Expert Systems With Applications 42 (2015) 9512–9527

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Forward and backward synchronizing algorithms

Adam Roman [a], Marek Szykuła [b,*]

[a] Institute of Computer Science, Jagiellonian University, Cracow, Poland
[b] Institute of Computer Science, University of Wrocław, Joliot-Curie 15, Wrocław PL-50-383, Poland

ARTICLE INFO

Keywords:
Synchronizing automaton
Synchronizing algorithm
Reset word
Reset length
Reset sequence
Sequential circuit

ABSTRACT

Automata synchronization has many important applications, mostly in conformance testing of electrical circuits, self-correcting codes and protocol testing. Finding a shortest synchronizing word cannot be done in polynomial time, assuming P ≠ NP. In some situations, especially for very large automata, finding such a word is almost impossible. Therefore, we accept any synchronizing word that is reasonably short and can be calculated in short time. The existing algorithms are either polynomial (quick, but not optimal) or exponential (exact, but useless in case of large automata). In this paper we present a flexible algorithmic framework for synchronization. It allows the user to parameterize the algorithm to obtain a desired balance in terms of a trade-off between memory usage, runtime and optimality. We also discuss many practical issues that affect efficiency of an implementation.

In particular, we design a new polynomial backward algorithm, which works significantly better than previously used heuristic algorithms. Finally, we present detailed results of experiments involving automata up to 2000 states, which compare our algorithms in various settings and the other known algorithms, and check the impact of different parameters on the results.

1. Introduction

A finite automaton is called synchronizing if there exists a word, called a synchronizing sequence or synchronizing word, that takes the automaton to a unique final state, regardless of the initial one. This property is not only a theoretical concept – it has an important application in conformance testing, mainly for embedded controllers, sequential circuits and communication protocols (Holzmann, 1991; Lai, 2002; Lee & Yannakakis, 1994; 1996; Sidhu & Leund, 1989). Synchronizing words have been used to generate test cases for synchronous circuits with no reset (Cho, Jeong, Somenzi, & Pixley, 1993; Pixley, Jeong, & Hachtel, 1992; Rho, F., & Pixley, 1993) (a reset here is a trivial synchronizing word of length 1).

In conformance testing one wants to check if a SUT – system under test (protocol, circuit etc.) – conforms to a given specification. In case of reactive systems the underlying model is usually a Mealy automaton – a finite state machine that generates output. Most of the conformance testing algorithms combine techniques for investigating particular states or transitions of a finite state machine. The techniques, among others, are state identification, state validation, homing and synchronizing sequences. Synchronizing word allows us to gain the control over the machine by identifying the (synchronized) state which we should currently be in. If a SUT does not conform to the model, we can easily verify that by analyzing an output for a so-called distinguishing sequence from the synchronizing state (Krichen, 2005).

From the practical point of view, the shorter the synchronizing word is, the better. Unfortunately, the problem of finding the *shortest* such sequence is $FP^{NP[log]}$-hard, and the related decision problem is both NP- and co-NP-hard (Olschewski & Ummels, 2010). Therefore we have to use either polynomial heuristic algorithms or exact ones, but with exponential runtime. Moreover, in case of sequential circuits the state space is usually very big, exponential in the number of latches, which complicates the problem. Rho et al. (1993) noticed that in the circuits with relatively long synchronizing sequences conformance testing is more difficult and, therefore, the benefit of using an explicit synchronizing word is more prominent in this case. The authors suggested an algorithm based on binary decision diagrams, which works better than the straightforward exponential one, but still for some hard examples there is a big gap between the computed word and the shortest synchronizing sequences. The fastest currently known algorithm for this problem (Kisielewicz, Kowalski, & Szykuła, 2013, 2015), which finds a *minimal* synchronizing word, works quite effectively for real-world example automata, and can deal with automata up to a few hundred of states. However, for larger examples, we are left with polynomial non-optimal solutions. For other results concerning development and improvement of synchronizing algorithms, both

* Corresponding author.
E-mail addresses: roman@ii.uj.edu.pl (A. Roman), msz@cs.uni.wroc.pl (M. Szykuła).

http://dx.doi.org/10.1016/j.eswa.2015.07.071
0957-4174/

exact and polynomial, see e.g. Eppstein (1990) Gerbush and Heeringa (2011); Güniçen, Erdem, and Yenigün (2013); Kudłacik, Roman, and Wagner (2012); Roman (2009); Skvortsov and Tipikin (2011); Trahtman (2003 2006).

In this paper we introduce a framework for building flexible synchronizing algorithms. Flexibility should be understood here as the ability to parametrize the algorithm to obtain a desired balance in terms of a trade-off between memory usage, runtime and synchronizing word length. We present two generic algorithms. One works in a top-down manner, similar to the exponential one. The second one is quite opposite, as it uses the bottom-up approach, starting from singletons and trying to reach the whole set of states of the input automaton. Both algorithms can be tuned in order to obtain the balance mentioned above. Also, they can be combined together, which in most cases is more effective than each of them separately. The combined version is able to handle very large automata, with several hundreds of states, making it very practical for real applications. According to our experiments, in average case it also finds shorter reset words than other heuristic polynomial algorithms used so far.

The main reason for designing our algorithm is as follows. There is a plethora of problems related to synchronization. Some of them require to find a shortest synchronizing word. In others we are satisfied when we find quickly a reasonably short word. In some situations we may require the word as shortest as possible, but because of some constraints (like: large number of states, not enough memory, short time) we need to make some trade-off. The flexible algorithm fits perfectly in all these situations. By choosing appropriate parameters we can knowingly balance between all existing constraints. In this sense our approach is universal for all types of synchronization problems.

We can mention that, according to the recent result of Gawrychowski and Straszak (2015), unless P = NP, it is not possible to approximate the length of the shortest reset word within a factor of $n^{1-\varepsilon}$, for any $\varepsilon > 0$ (cf. also Berlinkov, 2014a; 2014b; Gerbush and Heeringa, 2011 for hardness of approximation). On the other hand, all known synchronizing algorithms that rely on top-down manner (e.g. Eppstein (1990)) have $n-1$ approximation factor. Gerbush and Heeringa (2011) have generalized the Eppstein algorithm and have provided a polynomial algorithm with a slightly better, though still linear, approximation factor $\lceil \frac{n-1}{k-1} \rceil$, for a fixed k. However, its space complexity is $O(n^k)$, which makes it impractical even for very small values of k.

The paper is constructed as follows. In Section 2 we introduce the terminology that will be used throughout the paper. In Section 3 we describe a generic approach to finding synchronizing sequences. Then, in Sections 4 and 5, we describe and analyze our new algorithms. We also discuss various weight functions for the algorithms, that are heuristic evaluations directing the search; these are presented in Section 6. In Section 7 we describe the experimental results, including the comparison with other approaches for finding short synchronizing words for some benchmark data related to sequential circuits testing.

2. Basic notions

A finite automaton $\mathcal{A}$ is a triple $\langle Q, \Sigma, \delta \rangle$, where Q is a finite set of states, Σ is a finite alphabet and $\delta\colon Q \times \Sigma \to Q$ is a transition function. By Σ^* we denote a free monoid over Σ, that is a set of all finite words over Σ. Any element $w \in \Sigma^*$ is called a word and its length $|w|$ is the number of its letters. An empty word ε (which is a neutral element of Σ^*) has length 0. Transition function δ can be extended in a natural way to all words and subsets of states: for any $P \subseteq Q$, $a \in \Sigma$, $w \in \Sigma^*$ we define $\delta(P, aw) = \bigcup_{p \in P} \delta(\delta(p, a), w)$ and $\delta(P, \varepsilon) = P$. Sometimes, instead of writing $\delta(P, a)$ we will just write Pa. By Pa^{-1} we denote the preimage of the set P under the letter a, that is $Pa^{-1} = \{q \in Q : \delta(p, a) \in P\}$.

Let $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ be an automaton. A word $w \in \Sigma^*$ is *synchronizing*, or *reset*, for $\mathcal{A}$ if it takes all states to one state, that is, if $|\delta(Q, w)| = 1$. Equivalently, we can say that w is synchronizing for $\mathcal{A}$ if for any two states $p, q \in Q$ we have $\delta(p, w) = \delta(q, w)$. If there exists a synchronizing word w for $\mathcal{A}$, we say that $\mathcal{A}$ is synchronizing and that $s = \delta(Q, w)$ is a synchronizing state for w. Not all automata are synchronizing, for example automata whose underlying graph is not connected or automata in which all letters act like permutations on Q.

It is easy to observe that if w is a synchronizing word for $\mathcal{A}$ and $\delta(Q, w) = s$, then so is any word in the form vwu, $v, u \in \Sigma^*$, because $|\delta(Q, vwu)| = |\delta(P, wu)| = |\delta(s, u)| = 1$ for some $P = \delta(Q, v)$. So the natural question is: what is the shortest possible synchronizing word for a given automaton? We will call it a minimal synchronizing word and we will shortly call its length by *reset length* (sometimes called *reset threshold*). The famous Černý conjecture (Černý, 1964) states that the minimal synchronizing word for any n-state automaton has length $\leq (n-1)^2$. This conjectured bound is tight, as for any n there exists an automaton, called Černý automaton $\mathcal{C}_n$, with the reset length equal to $(n-1)^2$. It is defined on states $Q = \{0, 1, \ldots, n-1\}$ over binary alphabet $\Sigma = \{a, b\}$ with transition function defined as follows:

$$\delta(p, x) = \begin{cases} p + 1 \pmod n & \text{for } x = a, \\ p & \text{for } x = b \wedge p < n-1, \\ 0 & \text{for } x = b \wedge p = n-1. \end{cases}$$

The best currently known upper bound is $(n^3 - n)/6$. It was proved independently by Pin (1983) and Klyachko, Rystsov, and Spivak (1987).

A synchronizing algorithm takes as an input a finite automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ and returns its synchronizing word or says that $\mathcal{A}$ is not synchronizing.

There are two auxiliary constructions frequently used when designing synchronizing algorithms. First one is so-called *power automaton*. For a given $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ its power automaton is $\mathcal{P}(\mathcal{A}) = \langle 2^Q \setminus \{\emptyset\}, \Sigma, \delta^* \rangle$, where 2^Q are all possible subsets of Q and δ^* is just δ extended to subsets of the states: $\delta^*(p, a) = \bigcup_{q \in p} \delta(q, a)$. It is easy to see that the shortest path going from state Q to some singleton state corresponds to a minimal synchronizing word. The power automaton is a basis for the exact, exponential synchronizing algorithms. The second construction is so-called *pair automaton*, which is a subset of $\mathcal{P}(\mathcal{A})$, consisting only of sets of size less or equal 2. It is utilized in the Eppstein-type algorithms described below.

3. General framework

First, let us describe some existing forward-type algorithms and show how to fit them into a single generic model that uses the concepts of weight and list length, which will also be utilized in the backward algorithm described later in the next section. The parametrization of the generic algorithm will allow us to construct many different versions/extensions of the existing algorithms and therefore being flexible in terms of the algorithm precision and resource (time and memory) utilization.

The simplest and most naive is the exponential algorithm that does BFS search on the whole power automaton. It always finds the shortest synchronizing word, but its complexity is exponential in the number of states. Natarajan (1986) and Eppstein (1990) algorithms work in the polynomial time. The idea behind Natarajan's algorithm is simple: start with the whole set $P = Q$ of states and in each step pick arbitrary two states $p, q \in Q$ and find a word $\tau_{p,q} \in \Sigma^*$ such that $\delta(p, \tau_{p,q}) = \delta(q, \tau_{p,q})$. Next, put $P := \delta(P, \tau_{p,q})$ and repeat the same procedure until the singleton state is reached. The synchronizing word is the concatenation of all words found during these steps. This algorithm works in $O(n^4 + kn^3)$ time complexity.

Eppstein algorithm is in fact an improved version of Natarajan's one. It does some preprocessing at the beginning, which al-