RAIRO-Inf. Theor. Appl. **41** (2007) 137–145
DOI: 10.1051/ita:2007014

RETURNING AND NON-RETURNING PARALLEL COMMUNICATING FINITE AUTOMATA ARE EQUIVALENT

Ashish Choudhary[1], Kamala Krithivasan[2] and Victor Mitrana[3, 4]

Abstract. A parallel communicating automata system consists of several automata working independently in parallel and communicating with each other by request with the aim of recognizing a word. Rather surprisingly, returning parallel communicating finite automata systems are equivalent to the non-returning variants. We show this result by proving the equivalence of both with multihead finite automata. Some open problems are finally formulated.

Mathematics Subject Classification. 68Q45, 68Q68.

1. Introduction

The idea of considering several automata which work independently in parallel and cooperate with each other following different strategies with the aim of recognizing a word, has been considered in many papers. We mention here some of them: *multihead automaton* [6, 7], *multiprocessor automata* [1], *parallel communicating automata systems* [3,8] and *cooperating multi-stack pushdown automata* [4].

Keywords and phrases. Formal languages, parallel communicating finite automata system, multihead finite automaton, computational power.

[1] Dept of Computer Science and Engineering, Indian Institute of Technology, Madras, Chennai, 600036 India; ashishc@cse.iitm.ernet.in

[2] Dept of Computer Science and Engineering, Indian Institute of Technology, Madras, Chennai, 600036 India; kamala@iitm.ernet.in

[3] Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, 010014, Bucharest, Romania.

[4] Research Group in Mathematical Linguistics, Rovira i Virgili University, Pl. Imperial Tarraco 1, 43005, Tarragona, Spain; vmi@urv.net

Grammar systems introduced as grammatical approaches to some models in the problem solving theory are based on the same idea, see [2, 5] and their references.

We continue in this note the investigation of parallel communicating finite automata systems introduced in [8] by studying the relationships between returning and non-returning variants. In [8], it is already proved that the families of languages accepted by multihead finite automata and parallel communicating automata systems (*pcfa* for short) are equal. Furthermore, the equality is preserved with respect to the number of heads and components, respectively. Here we show a rather unexpected result: for every multihead finite automaton, we can construct an equivalent returning parallel communicating finite automata system (*rpcfa* for short) which accepts the same language as that one accepted by the given multihead finite automaton. Again, the relation is invariant with respect to the number of heads and components, respectively. By the above result, we conclude that *pcfa* and *rpcfa* are equivalent to each other.

The paper is organized as follows. The next section starts with the definitions of parallel communicating finite automata systems and their cooperation protocols. Then we recall some already existing results from [8] regarding parallel communicating automata systems and their relationships with multihead finite automata. Then we give our main result which proves the equivalence of *rpcfa* and multihead finite automata. Finally we discuss some open problems.

2. Parallel communicating finite automata systems

We assume the reader to be familiar with the fundamental concepts of formal language theory and automata theory, particularly the notions of grammars and finite automata [9].

An alphabet is always a finite set of letters. The set of all words over an alphabet V is denoted by V^*. The empty word is written as ε; moreover, $V^+ = V^* - \{\varepsilon\}$. For a finite set A we denote by $card(A)$ the cardinality of A.

A parallel communicating finite automata system (PCFA) of degree n is a construct

$$\mathcal{A} = (V, A_1, A_2, \ldots, A_n, K)$$

where:

- V is the input alphabet.
- Each $A_i = (Q_i, V, f_i, q_i, F_i), 1 \leq i \leq n$, is a finite automaton with the set of states $Q_i, q_i \in Q_i$ (the initial state of the automaton A_i), $F_i \subseteq Q_i$ (the set of final states of the automaton A_i) and f_i is the transition mapping of the automaton A_i defined as follows:

  $$f_i : Q_i \times (V \cup \{\varepsilon\}) \to 2^{Q_i}.$$

  Note that Q_i are not necessarily disjoint sets.
- $K = \{K_1, K_2, \ldots, K_n\} \subseteq \cup_{i=1}^{n} Q_i$ is the set of query states.

The automata $A_1, A_2, \ldots, A_n$ are called the *components* of the system $\mathcal{A}$. If there exists just one $1 \leq i \leq n$ such that $K \subseteq Q_i$, then the system is said to be *centralized*, the master of this system being the component i. For the sake of simplicity, whenever a system is centralized, the first component is its master. If the following conditions

(i) $card(f_i(s,a)) \leq 1$ for all $s \in Q_i$ and $a \in V \cup \{\varepsilon\}$;
(ii) if $card(f_i(s,\varepsilon)) \neq 0$ for some $s \in Q_i$, then $card(f_i(s,a)) = 0$ for all $a \in V$,

are fulfilled for all $1 \leq i \leq n$, then the automata system is *deterministic*.

By a configuration of a parallel communicating automata system as above, we mean a $2n-$tuple

$$(s_1, x_1, s_2, x_2, \ldots, s_n, x_n)$$

where

– s_i is the current state of the component i;
– x_i is the remaining part of the input word which has not been read yet by the component $i, 1 \leq i \leq n$.

We define two binary relations on the set of all configurations of $\mathcal{A}$ in the following way:

(I) $$(s_1, x_1, s_2, x_2, \ldots, s_n, x_n) \vdash (p_1, y_1, p_2, y_2, \ldots, p_n, y_n)$$

iff one of the two conditions holds:

- $K \cap \{s_1, s_2, \ldots, s_n\} = \emptyset$ and $x_i = a_i y_i, a_i \in V \cup \{\varepsilon\}, p_i \in f_i(s_i, a_i)$, $1 \leq i \leq n$,
- for all $1 \leq i \leq n$ such that $s_i = K_{j_i}$ and $s_{j_i} \notin K$ put $p_i = s_{j_i}, p_r = s_r$, for all the other $1 \leq r \leq n$, and $y_t = x_t, 1 \leq t \leq n$.

(II) $$(s_1, x_1, s_2, x_2, \ldots, s_n, x_n) \vdash_r (p_1, y_1, p_2, y_2, \ldots, p_n, y_n)$$

iff one of the following two conditions holds:

- $K \cap \{s_1, s_2, \ldots, s_n\} = \emptyset$ and $x_i = a_i y_i, a_i \in V \cup \{\varepsilon\}, p_i \in f_i(s_i, a_i)$, $1 \leq i \leq n$,
- for all $1 \leq i \leq n$ such that $s_i = K_{j_i}$ and $s_{j_i} \notin K$ put $p_i = s_{j_i}, p_{j_i} = q_{j_i}$, as well as $p_r = s_r$, for all the other $1 \leq r \leq n$, and $y_t = x_t, 1 \leq t \leq n$.

The difference between the two relations defined above may be easily noticed when the current states of some components are query states: these components get into communication with those components identified by the query states, which are forced to send their current states, if they are not query states, these states becoming the new states of the receiver components. The next states of the sender components remain the same in the case of relation $\vdash$ whereas they become the initial states when the relation $\vdash_r$ has been applied. A parallel communicating automata system whose moves are all based on the relation $\vdash_r$ is said to be *returning*.

Informally, the languages accepted by a PCFA $\mathcal{A}$ (in the non-returning and in the returning way), consist of all strings $x \in V^*$ such that the system starts in an initial configuration $(q_1, x, q_2, x, \ldots, q_n, x)$ and reaches a final configuration,

that is a configuration of the form $(s_1, \varepsilon, s_2, \varepsilon, \ldots, s_n, \varepsilon)$, where $s_i \in F_i$. Formally

$$Rec(\mathcal{A}) = \{x \in V^* | (q_1, x, q_2, x, \ldots, q_n, x) \vdash^* (s_1, \varepsilon, s_2, \varepsilon, \ldots, s_n, \varepsilon), \\ s_i \in F_i, 1 \leq i \leq n\},$$
$$Rec_r(\mathcal{A}) = \{x \in V^* | (q_1, x, q_2, x, \ldots, q_n, x) \vdash_r^* (s_1, \varepsilon, s_2, \varepsilon, \ldots, s_n, \varepsilon), \\ s_i \in F_i, 1 \leq i \leq n\}.$$

We shall denote by:

– $rcpcfa(n)$ - the class of returning centralized parallel communicating finite automata systems of degree n;

– $rpcfa(n)$ - the class of returning parallel communicating finite automata systems of degree n;

– $cpcfa(n)$ - the class of centralized parallel communicating finite automata systems of degree n;

– $pcfa(n)$ - the class of parallel communicating finite automata systems of degree n.

We add the prefix d in order to denote deterministic variants. If $x(n)$ denotes a class of automata systems, then $X(n)$ is the class of all languages accepted by automata systems in the class x. For example, $RCPCFA(n)$ is the class of all languages accepted by $rcpcfa(n)$ automata systems.

3. Previous results

In this section we recall some of the already existing results related to the comparison of the computational power of parallel communicating finite automata and multihead automata from [8]. For technical reasons, we shall use here the following definition of multihead finite automaton. A (non-deterministic) $k-$head finite automaton is a quintuple

$$A = (k, Q, V, f, q_0, F),$$

where Q, V, q_0, F have the same meaning as for a usual finite automaton, and f is a mapping from $Q \times (V \cup \{\varepsilon\})^k$ into the subsets of Q. The above definition is essentially similar to that found in [6] and [7]. Thus, $q \in f(s, a_1, a_2, \ldots, a_k)$ indicates that the automaton in state s each head i reading a_i may enter state q. The input heads are idealized in the sense that they pass over one another freely and they are prevented from going off the right end of the input. Moreover, if a head reads ε, it does not move to right and if it reads a symbol in V, it moves to the right one square. Acceptance is defined as follows: a string is accepted if the automaton starts in the initial state with the string on the input tape, all heads being positioned on the leftmost symbol of the input, and enters, after finitely many moves, in a final state, the input being completely read by all heads. In all other cases the input string is rejected. For a multihead finite automaton A as above denote by $Rec(A)$ the set of all strings accepted by A.

We denote the class of (deterministic) n-head finite automata by $fa(n)$ ($dfa(n)$) and the corresponding family of languages by $FA(n)$ ($DFA(n)$). The following two theorems are taken from [8].

Theorem 1. *Let $n \geq 1$.*
1. *$X(n) \subseteq FA(n)$ for all $X \in \{RCPCFA, RPCFA, CPCFA, PCFA\}$.*
2. *$X(n) \subseteq DFA(n)$ for all $X \in \{DRCPCFA, DRPCFA, DCPCFA, DPCFA\}$.*

Theorem 2. *Let $n \geq 1$.*
1. *$PCFA(n) = FA(n)$.*
2. *$DPCFA(n) = DFA(n)$.*

4. Equivalence of returning parallel communicating finite automata systems and multihead finite automata

In this section we prove the main result of our note. We show that any n-head automaton can be simulated by an automata system in $rpcfa(n)$.

Theorem 3. *$FA(n) \subseteq RPCFA(n)$ for all $n \geq 1$.*

Proof. Let $A = (n, Q, V, f, q_0, F)$ be an $n-$head finite automaton (obviously, we may restrict ourselves to the case $n \geq 2$). Construct the $rpcfa(n)$ $\mathcal{A} = (V, A_1, A_2, \ldots, A_n, K)$, with $A_i = (Q_i, V, f_i, q_0^{(i)}, F_i)$ and $K = \{K_i \mid 1 \leq i \leq n\}$. For each i, $1 \leq i \leq n$, the parameters of A_i are defined by

$$
\begin{aligned}
Q_i &= Q \cup X_i \cup Y_i \cup Z_i, \\
X_i &= \begin{cases} \{K_n\}, \text{ if } i = 1, \\ \{K_n, K_{i-1}\}, \text{ if } 2 \leq i \leq n-1, \\ \{K_{n-1}\}, \text{ if } i = n, \end{cases} \\
Y_i &= \begin{cases} Q \times (V \cup \{\varepsilon\}), \text{ if } i = 1, \\ (Q \times (V \cup \{\varepsilon\})^{i-1}) \cup (Q \times (V \cup \{\varepsilon\})^{i}), \text{ if } 2 \leq i \leq n-1, \\ (Q \times (V \cup \{\varepsilon\})^{n-1}) \cup \{p_0\}, \text{ if } i = n, \end{cases} \\
Z_i &= \begin{cases} \emptyset, \text{ if } (1 \leq i \leq 2 = n) \text{ or } (n = 3 \wedge i = 2), \\ \{s_2^{(n)}\}, \text{ if } i = n = 3, \\ \{s_2^{(i)}, s_3^{(i)}, \ldots, s_{n-i}^{(i)}\}, \text{ if } n \geq 3, 1 \leq i \leq r, \\ \{s_2^{(i)}, s_3^{(i)}, \ldots, s_{i-1}^{(i)}\}, \text{ if } n > 3, r < i \leq n, \end{cases}
\end{aligned}
$$

where $p_0 \notin Q$ and

$$
r = \begin{cases} \frac{n}{2}, \text{ if } n \text{ is even}, \\ \frac{n-1}{2}, \text{ if } n \text{ is odd}. \end{cases}
$$

Further on,

$$q_0^{(i)} = \begin{cases} q_0, \text{ if } 1 \leq i \leq n-1, \\ p_0, \text{ if } i = n, \end{cases} \qquad F_i = \begin{cases} F, \text{ if } 1 \leq i \leq n-1, \\ \{p_0\}, \text{ if } i = n, \end{cases}$$

and the transition mappings f_i are defined as follows:
• $i = 1$

$$\begin{aligned} f_1(q,a) &= \begin{cases} \{(q,a)\}, \text{ if } a \in V \cup \{\varepsilon\}, q \in Q, \\ \{(q,a), s_2^{(1)}\}, \text{ if } a = \varepsilon, q \in Q, n > 2, \\ \{(q,a), K_2\}, \text{ if } a = \varepsilon, q \in Q, n = 2, \end{cases} \\ f_1(s_j^{(1)}, \varepsilon) &= \begin{cases} \{s_{j+1}^{(1)}\}, \text{ if } 2 \leq j \leq n-2, \\ \{K_n\}, \text{ if } j = n-1. \end{cases} \end{aligned}$$

• $2 \leq i \leq n-1$

$$\begin{aligned} f_i((q, a_1, a_2, \ldots, a_{i-1}), b) &= \{(q, a_1, a_2, \ldots, a_{i-1}, b)\}, q \in Q, \\ &\quad a_j \in V \cup \{\varepsilon\}, 1 \leq j \leq i-1, b \in V \cup \{\varepsilon\}. \end{aligned}$$

We distinguish the following four cases:

Case 1. $(i = 2) \wedge (n = 3)$

$$f_2(q, \varepsilon) = \{K_1, K_3\}, q \in Q.$$

Case 2. $2 \leq i \leq r$, $n \geq 4$

$$\begin{aligned} f_i(q, \varepsilon) &= \begin{cases} \{s_2^{(i)}\}, \text{ if } i > 2, q \in Q, \\ \{s_2^{(i)}, K_1\}, \text{ if } i = 2, q \in Q, \end{cases} \\ f_i(s_j^{(i)}, \varepsilon) &= \begin{cases} \{s_{j+1}^{(i)}\}, \text{ if } j \leq n-i-1, j \neq i-1, \\ \{s_{j+1}^{(i)}, K_{i-1}\}, \text{ if } j = i-1, \end{cases} \\ f_i(s_{n-i}^{(i)}, \varepsilon) &= \{K_n\}. \end{aligned}$$

Case 3. $(r < i \leq n-1) \wedge (n \geq 4) \wedge (n \text{ is even})$ or $(r+1 < i \leq n-1) \wedge (n \geq 4) \wedge (n \text{ is odd})$, where r is defined as above.

$$\begin{aligned} f_i(q, \varepsilon) &= \begin{cases} \{s_2^{(i)}\}, \text{ if } i < n-1, q \in Q, \\ \{s_2^{(i)}, K_n\}, \text{ if } i = n-1, q \in Q, \end{cases} \\ f_i(s_j^{(i)}, \varepsilon) &= \begin{cases} \{s_{j+1}^{(i)}\}, \text{ if } j \leq i-2, j \neq n-i, \\ \{s_{j+1}^{(i)}, K_n\}, \text{ if } j = n-i, \end{cases} \\ f_i(s_{i-1}^{(i)}, \varepsilon) &= \{K_{i-1}\}. \end{aligned}$$

Case 4. $(i = r+1) \wedge ((n \geq 4) \wedge (n \text{ is odd}))$

$$\begin{aligned} f_{r+1}(q, \varepsilon) &= \{s_2^{(r+1)}\}, q \in Q, \\ f_{r+1}(s_j^{(r+1)}, \varepsilon) &= \begin{cases} \{s_{j+1}^{(r+1)}\}, \text{ if } j \leq r-1, \\ \{K_r, K_n\}, \text{ if } j = r. \end{cases} \end{aligned}$$

• $i = n$

$$\begin{aligned} f_n((q, a_1, a_2, \ldots, a_{n-1}), b) &= f(q, a_1, a_2, \ldots, a_{i-1}, b), q \in Q, \\ &\quad a_j \in V \cup \{\varepsilon\}, 1 \leq j \leq n-1, b \in V \cup \{\varepsilon\}, \end{aligned}$$

$$\begin{aligned} f_n(p_0, \varepsilon) &= \begin{cases} \{s_2^{(n)}\}, \text{ if } n \geq 3, \\ \{K_1\}, \text{ if } n = 2, \end{cases} \\ f_n(s_j^{(n)}, \varepsilon) &= \begin{cases} \{s_{j+1}^{(n)}\}, \text{ if } 2 \leq j \leq n-2, \\ \{K_{n-1}\}, \text{ if } j = n-1. \end{cases} \end{aligned}$$

In all the other cases not mentioned above each mapping f_i, $1 \leq i \leq n$, returns the empty set.

We give some explanations about the working mode of the system. The first component being in a state from Q, either effectively reads an input symbol or ε and "stores" this symbol or ε in the current state. This state is sent to the second component which has just asked for it. Since this is a $rpcfa(n)$, after sending its current state to the second component, the first component goes back to its initial state. We now force the first component to go from its initial state to the waiting state $s_2^{(1)}$ through an $\varepsilon-$ move, otherwise the system will get stuck into a deadlock. Indeed, the second component cannot continue its work being in a state from Q_1 other than a state of the form (q, a) for some state $q \in Q$ and $a \in V \cup \{\varepsilon\}$. On the other hand, if the second component has not issued its query to the first one, the system is blocked since the first component cannot continue its work in a state from $Q \times (V \cup \{\varepsilon\})$. It remains one more possibility, namely the first component enters the state $s_2^{(1)}$ and the second one has not issued any query. This case will be analyzed later. Note that this case cannot hold for $n = 2$. The beginning of a computation in the case $n = 2$ looks as follows:

$$\begin{aligned} &(q_0, ax, p_0, ax) \vdash_r ((q_0, a), x, K_1, ax) \vdash_r (q_0, x, (q_0, a), ax) \vdash_r \\ &(K_2, x, f(q_0, a, a), x) \vdash_r (f(q_0, a, a), x, p_0, x) \vdash_r \ldots \end{aligned}$$

The second component, in its turn, does the same, namely stores, the symbol read by its reading head in the current state, besides the symbol stored by the first component, and sends this new state to the third component which has just asked for it. After sending its state to the third component, the second component goes to the initial state as this is a $rpcfa(n)$. We now force the second component to go from its initial state to the waiting state $s_2^{(2)}$ through an $\varepsilon-$move, otherwise the

system will get stuck again into a deadlock. The remaining case from above can be continued in only one way: the second component enters $s_3^{(2)}$ while the third one enters $s_2^{(3)}$.

The process goes on this way until the last component receives the state which encodes the state of the first component when the process started as well as all symbols read, in turn, by all the other components so far. Now, according to the transition mapping of the $n-$head finite automaton the last component enters a state from Q which is sent to all the other components at the same time. Since this is a $rpcfa(n)$, after sending this state, the last component goes to its initial state. Consequently, this is the way in which the system simulates a move in the $n-$head finite automaton. Let us now consider the special case. More precisely, since we are forcing the i^{th} component to go from its initial state to the waiting state $s_2^{(i)}$, it may be possible that in the beginning all the components except the last one may directly go to their corresponding waiting states, but by doing this the system will get blocked as the last component is expecting a state from $Q \times (V \cup \{\varepsilon\})^{n-1}$ rather than a waiting state. From the construction itself, we can conclude that $Rec_r(\mathcal{A}) = Rec(A)$. □

Since $RPCFA(n) \subseteq FA(n)$ (by Th. 1) and by using the first statement of Theorem 2, we can conclude the following theorem.

Theorem 4. *$RPCFA(n) = PCFA(n)$.*

5. Conclusion

In this note we have shown that the class of languages accepted by $rpcfa(n)$ is the same as the class of languages accepted by $n-$head finite automata. This result is an extension of the results concerning the comparison of the computational power of parallel communicating finite automata and multihead automata from [8]. As a direct consequence of this result as well as some results form [8] we have shown that the computational power of returning and non-returning parallel communicating finite automata systems is the same.

We list here some open problems which appear attractive to us.

1. Can the statement of Theorem 3 be extended to deterministic variants? Remember that $DPCFA(n) = DFA(n)$.

2. What is the relationship between $CPCFA(n)$ and $RCPCFA(n)$?

3. Can an $n-$head finite automaton be simulated by an automata system in $cpcfa(n)$ or $rcpcfa(n)$?

References

[1] A. O. Buda, Multiprocessor automata. *Inform. Proces. Lett.* **25** (1977) 257–261.

[2] E. Csuhaj-Varju, J. Dassow, J. Kelemen and Gh. Paun, *Grammar Systems. A grammatical approach to distribution and cooperation*. Gordon and Breach (1994).

[3] E. Csuhaj-Varju, C. Martin-Vide, V. Mitrana and G. Vaszil, Parallel communicating pushdown automata systems. *Int. J. Found. Comput. Sci.* **11** (2000) 633–650.

[4] J. Dassow and V. Mitrana, stack cooperation in multi-stack pushdown automata. *J. Comput. Syst. Sci.* **58** (1999) 611–621.
[5] J. Dassow, Gh. Paun and G. Rozenberg, Grammar systems, in [9].
[6] J. Hartmanis, On nondeterminancy in simple computing devices. *Acta Inform.* **1** (1972) 336–344.
[7] O.H. Ibarra, On two-way multihead automata. *J. Comput. Syst. Sci.* **7** (1973) 28–36.
[8] C. Martín-Vide, A. Mateescu and V. Mitrana, Parallel finite automata systems communicating by states. *Int. J. Found. Comput. Sci.* **13** (2002) 733–749.
[9] G. Rozenberg, A. Salomaa (eds.), *The Handbook of Formal Languages*, Springer-Verlag (1997).

Communicated by Z. Esik.
Received May 24, 2005. Accepted March 30, 2006.