Information and Computation 260 (2018) 1–8

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Accepting runs in a two-way finite automaton

Oscar H. Ibarra [a,*], Zhe Dang [b,c], Qin Li [b]

[a] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
[b] School of Computer, Anhui University of Technology, Ma'anshan, Anhui, China
[c] School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164, USA

ARTICLE INFO

Article history:
Received 9 July 2017
Received in revised form 5 February 2018
Available online 21 March 2018

Keywords:
Two-way automata
Accepting run
Sampling

ABSTRACT

An accepting run in a two-way finite automaton M is a sequence of states that M enters during some accepting computation. The set of all such runs is denoted by $L_{\mathrm{run},M}$. We study the complexity of $L_{\mathrm{run},M}$ when M is a 2NFA (2DFA). We also look at the complexity of "position sampling" (the sequence of states that M enters in specified positions of some accepted input) in a 2NFA. In particular, we give some language properties of sampled runs of 2NFAs augmented with restricted unbounded storage.

1. Introduction

One way to understand the behavior of a software system is through observation. That is, when the system runs, we record a sequence of values of all or part of its state variables like the PC (program counter), variable values, pointer locations, stack frames, I/O, etc. Such a sequence, called a trace, can later be used for either off-line or online analysis. A trace contains valuable information such as information flow among the state variables. An automaton (e.g., a two-way finite automaton) can model/specify the execution of a program, where the PC values during the run of the program would correspond to the states of the automaton [2,3,11].

If M is a 2NFA (2DFA), let $L_{\mathrm{run},M} = \{\alpha \mid \alpha$ is a sequence of states that M enters during some accepting computation on some input$\}$. In this paper, we study the complexity of $L_{\mathrm{run},M}$ when M is a 2NFA (2DFA). We also look at the complexity of "position sampling" (the sequence of states that M enters in specified positions of some accepted input) in a 2NFA. In particular, we give some language properties of sampled runs of 2NFAs augmented with restricted unbounded storage.

We will use the following notations for language acceptors:

- DFA (NFA) = one-way deterministic (nondeterministic) finite automaton. DFAs and NFAs are equivalent, and they accept the same class of languages.
- 2DFA(2NFA) = two-way deterministic (nondeterministic) finite automaton with left and right input end-markers.
- finite-crossing 2DFA (2NFA) = 2DFA (2NFA) which crosses the boundary between any two adjacent cells of the input tape at most k times for some fixed $k \geq 1$.
- DPDA (NPDA) = one-way deterministic (nondeterministic) pushdown automaton, i.e., a DFA (NFA) augmented with a pushdown stack. NPDAs accept exactly the context-free languages (CFLs).

* Corresponding author.
E-mail addresses: ibarra@cs.ucsb.edu (O.H. Ibarra), zdang@eecs.wsu.edu (Z. Dang), linuxos2@163.com (Q. Li).

https://doi.org/10.1016/j.ic.2018.03.002
0890-5401/© 2018 Published by Elsevier Inc.

A *counter* is an integer variable that can be incremented by 1, decremented by 1, left unchanged, and tested for zero. It starts at zero and cannot store negative values. Thus, a counter is a pushdown stack on a unary alphabet, in addition to the bottom of the stack symbol which is never altered.

An automaton (DFA, NFA, 2DFA, 2NFA, DPDA, NPDA, etc.) can be augmented with a finite number of counters, where the “move” of the machine also now depends on the status (zero or non-zero) of the counters, and the move can update the counters. It is well known that a DFA augmented with two counters is equivalent to a deterministic Turing machine [12].

In this paper, we will restrict the augmented counter(s) to be reversal-bounded in the sense that each counter can only reverse (i.e., change mode from non-decreasing to non-increasing and vice-versa) at most r times for some given r. In particular, when $r = 1$, the counter reverses only once, i.e., once it decrements, it can no longer increment. Note that a counter that makes r reversals can be simulated by $\lceil \frac{r+1}{2} \rceil$ 1-reversal counters [8]. Closure and decidable properties of various machines augmented with reversal-bounded counters have been studied in the literature (see, e.g., [8,9]). We will use the notation DFCM, NFCM, 2DFCM, 2NFCM, DPCM, NPCM, etc., to denote a DFA, NFA, 2DFA, 2NFA, DPDA, NPDA, etc., augmented with reversal-bounded counters.

Automata with reversal-bounded counters can “count”, as seen in the following example.

Example 1.1. $L_k = \{x_1\# \cdots \#x_k\# \mid x_i \in (a+b)^+, x_i \neq x_j \text{ for } i \neq j\}$ can be accepted by an NFCM M_k with $k(k-1)/2$ 1-reversal counters.

For $1 \leq i < j \leq k$, M_k nondeterministically guesses that x_i and x_j are of

(i) different lengths, or (ii) disagree in at least one position.

To accomplish (i), M_k reads x_i and stores $|x_i|$ in counter C_i and then decrements the counter while reading x_j. Then $|x_i| \neq |x_j|$ if and only if C_i becomes zero before all of x_j is read or is positive after all of x_j is read. To accomplish (ii), M_k stores in counter C_i a “guessed” position p_i of x_i and records in the state the symbol a_{p_i} in that location. Then later, when it is scanning x_j, M_k, decrements C_i. When C_i becomes zero, M_k checks that the symbol under the head (on x_j) is not the same as a_{p_i}. Clearly, M_k uses $k(k-1)/2$ 1-reversal counters.

Let M be a 2NFA with input alphabet Σ, which is equipped with a two-way input tape (with left end-marker $\triangleright \notin \Sigma$ and right end-marker $\triangleleft \notin \Sigma$). Suppose that an input word, say $w \in \Sigma^*$, is given on the input tape (so the tape content is actually $\triangleright w \triangleleft$). The read head in M reads the input while performing a state transition drawn from a finite set T, which is called the transition table of M, or simply, the transitions of M. More precisely, a state transition is in the form of

$$(s, a, s', d)$$

where s, s' are states (there are only finitely many distinct states), $a \in \Sigma \cup \{\triangleright, \triangleleft\}$ is an input symbol or an end-marker, and $d \in \{+1, -1, 0\}$ is a direction (i.e., $+1$, -1, and 0 are respectively for moving to the right, moving to the left, and staying). When $d = 0$, the state transition is called a stationary move. For instance, the transition $(s, a, s', -1)$ means that, when M is at state s while symbol a is under the read head, the head moves to the right and the state is changed to s'. Sometimes, we explicitly indicate the direction for readability; e.g., (s, a, s', left). M is a 2DFA (i.e., deterministic) if for each pair (s, a) there is at most one (s', d) such that $(s, a, s', d) \in T$. In this case, we write $(s, a) \to (s', d)$ for the transition (s, a, s', d). M starts from its initial state and the read head is under the left end-marker. M then performs a sequence of state transitions, for some n,

$$(s_0, a_0, s_1, d_0)(s_1, a_1, s_2, d_1) \cdots (s_n, a_n, s_{n+1}, d_n)$$

while moving the two-way head on the w, where s_0 is the initial state, the first transition (s_0, a_0, s_1, d_0) reads the left end-marker and moves to the right. We note that the symbols $a_0, a_1, \cdots, a_n$ are actually the symbols under the read head when M executes the sequence of transitions on input $\triangleright w \triangleleft$.

M accepts w when M enters an accepting state (i.e., s_{n+1} is an accepting state in the above transition sequence). The state sequence $s_0 s_1 \cdots s_{n+1}$ in the state transition sequence witnessing the acceptance is called an accepting run. Since M is nondeterministic, there could be more than one accepting run for the given w. We use $L(M)$ to denote the set of all words w accepted by M and use $L_{\text{run},M} = \{\alpha \mid \alpha \text{ is an accepting run of } M \text{ on } w,\ w \in L(M)\}$ to denote the set of all accepting runs of M. Clearly, $L_{\text{run},M}$ is not necessarily a regular language on alphabet S (the states in M). In fact, stronger results will be shown in the next section.

2. Accepting runs in 2DFAs

Let k be a number. When a 2DFA (resp. 2NFA) makes at most k turns on all of its accepting runs, we call it a k-turn 2DFA (resp. k-turn 2NFA). In particular, when a k-turn 2DFA (resp. k-turn 2NFA) has no stationary moves, we call it a *non-stationary* k-turn 2DFA (resp. *non-stationary* k-turn 2NFA).

A finite-crossing 2NFCM (2DFCM) is a finite-crossing 2NFA (2DFA) augmented with reversal-bounded counters. The following was shown in [6]: