Fundamenta Informaticae 136 (2015) 177–197
DOI 10.3233/FI-2015-1149
IOS Press

Asynchronous Systems of Parallel Communicating Finite Automata

Marcel Vollweiler*
Fachbereich Elektrotechnik/Informatik
Universität Kassel,
34109 Kassel, Germany
vollweiler@theory.informatik.uni-kassel.de

Abstract. Synchronous systems of parallel communicating one-way finite automata have already been investigated. There, all components work stepwise in parallel, and the communication between the components is realized by requesting states in a one-directional manner. This means that one component can request information in form of the current state from another component, where the latter one sends its current state without realizing that a communication takes place. Here, we introduce asynchronous systems of parallel communicating one-way and two-way finite automata with a bidirectional communication protocol. A communication only takes place, when both components - the requesting and the responding component - are ready to communicate. It is shown that almost all language classes that are characterized by these systems coincide with the language classes that are characterized by multi-head finite automata.

Moreover, our communication protocol uses blocking point-to-point communications, i.e. a communication takes place between two components, and a communicating component is blocked until the communication has been finished. There have also been studied asynchronous systems of finite automata with non-blocking communication in the literature. Thus, we compare synchronous and asynchronous systems on the one hand and asynchronous systems with blocking and non-blocking communication on the other hand.

Finally, we give some results on the communication complexity of our systems, where the amount of communication is measured by counting each message which is sent from one component to another during a computation of a system. Particularly, we show that with constantly many communications our systems can only accept regular languages, and at most linearly (polynomially) many communications are needed for systems of one-way (two-way) components depending on the length of the input. Further, there exists no system that executes more than constantly many and less than linearly many communications.

*Address for correspondence: Fachbereich Elektrotechnik/Informatik, Universität Kassel, 34109 Kassel, Germany

1. Introduction

Systems of parallel communicating components (PC systems, for short) are an approach for distributed and parallel computing and cooperation in Formal Language Theory. The first type of such systems was formed by the so-called *parallel communicating grammar system with regular grammars* [17]. Later, PC systems were investigated for other types of grammars [4] and also for various types of automata: parallel communicating (one-way) finite automata systems [14], parallel communicating pushdown automata systems [5], parallel communicating Watson-Crick automata systems [6], and systems of parallel communicating restarting automata [18].

A *system of parallel communicating finite automata* (abbreviated by PCFA system) as introduced in [14] and investigated in, e.g., [1, 2, 3, 15] consists of a finite number of one-way finite automata (the *components*) that work together stepwise in parallel. This means that there exists a global clock that forces each component to execute exactly one computation step in each unit of time. Thus, the components work with the same speed. Particularly, this is why we call these systems synchronized in what follows. Information is transmitted between the components via communication that is realized by using special *communication states*. When two components, say A_i and A_j, want to communicate with each other, then A_i enters the communication state K_j in order to request the current state of A_j. Then, the system performs a communication step, where A_j sends its current state to A_i, and the finite control of A_i is set into this state. Whenever the requesting component enters a communication state, the communication step has to take place immediately, i.e. without waiting. A communication step can only be executed if the sending component is not itself in a communication state. In such a situation, the requested component has to communicate first. In a situation of cyclic requests, the system is blocked and rejects the input. After a communication step, both components continue their local computations. In addition, it is possible that several components request the current state of A_j at the same time by all entering state K_j. Then, A_j sends its state to all requesting components concurrently.

The described communication is *one-directional*. Only the requesting component determines when and with which other component a communication takes place. In fact, the sending component is not aware of any communication, i.e. it does neither know the requesting component nor the point in time of the communication step. It does not even realize that a communication took place at all. Further, the requesting component forgets everything about its previous computation during a computation step, since it has at most one communication state for each other component. Thus, a component cannot remember the correct computation path that leads to a communication state. In other words: reaching a communication state, a component forgets everything about its already read part of the input.

Here, we introduce systems of parallel communicating one-way and two-way finite automata with a fundamentally different communication protocol. For this, we define four types of communication states: request states, response states, receive states, and acknowledge states. Now, the communication works as follows: a component A_i can request some information from another component A_j by entering a request state req^j_ℓ. Component A_j can respond to A_i by entering a response state $\mathsf{res}^i_{\ell',c}$, where c is an information that is transmitted from A_j to A_i. The symbols ℓ and ℓ' are local information of A_i and A_j that are kept during a communication step but they are not sent. As long as only one communication partner has entered a communication state, and the other has not, the former one waits, while all the other components continue their computations independently. When A_i and A_j both reach the corresponding communication states, then a communication step is performed: A_i is set into the receive state $\mathsf{rec}^j_{\ell,c}$ (it has received information c from A_j), and A_j is set into the acknowledge state $\mathsf{ack}^i_{\ell',c}$ (the

receipt of the message is acknowledged). Thereafter, both components continue their local computations independently. Observe that both, the requesting and the responding component, are actively involved in a communication. Moreover, due to the fact that a requesting or responding component can wait for an answer arbitrarily long, the components not necessarily have to work stepwise in parallel anymore. This is why we call PCFA systems with our communication protocol *asynchronous systems of parallel communicating finite automata* (abbreviated by APCFA systems). Asynchronous PC systems were already considered for restarting automata [18] and for pushdown automata [16].

Our communication protocol is *blocking*, since a component is blocked whenever it enters a request or response state. Only when the communication step has been executed, components that are involved in the communication are allowed to continue their local computations. In contrast, Jurdziński et al. introduced asynchronous systems of finite automata (*ASFA*) with *non-blocking* communication [10, 12, 13]. These systems consist of deterministic finite two-way automata that are equiped with a message buffer of constant size for each other component. A component communicates with other components by writing messages into their message buffers. It does not wait until the receiver has processed its message but it continues its computation immediately. This is why these communication is *non-blocking*. If a message buffer is full, then either new messages are refused or the messages are shifted such that the oldest message in the buffer is droped.

The usage of non-blocking communication in asynchronous systems leads to different computations for the same input even in deterministic systems, since messages can be received in different chronological orders. At which point of time all messages are received is called time pattern. So there can exist different time patterns for the same input. Due to the definition [12], an ASFA accepts an input if, for all possible time patterns, a fixed component accepts after a finite number of computation steps. On the other hand, a system rejects the input if, for all possible time patterns, a fixed component rejects the input after a finite number of computation steps. Observe that this is a strong requirement for the construction of such a system, since a system is not allowed to reach an accepting and a rejecting configuration for the same input due to different time patterns. Moreover, it is not clear whether this property is decidable at all. In contrast, APCFA systems accept if and only if some component reaches an accepting configuration. However, besides blocking and non-blocking communication, the definition of acceptance is another fundamental difference between the asynchronous systems of finite automata defined by Jurdziński et al. and the APCFA systems that we introduce here.

This paper is structured as follows. First, we restate the definition of multi-head finite automata and finite automata. Then, we define asynchronous PC systems of finite automata. In Section 4, we show that the computational power of nondeterministic APCFA systems is equal to that of nondeterministic multi-head finite automata. In Section 5, the deterministic types of APCFA systems are compared with deterministic multi-head finite automata. Section 6 contains a comparison between the language classes that are characterized by synchronous PCFA systems, asynchronous PCFA systems, and ASFA. In Section 7, some results on the communication complexity of APCFA systems are given. Finally, we conclude with some open questions.

2. Multi-head finite automata

In this section, we restate the definition of multi-head finite automata following [9]. A *nondeterministic two-way n-head automaton* (2-NFA(n) for short) is a device that consists of a finite control, an input

tape, and n read-only heads. Formally, it is described by a tuple $M = (Q, \Sigma, n, \delta, ¢, \$, q_0, F)$, where Q is the set of states, Σ is the set of input symbols, n is the number of heads, q_0 is the initial state ($q_0 \in Q$), and F is the set of dedicated final states ($F \subseteq Q$). The special symbols ¢ and \$ are the left and the right tape markers, respectively. Moreover, δ is a mapping from $Q \times (\Sigma \cup \{¢, \$\})^n$ into the set of all (finite) subsets of $Q \times \{-1, 0, 1\}^n$. It is written as $(q, (d_1, d_2, \ldots, d_n)) \in \delta(p, (a_1, a_2, \ldots, a_n))$ if the automaton can change from state p into state q while reading symbol $a_i \in \Sigma \cup \{¢, \$\}$ with the i-th head and moving the i-th head $d_i \in \{-1, 0, 1\}$ positions to the right, $1 \leq i \leq n$. If d_i is -1, then the i-th head is moved to the left. In the cases of $d_i = 0$ or $d_i = 1$, the i-th head keeps its position or moves one step to the right, respectively. Move left and move right steps are only allowed if a head does not already read the left or the right marker, respectively. Otherwise, the head keeps its position.

A *configuration* κ of an n-head automaton is an n-tuple $\kappa = (x_1 q y_1, x_2 q y_2, \ldots, x_n q y_n)$, where q is the current state, $x_1 y_1 = x_2 y_2 = \ldots = x_n y_n = ¢w\$$ for an input $w \in \Sigma^*$, and the i-th head is positioned on the first symbol of y_i, for all $i = 1, \ldots, n$. The *initial configuration* for the initial state q_0 and an input word w is the n-tuple $(q_0 ¢ w\$, q_0 ¢ w\$, \ldots, q_0 ¢ w\$)$. An *accepting configuration* is a configuration $(x_1 q a_1 y_1, x_2 q a_2 y_2, \ldots, x_n q a_n y_n)$, where $q \in F$ and $\delta(q, (a_1, \ldots, a_n)) = \emptyset$.

For two configurations $(x_1 p a_1 y_1, x_2 p a_2 y_2, \ldots, x_n p a_n y_n)$ and $(x'_1 q y'_1, x'_2 q y'_2, \ldots, x'_n q y'_n)$, a *computation step*

$$(x_1 p a_1 y_1, x_2 p a_2 y_2, \ldots, x_n p a_n y_n) \vdash_M (x'_1 q y'_1, x'_2 q y'_2, \ldots, x'_n q y'_n)$$

can be performed if and only if $(q, (d_1, \ldots, d_n)) \in \delta(p, (a_1, \ldots, a_n))$ and, for all $i = 1, \ldots, n$, one of the following cases holds:

1. $d_i = -1$, $x_i = x''_i b_i$, $x'_i = x''_i$, and $y'_i = b_i a_i y_i$; or
2. $d_i = 0$, $x'_i = x_i$, and $y'_i = a_i y_i$; or
3. $d_i = 1$, $x'_i = x_i a_i$, and $y'_i = y_i$.

The reflexive and transitive closure of $\vdash_M$ is denoted by $\vdash^*_M$. A *computation* of an automaton M is a sequence of computation steps, and $\kappa \vdash^*_M \kappa'$ holds for two configurations κ and κ' if and only if there exists a computation from κ to κ'. The language that is accepted by an n-head automaton M is defined by

$$L(M) = \{w \in \Sigma^* \mid (q_0 ¢ w\$, q_0 ¢ w\$, \ldots, q_0 ¢ w\$) \vdash^*_M \kappa, \kappa \text{ is an accepting configuration}\}.$$

If δ is a (partial) function, then the automaton is called *deterministic* (2-DFA(n) for short). If we do not allow the heads to move to the left, i.e. δ is a mapping or a function into the set of all (finite) subsets of $Q \times \{0, 1\}^n$, then M is called a one-way automaton (1-NFA(n) in the nondeterministic case and 1-DFA(n) in the deterministic case). We get a standard finite automaton if we take $n = 1$ and let δ be a mapping into the set of all (finite) subsets of $Q \times \{0, 1\}$. Observe that in the deterministic case the automaton can either move its head or keeps its position (but not both), when being in a particular state and reading a particular symbol. A finite two-way automaton is just a two-way 1-head automaton.

The classes of languages that can be accepted by some 1-DFA(n), 1-NFA(n), 2-DFA(n), or 2-NFA(n) are denoted by $\mathcal{L}$(1-DFA(n)), $\mathcal{L}$(1-NFA(n)), $\mathcal{L}$(2-DFA(n)), and $\mathcal{L}$(2-NFA(n)), respectively. In addition, we define $\mathcal{L}(\mathsf{X}) = \bigcup_{\mathsf{n} \geq 1} \mathcal{L}(\mathsf{X}(\mathsf{n}))$, for all $\mathsf{X} \in \{\text{1-DFA}, \text{1-NFA}, \text{2-DFA}, \text{2-NFA}\}$.

3. Asynchronous systems of parallel communicating finite automata

An *asynchronous system of parallel communicating finite automata* (abbreviated as APCFA system) of *degree* n is an n-tuple $\mathcal{A} = (A_1, A_2, \ldots, A_n)$, where $A_i = (Q_i, \Sigma, 1, \delta_i, ¢, \$, q_i, F_i)$ are (one-head) finite automata for each $i = 1, \ldots, n$. The automata $A_1, \ldots, A_n$ are called the *components* of the system $\mathcal{A}$.

The communication between the automata is realized by communication states that are included in the sets of states. We distinguish four types of communication states: *request states* (req^i_ℓ), *response states* ($\mathsf{res}^i_{\ell,c}$), *receive states* ($\mathsf{rec}^i_{\ell,c}$), and *acknowledge states* ($\mathsf{ack}^i_{\ell,c}$). If a component A_j enters a request state req^i_ℓ, then it requests information from A_i, while it can store some local information ℓ that is kept during the communication step. The component A_i can respond to A_j by entering the response state $\mathsf{res}^j_{\ell',c}$, sending the information c and storing some local information ℓ'. Whenever a component has reached a request or response state, it waits until the communication partner reaches the corresponding communication state, while the other components continue their computations. If both communication partners have reached corresponding communication states, then the communication takes place. This is realized by setting A_j into the state $\mathsf{rec}^i_{\ell,c}$ (because it now has received information c) and setting A_i into the state $\mathsf{ack}^j_{\ell',c}$ (the receipt of the message is acknowledged). Thereafter, both components continue their local computations independently.

The notation of the communication states gives an intuitive representation of important aspects of a communication: *What* is communicated (message c)? *With whom* is communicated (receiver i)? *Which role* plays the component within the communication (request/ask or response/answer)? What is *local knowledge* and should not be transmitted (local information l)? In our context, the local information is achieved from the previous computation and is combined with the communication result. Formally, the local information is also used to distinguish different request/response states for the same communication partner and the same message.

A *configuration* K of an APCFA system $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ is an n-tuple $(x_1p_1a_1y_1, x_2p_2a_2y_2, \ldots, x_np_na_ny_n)$, where $x_ip_ia_iy_i$ is the current configuration of component A_i for each $1 \leq i \leq n$, i.e. p_i is the current state of A_i, $x_ia_iy_i = ¢w\$$ for the input w, and the head of A_i is currently positioned on the symbol a_i. The *initial configuration* of $\mathcal{A}$ for an input word w is $(q_1¢w\$, q_2¢w\$, \ldots, q_n¢w\$)$. An *accepting configuration* of $\mathcal{A}$ is a configuration that contains an accepting configuration of at least one component. Observe that this is exactly the case when at least one component is currently in a final state and none of its transitions can be applied in this situation.

For two configurations $K = (x_1p_1y_1, \ldots, x_np_ny_n)$ and $K' = (x'_1p'_1y'_1, \ldots, x'_np'_ny'_n)$, a *computation step* $K \vdash_{\mathcal{A}} K'$ of a system $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ is valid if and only if one of the following conditions holds for each component A_i, $1 \leq i \leq n$:

1. $x_ip_iy_i \vdash_{A_i} x'_ip'_iy'_i$ (local computation step),

2. $\exists j \in \{1, 2, \ldots, n\} \setminus \{i\}$: $p_i = \mathsf{req}^j_{\ell_i}$, $p_j = \mathsf{res}^i_{\ell_j,c}$,
   $p'_i = \mathsf{rec}^j_{\ell_i,c}$, $p'_j = \mathsf{ack}^i_{\ell_j,c}$,
   $x'_i = x_i$, $y'_i = y_i$ (incoming communication),

3. $\exists j \in \{1, 2, \ldots, n\} \setminus \{i\}$: $p_i = \mathsf{res}^j_{\ell_i,c}$, $p_j = \mathsf{req}^i_{\ell_j}$,
   $p'_i = \mathsf{ack}^j_{\ell_i,c}$, $p'_j = \mathsf{rec}^i_{\ell_j,c}$,
   $x'_i = x_i$, $y'_i = y_i$ (outgoing communication),

4. $p'_i = p_i$, $x'_i = x_i$, $y'_i = y_i$, $p_i \notin F_i$, and conditions 1 - 3 do not hold
   (components that are stuck or wait for a communication answer just keep their configurations and all the other components of the system continue their computations independently).

Observe that whenever at least one component reaches the accepting configuration, then the system cannot continue its computation. Thus, it halts and accepts. In particular, we assume that request and response states are not contained in the sets of final states.

The reflexive and transitive closure of $\vdash_{\mathcal{A}}$ is denoted by $\vdash^*_{\mathcal{A}}$. The language that is accepted by an APCFA system $\mathcal{A}$ is defined by

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid (q_1 ¢ w \$, q_2 ¢ w \$, \ldots, q_n ¢ w \$) \vdash^*_{\mathcal{A}} K, K \text{ is an accepting configuration}\},$$

where Σ is the input alphabet of each component.

If a system consists of n one-way (two-way) finite automata, then its type is denoted by 1-APCFA(n) (2-APCFA(n)). Systems are called *deterministic* if all components are deterministic. The types of deterministic systems are denoted by 1-DAPCFA(n) and 2-DAPCFA(n). The classes of languages that can be accepted by systems of these types are denoted by $\mathcal{L}$(1-APCFA(n)), $\mathcal{L}$(2-APCFA(n)), $\mathcal{L}$(1-DAPCFA(n)), and $\mathcal{L}$(2-DAPCFA(n)).

A system is said to be *centralized* if there exists one dedicated component (the master) such that all the other components (clients) are only allowed to communicate with the master but not with other clients. For this, the direction of the communication is not important, i.e. whether the master or the client sends the request or the response. Centralized systems are marked with the prefix 'C' (1-CAPCFA(n), 2-CAPCFA(n), 1-DCAPCFA(n), 2-DCAPCFA(n)).

Further, a system is said to be working in *returning mode* if a responding component is reset into its initial state instead into the acknowledge state after each communication. We denote the computation step relation of a system $\mathcal{A}$ working in returning mode with $\vdash_{\mathcal{A},r}$ and define it analogously to the relation $\vdash_{\mathcal{A}}$. We replace the second and the third item of the above definition with:

2. $\exists j \in \{1, 2, \ldots, n\} \setminus \{i\} : p_i = \mathsf{req}^j_{\ell_i}, \quad p_j = \mathsf{res}^i_{\ell_j, c},$
   $p'_i = \mathsf{rec}^j_{\ell_i, c}, \; p'_j = q_j,$
   $x'_i = x_i, \qquad y'_i = y_i \qquad$ (incoming communication),

3. $\exists j \in \{1, 2, \ldots, n\} \setminus \{i\} : p_i = \mathsf{res}^j_{\ell_i, c}, \; p_j = \mathsf{req}^i_{\ell_j},$
   $p'_i = q_i, \qquad p'_j = \mathsf{rec}^i_{\ell_j, c},$
   $x'_i = x_i, \qquad y'_i = y_i \qquad$ (outgoing communication).

The reflexive and transitive closure of $\vdash_{\mathcal{A},r}$ is denoted by $\vdash^*_{\mathcal{A},r}$ and the language that is accepted by a system $\mathcal{A}$ working in returning mode is defined by

$$L_r(\mathcal{A}) = \{w \in \Sigma^* \mid (q_1 ¢ w \$, q_2 ¢ w \$, \ldots, q_n ¢ w \$) \vdash^*_{\mathcal{A},r} K, K \text{ is an accepting configuration}\}.$$

A system that does not work in returning mode is said to work in *non-returning mode*. Observe that the property of working in returning or non-returning mode is a property of the computation and not of the system itself. However, if we consider a type of APCFA systems working only in returning-mode, then this is denoted by adding the prefix 'R'.

Finally, we distinguish the language classes $\mathcal{L}(\mathsf{X}(n))$ for any type X of APCFA systems, i.e. $\mathsf{X} \in \{1,2\}\{\text{-}\}\{\mathsf{D},\varepsilon\}\{\mathsf{R},\varepsilon\}\{\mathsf{C},\varepsilon\}\{\mathsf{APCFA}\}$, where ε denotes the empty word. For instance, $\mathcal{L}(\text{2-DRCAPCFA}(n))$ is the class of all languages that are accepted by centralized APCFA systems of deterministic two-way finite automata of degree n working in returning mode. Moreover, $\mathcal{L}(\mathsf{X}) = \bigcup_{n\geq 1} \mathcal{L}(\mathsf{X}(n))$.

The following example demonstrates how an APCFA system with three components accepts the non-contextfree language $L = \{w \in \{a,b,c\}^* \mid |w|_a \leq |w|_b, 2|w|_a \leq |w|_c\}$, where $|w|_x$ denotes the number of occurences of symbol x in word w.

Example 3.1. We present a system $\mathcal{M} = (M_1, M_2, M_3)$ that accepts the language $L = \{w \in \{a,b,c\}^* \mid |w|_a \leq |w|_b, 2|w|_a \leq |w|_c\}$. Basically, the three components M_1, M_2, and M_3 move their heads from left to right and collect the symbols a, b, and c, respectively. Whenever M_1 reads an a, then it initiates a communication with M_2 by entering the request state req^2. M_2 answers to M_1 by entering the response state res^1_b if and only if it has found any b. Thereafter, M_2 continues its local computation and searches for the next b, and M_1 enters the request state req^3 in order to get the information that M_3 has found two c's. Only if M_3 confirms this by entering the response state res^1_c, both continue their local computations and try to find another a or c, respectively. Reaching the end of the input, i.e. the right end marker \$, M_1 accepts by changing into state q_f. Observe that M_1 is only able to reach q_f if for each a there have been enough b's and c's. If there is a b or c missing, then a communication of M_1 cannot be resolved and thus M_1 get stuck. Formally, the transition functions are defined as follows:

$$
\begin{aligned}
\delta_1(q_1,\text{¢}) &= (q_1,1),\\
\delta_1(q_1,b) &= (q_1,1),\\
\delta_1(q_1,c) &= (q_1,1),\\
\delta_1(q_1,a) &= (\mathsf{req}^2,0),\\
\delta_1(\mathsf{rec}^2_b,a) &= (\mathsf{req}^3,0),\\
\delta_1(\mathsf{rec}^3_c,a) &= (q_1,1),\\
\delta_1(q_1,\$) &= (q_f,0),
\end{aligned}
\qquad
\begin{aligned}
\delta_2(q_2,\text{¢}) &= (q_2,1),\\
\delta_2(q_2,a) &= (q_2,1),\\
\delta_2(q_2,c) &= (q_2,1),\\
\delta_2(q_2,b) &= (\mathsf{res}^1_b,0),\\
\delta_2(\mathsf{ack}^1_b,b) &= (q_2,1),
\end{aligned}
\qquad
\begin{aligned}
\delta_3(q_3,\text{¢}) &= (q_3,1),\\
\delta_3(q_3,a) &= (q_3,1),\\
\delta_3(q_3,b) &= (q_3,1),\\
\delta_3(q_3,c) &= (q_c,1),\\
\delta_3(q_c,a) &= (q_c,1),\\
\delta_3(q_c,b) &= (q_c,1),\\
\delta_3(q_c,c) &= (\mathsf{res}^1_c,0),\\
\delta_3(\mathsf{ack}^1_c,c) &= (q_3,1).
\end{aligned}
$$

On input $ccbacbac$ $\mathcal{M}$ executes the following computation:

$$
\begin{aligned}
&(q_1\text{¢}ccbacbac\$ \quad , q_2\text{¢}ccbacbac\$ \quad , q_3\text{¢}ccbacbac\$ \quad)\\
\vdash\ &(\text{¢}q_1ccbacbac\$ \quad , \text{¢}q_2ccbacbac\$ \quad , \text{¢}q_3ccbacbac\$ \quad)\\
\vdash\ &(\text{¢}cq_1cbacbac\$ \quad , \text{¢}cq_2cbacbac\$ \quad , \text{¢}cq_ccbacbac\$ \quad)\\
\vdash\ &(\text{¢}ccq_1bacbac\$ \quad , \text{¢}ccq_2bacbac\$ \quad , \text{¢}c\mathsf{res}^1_ccbacbac\$)\\
\vdash\ &(\text{¢}ccbq_1acbac\$ \quad , \text{¢}cc\mathsf{res}^1_bbacbac\$, \text{¢}c\mathsf{res}^1_ccbacbac\$)\\
\vdash\ &(\text{¢}ccb\mathsf{req}^2acbac\$, \text{¢}cc\mathsf{res}^1_bbacbac\$, \text{¢}c\mathsf{res}^1_ccbacbac\$)\\
\vdash\ &(\text{¢}ccb\mathsf{rec}^2_bacbac\$, \text{¢}cc\mathsf{ack}^1_bbacbac\$, \text{¢}c\mathsf{res}^1_ccbacbac\$)\\
\vdash\ &(\text{¢}ccb\mathsf{req}^3acbac\$, \text{¢}ccbq_2acbac\$ \quad , \text{¢}c\mathsf{res}^1_ccbacbac\$)\\
\vdash\ &(\text{¢}ccb\mathsf{rec}^3_cacbac\$, \text{¢}ccbaq_2cbac\$ \quad , \text{¢}c\mathsf{ack}^1_ccbacbac\$)\\
\vdash\ &(\text{¢}ccbaq_1cbac\$ \quad , \text{¢}ccbacq_2bac\$ \quad , \text{¢}ccq_3bacbac\$ \quad)
\end{aligned}
$$

$$\begin{aligned}
&\vdash (\text¢ccbacq_1bac\$ \quad , \text¢ccbac\mathsf{res}_b^1bac\$ \, , \text¢ccbq_3acbac\$ \quad)\\
&\vdash (\text¢ccbacbq_1ac\$ \quad , \text¢ccbac\mathsf{res}_b^1bac\$ \, , \text¢ccbaq_3cbac\$ \quad)\\
&\vdash (\text¢ccbacb\mathsf{req}^2ac\$, \text¢ccbac\mathsf{res}_b^1bac\$ \, , \text¢ccbacq_cbac\$ \quad)\\
&\vdash (\text¢ccbacb\mathsf{rec}_b^2ac\$, \text¢ccbac\mathsf{ack}_b^1bac\$, \text¢ccbacbq_cac\$ \quad)\\
&\vdash (\text¢ccbacb\mathsf{req}^3ac\$, \text¢ccbacbq_2ac\$ \quad , \text¢ccbacbaq_cc\$ \quad)\\
&\vdash (\text¢ccbacb\mathsf{req}^3ac\$, \text¢ccbacbaq_2c\$ \quad , \text¢ccbacba\mathsf{res}_c^1c\$ \,)\\
&\vdash (\text¢ccbacb\mathsf{rec}_c^3ac\$, \text¢ccbacbacq_2\$ \quad , \text¢ccbacba\mathsf{ack}_c^1c\$)\\
&\vdash (\text¢ccbacbaq_1c\$ \quad , \text¢ccbacbacq_2\$ \quad , \text¢ccbacbacq_3\$ \quad)\\
&\vdash (\text¢ccbacbacq_1\$ \quad , \text¢ccbacbacq_2\$ \quad , \text¢ccbacbacq_3\$ \quad)\\
&\vdash (\text¢ccbacbacq_f\$ \quad , \text¢ccbacbacq_2\$ \quad , \text¢ccbacbacq_3\$ \quad).
\end{aligned}$$

4. Computational power of nondeterministic APCFA systems

In this section, the expressive power of nondeterministic APCFA systems is investigated. Obviously, each centralized system is a special case of an arbitrary system. Moreover, each system working in returning mode can be simulated by a system working in non-returning mode. For this, a sending component changes immediately from an acknowledge state into its initial state after each communication. Thus, we have the following inclusions:

Corollary 4.1. For all $n \geq 1$ and $\mathsf{X} \in \{1, 2\}$,

(a) $\mathcal{L}(\mathsf{X\text{-}CAPCFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}APCFA}(n))$,

(b) $\mathcal{L}(\mathsf{X\text{-}RCAPCFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}RAPCFA}(n))$,

(c) $\mathcal{L}(\mathsf{X\text{-}RCAPCFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}CAPCFA}(n))$,

(d) $\mathcal{L}(\mathsf{X\text{-}RAPCFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}APCFA}(n))$.

Now, we show that each nondeterministic asynchronous PCFA system can be simulated by a nondeterministic multi-head finite automaton.

Proposition 4.2. For all $n \geq 1$ and $\mathsf{X} \in \{1, 2\}$, $\mathcal{L}(\mathsf{X\text{-}APCFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}NFA}(n))$.

Proof:
Let $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ be a nondeterministic APCFA system with n finite automata $A_i = (Q_i, \Sigma, 1, \delta_i, \text¢, \$, q_i, F_i)$, $i = 1, \ldots, n$. We construct an n-head automaton $M = (Q, \Sigma, n, \delta, \text¢, \$, q_0, F)$ that simulates $\mathcal{A}$:

- $Q = Q_1 \times Q_2 \times \ldots \times Q_n$,
- $q_0 = (q_1, q_2, \ldots, q_n)$,
- $F = \{(p_1, p_2, \ldots, p_n) \in Q \mid p_i \in F_i \text{ for some } i \in \{1, \ldots, n\}\}$,
- $\delta((p_1, \ldots, p_n), a_1, \ldots, a_n) = \{((p'_1, \ldots, p'_n), d_1, \ldots, d_n) \mid \forall i = 1, \ldots, n :$

1) $(p_i', d_i) \in \delta_i(p_i, a_i)$ (local step of A_i)

or 2) $\exists j \in \{1, 2, \ldots, n\} \setminus \{i\} : p_i = \mathsf{req}^j_{\ell_i}, \quad p_j = \mathsf{res}^i_{\ell_j, c},$
$p_i' = \mathsf{rec}^j_{\ell_i, c}, \; p_j' = \mathsf{ack}^i_{\ell_j, c},$
$d_i = 0$ (incoming communication),

or 3) $\exists j \in \{1, 2, \ldots, n\} \setminus \{i\} : p_i = \mathsf{res}^j_{\ell_i, c}, \quad p_j = \mathsf{req}^i_{\ell_j},$
$p_i' = \mathsf{ack}^j_{\ell_i, c}, \; p_j' = \mathsf{rec}^i_{\ell_j, c},$
$d_i = 0$ (outgoing communication),

or 4) $p_i' = p_i, d_i = 0, p_i \notin F_i$, and conditions 1 - 3 do not hold $\}$.

Thus, system $\mathcal{A}$ performs a computation step

$$(x_1 p_1 a_1 y_1, \ldots, x_n p_n a_n y_n) \vdash_{\mathcal{A}} (x_1' p_1' y_1', \ldots, x_n' p_n' y_n')$$

if and only if the n-head finite automaton M simulates this step by the step

$$(x_1(p_1, \ldots, p_n) a_1 y_1, \ldots, x_n(p_1, \ldots, p_n) a_n y_n) \vdash_M (x_1'(p_1', \ldots, p_n') y_1', \ldots, x_n'(p_1', \ldots, p_n') y_n').$$

Whenever one component of $\mathcal{A}$ is in an accepting configuration, i.e. it is in a final state and cannot continue its computation, then M is also in a final state, halts, and thus accepts. On the other hand, if $\mathcal{A}$ cannot reach the accepting configuration (either no component reaches a final state or no component halts in a final state), then M can neither. Hence, $L(\mathcal{A}) = L(M)$. Moreover, if $\mathcal{A}$ consists of one-way (two-way) components, then M is a one-way (two-way) multi-head finite automaton. □

Further, we show how a multi-head finite automaton can be simulated by a centralized APCFA system working in returning mode. For this purpose, we modify the construction of the proof of Proposition 5.1 in [18], where PC systems of restarting automata are compared with multi-head finite automata.

Proposition 4.3. For all $n \geq 1$ and $\mathsf{X} \in \{1, 2\}$, $\mathcal{L}(\mathsf{X\text{-}NFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}RCAPCFA}(n))$.

Proof:
Let $M = (Q, \Sigma, n, \delta, ¢, \$, q_0, F)$ be an n-head finite automaton. A centralized APCFA system $\mathcal{A} = (A_1, \ldots, A_n)$ that simulates M in returning mode works as follows. Each client A_i, $2 \leq i \leq n$, nondeterministically chooses to send either its currently read input symbol or a guessed moving direction for its own head to the master A_1:

$$\delta_i(q_i, a) = \{(\mathsf{res}^1_a, 0), (\mathsf{res}^1_{-1}, -1), (\mathsf{res}^1_0, 0), (\mathsf{res}^1_1, 1)\},$$

for all $a \in \Sigma \cup \{¢, \$\}$. If M is a one-way multi-head finite automaton, then the transition $(\mathsf{res}^1_{-1}, -1) \in \delta_i(q_i, a)$ is omitted. Observe that, due to the returning mode, a client is reset into its initial state q_i after each communication. The master component A_1 requests the currently read input symbols from all clients, one after another, and stores them as local information of its communication states. Thereafter,

A_1 nondeterministically chooses a transition of M that is to be simulated, and then requests the guessed moving directions from the clients. Formally, we define, for all $(q, (d_1, \ldots, d_n)) \in \delta(p, (a_1, \ldots, a_n))$,

$$
\begin{aligned}
&\delta_1(p, a_1) &&= \{(\mathsf{req}^2_{\langle p\rangle}, 0)\},\\
&\delta_1(\mathsf{rec}^2_{\langle p\rangle, a_2}, a_1) &&= \{(\mathsf{req}^3_{\langle p, a_2\rangle}, 0)\},\\
&\delta_1(\mathsf{rec}^3_{\langle p, a_2\rangle, a_3}, a_1) &&= \{(\mathsf{req}^4_{\langle p, a_2, a_3\rangle}, 0)\},\\
&&&\vdots\\
&\delta_1(\mathsf{rec}^{n-1}_{\langle p, a_2, a_3, \ldots, a_{n-2}\rangle, a_{n-1}}, a_1) &&= \{(\mathsf{req}^n_{\langle p, a_2, \ldots, a_{n-1}\rangle}, 0)\},\\
&\delta_1(\mathsf{rec}^n_{\langle p, a_2, a_3, \ldots, a_{n-1}\rangle, a_n}, a_1) &&=\\
&\quad \{\, (\mathsf{req}^n_{\langle q, d_1, d_2, \ldots, d_n\rangle}, 0) \mid (q, (d_1, d_2, d_3, \ldots, d_n)) \in \delta(p, (a_1, \ldots, a_n))\,\},\\
&\delta_1(\mathsf{rec}^n_{\langle q, d_1, d_2, \ldots, d_n\rangle, d_n}, a_1) &&= \{(\mathsf{req}^{n-1}_{\langle q, d_1, d_2, \ldots, d_{n-1}\rangle}, 0)\},\\
&\delta_1(\mathsf{rec}^{n-1}_{\langle q, d_1, d_2, \ldots, d_{n-1}\rangle, d_{n-1}}, a_1) &&= \{(\mathsf{req}^{n-2}_{\langle q, d_1, d_2, \ldots, d_{n-2}\rangle}, 0)\},\\
&&&\vdots\\
&\delta_1(\mathsf{rec}^3_{\langle q, d_1, d_2, d_3\rangle, d_3}, a_1) &&= \{(\mathsf{req}^2_{\langle q, d_1, d_2\rangle}, 0)\},\\
&\delta_1(\mathsf{rec}^2_{\langle q, d_1, d_2\rangle, d_2}, a_1) &&= \{(q, d_1)\}.
\end{aligned}
$$

Only if the received guesses of all clients match the chosen transition of the multi-head finite automaton, A_1 can continue its computation. Whenever there is a wrong guess, i.e. a client sends a moving direction that differs from that one of the transition chosen by A_1, then A_1 is stuck and the system cannot accept the input. In addition, in all communications the first component is involved. Thus, the resulting system is centralized. Moreover, if M is a one-way (two-way) automaton, then system $\mathcal{A}$ consists of one-way (two-way) components. □

From the above results, we can conclude that each type of APCFA systems with n nondeterministic one-way (two-way) components has the same computational power as nondeterministic one-way (two-way) n-head finite automata.

Corollary 4.4. For all $n \geq 1$ and $\mathsf{X} \in \{1, 2\}$, $\mathcal{L}(\mathsf{X\text{-}RCAPCFA}(n)) = \mathcal{L}(\mathsf{X\text{-}RAPCFA}(n)) = \mathcal{L}(\mathsf{X\text{-}CAPCFA}(n)) = \mathcal{L}(\mathsf{X\text{-}APCFA}(n)) = \mathcal{L}(\mathsf{X\text{-}NFA}(n))$.

5. Computational power of deterministic APCFA systems

Corollary 4.1 and Proposition 4.2 can be immediately carried over to deterministic APCFA systems. Thus, we have:

Corollary 5.1. For all $n \geq 1$ and $\mathsf{X} \in \{1, 2\}$,

(a) $\mathcal{L}(\mathsf{X\text{-}DCAPCFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}DAPCFA}(n))$,

(b) $\mathcal{L}(\mathsf{X\text{-}DRCAPCFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}DRAPCFA}(n))$,

(c) $\mathcal{L}(\mathsf{X\text{-}DRCAPCFA}(n)) \subseteq \mathcal{L}(\mathsf{X\text{-}DCAPCFA}(n))$,

(d) $\mathcal{L}(\text{X-DRAPCFA}(n)) \subseteq \mathcal{L}(\text{X-DAPCFA}(n))$,

(e) $\mathcal{L}(\text{X-DAPCFA}(n)) \subseteq \mathcal{L}(\text{X-DFA}(n))$.

In the next proposition, we show that one-way deterministic centralized APCFA systems of degree n working in returning mode are at least as expressive as one-way deterministic n-head finite automata. The proof is similar to that of Theorem 12 in [16], where a multi-head pushdown automaton is simulated by an asynchronous PC system of pushdown automata.

Proposition 5.2. For all $n \geq 1$, $\mathcal{L}(\text{1-DFA}(n)) \subseteq \mathcal{L}(\text{1-DRCAPCFA}(n))$.

Proof:
We show that each deterministic one-way n-head finite automaton $M = (Q, \Sigma, n, \delta, ¢, \$, q_0, F)$ can be simulated by a deterministic and centralized APCFA system $\mathcal{A}$ with n one-way components working in returning mode. Basically, the first component of the system $\mathcal{A}$, i.e. the master A_1, requests the currently read symbols of the clients $A_2, \ldots, A_n$, one after another, and then simulates a transition of M. Whether a head of any client A_i, $i = 2, \ldots, n$, is moved one step to the right or keeps its position is controlled by A_1 just by requesting the next symbol from A_i or by avoiding this communication, respectively. The task of the clients is only to send their currently read symbol and move their heads one step to the right. Due to the returning mode, the clients are reset into their initial states and send the next read symbols after each communication with the master.

Formally, the transition mappings of the clients are defined by

$$\delta_i(q_i, a) = (\mathsf{res}^1_a, 1),$$

for each $i \in \{2, \ldots, n\}$ and each $a \in \Sigma \cup \{¢\}$. Whenever a client reaches the right end marker \$, it does not move its head:

$$\delta_i(q_i, \$) = (\mathsf{res}^1_\$, 0),$$

for each $i \in \{2, \ldots, n\}$.

The master A_1 has states of the form $\langle p, a_2, a_3, \ldots, a_n \rangle$, where p is a state of M and $a_2, \ldots, a_n$ are symbols of $\Sigma \cup \{*, ¢, \$\}$. Without loss of generality, we can assume that $* \notin \Sigma$. The initial state is $\langle q_0, *, \ldots, * \rangle$ and the set of final states is defined by $F_1 = \{\langle p, a_2, \ldots, a_n \rangle \mid p \in F, a_2, \ldots, a_n \in \Sigma \cup \{¢, \$\}\}$. Now, we distinguish two types of states:

$$Q_1^r = \{\langle p, a_2, a_3, \ldots, a_n \rangle \mid * \in \{a_2, \ldots, a_n\}\}$$

and

$$Q_1^s = \{\langle p, a_2, a_3, \ldots, a_n \rangle \mid * \notin \{a_2, \ldots, a_n\}\}.$$

States of Q_1^r imply that at least one symbol of a client is missing that must be requested by A_1 before a transition of M can be simulated. States of Q_1^s contain all necessary information for simulating a transition of M. Now, we define the transition mapping of A_1. For all states $\langle p, a_2, \ldots, a_{i-1}, *, \ldots, a_n \rangle \in Q_1^r$ with $* \notin \{a_2, \ldots, a_{i-1}\}$, A_1 behaves as follows:

$$\begin{aligned} \delta_1(\langle p, a_2, \ldots, a_{i-1}, *, \ldots, a_n \rangle, a_1) &= (\mathsf{req}^i_{\langle p, a_2, \ldots, a_{i-1}, *, \ldots, a_n \rangle}, 0), \\ \delta_1(\mathsf{rec}^i_{\langle p, a_2, \ldots, a_{i-1}, *, \ldots, a_n \rangle, a_i}, a_1) &= (\langle p, a_2, \ldots, a_{i-1}, a_i, \ldots, a_n \rangle, 0). \end{aligned}$$

If A_1 is in a state $\langle p, a_2, \ldots, a_n \rangle \in Q_1^s$, then a transition $\delta(p, a_1, \ldots, a_n) = (q, d_1, \ldots, d_n)$ of M is simulated by the following transition of A_1:

$$\delta_1(\langle p, a_2, \ldots, a_n \rangle, a_1) = (\langle q, b_2, \ldots, b_n \rangle, d_1),$$

where for all $i = 2, \ldots, n$, $b_i = a_i$ if $d_i = 0$ or $b_i = *$ if $d_i = 1$. Observe that only those input symbols of $a_2, \ldots, a_n$ are consumed for which the head of M is actually moved one step to the right. If a head of M is not moved to the right, i.e. $d_i = 0$, then the according symbol is kept by A_1. All consumed symbols are replaced by $*$ in order to request the next input symbols from the corresponding clients again.

If M is in state p reading $a_1, \ldots, a_n$ and no transition is applicable, then A_1 also halts in the state $\langle p, a_2, \ldots, a_n \rangle$ while reading a_1. Moreover, if $p \in F$, i.e. M accepts the input, then and only then $\langle p, a_2, \ldots, a_n \rangle$ is a final state of A_1. Thus, A_1 and therewith $\mathcal{A}$ accept if and only if M accepts the input. Hence, $L(\mathcal{A}) = L(M)$. □

We can conclude that all language classes that are characterized by some deterministic type of APCFA systems with one-way components coincide with $\mathcal{L}(\text{1-DFA})$.

Corollary 5.3. For all $n \geq 1$, $\mathcal{L}(\text{1-DRCAPCFA}(n)) = \mathcal{L}(\text{1-DCAPCFA}(n)) = \mathcal{L}(\text{1-DRAPCFA}(n)) = \mathcal{L}(\text{1-DAPCFA}(n)) = \mathcal{L}(\text{1-DFA}(n))$.

For two-way components, the construction in the proof of Proposition 5.2 does not work. Nevertheless, with an additional component, we can use a modified version of the *cyclic token method* from [2] in order to compare deterministic two-way APCFA systems with two-way multi-head finite automata. In this method, an information carrier - the token - is sent cyclically from one component to the next, where each component can get information from the token and store information in the token whenever it owns it.

Proposition 5.4. For all $n \geq 1$, $\mathcal{L}(\text{2-DFA}(n)) \subseteq \mathcal{L}(\text{2-DRAPCFA}(n+1))$.

Proof:
Let $M = (Q, \Sigma, n, \delta, ¢, \$, q_0, F)$ be a deterministic n-head finite automaton. Now, we describe, how M is simulated by a deterministic APCFA system $\mathcal{A} = (A_1, \ldots, A_{n+1})$ working in returning mode. Here, a token is a tuple $\langle p, a_2, d_2, \ldots, a_n, d_n \rangle$, where $p \in Q$, $a_2, \ldots, a_n \in \Sigma \cup \{¢, \$\}$, and $d_2, \ldots, d_n \in \{-1, 0, 1\}$.

For the simulation, an additional component A_{n+1} communicates with A_1 in each cycle of the token in order to inform A_1 of whether this is the first cycle or not (in the first cycle A_1 has to send the initial token to A_2; in all other cycles A_1 has to request the token from A_n). We set, for all $x \in \Sigma \cup \{\$\}$,

$$\delta_{n+1}(q_{n+1}, ¢) = (\text{res}^1_{\text{send\_token}}, 1),$$
$$\delta_{n+1}(q_{n+1}, x) = (\text{res}^1_{\text{request\_token}}, 0).$$

The message send_token is only sent once at the beginning of each computation. Thereafter, only the message request_token is sent. Observe that due to the returning mode, a component is reset into its initial state after sending information through a response.

The initial state of the first component A_1 is req^{n+1} to ask the additional component A_{n+1} whether it has to send the initial token to A_2, or whether it has to request the token from A_n. Then, for all $x \in \Sigma \cup \{¢, \$\}$,

$$\delta_1(\mathsf{rec}^{n+1}_{\mathsf{send\_token}}, ¢) = (\mathsf{res}^2_{\langle q_0, ¢, 0, \ldots, ¢, 0\rangle}, 0),$$
$$\delta_1(\mathsf{rec}^{n+1}_{\mathsf{request\_token}}, x) = (\mathsf{req}^n, 0).$$

The initial token $\langle q_0, ¢, 0, \ldots, ¢, 0\rangle$ contains the initial state of M, the current read input symbol of each client that are all initially ¢, and the initial moving directions that are all 0, since no transition is to be simulated right now.

If A_1 gets the token from A_n, then the token has completed a new cycle such that each client stored its current read input symbol within the token, and A_1 can simulate a computation step of M:

$$\delta_1(\mathsf{rec}^n_{\langle p, a_2, d_2, \ldots, a_n, d_n\rangle}, a_1) = (\mathsf{res}^2_{\langle q, a_2, d'_2, \ldots, a_n, d'_n\rangle}, d'_1)$$

if and only if

$$\delta(p, a_1, \ldots, a_n) = (q, d'_1, \ldots, d'_n).$$

Whenever M halts because of a non-defined transition, then and only then A_1 halts in the corresponding receive state due to the same reason. Moreover, the set of final states of A_1 is $F_1 = \{\mathsf{rec}^n_{\langle p, a_2, d_2, \ldots, a_n, d_n\rangle} \mid p \in F, a_2, \ldots, a_n \in \Sigma \cup \{¢, \$\}, d_2, \ldots, d_n \in \{-1, 0, 1\}\}$. Thus, A_1 halts in an accepting state if and only if M does so.

The main task of the clients of $\mathcal{A}$ is to get the token, to read out their moving directions, process the move of the head, store the currently read symbol within the token, and send the token to the successor component. Thus, we set, for all $i = 2, \ldots, n-1$,

$$\begin{aligned}
&\delta_i(q_i, x) && = (\mathsf{req}^{i-1}, 0) \quad \text{(for all } x \in \Sigma \cup \{¢, \$\}),\\
&\delta_i(\mathsf{rec}^{i-1}_{\langle p, a_2, d_2, \ldots, a_n, d_n\rangle}, a_i) && = (\langle p, a_2, d_2, \ldots, a_n, d_n\rangle, d_i),\\
&\delta_i(\langle p, a_2, d_2, \ldots, a_n, d_n\rangle, b_i) && = (\mathsf{res}^{i+1}_{\langle p, a_2, d_2, \ldots, a_{i-1}, d_{i-1}, b_i, d_i, a_{i+1}, d_{i+1}, \ldots, a_n, d_n\rangle}, 0).
\end{aligned}$$

Component A_n that simulates the last head of M sends the token to the first component:

$$\begin{aligned}
&\delta_n(q_n, x) && = (\mathsf{req}^{n-1}, 0) \quad \text{(for all } x \in \Sigma \cup \{¢, \$\}),\\
&\delta_n(\mathsf{rec}^{n-1}_{\langle p, a_2, d_2, \ldots, a_n, d_n\rangle}, a_n) && = (\langle p, a_2, d_2, \ldots, a_n, d_n\rangle, d_n),\\
&\delta_n(\langle p, a_2, d_2, \ldots, a_n, d_n\rangle, b_n) && = (\mathsf{res}^1_{\langle p, a_2, d_2, \ldots, a_{n-1}, d_{n-1}, b_n, d_n\rangle}, 0).
\end{aligned}$$

In addition, we set $F_i = \emptyset$, for all $i = 2, \ldots, n+1$, such that $\mathcal{A}$ accepts if and only if A_1 accepts. Since A_1 accepts if and only if M accepts, it follows $L(\mathcal{A}) = L(M)$. □

In the proof of Proposition 5.4, a deterministic two-way multi-head finite automaton is simulated by a deterministic non-centralized APCFA system with one additional component working in returning mode. In our next proposition, we show that each deterministic two-way multi-head finite automaton can also be simulated by a deterministic centralized APCFA system working in non-returning mode.

Proposition 5.5. For all $n \geq 1$, $\mathcal{L}(\mathsf{2\text{-}DFA}(n)) \subseteq \mathcal{L}(\mathsf{2\text{-}DCAPCFA}(n))$.

Proof:
Let $M = (Q, \Sigma, n, \delta, ¢, \$, q_0, F)$ be a deterministic two-way n-head finite automaton. A deterministic two-way APCFA system $\mathcal{A} = (A_1, \ldots, A_n)$ that simulates M in non-returning mode is defined similar to that in the proof of Proposition 4.3. Here, the clients ask for the moving directions of their heads instead of guessing it:

$$\begin{aligned}\delta_i(q_i, a) &= (\mathsf{res}_a^1, 0),\\ \delta_i(\mathsf{ack}_a^1, a) &= (\mathsf{req}^1, 0),\\ \delta_i(\mathsf{rec}_d^1, a) &= (q_i, d),\end{aligned}$$

for all $a \in \Sigma \cup \{¢, \$\}$, $d \in \{-1, 0, 1\}$, and $i = 2, \ldots, n$.

The master A_1 successively requests the current read symbols from all clients and stores them together with the current state of M as local information in its communication states. Thereafter, A_1 determines the transition of M that is to be simulated and then sends the moving directions to all clients. Thus, we define for each transition $\delta(p, (a_1, \ldots, a_n)) = (q, (d_1, \ldots, d_n))$ of M:

$$\begin{aligned}
&\delta_1(p, a_1) && = (\mathsf{req}^2_{\langle p\rangle}, 0),\\
&\delta_1(\mathsf{rec}^2_{\langle p\rangle, a_2}, a_1) && = (\mathsf{req}^3_{\langle p, a_2\rangle}, 0),\\
&\delta_1(\mathsf{rec}^3_{\langle p, a_2\rangle, a_3}, a_1) && = (\mathsf{req}^4_{\langle p, a_2, a_3\rangle}, 0),\\
& && \vdots\\
&\delta_1(\mathsf{rec}^{n-1}_{\langle p, a_2, a_3, \ldots, a_{n-2}\rangle, a_{n-1}}, a_1) && = (\mathsf{req}^n_{\langle p, a_2, \ldots, a_{n-1}\rangle}, 0),\\
&\delta_1(\mathsf{rec}^n_{\langle p, a_2, a_3, \ldots, a_{n-1}\rangle, a_n}, a_1) && = (\mathsf{res}^n_{\langle q, d_1, d_2, \ldots, d_{n-1}\rangle, d_n}, 0),\\
&\delta_1(\mathsf{ack}^n_{\langle q, d_1, d_2, \ldots, d_{n-1}\rangle, d_n}, a_1) && = (\mathsf{res}^{n-1}_{\langle q, d_1, d_2, \ldots, d_{n-2}\rangle, d_{n-1}}, 0),\\
& && \vdots\\
&\delta_1(\mathsf{ack}^3_{\langle q, d_1, d_2\rangle, d_3}, a_1) && = (\mathsf{res}^2_{\langle q, d_1\rangle, d_2}, 0),\\
&\delta_1(\mathsf{ack}^2_{\langle q, d_1\rangle, d_2}, a_1) && = (q, d_1).
\end{aligned}$$

Then, $(x_1 p a_1 y_1, \ldots, x_n p a_n y_n) \vdash_M^* (x'_1 q y'_1, \ldots, x'_n q y'_n)$ if and only if A_1 reaches the configuration $(x'_1 q y'_1)$ from the configuration $(x_1 p a_1 y_1)$ in the system $\mathcal{A}$. Together with the definition $F_1 = F$ and $F_i = \emptyset$, for all $i = 2, \ldots, n$, it follows that $L(\mathcal{A}) = L(M)$. □

This leads to the following result:

Corollary 5.6. For all $n \geq 1$, $\mathcal{L}(\text{2-DCAPCFA}(n)) = \mathcal{L}(\text{2-DAPCFA}(n)) = \mathcal{L}(\text{2-DFA}(n))$.

6. Comparison between ASFA, PCFA systems and APCFA systems

In this section, we compare the language classes that are characterized by synchronous PCFA systems, asynchronous PCFA systems with blocking communication, and asynchronous systems of finite automata (ASFA) with non-blocking communication. Due to the fact that the working modes of these systems are quite different, we use the correlations between the language classes of these systems and those of multi-head finite automata. For synchronous PCFA systems the connections to one-way multi-head finite automata were shown in [2, 3, 14], and for APCFA systems we take the results from Sections

4 and 5. For ASFA, we show the equivalence with deterministic two-way multi-head finite automata now.

An asynchronous system of finite automata (ASFA, [10, 12, 13]) is a finite collection of deterministic two-way finite automata: $\mathcal{A} = (A_1, \ldots, A_n)$. The automata work in parallel and independently of each other except for the communication that is realized by message passing. Each component has a message buffer of constant size for each other component. We denote the buffer of component A_i for incoming messages from component A_j by $B_{i,j}$. Thus, A_i has $n-1$ buffers $B_{i,1}$, ..., $B_{i,i-1}$, $B_{i,i+1}$, ..., $B_{i,n}$. When A_j wants to send a message to A_i, it just writes the message into the buffer $B_{i,j}$ of A_i. The behaviour of an automaton A_i is defined by function $\delta_i : Q_i \times (\Delta \cup \{\perp\})^{n-1} \times \Sigma \to Q_i \times (\Delta \cup \{\perp\})^{n-1} \times \{L, R, N\}$, where Q_i is the set of states of A_i, Δ is a message alphabet, and Σ is the input alphabet. The special symbol $\perp$ means that no message was received (the buffer is empty) or no message is sent. The symbols L, R, and N denote the possible moving directions left, right, and no move of the head. In detail, $\delta_i(q, m_1, \ldots, m_{i-1}, m_{i+1}, \ldots, m_n, a) = (q', m'_1, \ldots, m'_{i-1}, m'_{i+1}, \ldots, m'_n, d)$ means that if A_i is currently in state q and reads the messages m_1, ..., m_{i-1}, m_{i+1}, ..., m_n from its buffers and symbol a from its tape, then it changes into state q', sends messages m'_1, ..., m'_{i-1}, m'_{i+1}, ..., m'_n, and moves its head to the left if $d = L$, to the right if $d = R$, or keeps its position if $d = N$. Whenever a message from the buffer is processed, it is removed at the same time. Since the message buffers are of constant size, it must be determined what happens if a buffer is full. In this case, one can imagine either to refuse new incoming messages or to shift the messages such that the oldest one is dropped. In both cases, neither the sending nor the receiving component realizes the loss of the according message. We will take the former alternative in what follows.

It is demanded that each computation of such a system terminates [12]. This happens if a fixed component (let us say the first one) reaches an accepting or rejecting state that is connected with sending a terminating message that stops all components and includes the information 'accept' or 'reject'. Since there is no global clock that forces the components to work stepwise in parallel, there can occur different communication patterns for the same input. So it is in general not unique which message is sent before another one, and different sequences can appear in a message buffer for different computations on the same input. To get nevertheless a correct answer (accept or reject) of the system independently of the used communication pattern, we say that word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if, for all communication patterns, A_1 enters an accepting state. On the other hand, w is rejected by $\mathcal{A}$ if, for all communication patterns, A_1 enters a rejecting state. This implies that the system is constructed in a way such that, for an input w, A_1 cannot reach an accepting state through one communication pattern and a rejecting state through another communication pattern. The language that is accepted by a system $\mathcal{A}$ consists of all words that are accepted by $\mathcal{A}$ and is denoted by $L(\mathcal{A})$. The language class that is characterized by ASFA is denoted by $\mathcal{L}(\mathsf{ASFA})$. The set $\mathcal{L}(\mathsf{ASFA}(n))$ consists of all languages that can be accepted by ASFA with n components.

Now we show that the language class characterized by ASFA equals the language class of deterministic two-way multi-head finite automata.

Proposition 6.1. $\mathcal{L}(\mathsf{ASFA}(n)) \subseteq \mathcal{L}(\mathsf{2\text{-}DFA}(n))$, for all $n \geq 1$.

Proof:
Let $\mathcal{A}$ be an ASFA of degree n. Then an n-head finite automaton M simulates $\mathcal{A}$ as follows: M stores the current states of all components and the content of their message buffers in its internal control. This

can be done, since there exist $n-1$ message buffers of constant size for each component. Now, M cyclically simulates one computation step of each component using the i-th head to read the current symbol of the i-th component. Due to the definition, $\mathcal{A}$ must terminate. For this, the first component sends an according message that includes also whether $\mathcal{A}$ accepts or rejects. Therefore, M must reach this message, too. If $\mathcal{A}$ accepts, then M enters a final state and accepts. Otherwise, M enters a non-final state, gets stuck, and rejects the input. The order of the simulated components does not influence the accepting behaviour, because $\mathcal{A}$ accepts or rejects independently of the communication pattern. □

Proposition 6.2. $\mathcal{L}(\text{2-DFA}(n)) \subseteq \mathcal{L}(\text{ASFA}(2n))$, for all $n \geq 1$.

Proof:
Let M be a deterministic n-head two-way finite automaton. From M, a deterministic $2n$-head two-way finite automaton M' can be constructed such that $L(M) = L(M')$ and M' halts for all inputs. M' mainly simulates M with the first n heads. The additional n heads are used to count the number of computation steps of M. When the number of computation steps of M equals the number of different configurations for the given input, then M' halts and rejects, since M is in a computation loop.

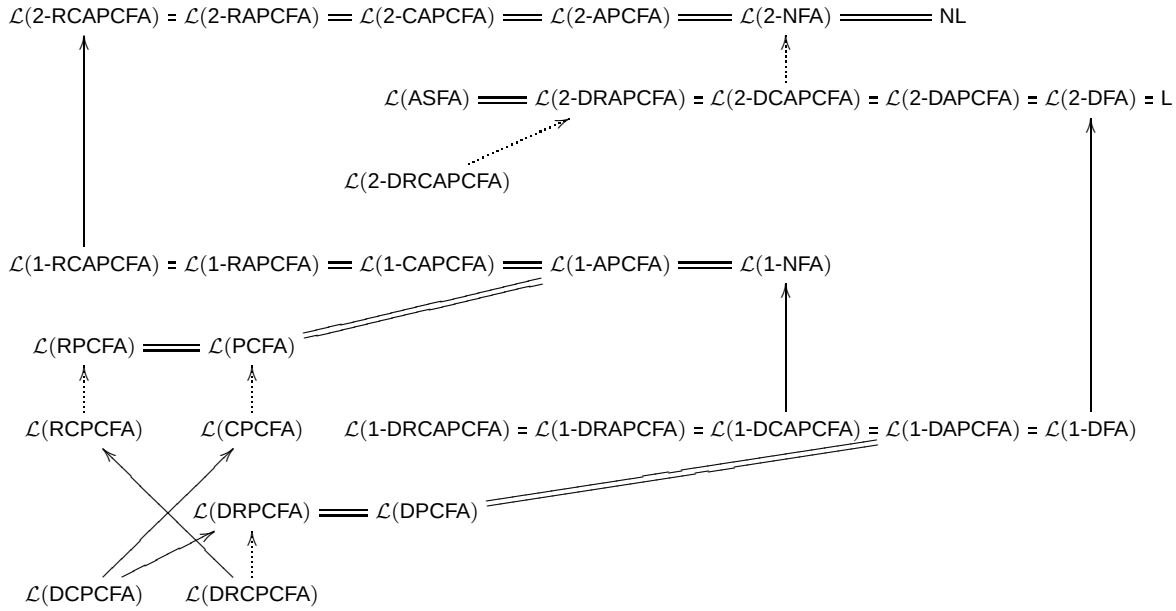
To be more precise, M' counts by explicitely moving the second n heads: the $(n+1)$-th head is stepwise moved from the ¢-symbol at the left border of the tape to the \$-symbol at the right border of the tape. With an additional step, these are $k+2$ steps for an input of length k. Now, the $(n+2)$-th head is moved one step to the right and the $(n+1)$-th head starts to move stepwise back to the left tape border. Reaching it, the $(n+2)$-th head is again moved one position to the right, and so forth. This is generalized such that whenever head $n+i$ had crossed the whole tape, then head $n+i+1$ is moved one step. Thus, when the $2n$-th head reaches the right border of the tape, then M' has counted to $(k+2)^n$, that is the number of all different positions where n heads can be placed at. With additional states the counted number can be increased by a factor such that M' counts to the number of all different configurations of M with an input word of length k. In addition, this procedure is deterministic.

We construct a system $\mathcal{A}$ of degree $2n$ that simulates M'. Basically, the first component A_1 of $\mathcal{A}$ simulates M', and the i-th head of M' is simulated by the i-th component A_i of $\mathcal{A}$, for $i = 2, \ldots, 2n$. The components are synchronized by the fact that a local computation step is only possible if according messages are read from the message buffers. If a message is missing and the corresponding message buffer is empty, then a component cannot perform a local computation step and, thus, waits.

In more detail, $\mathcal{A}$ simulates a computation step of M' as follows: A_1 writes the message 'symbol' into the buffers $B_{2,1}$, $B_{3,1}$, …, $B_{2n,1}$ of A_2, …, A_{2n} and awaits an answer from all these components. For this, a local computation step of A_1 can only be executed if it reads messages from all its own buffers $B_{1,2}$, $B_{1,3}$, …, $B_{1,2n}$ containing the current symbols of A_2, …, A_{2n}. Reading the message 'symbol', these components just put their currently read input symbols into the message buffers $B_{1,2}$, $B_{1,3}$, …, $B_{1,2n}$ of A_1 and await the next message from A_1. If all the buffers of A_1 are filled with the symbols that are currently read by $A_2, \ldots, A_{2n}$, then it simulates the corresponding transition of M'. It stores the successor state of M' in its finite control, and sends the moving directions of all heads to the components, i.e., it writes 'left', 'right', or 'no move' into the buffers $B_{2,1}$, $B_{3,1}$, …, $B_{2n,1}$. Thereafter, $A_2, \ldots, A_{2n}$ move their heads accordingly and await the message 'symbol' of A_1 again. This proceeds until M' halts. If M' halts in an accepting state, then A_1 sends the terminating message with the information 'accept'. If M' halts in a non-accepting state, then A_1 sends the terminating message with the information 'reject'.

Since each computation of M' terminates, each computation of $\mathcal{A}$ terminates, too. It can be easily verified that $\mathcal{A}$ accepts if and only if M' accepts. Hence, $L(M) = L(M') = L(\mathcal{A})$. □

The following figure depicts the connections between the considered language classes, where an arrow denotes a proper inclusion, and a dotted arrow denotes an inclusion that is not yet known to be proper. Additionally, the classes L and NL of all languages which can be accepted by a deterministic or nondeterministic logarithmically space bounded Turing machine, respectively, are included in the comparison, since they are directly connected with multi-head finite automata [7]. The relations between L, NL, the language classes which are characterized by synchronous PCFA systems, by one-way multi-head finite automata, and by two-way multi-head finite automata were shown in the cited literature.



7. Communication complexity

In this section, we investigate the communication complexity of APCFA systems. For this, we consider upper bounds of the number of communications that a component is allowed to execute during a computation. With $\mathcal{L}_c(\mathsf{X})$, $c \in \mathbb{N}$, we denote the class of languages that can be accepted by an APCFA system of type X such that each component is allowed to perform at most c communications. With $\mathcal{L}_\infty(\mathsf{X})$, we denote the class of languages that can be accepted by APCFA systems of type X such that the components can perform arbitrarily many communications. Obviously, $\mathcal{L}_\infty(\mathsf{X}) = \mathcal{L}(\mathsf{X})$ for all types X of APCFA systems. Thus, the computational power of multi-head finite automata is an upper bound for APCFA systems with an arbitrary amount of communications.

Clearly, an APCFA system that accepts without any communication can be easily simulated by a nondeterministic (two-way) finite automaton that simply nondeterministically choses and simulates the accepting component of the system. This is in contrast to synchronous PCFA systems. For instance, from [14] we know that the non-contextfree language $\{a^n b^n c^n \mid n \geq 1\}$ is accepted by a synchronous PCFA system of degree three without any communication.

Now, we prove that also systems with at most constantly many communications can be simulated by two-way finite automata, too.

Proposition 7.1. $\mathcal{L}_k(\mathsf{X})$ is equal to the set of all regular languages, for all $k \in \mathbb{N}$ and all types X of one-way and two-way APCFA systems.

Proof:
Let $\mathcal{A} = (A_1, \ldots, A_n)$ be an arbitrary APCFA system, where each component can perform at most k communications for a fixed constant $k \in \mathbb{N}$. We construct a nondeterministic two-way finite automaton M that simulates $\mathcal{A}$. The states of M have the form (q, T), where $T = \langle S_1, \ldots, S_n \rangle$ is an n-tuple that contains, for each component A_i, one queue S_i of length at most $k+1$. The queues can contain receive states, acknowledge states, or the entry 'accept'. An entry can only be added at the end of the queue or removed from the beginning of the queue. Observe that there are only constantly many different tuples (constantly many queues of constant length with constantly many different entries).

The initial state of M is (q_1, T), where q_1 is the initial state of A_1 and T contains only empty queues. Automaton M behaves as follows: it starts with simulating A_1 and stores the current state of A_1 within its finite control. When A_1 reaches a request state, then M choses nondeterministically a possible answer from the set of the corresponding response states of the communication partner (these are only constantly many), stores the corresponding receive state at the end of the first queue S_1, and continues the simulation of A_1. If A_1 reaches a response state, then M adds the corresponding acknowledge state to S_1 and continues the simulation. At any time during the simulation of A_1, M can nondeterministically chose to finish this simulation and to start to simulate the next component A_2 or to continue the simulation of A_1. When A_1 reaches an accepting configuration, then M stores 'accept' at the end of S_1, and starts immediately to simulate the next component A_2. For this, M moves its head to the left end marker ¢ and changes into the state (q_2, T), where q_2 is the initial state of A_2 and T is the current n-tuple.

This proceeds until each component is simulated. Thereafter, M starts to resolve the communications that are stored within the tuple T. Whenever two queues start with two corresponding communication states (receive and acknowledge state), these states are removed. This is realized just by a state change. It is not important, in which order the communications are resolved (at least if only the first entries of the queues are considered). M resolves communications until either a queue contains only 'accept', or there are no corresponding communication states left (and no queue contains only 'accept'). In the former case, M changes into a final state, halts, and accepts. In the latter case, M is stuck and rejects the input.

Finally, M accepts the input if and only if there is a queue that contains only 'accept'. This means that all necessary communications are resolved and at least one component has reached a final configuration. Observe that not all guessed communications must be correct but at least those that are necessary to reach an accepting configuration. In addition, if none of the components of $\mathcal{A}$ can reach an accepting configuration, then M cannot accept, either. Hence, $L(M) = L(\mathcal{A})$. Moreover, $L(\mathcal{A})$ is regular, since two-way finite automata are known to accept exactly the set of regular languages [8]. □

We have seen that an APCFA system needs more than constantly many communications to accept non-regular languages. On the other hand, at most $\mathcal{O}(m)$ and $\mathcal{O}(m^n)$ communications are needed for the simulation of an accepting computation of a one-way or two-way n-head finite automaton, respectively, with input of length m. This follows from the simulations of Propositions 4.3, 5.2, and 5.4 and the fact that the length of a shortest accepting computation of a one-way (two-way) multi-head finite automaton is in $\mathcal{O}(m)$ ($\mathcal{O}(m^n)$). Thus, for each APCFA system we can construct an equivalent system with the above upper bounds of communications.

Corollary 7.2. For each one-way (two-way) APCFA system $\mathcal{A}$, one can effectively construct a one-way (two-way) APCFA system $\mathcal{A}'$ such that $L(\mathcal{A}) = L(\mathcal{A}')$, and each component performs at most $\mathcal{O}(m)$ ($\mathcal{O}(m^n)$) communications for any input of length m and number of components n.

Now we have a look at systems whose ammount of communication lies between constant and linear in the one-way case and between constant and polynomially in the two-way case. In [10] it is shown that there exists a gap between constant and linear for ASFA, i.e. there does not exist any ASFA with communication complexity $\omega(1)$ and $o(n)$. We follow the proof idea and show the same result for deterministic one-way and two-way APCFA systems. With $c(\mathcal{A}, w)$ we denote the number of communications that $\mathcal{A}$ needs to accept w.

Proposition 7.3. Let L be a non-regular language that is accepted by a deterministic APCFA system $\mathcal{A}$. Then $c(\mathcal{A}, w) \in \Omega(|w|)$ for infinitely many $w \in L$.

Proof:
Let Σ be an alphabet, $L \in \mathcal{L}(\mathsf{2\text{-}DAPCFA})$ a non-regular language over Σ, and $\mathcal{A}$ a deterministic APCFA system with $L(\mathcal{A}) = L$. Due to Proposition 7.1, for each $m \in \mathbb{N}$, there exists a word $w \in L$ such that $c(\mathcal{A}, w) > m$. We take a shortest such w, consider the accepting computation of $\mathcal{A}$ on w and factorize w into $w = u_1 s_1 u_2 s_2 \dots u_r s_r u_{r+1}$, where $s_i \in \Sigma$, $i = 1, \dots, r$, are symbols on which some component of $\mathcal{A}$ enters a request or response state, and $u_j \in \Sigma^*$, $j = 1, \dots, r+1$, are subwords of w, where no component enters a request or response state. Then $\frac{1}{2}|s_1 s_2 \dots s_r| \leq c(\mathcal{A}, w)$, since one communication consists of a request and a response that can in general be reached on different symbols. Now, we show that $|u_1 u_2 \dots u_{r+1}| \in \mathcal{O}(1)$, which implies that $|s_1 s_2 \dots s_r| \in \Omega(|w|)$ and, thus, $c(\mathcal{A}, w) \in \Omega(|w|)$.

The factors u_j, $j = 1, \dots, r+1$, are exactly those parts of w, where no component enters a request or response state. Moreover, we can assume that also receive and acknowledge states cannot be reached through a local computation. In other words, all components perform only local computation steps without any communication on these factors. If u_j is longer than a constant t, then there exists a factor x of u_j such that, for each component, the crossing sequences entering x from the left and leaving x to the right are equal. This constant t only depends on the numbers of different crossing sequences of the components, and these depend only on the number of (non-communication) states of the components. The factor x can be cut from u_j such that we get a shorter word w' that is also accepted by $\mathcal{A}$ with the same number of communications, i.e. $c(\mathcal{A}, w') = c(\mathcal{A}, w)$. But this is a contradiction to our assumption that w is a shortest word with $c(\mathcal{A}, w) > m$.

It follows that $|s_1 s_2 \dots s_r| \in \Omega(|w|)$ and thus $c(\mathcal{A}, w) \in \Omega(|w|)$, which completes the proof. □

Observe that the proof of Proposition 7.3 is not sufficient for nondeterministic APCFA systems, because in general there may exist another computation of the shortened word w' such that $c(\mathcal{A}, w') \leq m$. It is still open whether this gap of communication complexity also holds for nondeterministic systems.

For the sake of completeness, it is mentioned that in [11] the authors consider synchronous systems of finite automata which differ from the PCFA systems we considered here. They show several results on communication complexity for these systems. For example, they prove a hierarchy for a constant number of messages that is in contrast to the asynchronous systems.

8. Conclusion

We introduced asynchronous systems of parallel communicating one-way and two-way finite automata with blocking communication and compared them with synchronous systems of parallel communicating finite automata, multi-head finite automata, and asynchronous systems of finite automata with non-blocking communication. We have shown that all language classes characterized by any type of asynchronous PCFA systems except for deterministic centralized two-way APCFA systems working in returning mode coincide with the corresponding language classes of multi-head finite automata.

For the language class of deterministic two-way APCFA systems working in returning mode, we could not find a precise characterization in terms of multi-head finite automata. So, it is an open question whether the inclusion $\mathcal{L}(\text{2-DRCAPCFA}) \subseteq \mathcal{L}(\text{2-DFA})$ is proper or not. Moreover, it would be interesting to know whether there exists an inclusion between $\mathcal{L}(\text{2-DRCAPCFA})$ and $\mathcal{L}(\text{1-NFA})$.

The close relationship between APCFA systems and multi-head finite automata shows how the processing of a multi-head finite automaton can be distributed with very loosely connected and simple components and a very simply structured communication protocol. The number of communications is strictly connected to the computational power of APCFA systems. With at most constantly many communications, APCFA systems can only accept regular languages. On the other hand, linearly many and polynomially many communications (with respect to the length of the input) are enough for one-way and two-way systems, respectively. Moreover, there is no language that would require a deterministic one-way or two-way APCFA system that has communication complexity between constant and linear. Whether this holds also for nondeterministic systems is an open problem as well as the question of whether there exist languages such that two-way systems really need polynomially many communications. Is there a hierarchy according to the number of communications between constant and polynomially?

References

[1] Bordihn, H., Kutrib, M., Malcher, A.: Undecidability and Hierarchy Results for Parallel Communicating Finite Automata, *International Journal of Foundations of Computer Science*, **22**(7), 2011, 1577–1592.

[2] Bordihn, H., Kutrib, M., Malcher, A.: On the Computational Capacity of Parallel Communicating Finite Automata, *International Journal of Foundations of Computer Science*, **23**(03), 2012, 713–732.

[3] Choudhary, A., Krithivasan, K., Mitrana, V.: Returning and non-returning parallel communicating finite automata are equivalent, *Informatique Théorique et Applications*, **41**(2), 2007, 137–145.

[4] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach Science Publishers, Inc., Newark, NJ, USA, 1994.

[5] Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V., Vaszil, G.: Parallel Communicating Pushdown Automata Systems, *International Journal of Foundations of Computer Science*, **11**(4), 2000, 631–650.

[6] Czeizler, E., Czeizler, E.: Parallel Communicating Watson-Crick Automata Systems, *Automata and Formal Languages, 11th International Conference, AFL 2005, Dobogókő, Hungary, May 17-20* (Z. Ésik, Z. Fülöp, Eds.), Institute of Informatics, University of Szeged, 2005.

[7] Hartmanis, J.: On Non-Determinancy in Simple Computing Devices, *Acta Informatica*, **1**(4), 1972, 336–344.

[8] Hopcroft, J. E., Ullman, J. D.: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[9] Ibarra, O. H.: On Two-Way Multihead Automata, *Journal of Computer and System Sciences*, **7**(1), 1973, 28–36.

[10] Jurdziński, T.: *Communication Aspects of Computation of Systems of Finite Automata*, Ph.D. Thesis, Institute of Computer Science, University of Wrozław, Wrocław, 2000.

[11] Jurdziński, T., Kutyłowski, M., Lorýs, K.: Multi-party Finite Computations, in: *Computing and Combinatorics* (T. Asano, H. Imai, D. Lee, S.-i. Nakano, T. Tokuyama, Eds.), vol. 1627 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 1999, 318–329.

[12] Jurdziński, T., Kutyłowski, M., Zatopiański, J.: Communication Complexity for Asynchronous Systems of Finite Devices, *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01), Los Alamitos, CA, April 23-27, 2001*, IEEE Computer Society, 2001.

[13] Jurdziński, T., Kutyłowski, M., Zatopiański, J.: Efficient Simulation of Synchronous Systems by Multi-speed Systems, *ITA*, **39**(2), 2005, 403–419.

[14] Martín-Vide, C., Mateescu, A., Mitrana, V.: Parallel Finite Automata Systems Communicating by States, *International Journal of Foundations of Computer Science*, **13**(5), 2002, 733–749.

[15] Martín-Vide, C., Mitrana, V.: Some undecidable problems for parallel communicating finite automata systems, *Information Processing Letters*, **77**(5-6), 2001, 239–245.

[16] Otto, F.: Asynchronous PC Systems of Pushdown Automata, in: *Language and Automata Theory and Applications* (A.-H. Dediu, C. Martín-Vide, B. Truthe, Eds.), vol. 7810 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, Berlin, 2013, 456–467.

[17] Păun, G., Santean, L.: Parallel Communicating Grammar Systems: The Regular Case, *Analele Universitatii din Bucuresti, Seria matematica-informatica*, **2**, 1989, 55–63.

[18] Vollweiler, M., Otto, F.: Systems of Parallel Communicating Restarting Automata, *4th Workshop on Non-Classical Models for Automata and Applications* (R. Freund, M. Holzer, B. Truthe, U. Ultes-Nitsche, Eds.), Österreichische Computer Gesellschaft, Fribourg, Switzerland, 2012.