Information and Computation 209 (2011) 528–535

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Optimal simulation of self-verifying automata by deterministic automata

Galina Jirásková [a,1], Giovanni Pighizzini [b,*,2]

[a] Mathematical Institute, Slovak Academy of Sciences, Grešákova 6, 040 01, Košice, Slovakia
[b] Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, via Comelico 39, I-20135 Milano, Italy

ARTICLE INFO

Article history:
Available online 18 November 2010

Keywords:
Finite automata
Nondeterminism
Self-verifying automata
Descriptional complexity

ABSTRACT

Self-verifying automata are a special variant of finite automata with a symmetric kind of nondeterminism. We study the conversion of self-verifying automata to deterministic automata from a descriptional complexity point of view. The main result is the exact cost, in terms of the number of states, of such a simulation.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

In automata theory and theoretical computer science, several kinds of devices able to recognize formal languages have been proposed and investigated. Different classes of devices can be compared, first of all, from the point of view of their recognition powers. We mention just two examples of classical results in this area: the equivalence between deterministic and nondeterministic finite automata, and the fact that deterministic pushdown automata are strictly less powerful than nondeterministic ones. With a deeper investigation, classes of devices can be compared from the point of view of their descriptional complexity [1]. The classical example is the simulation of n-state nondeterministic automata by deterministic automata that can be done using 2^n states [2], and cannot be done, in the worst case, with less than 2^n states [3–5].

In this paper, we continue this line of research by considering *self-verifying automata*, a special kind of finite automata, introduced in [6], with a symmetric form of nondeterminism called *self-verifying nondeterminism* [7]. This kind of nondeterminism was mainly considered in connection with randomized Las Vegas computations, but as pointed out in [8], it is interesting also *per se*.

Roughly speaking, in self-verifying nondeterminism, computation paths can give three types of answers: *yes*, *no*, and *I do not know*. On each input string, at least one path must give answer "yes" or "no". Furthermore, on the same string, two paths cannot give contradictory answers, namely both the answers "yes" and "no" are not possible.

Hence, the existence of a computation path ending in an accepting state (an answer "yes") definitively proves the membership of the string in the language. This is exactly the same as for nondeterministic automata. Furthermore, in self-verifying automata the existence of a computation path ending in a rejecting state (an answer "no") definitively proves that the string does not belong to the language. This is in contrast with nondeterministic automata, where the existence of a rejecting path leaves open the possibility that the input could be accepted by a different path. Thus, the main feature of self-verifying automata is that, even if the transitions are nondeterministic, when a computation accepts or rejects, the answer is definitively correct, that is, the automaton "can trust" the outcome of that computation. The name "self-verifying" derives from this property.

* Corresponding author.
E-mail addresses: jiraskov@saske.sk (G. Jirásková), pighizzini@dico.unimi.it (G. Pighizzini).

[1] Supported by VEGA grant 2/0111/09.
[2] Partially supported by MIUR under the project PRIN "Aspetti matematici e applicazioni emergenti degli automi e dei linguaggi formali: metodi probabilistici e combinatori in ambito di linguaggi formali".

0890-5401/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2010.11.017

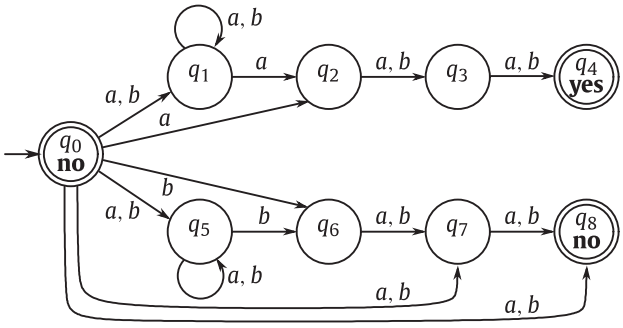

Fig. 1. A self-verifying automaton for the language $\{uav \mid u, v \in \{a, b\}^* \text{ and } |v| = 2\}$.

Self-verifying automata are as powerful as deterministic automata; in particular, the standard subset construction can be used to convert them to deterministic automata. Hence the question arises of investigating this equivalence from the descriptional point of view. This problem was previously considered by Assent and Seibert in [9], who proved that in the deterministic automaton obtained by applying the standard subset construction to a self-verifying automaton certain states must be equivalent. As a consequence, they were able to show that each n-state self-verifying automaton can be simulated by a deterministic automaton with $O(2^n/\sqrt{n})$ states.

Here we further deepen this investigation by showing that such an upper bound can be lowered to a function $g(n)$ which grows like $3^{n/3}$. We give the *exact* value of $g(n)$ in the paper. In particular, we associate with each n-state self-verifying automaton a certain graph with n vertices, and we prove that there exists a deterministic automaton equivalent to to the given self-verifying automaton whose state set is isomorphic to the set of the maximal cliques of such a graph. Using a result from graph theory stating the number of possible maximal cliques in a graph [10], we get the upper bound $g(n)$. In the second part of the paper, we prove the optimality of this upper bound. For every positive integer n, we describe a *binary* language accepted by an n-state self-verifying automaton such that the minimal equivalent deterministic automaton must have *exactly* $g(n)$ states.

We next allow self-verifying automata to have multiple initial states, and prove that the cost of the simulation of n-state self-verifying automata with multiple initial states by deterministic automata is essentially the same as in the case of self-verifying automata with only one initial state. This cost does not reduce, even if we restrict to self-verifying automata that use only deterministic transitions and make the only nondeterministic decision at the beginning of the computation, to choose the initial state. We conclude the paper by presenting some considerations concerning the case of automata defined over a one-letter alphabet.

2. Preliminaries

We fix an alphabet Σ. Given a language $L \subseteq \Sigma^*$, we denote by L^c the complement of L, namely the set $\Sigma^* - L$. We assume that the reader is familiar with the notions of deterministic and nondeterministic finite automata. For short, we denote them as dfa's and nfa's, respectively.

Definition 1. A *self-verifying finite automaton* (svfa) is a 6-tuple $A = (Q, \Sigma, \delta, q_0, F^a, F^r)$, where Q, Σ, δ, q_0 are defined as for standard nondeterministic automata, and F^a, $F^r \subseteq Q$ are the sets of *accepting* and *rejecting* states, respectively. The remaining states, namely the states belonging to $Q - (F^a \cup F^r)$, are called *neutral* states.

It is required that for each input string w in Σ^*, there exists at least one computation ending in an accepting or in a rejecting state, that is, $\delta(q_0, w) \cap (F^a \cup F^r) \neq \emptyset$, and there are no strings w such that both $\delta(q_0, w) \cap F^a$ and $\delta(q_0, w) \cap F^r$ are nonempty.

The language accepted by A, denoted as $L^a(A)$, is the set of all input strings having a computation ending in an accepting state, while the language rejected by A, denoted as $L^r(A)$, is the set of all input strings having a computation ending in a rejecting state.

It follows directly from the definition that $L^a(A) = (L^r(A))^c$ for each svfa A. Hence, when we say that an svfa A *recognizes* a language L, we mean that $L = L^a(A)$ and $L^c = L^r(A)$.

Example 1. Consider the automaton of Fig. 1. Accepting and rejecting states are marked with double circles with "yes" or "no". Hence, $F^a = \{q_4\}$ and $F^r = \{q_0, q_8\}$. The remaining states are neutral. We now show that the automaton is self-verifying.

First of all, the empty string is rejected. Now consider a string w in $\{a, b\}^*$, with $1 \leqslant |w| \leqslant 2$. There exists one computation on w ending in the rejecting state q_8. All the other possible computations on w end in some neutral state. Hence, each string of length 1 or 2 is rejected. Finally, we consider strings of length at least 3, namely strings of the form $w = u\sigma v$, where

Download English Version:

https://daneshyari.com/en/article/10330785

Download Persian Version:

https://daneshyari.com/article/10330785

Daneshyari.com