A Note on Emptiness for Alternating Finite Automata with a One-Letter Alphabet ⋆

Petr Jančar   Zdeněk Sawa

Center of Applied Cybernetics, Dept. of Computer Science,
Technical University of Ostrava (FEI VŠB-TUO)
17. listopadu 15, CZ-70833 Ostrava, Czech Republic
{Petr.Jancar,Zdenek.Sawa}@vsb.cz

Abstract

We present a new proof of PSPACE-hardness of the emptiness problem for alternating finite automata with a singleton alphabet. This result was shown by Holzer (1995) who used a proof relying on a series of reductions from several papers. The new proof is simple, direct and self-contained.

Key words: computational complexity, alternating finite automaton, emptiness

1 Introduction

Checking emptiness, i.e. checking whether the language accepted by a given automaton is (non-)empty, is a natural problem studied in automata theory. It is well known that the emptiness problem is PSPACE-complete for alternating finite automata (AFA), the hardness being implied by the PSPACE-completeness of the universality problem for nondeterministic finite automata. It is probably less well known that the problem 1L-Afa-Emptiness, the emptiness problem for AFA with a singleton alphabet, is also PSPACE-hard; this was shown by Holzer in [3], who thus completed the results of [5].
During the conference presentation of [8], Markus Lohrey noted that Holzer's result can help strengthen some presented complexity lower bounds. In fact, it also helps strengthen some results in [4], and Jiří Srba was inspired to use the result in [9].
If one is interested in the actual proof of PSPACE-hardness of

⋆ The authors acknowledge the support by the Czech Ministry of Education, Grant No. 1M0567.

1L-Afa-Emptiness, it is a bit unpleasant to find that Holzer uses the emptiness problem for so called EP0L systems [7] which was shown to be PSPACE-complete in [6], where the proof of PSPACE-hardness (solving a long-term open question) uses a series of reductions among several problems, one of these reductions being handled by a reference to [2].
In this note we observe that the PSPACE-hardness of 1L-Afa-Emptiness can be shown directly by a "master reduction," and we note that the idea was implicitly present already in the seminal paper on alternation [1]. In fact, a little adjustment of the construction could also serve to show the PSPACE-hardness of all problems in the above mentioned series in [6].

2 The main observation

Let us consider a fixed deterministic Turing machine M with space bounded by $f(n)$. For any input w for M we will show how to construct a one-letter-alphabet AFA (1L-AFA) A_w with $O(f(|w|))$ states so that M accepts w iff $L(A_w) \neq \emptyset$; by $|w|$ we denote the length of w.
We start by recalling the basic definitions.

For a set X we use $Bool^+(X)$ to denote the set of (positive) boolean formulas that only use $\wedge$ and $\vee$ as boolean connectives and elements of X as variables. By $[\phi]_\nu$ we denote the truth value (0 or 1) of formula $\phi \in Bool^+(X)$ under the boolean assignment $\nu : X \to \{0,1\}$.

An *alternating finite automaton* (AFA) is a structure $A = (Q, \Sigma, \delta, q_0, F)$ where Q is the finite set of *states*, Σ is the finite *alphabet*, $\delta : Q \times \Sigma \to Bool^+(Q)$ is the *transition function*, q_0 is the *initial state*, and $F \subseteq Q$ is the set of *accepting states*.
We define the predicate $Acc \subseteq Q \times \Sigma^*$ by induction on the length of the second component; $Acc(q, w)$ is to be read as "A starting in q accepts w."

- $Acc(q, \varepsilon)$ iff $q \in F$.
- $Acc(q, aw)$ iff $[\delta(q,a)]_\nu = 1$ for the boolean assignment ν satisfying $(\, \nu(q') = 1 \Leftrightarrow Acc(q', w)\,)$ for all $q' \in Q$.

AFA A accepts the language $L(A) = \{w \in \Sigma^* \mid Acc(q_0, w)\}$.
When $|\Sigma| = 1$, we say that A is a 1L-AFA (1L being read "one letter").
We are interested in the problem 1L-Afa-Emptiness:

Instance: 1L-AFA A.
Question: Is $L(A) = \emptyset$?

A *deterministic Turing machine* (deciding a problem, or accepting a language) is a structure $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where Q is the finite set of (control) *states*, Σ is the finite *input alphabet*, Γ is the finite *tape alphabet* where $\Sigma \subseteq \Gamma$, $\delta : (Q - \{q_{acc}, q_{rej}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 0, +1\}$ is the *transition function*, and $q_0, q_{acc}, q_{rej} \in Q$ are the *initial state*, the *accepting final state* and the *rejecting final state*, respectively. The tape alphabet Γ contains a special *blank* symbol $\Box \notin \Sigma$. We assume that M starts with scanning the tape cell with the leftmost symbol of an input word $w \in \Sigma^+$ and never moves left from that cell. W.l.o.g. we only consider nonempty input words.

Technically we view the tape cells as numbered by nonnegative integers, i.e. by elements of $\mathbb{N} = \{0, 1, 2, \ldots\}$. A *configuration* C is then a function $C : \mathbb{N} \rightarrow \Delta$ where $\Delta = \Gamma \cup (Q \times \Gamma)$; the state and the head position are determined by the pair $C(j) \in (Q \times \Gamma)$. Given a (nonempty) input $w = a_1 a_2 \ldots a_n$, the *initial configuration* C_0^w is defined as $C_0^w(1) = (q_0, a_1)$, $C_0^w(j) = a_j$ for $2 \leq j \leq n$, and $C_0^w(j) = \Box$ elsewhere.

The *computation* of M on w is the (finite or infinite) sequence of configurations $C_0^w, C_1^w, C_2^w, \ldots$ determined by the input w and the transition function δ in the usual manner. We use the cell 0 for technical convenience; necessarily, $C_i^w(0) = \Box$ for all i. It is important that $C_{i+1}^w(j)$, for $j \geq 1$, is determined by the triple $(C_i^w(j-1), C_i^w(j), C_i^w(j+1))$, not depending on the actual i, j, w. For any $z \in \Delta$ we can thus define the following easily constructible set:

$$\texttt{Preds}(z) = \{(z_1, z_2, z_3) \in \Delta^3 : (\forall i, j, w)(\,(C_i^w(j-1), C_i^w(j), C_i^w(j+1)) = (z_1, z_2, z_3) \text{ implies } C_{i+1}^w(j) = z\,)\}\,.$$

For technical convenience we also assume that if M enters q_{acc} then the head scans cell 1 which currently contains $\Box$. Thus we can define that M *accepts* w iff there is $i \in \mathbb{N}$ such that $C_i^w(1) = (q_{acc}, \Box)$.

Now we come to the crucial construction. We assume a fixed deterministic Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ with space bounded by a function f; this means that M can only visit the cells numbered $1, 2, \ldots, f(n)$ in the computation starting on an input w with $|w| = n$. The function $f : \mathbb{N} \rightarrow \mathbb{N}$ is supposed to satisfy $f(n) \geq n$ for all n, which also means that $C_i^w(j) = \Box$ for $j > f(n)$ in the computation of M on w with $|w| = n$.

For any $w = a_1 a_2 \ldots a_n$ we define the following 1L-AFA $A_w = (Q', \{\Diamond\}, \delta', q'_0, F')$:

- $Q' = \{0, 1, 2, \ldots, f(n)+1\} \times \Delta$ (where $\Delta = \Gamma \cup (Q \times \Gamma)$),
- $q'_0 = (1, (q_{acc}, \Box))$,
- $F' = \{\, (j, z) \in Q' \mid C_0^w(j) = z\,\}$,
- for $j \in \{0, f(n)+1\}$ we put $\delta'((j, \Box), \Diamond) = 1$ (constantly true) and $\delta'((j, z), \Diamond) = 0$ (constantly false) for $z \neq \Box$,
- for $1 \leq j \leq f(n)$ we define:

$$\delta'((j, z), \Diamond) = \bigvee_{(z_1, z_2, z_3) \in \texttt{Preds}(z)} (j-1, z_1) \;\wedge\; (j, z_2) \;\wedge\; (j+1, z_3)\,.$$

The next proposition can be easily shown by induction on i. It relates the computation $C_0^w, C_1^w, C_2^w, \ldots$ of the deterministic Turing machine M on w and the predicate Acc corresponding to the AFA $A_w = (Q', \{\diamondsuit\}, \delta', q_0', F')$.

Proposition 1 *For all $i \in \mathbb{N}$ and $(j, z) \in Q'$ we have:*
$C_i^w(j) = z \;\Leftrightarrow\; Acc((j, z), \diamondsuit^i)$ *.*

Corollary 2 *M accepts w iff $\exists i : C_i^w(1) = (q_{acc}, \Box)$ iff $\exists i : Acc(q_0', \diamondsuit^i)$ iff $L(A_w) \neq \emptyset$.*

Theorem 3 1L-AFA-EMPTINESS *is PSPACE-complete.*

PROOF. Any problem P in PSPACE is decided by a deterministic Turing machine M with space bounded by a polynomial $p(n)$. Given such M, our (algorithmic) construction of A_w can be obviously done in polynomial time, and logarithmic space, wrt $|w|$. Hence every problem in PSPACE is logspace-reducible to 1L-AFA-EMPTINESS.
The membership of the emptiness problem in PSPACE is straightforward, even in the case of general AFA; it was shown in [5]. $\Box$

For deriving other PSPACE-hardness results, it is useful to have special simple forms of 1L-AFA for which the emptiness problem is still PSPACE-hard. We present one such form.
We call a *1L-AFA $A = (Q, \{\diamondsuit\}, \delta, q_0, F)$ simple* if each formula $\delta(q, \diamondsuit)$ is either a variable q' or is in the form $q_1 \wedge q_2$ or in the form $q_1 \vee q_2$.

Proposition 4 *The emptiness problem for simple 1L-AFA is PSPACE-hard.*

PROOF. We reduce 1L-AFA-EMPTINESS to the emptiness problem for simple 1L-AFA.
Let us consider a 1L-AFA $A = (Q, \{\diamondsuit\}, \delta, q_0, F)$. By f_q we denote a "fully-parenthesized form" of the formula $\delta(q, \diamondsuit)$; any subformula f of f_q is either a variable q' or is in the form $(f_1 \wedge f_2)$ or in the form $(f_1 \vee f_2)$. By $depth(f)$ we denote the depth of nesting in f: $depth(q) = 1$ and $depth(f_1 \wedge f_2) = depth(f_1 \vee f_2) = 1 + max\{depth(f_1), depth(f_2)\}$.
Let $m = max\{depth(f_q) : q \in Q\}$.
The above 1L-AFA A can be transformed to a simple 1L-AFA $A' = (Q', \{\diamondsuit\}, \delta', q_0', F')$ defined as follows:
$Q' = \{(1, q_0)\} \cup \{(i, f) : f$ is a subformula of some f_q and $m \geq i \geq depth(f)\}$,
$q_0' = (1, q_0)$,
$F' = \{(1, q) : q \in F\}$,
$\delta'((1, q), \diamondsuit) = (m, f_q)$,
if $i > depth(f)$ then $\delta'((i, f), \diamondsuit) = (i{-}1, f)$,

if $i = depth(f)$ and $f = (f_1\, op\, f_2)$ then $\delta'((i,f),\diamondsuit) = (i{-}1, f_1)\, op\, (i{-}1, f_2)$ for $op \in \{\wedge, \vee\}$.
It is obvious that the length of every word in $L(A')$ is divisible by m, and that $\diamondsuit^j \in L(A)$ iff $\diamondsuit^{jm} \in L(A')$. Thus $L(A) = \emptyset$ iff $L(A') = \emptyset$. $\square$

3 Additional remarks

We note that the idea of the above construction showing PSPACE-hardness of 1L-Afa-Emptiness is implicitly present in the seminal paper [1]. The proof of Theorem 3.4. in [1] shows that, given a deterministic Turing machine M with time (and thus also space) bounded by $f(n)$, we can construct an equivalent *alternating* Turing machine M' with space $O(\log f(n))$. The work of M' can be interpreted in our terms as follows: given w, M' checks if there is $i \leq f(|w|)$ such that A_w (defined wrt M) accepts $\diamondsuit^i$. M' cannot construct A_w explicitly; it just generates the binary description of a guessed $i \leq f(n)$ and then simulates i steps of A_w. M' has to be able to remember the current state (j,z) of A_w but this is no problem since it can use the tape for storing (the binary description of) j. The ability of M' to simulate A_w is obvious since the corresponding instructions of M' depend only on M, not on w.

It is also worth to note that 1L-Afa-Emptiness can be easily reduced to the emptiness problem for EP0L (as was also observed in [3]), for which the question of PSPACE-hardness had been an open problem until the solution in [6]. The other problems which were shown PSPACE-hard in [6], the emptiness (and other problems) for *binary systolic tree automata* (BSTA) and for the auxiliary model of "set systems," could be directly derived by using a simple adjustment of the idea used in the construction of A_w; we now sketch this adjustment.
In the computation $C_0^w, C_1^w, C_2^w, \ldots$ of a deterministic Turing machine M on w, the values $C_i^w(j{-}1), C_i^w(j), C_i^w(j{+}1)$ can be seen as a *substantiation* of $C_{i+1}^w(j)$; we can think of *substantiation rules* of the form

$$(j,z) \Leftarrow ((j{-}1, z_1), (j, z_2), (j{+}1, z_3))$$

where $(z_1, z_2, z_3) \in \mathtt{Preds}(z)$. Looking more closely, we note that each $C_{i+1}^w(j)$ can be substantiated by just two elements of C_i^w, namely by the pair $(C_i^w(j), C_i^w(j'))$ where $C_i^w(j') \in Q \times \Gamma$; in the case $C_i^w(j) \in Q \times \Gamma$ we have $j = j'$, a substantiation by one element of C_i^w – but this can still be viewed as a substantiation by the pair $(C_i^w(j), C_i^w(j))$ when needed for uniformity. Assuming M has space bounded by $f(n)$, for any input w with $|w| = n$ we can obviously construct $O((f(n))^2)$ substantiation rules

$$(j,z) \Leftarrow ((j, z_1), (j', z_2))$$

(where $j, j' \in \{1, 2, \ldots, f(n)\}$). We also note the following *determinism* (important for BSTA): for every pair $((j, z_1), (j', z_2))$ there is at most one (j, z) such that $(j, z) \Leftarrow ((j, z_1), (j', z_2))$ is a rule.

References

[1] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer, Alternation, J. ACM 28 (1) (1981) 114–133.

[2] K. Culik, J. Gruska, A. Salomaa, On a family of L languages resulting from systolic tree automata, Theoretical Comput. Sci. 23 (3) (1983) 231–242.

[3] M. Holzer, On emptiness and counting for alternating finite automata, in: J. Dassow, G. Rozenberg, A. Salomaa (eds.), Proceedings of Developments in Language Theory II (Magdeburg, Germany, 17-21 July 1995), World Scientific, 1996.

[4] P. Jančar, A. Kučera, F. Moller, Z. Sawa, DP lower bounds for equivalence-checking and model-checking of one-counter automata, Information and Computation 188 (2004) 1–19.

[5] T. Jiang, B. Ravikumar, A note on the space complexity of some decision problems for finite automata, Inf. Process. Lett. 40 (1) (1991) 25–31.

[6] A. Monti, A. Roncato, Completeness results concerning systolic tree automata and E0L languages, Inf. Process. Lett. 53 (1) (1995) 11–16.

[7] G. Rozenberg, A. Salomaa, The Mathematical Theory of L Systems, vol. 90 of Pure and Applied Mathematics, Academic Press, 1980.

[8] O. Serre, Parity games played on transition graphs of one-counter processes, in: L. Aceto, A. Ingólfsdóttir (eds.), Proceedings of FOSSACS 2006, vol. 3921 of Lecture Notes in Computer Science, Springer, 2006.

[9] J. Srba, Visibly pushdown automata: From language equivalence to simulation and bisimulation, in: Z. Ésik (ed.), Proceedings of CSL 2006, vol. 4207 of Lecture Notes in Computer Science, Springer-Verlag, 2006.