ELSEVIER

Theoretical Computer Science 231 (2000) 297–308

Theoretical Computer Science

www.elsevier.com/locate/tcs

Intractability of decision problems for finite-memory automata

Hiroshi Sakamoto *, Daisuke Ikeda

Department of Informatics, Kyushu University, Fukuoka 812-8581, Japan

Abstract

This paper deals with *finite-memory automata*, introduced in Kaminski and Francez (Theoret. Comput. Sci. 134 (1994) 329–363). With a restricted memory structure that consists of a finite number of registers, a finite-memory automaton can store arbitrary input symbols. Thus, the language accepted by a finite-memory automaton is defined over a potentially infinite alphabet. The following decision problems are studied for a general finite-memory automata A as well as for deterministic ones: the *membership problem*, i.e., given an A and a string w, to decide whether w is accepted by A, and the *non-emptiness problem*, i.e., given an A, to decide whether the language accepted by A is non-empty. The membership problem is P-complete, provided a given automaton is deterministic, and each of the other problems is NP-complete. Thus, we conclude that the decision problems considered are intractable.

Keywords: Finite-memory automata; Decision problems; Complexity; Formal languages

1. Introduction

Recently, Kaminski and Francez [3] introduced a new model of computation, the so-called *finite-memory automaton*, for dealing with languages defined over infinite alphabets. In this model, we assume a very restricted memory structure, such as a work tape, which consists of a finite number of registers, each of which may store an arbitrary symbol. When a finite-memory automaton reads an input symbol, it can check which register contains the symbol, or if none of them does. If the input symbol has already been stored, the finite-memory automaton changes its state using information exclusively in the register of its work tape in which the symbol is stored. If the input symbol has not been stored, it is written into a register depending only on the current state. Consequently, a finite-memory automaton deals with any finite and potentially infinite alphabet.

* Corresponding author.

E-mail addresses: hiroshi@i.kyushu-u.ac.jp (H. Sakamoto), daisuke@cc.kyushu-u.ac.jp (D. Ikeda)

0304-3975/00/$ - see front matter
PII: S0304-3975(99)00105-X

Compared to other computational models, finite-memory automata have the following features. First, they are a natural extension of finite-state automata (finite automata for short). Next, a finite-memory automaton can be finitely described analogously to other natural computational models. Finally, the class of finite-memory automata possesses several useful closure properties (cf. [3]).

However, there are some problems not solved in [3]. One of the open questions is the *inclusion problem* for finite-memory automata. Kaminski and Francez [3] gave a partial solution for this problem, i.e., they proved that the problem is decidable if one of the given automata is restricted to having at most two registers. The difficulty of a decision problem with a finite-memory automaton is caused by the fact that we cannot compute all the configurations defined over a potentially infinite alphabet.

On the other hand, when restricted to finite alphabets, every language accepted by a finite-memory automaton A is regular. Let $L(A)$ be the language accepted by A over an infinite alphabet and let $\Sigma_1, \Sigma_2, \ldots, \Sigma_n, \ldots$ be a sequence of finite alphabets such that $\Sigma_i \subset \Sigma_{i+1}$ for all $i \geqslant 1$. Then, we have $L(A) \cap \Sigma_i^* \subset L(A) \cap \Sigma_{i+1}^*$.

Thus, many decision problems with finite-memory automata arrive at the question as to whether we can efficiently find an evidence string $w \in \Sigma_n^*$ such that $w \notin L(A) \cap \Sigma_n^*$, and indeed, we have been able to solve several decision problems for finite-memory automata by reducing them to this question. In particular, we show that a polynomial upper bound for the length of a minimum string in $L(A)$ exists, provided $L(A) \neq \emptyset$.

Furthermore, admitting the setting of finite-memory automata to be quite reasonable, the following problem arises naturally:

Can finite-memory automata represent regular languages more compactly than finite automata?

Thus, we study the gap between finite-memory automata and finite automata with respect to computational complexity of several decision problems. The complexity of decision problems is also interesting in its own right, and these problems remained open in [3]. In particular, we shall deal with the following problems:

Membership: *Given a finite-memory automaton A and a string w, decide whether w is accepted by A* and

Non-emptiness: *Given a finite-memory automaton A, decide whether at least one non-empty string is accepted by A.*

Along with finite automata, the class of *deterministic* finite-memory automata has been introduced in [3]. The class of languages accepted by deterministic finite-memory automata is properly contained in the class of languages accepted by non-deterministic finite-memory automata. Hence, it is natural to study the decision problems for the case of deterministic finite-memory automata.

The corresponding decision problems for finite automata are defined analogously. The membership problem for finite automata is in NLOG, while it is in DLOG in the deterministic case. The non-emptiness problem for finite automata is NLOG-complete, even in the deterministic case (cf. [4]).

In the case of a finite-memory automaton, as one must keep the contents of all registers in order to simulate a computation, it cannot be expected that our membership problem is in NLOG. We have polynomial-time decidability of the membership problem in the deterministic case, due to the uniqueness of a computation for every string, and thus, it is also clear that the non-deterministic case of the membership problem is in NP. In this work we demonstrate P-completeness and NP-completeness, respectively, for the two cases.

Compared with the membership problem, it is not trivial that the non-emptiness problem is in NP because we cannot conclude that a *short string* in $L(A)$ exists, provided $L(A) \neq \emptyset$ by any result in [3]. As we mentioned above, we first show that the length of a shortest string in $L(A)$ is bounded by at least a polynomial in the number of states and registers of A. As a result, the non-emptiness problem is in NP. Moreover, we prove that the problems of the deterministic and non-deterministic cases are both NP-complete.

2. Preliminaries

We assume familiarity with formal languages and computational complexity theory (cf., e.g., [2, 4]). By $\mathbb{N} = \{0, 1, 2, \ldots\}$, we denote the set of all natural numbers. For a finite set S, let $\|S\|$ denote its cardinality. An *alphabet* is a set of symbols. For every alphabet $\mathscr{A}$, we denote by $\mathscr{A}^*$ the free monoid over $\mathscr{A}$ (cf. [2]). An element of $\mathscr{A}^*$ is called a *string*. For $\alpha \in \mathscr{A}^*$, we use $|\alpha|$, $[\alpha]$ and $\alpha[i]$ to denote the *length*, the *range* and the ith symbol of α, respectively, where the range of α is the set of the symbols appearing in α. Furthermore, for all $n \in \mathbb{N}$, we define $\mathscr{A}^n = \{\alpha \in \mathscr{A}^* \mid |\alpha| = n\}$. Any set $L \subseteq \mathscr{A}^*$ is called a *language*. A *class* of languages is a set of languages containing at least one non-empty language.

In particular, let $\Omega = \{a_i \mid i \in \mathbb{N}\}$ be any fixed countably infinite alphabet. By Σ, we denote any finite subset of Ω. Let $\sharp$ be a special symbol not belonging to Ω. An *assignment* is a finite string $x_1 x_2 \cdots x_n \in (\Omega \cup \{\sharp\})^n$ such that if $x_i = x_j$ and $i \neq j$, then $x_i = \sharp$ for all $1 \leqslant i, j \leqslant n$.

Let S, S_1, S_2 be sets; for a mapping $\psi : S_1 \mapsto S_2$, we write ψ^{-1} to denote the inverse of ψ, provided it exists. A mapping $\pi : S \mapsto S$ is said to be a *permutation* over S if π is a one-to-one and onto mapping over S.

Definition 1 (*cf. Kaminski and Francez* [3]). A *finite-memory automaton* is denoted by a 6-tuple $A = \langle Q, q_0, \tau, \varrho, \delta, F \rangle$, where (1) Q is a finite set and $q_0 \in Q$, the elements of Q are called *states* and q_0 is called the *initial state*, (2) $\tau \in (\Omega \cup \{\sharp\})^k$ is an assignment of length k called the *initial assignment*, (3) ϱ, called the *reassignment*, is a partial function from Q to $\{1, 2, \ldots, k\}$, that is, for each $p \in Q$, $\varrho(p) \in \{1, 2, \ldots, k\}$ or undefined, (4) $\delta \subseteq Q \times \{1, 2, \ldots, k\} \times Q$ is called the *transition relation*, and (5) $F \subseteq Q$ is called the set of *final states*.

Let A be a finite-memory automaton. We call a register of A a *window*. When A is in a state p and reads a symbol a, then A changes its state into q if $(p,i,q)\in\delta$ provided a is the ith symbol of A's current assignment α. If $a\notin[\alpha]$, then A rewrites the $\varrho(p)$th window by a and changes the state if $(p,\varrho(p),q)\in\delta$.

More formally, the computations of a finite-memory automaton A are defined as follows. Every pair of a state and an assignment is called a *configuration*. The pair (q_0,τ) is referred to as the *initial configuration*. All configurations with final states are called *final configurations*. We define a binary relation $\vdash$ as follows. Let α and β be assignments, and let $p,q\in Q$. Then, $(p,\alpha)\vdash(q,\beta)$ if there exist $a\in\Omega$, $i\in\{1,2,\ldots,k\}$, and $(p,i,q)\in\delta$ such that:

1. $a=\alpha[i]$ and $\alpha=\beta$ or
2. $a\notin[\alpha]$, $\varrho(p)=i$, $\beta[i]=a$, and $\alpha[j]=\beta[j]$ for all $j\neq i$.

When necessary, we write $(p,\alpha)\vdash^a(q,\beta)$ by specifying the symbol a. If there exists a sequence of configurations $c_0,c_1,\ldots,c_n$ such that $c_i\vdash^{b_i}c_{i+1}$ for all $0\leqslant i\leqslant n$, then we write $c_0\vdash^w c_n$, where $w=b_0b_1\cdots b_n$. Finally, we use $c_0\vdash^* c_n$ provided $w\in\Omega^*$ exists such that $c_0\vdash^w c_n$. A is said to *accept* a string w if $c_0\vdash^w c_n$, where c_0 is the initial configuration and c_n is a final configuration of A. The language accepted by A is denoted by $L(A)$ and defined to be the set of all strings $w\in\Omega^*$ accepted by A.

Definition 2 (*cf. Kaminski and Francez* [3]). A finite-memory automaton is said to be *deterministic* if for each $p\in Q$ and each $1\leqslant i\leqslant k$, the value of $\varrho(p)$ is defined and there exists exactly one $q\in Q$ such that $(p,i,q)\in\delta$, where k is the length of the initial assignment.

By Definition 2, the class of deterministic finite-memory automata is a subset of the set of all finite-memory automata. To see that it is a *proper* subset, let $A=\langle Q,q_0,\tau,\varrho,\delta,F\rangle$ be any deterministic finite-memory automaton. By exchanging F with $Q\backslash F$, we obtain a deterministic finite-memory automaton A' such that $L(A')=\Omega^*\backslash L(A)$, since the computation of A is unique for each input. Thus, the class of deterministic finite-memory automata is closed under complement but the class of all finite-memory automata is not (cf. [3]).

Similar to the case of finite automata, a finite-memory automaton can be described as a directed graph whose nodes denote states and edges denote transitions. There is an edge labeled by k from node s_i to node s_j if a relation (s_i,k,s_j) is defined. A node s is labeled by $\varrho(s)$ if ϱ is defined for s.

3. The membership problem

In this section, we study the membership problem for finite-memory automata as well as for deterministic ones. An input is a finite-memory automaton A and a string [1]

[1] In decision problems, we assume a symbol $a_i\in\Omega=\{a_j\mid j\in\mathbb{N}\}$ to be encoded by a followed by $\mathrm{bin}(i)\in\{0,1\}^*$, where $\mathrm{bin}(i)$ denotes i in binary.

w. The problem is then to decide whether $w \in L(A)$. The membership problems for deterministic finite-memory automata, and finite-memory automata in general, are denoted by MEMD and MEM, respectively.

Let $A = \langle Q, q_0, \tau, \varrho, \delta, F\rangle$ be a finite-memory automaton and $w \in \Omega^*$. Since for any $p \in Q$, at most one $\varrho(p)$ is defined, then $||\varrho|| \leqslant ||Q||$. Moreover, for any actual configuration of A on w, at most $|\tau|$ different symbols exist in τ. Thus, the time to decide whether $w \in L(A)$ is $O(||Q||^2 \cdot |\tau| + |w|)$. It directly follows that MEMD $\in$ P and MEM $\in$ NP.

A Turing machine can simulate a computation of a finite automaton for a string in NLOG because there exist, at most, n configurations, where n is the number of states of the automaton. On the other hand, the number of different configurations of a finite-memory automaton is potentially exponential due to the possibility of mapping of symbols into the windows of the initial assignment. Thus, there is no possibility that the problem MEMD is also in NLOG, even if the automaton is deterministic. In this section, we demonstrate NP-hardness of MEM and P-hardness of MEMD.

We begin with the P-hardness of MEMD by reducing the *monotone circuit value problem*, denoted by monotone CVP. The monotone CVP is defined as follows (cf. [1]). Let $X = \{x_i \mid i \in \mathbb{N}\}$ be the set of Boolean variables. We denote a circuit C over X by a sequence $C_1, C_2, \ldots, C_n$. Each component C_i $(1 \leqslant i \leqslant n)$ is called a *gate* of C.[2] A circuit C is called *monotone* if C contains no negation. A *truth assignment* of a circuit of m variables is a mapping $f : \{x_1, \ldots, x_m\} \mapsto \{0, 1\}$. The *value* of C with respect to f, denoted by $f(C)$, is defined as usual (cf. [4]). Now, the monotone CVP is, *given a monotone circuit* C *and a truth assignment* f, *to decide whether* $f(C) = 1$.

Theorem 1. MEMD *is* P-*complete.*

Proof. We show that monotone CVP is log-space reducible to MEMD. Let $C = C_1, \ldots, C_n$ be a monotone circuit defined by variables $x_1, \ldots, x_m$ and let f be any fixed truth assignment of C. First, we construct a string $w \in \Omega^*$ from C as follows. For each $1 \leqslant i \leqslant n$, define the string $w(i)$ as

$$w(i) = \begin{cases} a_i & \text{if } C_i = x_i, \\ a_k a_j a_i & \text{if } C_i = C_k \wedge C_j \text{ or } C_i = C_k \vee C_j. \end{cases}$$

The string w is the concatenation of all $w(i)$'s for $1 \leqslant i \leqslant n$, that is, $w = w(1)w(2)\cdots w(n)$.

Next, we must construct a corresponding finite-memory automaton $A = \langle Q, q_0, \tau, \varrho, \delta, F\rangle$. We set $\tau = \sharp^{2n}$ and define the set Q of states as well as the reassignment ϱ by distinguishing the following cases for all $i \in \{1, \ldots, n\}$. If $C_i = x_i$ we add the state p_i to Q and set $\varrho(p_i) = i$ provided $f(x_i) = 1$ and $\varrho(p_i) = n + i$ otherwise. If C_i is not an input gate, we add four states $p_i, p_{i_1}, p_{i_2}, p_{i_3}$. Furthermore, we set $\varrho(p_{i_2}) = i$, $\varrho(p_{i_3}) = n + i$ in case $C_i = C_k \wedge C_j$ and $\varrho(p_{i_2}) = n + i$, $\varrho(p_{i_3}) = i$ in case $C_i = C_k \vee C_j$.

[2] Without loss of generality, we assume the m input gates of a circuit C to be $C_1, \ldots, C_m$, i.e., $C_i = x_i$ for $i = 1, \ldots, m$.

For each $1 \leqslant i \leqslant n$, the transition relation δ is defined as follows:

1. If $C_i = x_i$ then define $(p_i, \varrho(p_i), p_{i+1})$.
2. Let $C_i = C_k \wedge C_j$; we distinguish the following cases:
   (a) if $i < n$, define (p_i, k, p_{i_1}), $(p_i, n+k, p_{i_3})$, (p_{i_1}, j, p_{i_2}), $(p_{i_1}, n+j, p_{i_3})$, (p_{i_2}, i, p_{i+1}), (p_{i_3}, j, p_{i_3}), $(p_{i_3}, n+j, p_{i_3})$, and $(p_{i_3}, n+i, p_{i+1})$,
   (b) if $i = n$, define (p_i, k, p_{i_1}), (p_{i_1}, j, p_{i_2}), and (p_{i_2}, i, p_{i+1}).
3. Let $C_i = C_k \vee C_j$; we distinguish the following cases:
   (a) if $i < n$, define $(p_i, n+k, p_{i_1})$, (p_i, k, p_{i_3}), $(p_{i_1}, n+j, p_{i_2})$, (p_{i_1}, j, p_{i_3}), $(p_{i_2}, n+i, p_{i+1})$, (p_{i_3}, j, p_{i_3}), $(p_{i_3}, n+j, p_{i_3})$, and (p_{i_3}, i, p_{i+1}),
   (b) if $i = n$, define $(p_i, n+k, p_{i_1})$, (p_i, k, p_{i_3}), (p_{i_1}, j, p_{i_3}), (p_{i_3}, j, p_{i_3}), $(p_{i_3}, n+j, p_{i_3})$, and (p_{i_3}, i, p_{i+1}).

Finally, we set $q_0 = p_1$ and $F = \{p_{n+1}\}$. For each state in Q, at most three transitions are defined. Thus, A can be computed in $O(\log n)$ space. It remains to show that $w \in L(A)$ iff $f(C) = 1$.

Claim. *Let $i \geqslant 1$, $(q_0, \tau) \vdash^{w(1)w(2)\cdots w(i)} (p_{i+1}, \alpha_{i+1})$ and a the last symbol of $w(i)$. Then, $\alpha_{i+1}[i] = a$ if $f(C_i) = 1$ and $\alpha_{i+1}[n+i] = a$ if $f(C_i) = 0$.*

For each $1 \leqslant i \leqslant m$, $|w(i)| = 1$, and for all $p \neq q \in Q$, $\varrho(p) \neq \varrho(q)$. Thus, by the definition of A, *Claim* is true for each $i = 1, \ldots, m$. Let us take an $i \geqslant m$. Then, $C_{i+1} = C_k \wedge C_j$ or $C_{i+1} = C_k \vee C_j$, where $k < j \leqslant i$.

First, consider $C_{i+1} = C_k \wedge C_j$. In the case that $f(C_{i+1}) = 1$, $f(C_k) = 1$ and $f(C_j) = 1$. By the induction hypothesis, $\alpha_{k+1}[k] = a_k$ and $\alpha_{j+1}[j] = a_j$. Since these two symbols are never replaced, $\alpha_{i+1}[k] = a_k$ and $\alpha_{i+1}[j] = a_j$. Thus, $(p_{i+1}, \alpha_{i+1}) \vdash^{a_k a_j} (p_{(i+1)_2}, \alpha_{i+1})$.

Clearly, $a_{i+1} \notin [\alpha_{i+1}]$. It follows that $(p_{(i+1)_2}, \alpha_{i+1}) \vdash^{a_{i+1}} (p_{i+2}, \alpha_{i+2})$ and $\alpha_{i+2}[i+1] = a_{i+1}$ since $\varrho(p_{(i+1)_2}) = i+1$. Thus, $(p_{i+1}, \alpha_{i+1}) \vdash^{w(i+1)} (p_{i+2}, \alpha_{i+2})$ and $\alpha_{i+2}[i+1] = a_{i+1}$.

In case of $f(C_{i+1}) = 0$, either $f(C_k) = 0$ or $f(C_j) = 0$. Suppose that $f(C_k) = 0$. By the induction hypothesis, $\alpha_{k+1}[n+k] = a_k$, and thus, $\alpha_{i+1}[n+k] = a_k$. Hence, $(p_{i+1}, \alpha_{i+1}) \vdash^{a_k} (p_{(i+1)_3}, \alpha_{i+1}) \vdash^{a_j} (p_{(i+1)_3}, \alpha_{i+1})$. Then, we also have $(p_{i+1}, \alpha_{i+1}) \vdash^{a_k a_j a_{i+1}} (p_{i+2}, \alpha_{i+2})$.

Suppose the contrary $f(C_{i+1}) = 0$ such that $f(C_k) = 1$ and $f(C_j) = 0$. Then, $\alpha_{i+1}[k] = a_k$ and $\alpha_{i+1}[n+j] = a_j$. Thus, $(p_{i+1}, \alpha_{i+1}) \vdash^{a_k} (p_{(i+1)_1}, \alpha_{i+1})$ and $(p_{(i+1)_1}, \alpha_{i+1}) \vdash^{a_j} (p_{(i+1)_3}, \alpha_{i+1})$. Since $\varrho(p_{(i+1)_3}) = n+i+1$, we have $(p_{i+1}, \alpha_{i+1}) \vdash^{a_k a_j a_{i+1}} (p_{i+2}, \alpha_{i+2})$. Hence, $(p_{i+1}, \alpha_{i+1}) \vdash^{w(i+1)} (p_{(i+2)}, \alpha_{i+2})$ and $\alpha_{i+2}[n+i+1] = a_{i+1}$.

Second, consider $C_{i+1} = C_k \vee C_j$. The reassignment ϱ on C_{i+1} is obtained from the reassignment for $C_{i+1} = C_k \wedge C_j$ by exchanging $i+1$ with $n+i+1$, j with $n+j$, and k with $n+k$, and the same exchanges apply to δ. Thus, this case can be handled analogously and Claim is true.

Finally, for each $1 \leqslant i \leqslant n-1$, we have $(p_1, \alpha_1) \vdash^{w(1)\cdots w(n-1)} (p_n, \alpha_n)$. The computation $(p_n, \alpha_n) \vdash^{w(n)} (p_{n+1}, \alpha_{n+1})$ is defined only if $f(n) = 1$. Therefore, $f(C) = 1$ iff $w \in L(A)$ because $F = \{p_{n+1}\}$. □

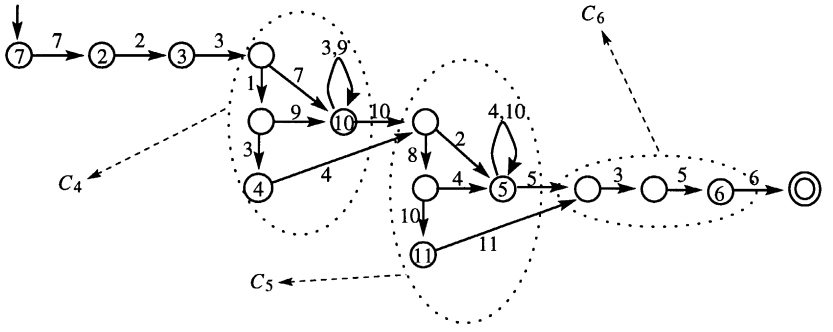

Fig. 1. The finite-memory automaton computed in Example 1.

Example 1. We illustrate our reduction by using a monotone circuit C and a truth assignment f. Let $C = C_1, \ldots, C_6$ such that each C_i for $1 \leqslant i \leqslant 3$ is the variable, $C_4 = C_1 \wedge C_3$, $C_5 = C_2 \vee C_4$, and $C_6 = C_3 \wedge C_5$. Let $f(x_1) = 0$ and $f(x_2) = f(x_3) = 1$. The string w is the concatenation of $w(1) = a_1$, $w(2) = a_2$, $w(3) = a_3$, $w(4) = a_1a_3a_4$, $w(5) = a_2a_4a_5$, and $w(6) = a_3a_5a_6$. The initial assignment of A is $\sharp^{12}$. The resulting A is shown in Fig. 1.

We next show the NP-hardness of MEM by reducing the circuit SAT problem [4] to it. For a given circuit C, the circuit SAT is the problem of deciding whether there exists a truth assignment f such that $f(C) = 1$. Since there are 2^m possible truth assignments for m variables, the main part of the proof is a one-to-one mapping between the truth assignments of C and the configurations of a finite-memory automaton A constructed from C.

Although we can construct a trivial A that has 2^m configurations for a fixed finite alphabet by introducing many states, because we want to show a log-space reduction, we must represent A in more compact way. Moreover, it is easy to construct a compact A by using a loop of transitions. However, the number of configurations of such an A is potentially infinite, even if an alphabet is fixed to be finite. In this way, we cannot assume a one-to-one mapping preserving the consistency $f(C) = 1$ iff $w \in L(A)$. In the following theorem, we present a way to represent a compact and loop-free set of transitions corresponding to the truth assignments.

Theorem 2. MEM *is* NP-*complete.*

Proof. We show that circuit SAT is log-space reducible to MEM. The construction of the string w remains unchanged except that we add a_0 to it as its first symbol. Suppose the target circuit $C = C_1, C_2, \ldots, C_n$ has m variables and $C_i = x_i$ for all $i = 1, \ldots, m$.

First, define $A = \langle Q, q_0, \tau, \varrho, \delta, \{p\}\rangle$ as follows. Let $Q = \{q_0, p\} \cup \{p_i,\ q_i \mid 1 \leqslant i \leqslant m\}$ and $\tau = \sharp^{2m+1}$. Let $\varrho(q_0) = 2m + 1$ and $(q_0, 2m + 1, p_1)$, $(q_0, 2m + 1, q_1) \in \delta$. For each $1 \leqslant i \leqslant m$, let $\varrho(p_i) = i$ and $\varrho(q_i) = m + i$ as well as (p_i, i, p_{i+1}), $(p_i, i, q_{i+1}) \in \delta$

and $(q_i, m+i, q_{i+1})$, $(q_i, m+i, p_{i+1}) \in \delta$, where ϱ and δ are undefined on the final state p.

This automaton A is loop-free. Thus, by the definition A, it is clear that for every $w \in \Omega^*$, $w \in L(A)$ iff $|w| \geqslant m+1$. Let $C(A,p) = \{\alpha \mid (q_0, \tau) \vdash^* (p, \alpha)\}$. For each $1 \leqslant i \leqslant 2m+1$, there exists exactly one $q \in Q$ such that $\varrho(q) = i$. Thus, there exists a one-to-one mapping from $\alpha \in C(A,p)$ to $\beta \in \{0,1\}^m$ such that for each $1 \leqslant j \leqslant m$, $\alpha[j] = \sharp$ and $\alpha[m+j] \neq \sharp$ iff $\beta[j] = 0$, and $\alpha[j] \neq \sharp$ and $\alpha[m+j] = \sharp$ iff $\beta[j] = 1$.

Thus, we can construct a finite-memory automaton A that simulates each assignment $\beta \in \{0,1\}^m$ for the variables of C by the computations $(q_0, \tau) \vdash^{a_0 w(1) \cdots w(m)} (p, \alpha)$.

The other part of the definition of A for $C_i = C_j \wedge C_k$ and $C_i = C_j \vee C_k$ is exactly the same as in Theorem 1. It remains to handle the case $C_i = \neg C_j$. We add the states p_i, p_{i_1}, p_{i_2} and p_{i+1} and define ϱ and δ as follows. Let $\varrho(p_{i_1}) = m+i$ and $\varrho(p_{i_2}) = i$ and in case $i < n$, let $(p_i, j, p_{i_1}), (p_{i_1}, m+i, p_{i+1}), (p_i, m+j, p_{i_2}), (p_{i_2}, i, p_{i+1}) \in \delta$, and in case $i = n$, let $(p_n, m+j, p_{n_2}), (p_{n_2}, n, p_{n+1}) \in \delta$.

The gate $C_i = \neg C_j$ corresponds to the string $w(i) = a_j a_i$. The automaton A reads a_j on the state p_i and remembers the value C_j as the position of the window containing a_j. According to the position j or $m+j$, it changes its state to p_{i_1} or p_{i_2} and memorizes the value of C_i by putting the next input a_i into the $(m+i)$th or ith window, respectively. Thus, it is consistent that $f(C) = 1$ iff $w \in L(A)$. □

4. The non-emptiness problem

In this section, we study a further decision problem for finite-memory automata, the non-emptiness problem, denoted by ¬ EMP, which is, *given a finite-memory automaton A, to decide whether* $L(A) \neq \emptyset$.

A restriction of ¬ EMP to deterministic finite-memory automata is denoted by ¬ EMP. Although a finite-memory automaton deals with an infinite alphabet, we can reduce it to a finite alphabet using the next proposition.

Proposition 1 (cf. Kaminski and Francez [3]). *Let A be a finite-memory automaton, τ be the initial assignment of A, and π be a permutation over Ω with $\pi(a) = a$ for all $a \in [\tau]$. Then, $\pi(L(A)) = L(A)$, where $\pi(L(A)) = \{\pi(w) \mid w \in L(A)\}$.*

Proposition 1 shows that if a string w is accepted by A, then the string $\pi(w)$ for any π is accepted by A. Vice versa, if $w \notin L(A)$, then so is $\pi(w)$ for any permutation π.

Moreover, Kaminski and Francez [3] showed that if A accepts a string w of length n, then A also accepts a string w' of length n such that $\|[w']\| \leqslant k$, where k is the length of the initial assignment. Therefore, the $\|[w']\|$ of any string w accepted by A can become finite. Thus, we can show that ¬ EMP is in PSPACE as follows: a Turing machine selects a symbol among a finite alphabet as the next input symbol, simulates the move of a given finite-memory automaton using its work tape to keep the contents of the assignment, and accepts the input if the automaton accepts the concatenation of

all selected symbols. In this way, the computation time depends on the length of the concatenation.

Given a finite-memory automaton A with a finite alphabet, Kaminski and Francez [3] constructed an equivalent finite automaton such that the number of the state is exponential in the size of A. Therefore, there exists a string such that its length is exponential in the size of A and it is accepted by A. This means that we cannot conclude $\neg$ EMP $\in$ NP since it does not provide an upper bound on the length of a minimum string in $L(A)$.

Now, we show an upper bound on the length of minimum strings accepted by A. Clearly, in order to establish $\neg$ EMP $\in$ NP, this upper bound must be polynomial in the size of A.

Let $A = \langle Q, q_0, \tau, \varrho, \delta, F\rangle$ be a finite-memory automaton and let α be an assignment of A, where $\|Q\| = n$ and $|\tau| = k$. By $C(A)$, we denote the set $\{(p,\alpha) \mid (q_0,\tau) \vdash^* (p,\alpha)\}$. A state p is said to be *reachable* if $(p,\alpha) \in C(A)$ for some α. By $\hat{\alpha}$, we denote the set $\{i \mid 1 \leqslant i \leqslant k,\ \alpha[i] \neq \sharp\}$.

Theorem 3. *If a finite-memory automaton $A = \langle Q, q_0, \tau, \varrho, \delta, F\rangle$ accepts at least one string w, then there is also a string w' such that $|w'| \leqslant kn$ and $w' \in L(A)$, where $k = |\tau|$ and $n = \|Q\|$.*

Proof. Let $A = \langle Q, q_0, \tau, \varrho, \delta, F\rangle$ be a finite-memory automaton and $w = b_1 b_2 \cdots b_m$ $(m \geqslant 1)$ be in $L(A)$. Since $w \in L(A)$, there exists a computation $c = (p_0,\alpha_0) \vdash^{b_1} \cdots \vdash^{b_m} (p_m,\alpha_m)$, where $p_0 = q_0$, $\alpha_0 = \tau$, and $p_m \in F$. Without loss of generality, we can assume that p_m is the first state in c belonging to F. Consider any computation step of the form $(p_{i-1},\alpha_{i-1}) \vdash^{b_i} (p_i,\alpha_i)$ such that $\hat{\alpha}_{i-1} = \hat{\alpha}_i$. There are two cases to be distinguished.

Case 1: $b_i \in [\alpha_{i-1}]$. Consider a sequence of transitions $(p_{i-1},\alpha_{i-1})\ldots$ of length n. If the last state is not in F, then a cycle occurs, which can be eliminated from w.

Case 2: $b_i \notin [\alpha_{i-1}]$. Then, the reassignment ϱ is applied and we have two cases to consider. Let $j = \varrho(p_{i-1})$ such that $\alpha_{i-1}[j] \neq \sharp$. Hence, there has already been a symbol in $\alpha_{i-1}[j]$, say t_j. In this case, we can replace b_i in w by t_j without altering the computation, and hence, we go back to Case 1. Now, suppose $\hat{\alpha}_{i-1} \neq \hat{\alpha}_i$, namely, $\alpha_{i-1}[j] = \sharp$, where $j = \varrho(p_{i-1})$. Then, we replace $\sharp$ by b_i. Note that this case can happen only k times. Thus, the resulting string w' can have a length of, at most, kn. □

By Proposition 1 and Theorem 3, it is easy to construct a non-deterministic Turing machine that accepts $\neg$ EMP in polynomial time.In order to prove the NP-hardness of $\neg$ EMP, we consider the *satisfiability problem*, denoted by SAT, which we define next. Let $n \geqslant 1$; a *Boolean formula* G over $X = \{x_1,\ldots,x_{2n}\}$, is said to be in *conjunctive normal form* provided $G = \bigwedge_{j=1}^{m} C_j$, where each C_j is a clause, that is, $C_j = \ell_{j_1} \vee \ell_{j_2} \vee \cdots \vee \ell_{j_k}$. Here all $\ell_{j_z} \in X$ for $z = 1,\ldots,k$. Note that $k \leqslant 2n$. We refer to $\ell_1,\ldots,\ell_{2n}$ as literals, $x_1,\ldots,x_n$ variables, and $x_{n+1},\ldots,x_{2n}$ negated variables denoted $x_{n+i} = \neg x_i$ for all

$i=1,\ldots,n$. For a clause C_j we use $\langle C_j\rangle$ to denote the set $\{j_1,j_2,\ldots,j_k\}$. Now, SAT is the problem, *given a Boolean formula G in conjunctive normal form, to decide whether there exists a truth assignment f of G such that* $f(G)=1$. SAT is NP-complete (cf. [4]).

Theorem 4. $\neg$ EMP *is* NP-*complete.*

Proof. We show a log-space reduction from SAT to our $\neg$ EMP. Let $n\geqslant 1$, $X=\{x_1,\ldots,x_{2n}\}$, and let $G=\bigwedge_{j=1}^{m} C_j$ be a Boolean formula over X in conjunctive normal form. The desired finite-memory automaton $A=\langle Q,q_0,\tau,\varrho,\delta,F\rangle$ is defined as follows. $Q=\{q_i \mid 0\leqslant i\leqslant 2n+m+1\}$, $\tau=\sharp^{2n+1}$, $F=\{q_{2n+m+1}\}$, $\varrho(q_0)=2n+1$ and for each $i\leqslant 2n$, we set $\varrho(q_i)=i$. Finally, we define $\delta=\delta_1\cup\delta_2\cup\delta_3$, where

$$\delta_1=\{(q_0,2n+1,q_1),(q_0,2n+1,q_{n+1}),(q_n,n,q_{2n+1}),(q_{2n},2n,q_{2n+1})\},$$

$$\delta_2=\{(q_i,i,q_{i+1}),(q_i,i,q_{n+i+1}) \mid 1\leqslant i\leqslant n-1\}\cup\{(q_i,i,q_{i+1}),(q_i,i,q_{i-n+1}) \mid n+1\leqslant i\leqslant 2n-1\},\text{ and}$$

$$\delta_3=\{(q_{2n+i},j,q_{2n+i+1}) \mid 1\leqslant i\leqslant m, j\in\langle C_i\rangle\}.$$

Note that the finite-memory automaton A is loop-free. While A moves from either q_1 or q_{n+1} to q_{2n+1}, A writes n distinct symbols on either the ith or the $(n+i)$th window $(1\leqslant i\leqslant n)$. Let α be the assignment such that n distinct symbols have already been written on the windows. Then, α corresponds to a truth assignment f. That is, if $f(x_i)=1$ then $\alpha[i]\neq\sharp$ and $\alpha[n+i]=\sharp$, and if $f(\neg x_i)=1$ then $\alpha[i]=\sharp$ and $\alpha[n+i]\neq\sharp$.

Now, we prove that G is satisfiable iff q_{2n+m+1} is reachable. Since ϱ is undefined for each q_{2n+i} $(1\leqslant i\leqslant m+1)$, all computations from q_{2n+1} depend only on α. For each $1\leqslant k\leqslant m$, x_i appearing in C_k exists such that $f(x_i)=1$ $(1\leqslant i\leqslant n)$ iff $\alpha[i]\neq\sharp$ and $\alpha[n+i]=\sharp$. Therefore, C_k is satisfiable iff q_{2n+k+1} is reachable from q_{2n+k} by δ_3. Similarly, in the case of $\neg x_i$, we have that q_{2n+1} is definitely reachable from q_0. Thus, G is satisfiable iff q_{2n+m+1} is reachable. □

Example 2. We exemplify the construction outlined above by using a 3-CNF $G=(x_1\vee x_2\vee x_3)\wedge(\neg x_1\vee\neg x_2\vee\neg x_3)\wedge(\neg x_1\vee x_2\vee\neg x_3)$. The resulting automaton is displayed in Fig 2. The initial assignment is $\sharp^7$.

Next, we show that $\neg$ EMP remains NP-complete when restricted to deterministic finite-memory automata. Obviously, $\neg$ EMPD $\in$ NP by Theorem 4. We show that SAT is also log-space reducible to $\neg$ EMPD. However, the technique used in Theorem 4 cannot be applied directly to the deterministic case because the automaton constructed there is non-deterministic. Just converting the obtained non-deterministic automata into deterministic ones is not feasible, since it could lead to automata having size exponential in the length of the formulae G. Therefore, we must reduce SAT directly to $\neg$ EMPD.

Theorem 5. $\neg$ EMPD *is* NP-*complete.*

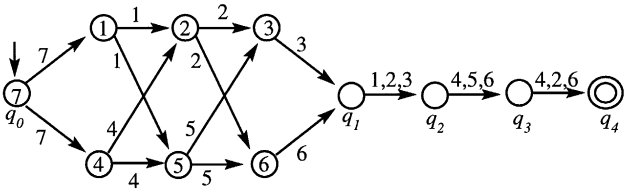

Fig. 2. The finite-memory automaton A reduced from G in Example 2.

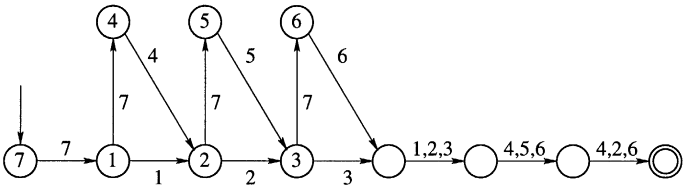

Fig. 3. The deterministic A computed from G in Example 2.

Proof. Let $n \geqslant 1$, $X = \{x_1, \ldots, x_{2n}\}$, and $G = \bigwedge_{j=1}^{m} C_j$ a Boolean formula over X in conjunctive normal form. Then, a finite-memory automaton $A = \langle Q, q_0, \tau, \varrho, \delta, F \rangle$ is computed as follows. Let $Q = \{q_i \mid 0 \leqslant i \leqslant 2n + m + 1\}$, let $\tau = \sharp^{2n+1}$, let $F = \{q_{2n+m+1}\}$, and set $\varrho(q_0) = 2n + 1$ as well as $\varrho(q_i) = i$ for all $1 \leqslant i \leqslant 2n$. Furthermore, we set $\delta = \delta_1 \cup \delta_2 \cup \delta_3$, where

1. $\delta_1 = \{(q_0, 2n + 1, q_1), (q_n, n, q_{2n+1}), (q_{2n}, 2n, q_{2n+1})\}$,
2. $\delta_2 = \{(q_i, i, q_{i+1}), (q_i, 2n + 1, q_{n+i}), (q_{n+i}, n + i, q_{i+1}) \mid 1 \leqslant i \leqslant n - 1\}$,
3. $\delta_3 = \{(q_{2n+i}, j, q_{2n+i+1}) \mid 1 \leqslant i \leqslant m,\ j \in \langle C_i \rangle\}$.

Note that each path from the initial state q_0 to q_{2n+2} contains no loop. Each path from q_0 to q_{2n+2} contains exactly one of two edges labeled i and $n+i$ for all $1 \leqslant i \leqslant n$. Then, for every string w we have $(q_0, \tau) \vdash^w (q_{2n+2}, \alpha)$ iff either $\alpha[i] \neq \sharp$ or $\alpha[n + i] \neq \sharp$ for all $1 \leqslant i \leqslant n$. Moreover, there exist exactly 2^n distinct configurations (q_{2n+2}, α). Thus, there exists a one-to-one mapping from the truth assignments for all variables $x_1, x_2, \ldots, x_n$ to the possible assignments on q_{2n+2}. The remaining part of the definition of the automaton A is the same as in the proof of Theorem 4. Thus, G is satisfiable if the final state q_{2n+m+1} is reachable.

However, since some states are not labeled and some transitions are not defined, the automaton is still not deterministic. By adding an extra state as a trash box, it is easy to complete the definition of A so as to be deterministic. We omit the details. Clearly, the computation of A is in log-space. □

Example 3. Again, let us take the 3-CNF in Example 2 and show the reduced deterministic finite-memory automaton A in Fig. 3. This automaton A also has the initial assignment $\sharp^7$. Although some transitions are not defined in A, for instance, there is no transition for the initial state and the first window, we put a special state p as a trash box for undefined transitions. Thus, this automaton is essentially deterministic.

Table 1
The complexity of decision problems

	Membership	Non-emptiness
DFMA	P-complete	NP-complete
NFMA	NP-complete	NP-complete

5. Summary and conclusions

We studied the decision problems MEM and ¬ EMP for non-deterministic finite-memory automata as well as for their deterministic counterparts. All results obtained are listed in Table 1. Looking at Table 1, we conclude that there is strong evidence that finite-memory automata are more compact representation for regular languages than finite automata unless $P = NP$.

The membership problems for deterministic and non-deterministic finite automata are in DLOG and NLOG. The problems for deterministic and non-deterministic finite-memory automata shift to P and NP, respectively. Thus, we conclude that the problems are intractable. Similarly, we conclude that the non-emptiness problem for finite-memory automata is also intractable. Moreover, while the non-emptiness problems for both deterministic and non-deterministic finite automata are NLOG-complete, the same problems for both deterministic and non-deterministic finite-memory automata shift to NP.

Unfortunately, the inclusion problem is not known to be decidable. Given two finite-memory automata, the inclusion problem is whether the language of one of them is included by the other. We only know the result that the inclusion problem is decidable if one of the two finite-memory automata is deterministic and is restricted to be of at most two registers (cf. [3]).

References

[1] L.M. Goldschlager, The monotone and planar circuit value problems are log space complete for P, SIGACT News 9 (1977) 25–29.
[2] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company, Reading, MA, 1979.
[3] M. Kaminski, N. Francez, Finite-memory automata, Theoret. Comput. Sci. 134 (1994) 329–363.
[4] C.H. Papadimitriou, Computational Complexity, Addison-Wesley Publishing Company, Reading, MA, 1994.