Information and Computation 237 (2014) 294–302

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Oblivious two-way finite automata: Decidability and complexity ☆,☆☆

Martin Kutrib [a], Andreas Malcher [a], Giovanni Pighizzini [b,*]

[a] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
[b] Dipartimento di Informatica, Università degli Studi di Milano, Via Comelico 39, 20135 Milano, Italy

ARTICLE INFO

Article history:
Received 12 June 2013
Received in revised form 3 March 2014
Available online 16 April 2014

Keywords:
Two-way finite automata
Oblivious computations
Descriptional complexity
Decidability

ABSTRACT

We investigate the descriptional complexity and decidability of obliviousness for two-way finite automata. In particular, we consider the simulation of two-way deterministic finite automata (2DFAs) by oblivious 2DFAs, the simulation of oblivious 2DFAs by sweeping 2DFAs and one-way nondeterministic finite automata (1NFAs) as well as the simulation of sweeping 2DFAs by 1NFAs. In all cases exponential lower bounds on the number of states are obtained for languages over an alphabet with at most four letters. Finally, a procedure for deciding obliviousness of an arbitrary 2DFA is given and, moreover, the problem is shown to be PSPACE-complete.

1. Introduction

Finite automata are used in several applications and implementations in software engineering, programming languages and other practical areas in computer science. They are one of the first and most intensely investigated computational models. Since deterministic or nondeterministic as well as one-way or two-way finite automata are all well known to capture the regular languages, it is natural to investigate their descriptional complexity [10], that is, the succinctness of the representation of a regular language by the different types of automata in order to optimize the space requirements. For example, it is well known that nondeterministic one-way finite automata (1NFAs) can offer exponential savings in space compared with deterministic one-way finite automata (1DFAs). More precisely, given some n-state 1NFA one can always construct a language-equivalent 1DFA with at most 2^n states [31]. Later it was shown independently by several authors that this exponential upper bound is the best possible, that is, for every n there is an n-state 1NFA which cannot be simulated by any 1DFA with strictly less than 2^n states [23,25,27]. The state complexity of the simulation of two-way by one-way finite automata has been solved by establishing a tight bound of $\binom{2n}{n+1}$ for the simulation of two-way deterministic as well as nondeterministic finite automata by 1NFAs [15]. In the same paper tight bounds of $n(n^n-(n-1)^n)$ and $\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\binom{n}{i}\binom{n}{j}(2^i-1)^j$ are shown for two-way deterministic and two-way nondeterministic finite automata simulations by 1DFAs. The bounds reveal that two-way head motion is a very powerful resource with respect to the number of states. Interestingly, when simulating

☆ Supported by CRUI/DAAD under the project "Programma Vigoni: Descriptional Complexity of Non-Classical Computational Models," project codes CRUI E65E06000080001/DAAD 50774743.
☆☆ A preliminary version of this work was presented at LATIN 2012: Theoretical Informatics — 10th Latin American Symposium, Arequipa, Peru, April 16–20, 2012.
* Corresponding author.
E-mail addresses: kutrib@informatik.uni-giessen.de (M. Kutrib), malcher@informatik.uni-giessen.de (A. Malcher), pighizzini@di.unimi.it (G. Pighizzini).

http://dx.doi.org/10.1016/j.ic.2014.04.001
0890-5401/

two-way devices by 1NFAs, it does not matter whether the two-way device is nondeterministic or not. From this point of view, two-way head motion can compensate for nondeterminism.

Nevertheless, challenging problems are still open. The question of the costs for trading two-way head motion for nondeterminism, that is, the costs for simulating (two-way) NFAs by 2DFAs is unanswered for decades. It was raised by Sakoda and Sipser [32]. While from standard simulations we can obtain an exponential upper bound, the best lower bound currently known is $\Omega(n^2)$. This result was proved by Chrobak [4], evaluating the costs of *unary* automata simulations, as also suggested by Sipser [36]. It turned out that the unary case is essentially different from the general one. The costs of the unary 2DFAs and 1NFA to 1DFA simulation both reduce to the bound $e^{\Theta(\sqrt{n \cdot \ln n})}$ [4]. This result was generalized by sophisticated studies showing that the cost of unary 2NFA to 1DFA simulation is the same [24]. Furthermore, again by Chrobak [4], the Sakoda–Sipser problem for 1NFAs has been solved for the unary case. The tight bound in the order of magnitude is $\Theta(n^2)$. However the tight bound for the simulation of 2NFAs by 2DFAs is still open, even in the unary case. The best known upper bound is of order $2^{O(\ln^2 n)}$ [6]. A tightness proof of such a bound (or any other superpolynomial lower bound) should imply the separation between the classes L and NL, thus solving another long-standing open problem [7]. Other interesting connections between the problem of Sakoda and Sipser and the question whether L equals NL have been proved by Berman and Lingas [1] and, recently, by Kapoutsis [16,18].

In order to attack and to solve the problem of Sakoda and Sipser at least for subclasses, sweeping automata, that is, 2DFAs that may halt and change their head direction only at the endmarkers, are investigated. Sipser [36] found a language over an alphabet of size 2^{n^2}, that is accepted by an n-state 1NFA but any equivalent sweeping 2DFA needs at least 2^n states. The result has been improved to a binary alphabet [21]. Using again a growing alphabet it has been shown that there is an $O(n)$-state 2DFA such that any equivalent sweeping 2DFA has at least 2^n states [2,26,34].

Each sweeping automaton can be converted into an equivalent one that makes no stationary moves and performs a constant number of sweeps over the input only. The cost for this conversion is a quadratic number of states. However, such a normalized sweeping automaton has the same trajectory of the head movement on all inputs of the same length, it is said to be data-independent or oblivious. Relaxing the sweeping condition and restricting a general 2DFA to be oblivious yields a new type of two-way finite automata that is a generalization of normalized sweeping automata and a restriction of general 2DFAs. The notion of oblivious Turing programs was introduced in [28]. Later tis concept was used to simulate Turing machines by logical networks [30,33]. Moreover, obliviousness was studied in the context of parallel random access machines (PRAMs), in order to obtain characterizations of polynomial size, poly-logarithmic depth unbounded and bounded fan-in circuit classes by variants of PRAMs with oblivious or non-oblivious read- and write-structures [20]. Petersen [29] showed that obliviousness is no restriction for multi-head one-counter automata, multi-head non-erasing stack automata, and multi-head stack automata. The same is obviously true for one-head finite automata. Oblivious multi-head finite automata are studied as well [9,11].

The known results on the simulation of 1NFAs by sweeping 2DFAs have been extended to oblivious 2DFAs as follows [12]. It is shown that there is an $O(n)$-state 1NFA such that every equivalent oblivious 2DFA has at least 2^n states. Relaxing the obliviousness condition in such a way that a sublinear number of different trajectories is allowed, the result is that there exists an $O(n)$-state 1NFA such that any equivalent sublinearly oblivious 2DFA still needs at least $2^{\Omega(n)}$ states. Again, these results are for a growing alphabet of size 2^{n^2}.

Here we further investigate oblivious two-way deterministic finite automata. Our main interest is the decidability and descriptional power of obliviousness, that is, the question to what extent oblivious machines can represent regular languages more succinctly than sweeping machines or one-way automata and, vice versa, how many states are necessary for oblivious machines to simulate general deterministic two-way finite automata. In particular, we consider the simulation of 2DFAs by oblivious 2DFAs, the simulation of oblivious 2DFAs by sweeping 2DFAs and 1NFAs as well as the simulation of sweeping 2DFAs by 1NFAs. In all cases exponential lower bounds on the number of states are obtained for languages over an alphabet with at most four letters. Finally, it is shown that obliviousness is decidable for 2DFAs and the problem is shown to be PSPACE-complete.

2. Oblivious two-way finite automata

We denote the set of non-negative integers by $\mathbb{N}$. We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ. The length of a word w is denoted by $|w|$.

A two-way finite automaton is a finite automaton having a single read-only input tape whose inscription is the input word in between two endmarkers. The head of the automaton can move freely on the tape but not beyond the endmarkers. More precisely, a *deterministic two-way finite automaton* (2DFA) is a system $M = \langle Q, \Sigma, \delta, \triangleright, \triangleleft, q_0, F\rangle$, where Q is the finite set of *internal states*, Σ is the set of *input symbols*, $\triangleright \notin \Sigma$ and $\triangleleft \notin \Sigma$ are the *left and right endmarkers*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states*, δ is the transition function mapping $Q \times (\Sigma \cup \{\triangleright, \triangleleft\})$ to $Q \times \{-1, 0, 1\}$, where 1 means to move the head one square to the right, -1 means to move it one square to the left, and 0 means to keep the head on the current square. Whenever $(q', d) = \delta(q, a)$ is defined, then $d \in \{0, 1\}$ if $a = \triangleright$, and $d \in \{-1, 0\}$ if $a = \triangleleft$.

A 2DFA M starts with its head on the left endmarker. It halts when the transition function is not defined for the current situation. A computation can also enter an infinite loop. However, the *input is accepted* if and only if M halts in an accepting state.