.

Thus, for computing $\boldsymbol{T}_n$, we may restrict ourselves to the runs of $\boldsymbol{A}$ on the n-node Σ'-trees. The latter set is finite and the set of all labelings associated with runs of $\boldsymbol{A}$ on a given binary tree is computable. □

Lemma 28. *(Cf. Lemma 18.) A positive integer N satisfying the prerequisites of Lemma 26 is computable.*

Proof. The proof is similar to that of Lemma 18. For each positive integer N, applying Lemma 27, we can check whether for each N-node Σ-tree T, all labelings associated with runs of $\boldsymbol{A}$ on T are reducible. Therefore, if a positive integer N satisfying the prerequisites of Lemma 26 exists, it can be found by checking all one-node binary trees, then all two-node binary trees, etc. (This process must eventually terminate when we arrive at the right N.)

To complete the proof we have to show that there indeed always exists a positive integer N satisfying the prerequisites of Lemma 26. For this, we impose on the set $\boldsymbol{T}$ the following (unranked) tree structure.

- Labeling $\boldsymbol{f}' : T' \to (2^S)^\Delta \times \mathbb{N}^{2^S\setminus\{\emptyset\}}$ is a successor of labeling $\boldsymbol{f} : T \to (2^S)^\Delta \times \mathbb{N}^{2^S\setminus\{\emptyset\}}$ if and only if T is obtained form T' by deleting two sibling leaves and $\boldsymbol{f}$ is the restriction of $\boldsymbol{f}'$ to T.

It follows from the proof of Lemma 27 that the tree $\boldsymbol{T}$ is of a finite branching degree and that for any two nodes of $\boldsymbol{T}$ it is decidable whether one is a successor of the other.

For the proof of the lemma, assume to the contrary that there is no such N. That is, for each $N = 1, 2, \ldots$ there exists an irreducible node from the set $\boldsymbol{T}_N$ in the tree $\boldsymbol{T}$.

Since the number of such nodes is infinite and the tree $\boldsymbol{T}$ is of a finite branching degree, by a straightforward modification of König's Infinity Lemma [9], there exists an infinite path $\boldsymbol{f}_0, \boldsymbol{f}_1, \ldots, \boldsymbol{f}_m, \ldots$ in $\boldsymbol{T}$ such that for each m there are a positive integer N_m and an irreducible node $\boldsymbol{f}'_m : T_{N_m} \to (2^S)^\Delta \times \mathbb{N}^{2^S\setminus\{\emptyset\}}$ in $\boldsymbol{T}_{N_m}$ such that

$$\boldsymbol{f}_m \preccurlyeq \boldsymbol{f}'_m. \tag{4}$$

By the definition of the successor relation in $\boldsymbol{T}$, $\boldsymbol{f}_0, \boldsymbol{f}_1, \ldots, \boldsymbol{f}_m, \ldots$ is an increasing infinite sequence of upper subtrees of a labeling $\boldsymbol{f} : T \to (2^S)^\Delta \times \mathbb{N}^{2^S\setminus\{\emptyset\}}$ of some infinite full binary tree T with pairs of functions from $(2^S)^\Delta \times \mathbb{N}^{2^S\setminus\{\emptyset\}}$.

Applying König's Infinity Lemma to T, we obtain an infinite path $v_0, v_1, \ldots$ of nodes of T.

Let $i, j = 0, 1, \ldots$ be such that $i < j$ and let m be such that $v_i, v_j \in T_{N_m}$. Then, by (4), $\boldsymbol{f}(v_i) = \boldsymbol{f}'_m(v_i)$ and $\boldsymbol{f}(v_j) = \boldsymbol{f}'_m(v_j)$. Since $v_i \prec v_j$ and the labeling $\boldsymbol{f}'_m$ of T_{N_m} is irreducible, $\boldsymbol{f}(v_i) \not\preccurlyeq \boldsymbol{f}(v_j)$. However, by Remark 10, this contradicts [8, Lemma 4.1] stating that exactly the opposite holds. □

Now we can implement the decision procedure presented in the beginning of this section.

Acknowledgement

The authors are grateful to Tony Tan for his comments on the first version of the paper.

References

[1] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer, Alternation, J. ACM 28 (1) (1981) 114–133.
[2] S. Demri, R. Lazić, LTL with the freeze quantifier and register automata, in: Proceedings of the 21st IEEE Symposium on Logic in Computer Science (LICS 2006), IEEE Computer Society, 2006, pp. 17–26.
[3] S. Demri, R. Lazić, LTL with the freeze quantifier and register automata, ACM Trans. Comput. Log. 10 (2009), Article 16.
[4] M. Jurdziński, R. Lazić, Alternating automata on data trees and XPath satisfiability, ACM Trans. Comput. Log. 12 (2011), Article 19.
[5] M. Kaminski, N. Francez, Finite-memory automata, in: Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 683–688.
[6] M. Kaminski, N. Francez, Finite-memory automata, Theor. Comput. Sci. 134 (1994) 329–363.
[7] M. Kaminski, T. Tan, Tree automata over infinite alphabets, in: A. Avron, N. Dershowitz, A. Rabinovich (Eds.), Pillars of Computer Science – Essays Dedicated to Boris (Boaz) Trakhtenbrot, in: Lect. Notes Comput. Sci., vol. 4800, Springer, Berlin, 2008, pp. 386–423.
[8] R.M. Karp, R.E. Miller, Parallel program schemata, J. Comput. Syst. Sci. 3 (1969) 147–195.
[9] D. König, Sur les correspondences multivoques des ensembles, Fundam. Math. 8 (1926) 114–134.
[10] F. Neven, T. Schwentick, V. Vianu, Towards regular languages over infinite alphabets, in: J. Sgall, A. Pultr, P. Kolman (Eds.), Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001, in: Lect. Notes Comput. Sci., vol. 2136, Springer, Berlin, 2001, pp. 560–572.
[11] F. Neven, T. Schwentick, V. Vianu, Finite state machines for strings over infinite alphabets, ACM Trans. Comput. Log. 5 (2004) 403–435.
[12] M. Rabin, D. Scott, Finite automata and their decision problems, IBM J. Res. Dev. 3 (1959) 114–125.
[13] T. Tan, On pebble automata for data languages with decidable emptiness problem, in: R. Královic, D. Niwinski (Eds.), Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009, in: Lect. Notes Comput. Sci., vol. 5734, Springer, Berlin, 2009, pp. 712–723.
[14] T. Tan, On pebble automata for data languages with decidable emptiness problem, J. Comput. Syst. Sci. 76 (2010) 778–791.