Measuring Communication in Parallel Communicating Finite Automata

Henning Bordihn
Institut für Informatik, Universität Potsdam,
August-Bebel-Str. 89, 14482 Potsdam, Germany
henning@cs.uni-potsdam.de

Martin Kutrib and Andreas Malcher
Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,malcher}@informatik.uni-giessen.de

Systems of deterministic finite automata communicating by sending their states upon request are investigated, when the amount of communication is restricted. The computational power and decidability properties are studied for the case of returning centralized systems, when the number of necessary communications during the computations of the system is bounded by a function depending on the length of the input. It is proved that an infinite hierarchy of language families exists, depending on the number of messages sent during their most economical recognitions. Moreover, several properties are shown to be not semi-decidable for the systems under consideration.

1 Introduction

Communication is one of the most fundamental concepts in computer science: objects of object-oriented programs, roles or pools in business processes, concurrent processes in computer networks or in information or operating systems are examples of communicating agents.

Parallel communicating finite automata systems (PCFA) have been introduced in [12] as a simple automaton model of parallel processes and cooperating systems, see also [1, 2, 4]. A PCFA consists of several finite automata, the components of the system, that process a joint input string independently of each other. However, their transitions are synchronized according to a global clock. The cooperation of the components is enabled by communication steps in which components can request the state reached by another component. The system can work in returning or non-returning mode. In the former case each automaton which sends its current state is set back to its initial state after this communication step. In the latter case the state of the sending automaton is not changed. Recently, these communication protocols have been refined in [15] and further investigated for the case of parallel communicating systems of pushdown automata [14]. There, the communication process is performed in an asynchronous manner, reflecting the technical features of many real communication processes. In the sequel of this paper and as a first step towards an investigation of the influence of restricted communication to parallel communicating systems of automata, we stick with the simpler model having synchronized communication steps.

In a PCFA, one also distinguishes between centralized systems where only one designated automaton, called master, can request information from other automata, and non-centralized systems where every automaton is allowed to request information from others. Taking the distinction between returning and non-returning systems into account, we are led to four different working modes. Moreover, one

Z. Ésik and Z. Fülöp (Eds.): Automata and Formal Languages 2014 (AFL 2014)
EPTCS 151, 2014, pp. 124–138, doi:10.4204/EPTCS.151.8

distinguishes between deterministic and nondeterministic PCFA. The system is deterministic, if all its components are deterministic finite automata.

It is known from [2, 4, 12] that deterministic (nondeterministic) non-centralized PCFA are equally powerful as deterministic (nondeterministic) one-way multi-head finite automata [6], both in returning and non-returning working modes. Moreover, it is proved in [2] that nondeterminism is strictly more powerful than determinism for all the four working modes, and that deterministic centralized returning systems are not weaker than deterministic centralized non-returning ones.

All variants of PCFA accept non-regular languages due to the feature that communication between the components of the system is allowed. Thus it is of interest to measure the amount of communication needed for accepting those languages. Mitrana proposed in [13] a dynamical measure of descriptional complexity as follows: The degree of communication of a PCFA for a given word is the minimal number of communications necessary to recognize the word. Then, the degree of communication of a PCFA is the supremum of the degrees of communication taken over all words recognized by the system, while the degree of communication of a language (with respect to a PCFA of type X) is the infimum of the degrees of communication taken over all PCFA of type X that accept the language. Mitrana proved that this measure cannot be algorithmically computed for languages accepted by nondeterministic centralized or non-centralized non-returning PCFA. The computability status of the degree of communication for the other types of PCFA languages as well as for all types of PCFA is stated as open question in [13].

In this paper, we study PCFA where the degree of communication is bounded by a function in the length of the input word. We restrict ourselves to one of the simplest types of PCFA, namely to deterministic centralized returning systems of finite automata. In the next section, the basic definitions and two examples of languages accepted by communication bounded PCFA are presented. In Section 3, we show that bounding the degree of communication by logarithmic, square root or linear functions leads to three different families of languages. For the strictness results, we use similar witness languages and a proof technique based on Kolmogorov complexity as in [9], where the second and the third author investigated the computational power of two-party Watson-Crick systems, that is, synchronous systems consisting of two finite automata running in opposite directions on a shared read-only input and communicating by broadcasting messages.

In Section 4, non-semi-decidability results are proved for deterministic returning centralized PCFA and their languages, thus partially answering questions listed as open in [13]. Similarly to [1] the proofs rely on properties of one-way cellular automata and their valid computations. Finally, Section 5 refines the three-level hierarchy from Section 3 to an infinite hierarchy.

2 Preliminaries and Definitions

We write Σ^* for the set of all words over the finite alphabet Σ, and $\mathbb{N}$ for the set $\{0,1,2,\dots\}$ of non-negative integers. The *empty word* is denoted by λ. For the *length* of w we write $|w|$. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*.

Next we turn to the definition of parallel communicating finite automata systems. The nondeterministic model has been introduced in [12]. Following [1], the formal definition is as follows.

A *deterministic parallel communicating finite automata system of degree k* (DPCFA(k)) is a construct $A = \langle \Sigma, A_1, A_2, \dots, A_k, Q, \triangleleft \rangle$, where

1. Σ is the set of *input symbols*,
2. each $A_i = \langle S_i, \Sigma, \delta_i, s_{0,i}, F_i \rangle$, $1 \le i \le k$, is a *deterministic finite automaton* with finite state set S_i, *partial* transition function $\delta_i : S_i \times (\Sigma \cup \{\lambda, \triangleleft\}) \to S_i$ (requiring that $\delta_i(s,a)$ is undefined for all

$a \in \Sigma \cup \{\lhd\}$, if $\delta_i(s,\lambda)$ is defined), initial state $s_{0,i} \in S_i$, and set of accepting states $F_i \subseteq S_i$,

3. $Q = \{q_1, q_2, \ldots, q_k\} \subseteq \bigcup_{1 \leq i \leq k} S_i$ is the set of *query states*, and
4. $\lhd \notin \Sigma$ is the *end-of-input symbol*.

The single automata are called *components* of the system A. A *configuration* $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k)$ of A represents the current states s_i as well as the still unread parts x_i of the tape inscription of all components $1 \leq i \leq k$. System A starts with all of its components scanning the first square of the tape in their initial states. For input word $w \in \Sigma^*$, the initial configuration is $(s_{0,1}, w\lhd, s_{0,2}, w\lhd, \ldots, s_{0,k}, w\lhd)$.

Basically, a computation of A is a sequence of configurations beginning with an initial configuration and ending with a halting configuration, when no successor configuration exists. Each step can consist of two phases. In a first phase, all components are in non-query states and perform an ordinary (non-communicating) step independently. The second phase is the communication phase during which components in query states receive the requested states as long as the sender is not in a query state itself. That is, if a component A_i is in query state q_j, then A_i is set to the current state of component A_j. This process is repeated until all requests are resolved, if possible. If the requests are cyclic, no successor configuration exists. For the first phase, we define the successor configuration relation $\vdash$ by $(s_1, a_1y_1, s_2, a_2y_2, \ldots, s_k, a_ky_k) \vdash (p_1, z_1, p_2, z_2, \ldots, p_k, z_k)$, if $Q \cap \{s_1, s_2, \ldots, s_k\} = \emptyset$, $a_i \in \Sigma \cup \{\lambda, \lhd\}$, $p_i \in \delta_i(s_i, a_i)$, and $z_i = \lhd$ for $a_i = \lhd$ and $z_i = y_i$ otherwise, $1 \leq i \leq k$. For non-returning communication in the second phase, we set $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k) \vdash (p_1, x_1, p_2, x_2, \ldots, p_k, x_k)$, if, for all $1 \leq i \leq k$ such that $s_i = q_j$ and $s_j \notin Q$, we have $p_i = s_j$, and $p_r = s_r$ for all the other r, $1 \leq r \leq k$. Alternatively, for returning communication in the second phase, we set $(s_1, x_1, s_2, x_2, \ldots, s_k, x_k) \vdash (p_1, x_1, p_2, x_2, \ldots, p_k, x_k)$, if, for all $1 \leq i \leq k$ such that $s_i = q_j$ and $s_j \notin Q$, we have $p_i = s_j$, $p_j = s_{0,j}$, and $p_r = s_r$ for all the other r, $1 \leq r \leq k$.

A computation *halts* when the successor configuration is not defined for the current situation. In particular, this may happen when cyclic communication requests appear, or when the transition function of one component is not defined. The language $L(A)$ accepted by a DPCFA(k) A is precisely the set of words w such that there is some computation beginning with $w\lhd$ on the input tape and halting with at least one component having an undefined transition function and being in an accepting state. Let $\vdash^*$ denote the reflexive and transitive closure of the successor configuration relation $\vdash$ and define $L(A)$ as

$$\{ w \in \Sigma^* \mid (s_{0,1}, w\lhd, s_{0,2}, w\lhd, \ldots, s_{0,k}, w\lhd) \vdash^* (p_1, a_1y_1, p_2, a_2y_2, \ldots, p_k, a_ky_k), \text{ such that } p_i \in F_i \text{ and } \delta_i(p_i, a_i) \text{ as well as } \delta_i(p_i, \lambda) \text{ are undefined for some } 1 \leq i \leq k \}.$$

Whenever the degree is missing in the notation DPCFA(k), we mean systems of arbitrary degree. The absence or presence of an R in the type of the system denotes whether it works in *non-returning* communication, that is, the sender remains in its current state, or *returning* communication, that is, the sender is reset to its initial state. If there is just one component, say A_1, that is allowed to query for states, that is, $S_i \cap Q = \emptyset$, for $2 \leq i \leq k$, then the system is said to be *centralized*. In this case, we refer to A_1 as the *master component* and add a C to the notation of the type of the system. The *family of languages accepted* by devices of type X with arbitrary degree (with degree k) is denoted by $\mathscr{L}(X)$ ($\mathscr{L}(X(k))$).

In the following, we study the impact of communication in PCFA. The communication is measured by the *total number of queries sent during a computation*. That is, we count the number of time steps at which a component enters a query state and consider the sum of these numbers for all components. Let $f : \mathbb{N} \to \mathbb{N}$ be a mapping. If all $w \in L(A)$ are accepted with computations where the total number of queries sent is bounded by $f(|w|)$, then A is said to be *communication bounded by* f.

We denote the class of devices of type X (with degree k) that are communication bounded by some function f by f-X (f-X(k)).

In order to clarify the notation we give two examples. Whenever we refer to a time t of a computation of a DPCFA, then the configuration reached after exactly t computation steps is considered.

Example 1 The language $L_{expo} = \{\, \$a^{2^0}ba^{2^1}b\cdots ba^{2^m}\& \mid m \geq 1 \,\}$ belongs to $\mathscr{L}(f\text{-DRCPCFA}(2))$ with $f \in O(\log(n))$. Roughly, the idea of the construction is that the lengths of adjacent a-blocks (separated by a b) are compared. To this end, the master reads the left block with half speed, that is, moving one symbol to the right in every other time step, while the non-master component reads the right block with full speed, that is, moving one symbol to the right in every time step. If the master reaches a b, it queries the non-master whether it has also reached a b. If this is true, the comparison of the next two a-blocks is started. The input is accepted if the master obtains the symbol & from the non-master component and the remaining input is in $a^+\&\triangleleft$.

Formally, we define $A = \langle\{a,b,\$,\&\}, A_1, A_2, \{q_2\}, \triangleleft\rangle$ to be a DRCPCFA(2) with master component $A_1 = \langle\{s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}, s_{4,1}, s_{5,1}, s_b, s_\&, q_2, accept\}, \{a,b,\$,\&\}, \delta_1, s_{0,1}, \{accept\}\rangle$, second component $A_2 = \langle\{s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}, s_b, s_\&, s_\triangleleft\}, \{a,b,\$,\&\}, \delta_2, s_{0,2}, \emptyset\rangle$, and transition functions δ_1 and δ_2 as follows.

The non-master component A_2:

1. $\delta_2(s_{0,2}, \$) = s_{1,2}$
2. $\delta_2(s_{1,2}, a) = s_{2,2}$
3. $\delta_2(s_{2,2}, b) = s_{3,2}$
4. $\delta_2(s_{3,2}, a) = s_{3,2}$
5. $\delta_2(s_{3,2}, b) = s_b$
6. $\delta_2(s_{3,2}, \&) = s_\&$
7. $\delta_2(s_{0,2}, a) = s_{3,2}$
8. $\delta_2(s_{0,2}, \triangleleft) = s_\triangleleft$
9. $\delta_2(s_\triangleleft, \lambda) = s_\triangleleft$

The component reads the input prefix $\$ab$ in the first three time steps (rules 1,2,3). Subsequently, it reads an a-block in state $s_{3,2}$ (rule 4). Whenever it moves on a symbol b it changes into state s_b (rule 5). So, it enters state s_b at time step 3 plus the length of the second a-block plus 1. The component halts in state s_b unless it is reset to its initial state by a query. In this case it reads the current a-block and the next b and enters state s_b again after a number of time steps that is the length of the a-block plus one (rules 7,4,5). Rule 6 is used when & appears in the input instead of b. After being reset into the initial state on the endmarker, the component enters state $s_\triangleleft$ and loops with λ-moves.

The master component A_1:

1. $\delta_1(s_{0,1}, \$) = s_{1,1}$
2. $\delta_1(s_{1,1}, \lambda) = s_{2,1}$
3. $\delta_1(s_{2,1}, \lambda) = s_{3,1}$
4. $\delta_1(s_{3,1}, a) = s_{4,1}$
5. $\delta_1(s_{4,1}, \lambda) = s_{3,1}$
6. $\delta_1(s_{3,1}, b) = q_2$
7. $\delta_1(s_b, a) = s_{4,1}$
8. $\delta_1(s_\&, a) = s_\&$
9. $\delta_1(s_\&, \&) = s_{5,1}$
10. $\delta_1(s_{5,1}, \triangleleft) = accept$

The master reads the input prefix $\$ab$ in the first six time steps and enters the query state q_2 (rules 1–6). Exactly at that time the non-master component enters state s_b. Being in state s_b received the master reads the current a-block and the next b and enters state q_2 again after a number of time steps that is two times the length of the a-block plus one (rules 7,4,5,6). Exactly at this time the non-master component enters state s_b again provided that the a-block read by the non-master component is twice as long as the a-block read by the master. When the master receives state $s_\&$ instead of s_b, it reads the remaining suffix (rules 8,9), enters the accepting state on the endmarker (rule 10) and halts.

Finally, the length of a word $w \in L_{expo}$ is $|w| = m+2+\sum_{i=0}^{m} 2^i = 2^{m+1}+m+1$, for some $m \geq 1$. In its accepting computation, a communication takes place for every symbol b and the endmarker. So there are $m+1$ communications which is of order $O(\log(|w|))$. □

The construction of the next example is similar to the one given in Example 1.

Example 2 The language $L_{poly} = \{\, \$aba^3ba^5b\cdots ba^{2m+1}\& \mid m \geq 0 \,\}$ belongs to $\mathscr{L}(f\text{-DRCPCFA}(2))$ with $f \in O(\sqrt{n})$. □

3 Computational Capacity

In this section we consider aspects of the computational capacity of f-DRCPCFA(k). Examples 1 and 2 already revealed that there are non-semilinear languages accepted by systems with two components and sublinear communication. The next simple result is nevertheless important for the size of representations that will be used in connection with Kolmogorov arguments to separate language classes.

Lemma 3 *Let $k \geq 1$ and A be a DRCPCFA(k) with $S_1, S_2, \ldots, S_k$ being the state sets of the single components. If $w \in L(A)$, then w is accepted after at most $|S_1|\cdot|S_2|\cdots|S_k|\cdot(|w|+1)$ time steps, that is, in linear time.*

Proof During a computation some component A_i may be in $|S_i|$ different states. So after $|S_1|\cdot|S_2|\cdots|S_k|$ time steps the whole system runs through a loop if none of the components moves. Therefore, as long as no halting configuration is reached, at least one component must move after at most $|S_1|\cdot|S_2|\cdots|S_k|$ time steps. □

The language of the next lemma combines the well-known non-context-free copy language with L_{expo} from above. It plays a crucial role in later proofs.

Lemma 4 *The language*

$$L_{expo,wbw} = \{\, \$w_1w_2\cdots w_mba^{2^0}w_1w_1a^{2^1}w_2w_2\cdots a^{2^{m-1}}w_mw_m\& \mid m \geq 1, w_i \in \{0,1\}, 1 \leq i \leq m \,\}$$

belongs to $\mathscr{L}(O(\log(n))\text{-DRCPCFA}(3))$.

Proof A formal construction of a $O(\log(n))$-DRCPCFA(3) accepting $L_{expo,wbw}$ is given through the transition functions below, where $s_{0,i}$ is the initial state of component A_i, $1 \leq i \leq 3$, the sole accepting state is *accept*, and $\sigma \in \{0,1\}$.

The second non-master component A_3 initially passes over the \$ and, then, it reads a symbol, remembers it in its state, and loops without moving (rules 1,2,3,8,9). Whenever the component is reset into its initial state after a query, it reads the next symbol, remembers it, and loops without moving (rules 4–11). This component is used by the master to match the w_i from the prefix with the w_i from the suffix.

The non-master component A_3:

1. $\delta_3(s_{0,3}, \$) = s_{1,3}$
2. $\delta_3(s_{1,3}, 0) = s_0$
3. $\delta_3(s_{1,3}, 1) = s_1$
4. $\delta_3(s_{0,3}, 0) = s_0$
5. $\delta_3(s_{0,3}, 1) = s_1$
6. $\delta_3(s_{0,3}, b) = s_b$
7. $\delta_3(s_{0,3}, a) = s_a$
8. $\delta_3(s_0, \lambda) = s_0$
9. $\delta_3(s_1, \lambda) = s_1$
10. $\delta_3(s_b, \lambda) = s_b$
11. $\delta_3(s_a, \lambda) = s_a$

The first non-master component A_2 initially passes over the prefix $\$w_1w_2\cdots w_m$ (rules 1,2), the b (rule 3), and the adjacent infix $aw_1w_1aaw_2w_2$ (rules 4–13). On its way it checks whether the neighboring symbols w_i are in fact the same (rules 5–8 and 10–13). If the second check is successful the component enters state s_{ww}. Exactly at that time it has to be queried by the master, otherwise it blocks the computation. Subsequently, it repeatedly continues to read the input, where each occurrence of neighboring symbols w_i are checked for equality (rules 14 and 9–13), which is indicated by entering state s_{ww} again. This component is used to verify that all neighboring symbols w_i in the suffix are equal and, by the master, to check the lengths of the a-blocks in the same way as in Example 1. Note that the component is at time $m+9$ on the first symbol after w_2w_2. After being reset to its initial state, it takes a number of time steps equal to the length of the next a-block plus 2 to get on the first symbol after the next w_iw_i.

The non-master component A_2:

1. $\delta_2(s_{0,2},\$) = s_{1,2}$
2. $\delta_2(s_{1,2},\sigma) = s_{1,2}$
3. $\delta_2(s_{1,2},b) = s_{2,2}$
4. $\delta_2(s_{2,2},a) = s_{3,2}$
5. $\delta_2(s_{3,2},0) = s^0_{4,2}$
6. $\delta_2(s_{3,2},1) = s^1_{4,2}$
7. $\delta_2(s^0_{4,2},0) = s_{5,2}$
8. $\delta_2(s^1_{4,2},1) = s_{5,2}$
9. $\delta_2(s_{5,2},a) = s_{5,2}$
10. $\delta_2(s_{5,2},0) = s^0_{6,2}$
11. $\delta_2(s_{5,2},1) = s^1_{6,2}$
12. $\delta_2(s^0_{6,2},0) = s_{ww}$
13. $\delta_2(s^1_{6,2},1) = s_{ww}$
14. $\delta_2(s_{0,2},a) = s_{5,2}$
15. $\delta_2(s_{0,2},\&) = s_\&$
16. $\delta_2(s_\&,\lambda) = s_\&$
17. $\delta_2(s_{0,2},\triangleleft) = s_\triangleleft$
18. $\delta_2(s_\triangleleft,\lambda) = s_\triangleleft$

The master component A_1 initially passes over the prefix $\$w_1w_2\cdots w_m$ (rules 1,2), the b (rule 3), and the first a (rules 4–8). Then it reads the first of two adjacent symbols w_i and enters the query state q_3 (rule 9) (the equality of the symbols w_i has already been checked by component A_2). From component A_3 it receives the information about the matching symbol w_i from the prefix. If this symbol is the same as the next input symbol, then the computation continues (rules 10,11) by entering query state q_2. Note that this happens exactly at time step $m+9$. If the master receives state s_{ww} the length of the first two a-blocks are verified. Now the master repeatedly continues to read the input (rule 12,7,8), where on each occurrence of neighboring symbols w_i the equality with the corresponding symbol in the prefix is checked by querying component A_3 and the lengths of the a-blocks are compared by querying component A_2. After querying component A_2, it takes a number of time steps equal to the length of the adjacent a-block (processed by component A_2) plus 2 to get into state q_2 again. Finally, when the master component has checked the last symbol w_m and gets the information that A_2 has read symbol &, it queries component A_3 (rule 13). If it receives a b, the input is accepted (rule 14). In all other cases it is rejected.

The master component A_1:

1. $\delta_1(s_{0,1},\$) = s_{1,1}$
2. $\delta_1(s_{1,1},\sigma) = s_{1,1}$
3. $\delta_1(s_{1,1},b) = s_{2,1}$
4. $\delta_1(s_{2,1},\lambda) = s_{3,1}$
5. $\delta_1(s_{3,1},\lambda) = s_{4,1}$
6. $\delta_1(s_{4,1},\lambda) = s_{5,1}$
7. $\delta_1(s_{5,1},a) = s_{6,1}$
8. $\delta_1(s_{6,1},\lambda) = s_{5,1}$
9. $\delta_1(s_{5,1},\sigma) = q_3$
10. $\delta_1(s_0,0) = q_2$
11. $\delta_1(s_1,1) = q_2$
12. $\delta_1(s_{ww},a) = s_{6,1}$
13. $\delta_1(s_\&,\&) = q_3$
14. $\delta_1(s_b,\triangleleft) = accept$

The length of a word $w \in L_{expo,wbw}$ is $|w| = 3m+3+\sum_{i=0}^{m-1} 2^i = 2^m+3m+2$, for some $m \geq 1$. In its accepting computation, two communications take place for every w_iw_i and one more communication on

the endmarker. So there are $2m+1$ communications which is of order $O(\log(|w|))$. □

For the proof of the following theorem we use an incompressibility argument. General information on Kolmogorov complexity and the incompressibility method can be found in [10]. Let $w \in \{0,1\}^+$ be an arbitrary binary string. The Kolmogorov complexity $C(w)$ of w is defined to be the minimal size of a program describing w. The following key argument for the incompressibility method is well known. There are binary strings w of *any* length so that $|w| \leq C(w)$.

Lemma 5 *The language* $L_{wbw} = \{ w_1 w_2 \cdots w_m b w_1 w_2 \cdots w_m \mid m \geq 1, w_i \in \{0,1\}, 1 \leq i \leq m \}$ *is accepted by some* $O(n)$*-DRCPCFA*(2) *but, for any* $k \geq 1$*, does not belong to* $\mathscr{L}(f\text{-DRCPCFA}(k))$ *if* $f \in \frac{n}{\omega(\log(n))}$.

Proof First, we sketch the construction of a $O(n)$-DRCPCFA(2) accepting L_{wbw}. Initially, the master component proceeds to the center marker b, while the non-master component reads the first input symbol w_1 and remembers this information in its state. Next, the master queries the non-master and matches the information received with the first symbol following b, while the non-master reads the next input symbol and remembers it in its state. Subsequently, this behavior is iterated, that is, the master queries the non-master again and matches its next input symbol, while the non-master reads and remembers the next symbol. The input is accepted when the master receives a b at the moment it reaches the right endmarker. Clearly, the number of communications on input length $n = 2m+1$ is $m+1 \in O(n)$.

Second, we turn to show that $L_{wbw} \notin \mathscr{L}(f\text{-DRCPCFA}(k))$ if $f \in \frac{n}{\omega(\log(n))}$. In contrast to the assertion, we assume that L_{wbw} is accepted by some f-DRCPCFA(k) $A = \langle \Sigma, A_1, A_2, \ldots, A_k, Q, \triangleleft \rangle$ with $f(n) \in \frac{n}{\omega(\log(n))}$. Let $z = wbw$, for some $w \in \{0,1\}^+$, and $K_0 \vdash \cdots \vdash K_{acc}$ be the accepting computation on input z, where K_0 is the initial configuration and K_{acc} is an accepting configuration.

Next, we consider snapshots of configurations at every time step at which the master component queries some other component or at which a component enters the middle marker b. For every such configuration, we take the time step t_i, the current states $s_1^{(i)}, s_2^{(i)}, \ldots, s_k^{(i)}$, and the positions $p_1^{(i)}, p_2^{(i)}, \ldots, p_k^{(i)}$ of the components. Thus, the ith snapshot is represented by the tuple $(t_i, s_1^{(i)}, p_1^{(i)}, s_2^{(i)}, p_2^{(i)}, \ldots, s_k^{(i)}, p_k^{(i)})$. Since there are altogether at most $f(2|w|+1)$ communications, the list of snapshots Λ contains at most $f(2|w|+1)+k$ entries.

We claim that each snapshot can be represented by at most $O(\log(|w|))$ bits. Due to Lemma 3 acceptance is in linear time and, therefore, each time step can be represented by at most $O(\log(|w|))$ bits. Each position of a component can also be represented by at most $O(\log(|w|))$ bits. Finally, each state can be represented by a constant number of bits. Altogether, each snapshot can be represented by $O(\log(|w|))$ bits. So, the list Λ can be represented by $(f(2|w|+1)+k) \cdot O(\log(|w|)) = \frac{|w|}{\omega(\log(|w|))} \cdot O(\log(|w|)) = o(|w|)$ bits.

Now we show that the list Λ of snapshots together with a snapshot of K_{acc} and the knowledge of A and $|w|$ is sufficient to reconstruct w. The reconstruction is implemented by the following algorithm P. First, P sequentially simulates A on all $2^{|w|}$ inputs xbx where $|x| = |w|$. Additionally, it is checked whether the computation simulated has the same snapshots as in the list Λ and the accepting configuration. In this way, the string w can be identified. We have to show that there is no other string $w' \neq w$ which can be identified in this way as well. Let us assume that such a w' exists. Then all snapshots of accepting computations on input wbw and $w'bw'$ are identical. This means that both computations end at the same time step and all components are in the same state and position. Additionally, in both computations communications take place at the same time steps, all components are in the same state and position at that moment. Moreover, the right half of the respective words is entered in the same states and in

the same time steps on both input words wbw and wbw'. So, both computations are also accepting on input wbw' which is a contradiction.

Thus, w can be reconstructed given the above program P, the list of snapshots Λ, the snapshot of the accepting configuration, A, and $|w|$. Since the sizes of P and A are bounded by a constant, the size of Λ is bounded by $o(|w|)$, and $|w|$ as well as the size of the remaining snapshot is bounded by $O(\log(|w|))$ each, we can reconstruct w from a description of total size $o(|w|)$. Hence, the Kolmogorov complexity $C(w)$, that is, the minimal size of a program describing w is bounded by the size of the above description, and we obtain $C(w) \in o(|w|)$. On the other hand, we know that there are binary strings w of arbitrary length such that $C(w) \geq |w|$. This is a contradiction for w being long enough. □

The language of the next lemma is used in later proofs.

Lemma 6 *The language*

$$L_{poly,wbw} = \{\, \$w_1w_2\cdots w_m b a^1 w_1 w_1 a^3 w_2 w_2 a^5 w_3 w_3 \cdots a^{2m-1} w_m w_m \& \mid m \geq 1, w_i \in \{0,1\}, 1 \leq i \leq m \,\}$$

is accepted by some $O(\sqrt{n})$-DRCPCFA(3) but, for any $k \geq 1$, does not belong to $\mathscr{L}(f$-DRCPCFA$(k))$ if $f \in O(\log(n))$.

Proof Using the construction idea of Lemma 4, one shows $L_{poly,wbw} \in \mathscr{L}(O(\sqrt{n})\text{-DRCPCFA}(3))$.

The claimed non-containment is shown similarly to Lemma 5: in contrast to the assertion, we assume that $L_{poly,wbw}$ is accepted by some f-DRCPCFA(k) $A = \langle \Sigma, A_1, A_2, \ldots, A_k, Q, \lhd \rangle$ with $f(n) \in O(\log(n))$. Let

$$z = \$w_1w_2\cdots w_m b a^1 w_1 w_1 a^3 w_2 w_2 a^5 w_3 w_3 \cdots a^{2m-1} w_m w_m \& \in L_{poly,wbw},$$

where $w = w_1w_2\cdots w_m$, and $K_0 \vdash \cdots \vdash K_{acc}$ be the accepting computation on input z, where K_0 is the initial configuration and K_{acc} is an accepting configuration.

We use again an incompressibility argument and write down the list of snapshots of configurations in which communication takes place and the accepting configuration K_{acc}, and descriptions of A and $|w|$. Similar to the proof of Lemma 5, a program P can be described which reconstructs w uniquely from the information given.

Next, we determine the size of such a description. Program P and the system A can be represented by a constant number of bits. The length $|w|$ can be described by $\log(|w|) \in O(\log(m))$ bits. Since $|z| = 3m+3+\sum_{i=1}^{m} 2i-1 = 3m+3+m^2$ and acceptance is in linear time (Lemma 3), each time step can be represented by $O(\log(|z|)) = O(\log(m^2))$ bits. Moreover, the k states can be described by $O(1)$ bits, and the k positions by $k \cdot \log(|z|) = k \cdot \log(m^2+3m+3) \in O(\log(m))$ bits. So, altogether one snapshot can be represented by $O(\log(m))$ bits. Since at most $f(|z|) \in O(\log(|z|)) = O(\log(m))$ snapshots have to be listed, the list of all snapshots can be described by $O((\log(m))^2)$ bits. Therefore, the total size of a description of w is bounded by $O((\log(m))^2)$ as well. Thus, the Kolmogorov complexity $C(w)$ of w is bounded by $O((\log(m))^2)$. On the other hand, there are binary strings w of arbitrary length such that $C(w) \geq |w| = m$. This is a contradiction for w being long enough. □

The previous theorems showed that there are proper inclusions

$$\mathscr{L}(O(\log(n))\text{-DRCPCFA}(k)) \subset \mathscr{L}(O(\sqrt{n})\text{-DRCPCFA}(k))$$

for every $k \geq 3$, and

$$\mathscr{L}(O(\sqrt{n})\text{-DRCPCFA}(k)) \subset \mathscr{L}(O(n)\text{-DRCPCFA}(k))$$

for every $k \geq 2$.

Later, we will prove an infinite hierarchy in between the classes $\mathscr{L}(O(\log(n))$-DRCPCFA$(k))$ and $\mathscr{L}(O(\sqrt{n}))$-DRCPCFA(k), for every $k \geq 4$.

4 Decidability and Undecidability Results

4.1 Undecidability of Emptiness and Classical Questions

First, we show undecidability of the classical questions for models with a logarithmic amount of communication. To this end, we adapt the construction given in [1] which is based on the valid computations of *one-way cellular automata* (OCA), a parallel computational model (see, for example, [7, 8]). More precisely, the undecidability is shown by reduction of the corresponding problems for OCA which are known not even to be semi-decidable [11]. To this end, histories of OCA computations are encoded in single words that are called *valid computations* (cf., for example, [5]).

A one-way cellular automaton is a linear array of identical deterministic finite automata, sometimes called cells. Except for the leftmost cell each one is connected to its nearest neighbor to the left. The state transition depends on the current state of a cell itself and the current state of its neighbor, where the leftmost cell receives information associated with a boundary symbol on its free input line. The state changes take place simultaneously at discrete time steps. The input mode for cellular automata is called parallel. One can suppose that all cells fetch their input symbol during a pre-initial step.

More formally, an OCA is a system $M = \langle S, \#, T, \delta, F \rangle$, where S is the nonempty, finite set of cell states, $\# \notin S$ is the boundary state, $T \subseteq S$ is the input alphabet, $F \subseteq S$ is the set of accepting cell states, and $\delta : (S \cup \{\#\}) \times S \to S$ is the local transition function.

A configuration of an OCA at some time step $t \geq 0$ is a description of its global state, which is formally a mapping $c_t : \{1, 2, \ldots, n\} \to S$, for $n \geq 1$. The initial configuration at time 0 on input $w = x_1 x_2 \ldots x_n$ is defined by $c_{0,w}(i) = x_i$, $1 \leq i \leq n$. Let c_t, $t \geq 0$, be a configuration with $n \geq 2$, then its successor c_{t+1} is defined as follows: $c_{t+1}(1) = \delta(\#, c_t(1))$ and $c_{t+1}(i) = \delta(c_t(i-1), c_t(i))$, $2 \leq i \leq n$.

An input is accepted if at some time step during its computation the rightmost cell enters an accepting state. Without loss of generality and for technical reasons, one can assume that any accepting computation has at least three steps.

Now we turn to the valid computations of an OCA $M = \langle S, \#, T, \delta, F \rangle$. The computation of a successor configuration c_{t+1} of a given configuration c_t is written down in a sequential way as follows. Assume c_{t+1} is computed cell by cell from left to right. That is, we are concerned with subconfigurations of the form $c_{t+1}(1) \cdots c_{t+1}(i) c_t(i+1) \cdots c_t(n)$, where n is the length of the input. For technical reasons, in $c_{t+1}(i)$ we have to store both the successor state, which is entered in time step $t+1$ by cell i, and its former state. In this way, the computation of the successor configuration of M can be written as a sequence of n subconfigurations, and configuration c_{t+1} can be represented by $w^{(t+1)} = w_1^{(t+1)} \cdots w_n^{(t+1)}$ such that $w_i^{(t+1)} \in \#S^*(S \times S)S^*$, for $1 \leq i \leq n$, with $w_i^{(t+1)} = \#c_{t+1}(1) \cdots c_{t+1}(i-1)(c_{t+1}(i), c_t(i))c_t(i+1) \cdots c_t(n)$. The valid computations VALC(M) are now defined to be the set of words of the form $w^{(0)} w^{(1)} \cdots w^{(m)}$, where $m \geq 3$, $w^{(t)} \in (\#S^*(S \times S)S^*)^+$ are configurations of M, $1 \leq t \leq m$, $w^{(0)}$ is an initial configuration having the form $\#(T')^+$, where T' is a primed copy of the input alphabet T with $T' \cap S = \emptyset$, $w^{(m)}$ is an accepting configuration of the form $(\#S^*(S \times S)S^*)^* \#S^*(F \times S)$, and $w^{(t+1)}$ is the successor configuration of $w^{(t)}$, for $0 \leq t \leq m-1$.

For the constructions of DRCPCFA accepting the set VALC(M), we provide an additional technical transformation of the input alphabet. Let $S' = S \cup T'$ and $A = \{\#\} \cup S' \cup S'^2$ be the alphabet

over which VALC(M) is defined. We consider the mapping $f : A^+ \to (A \times A)^+$ which is defined for words of length at least two by $f(x_1x_2\cdots x_n) = [x_1,x_2][x_2,x_3]\cdots[x_{n-1},x_n]$. From now on we consider VALC(M) $\subseteq (A\times A)^+$ to be the set of valid computations to which f has been applied. The set of *invalid computations* INVALC(M) is then the complement of VALC(M) with respect to the alphabet $A \times A$.

The following example illustrates the definitions.

Example 7 We consider the following computation of an OCA M over the input alphabet $\{c,d\}$. The initial configuration is $c_0 = (c,d,d)$. Let the successor configurations be $c_1 = (p_1,r_1,s_1)$, $c_2 = (p_2,r_2,s_2)$, and $c_3 = (p_3,r_3,s_3)$. Furthermore, let s_3 be an accepting state, that is, cdd is an accepted input. These configurations are written down as sequences of subconfigurations as follows.

$$\begin{aligned} w^{(0)} &= \#c'd'd' \\ w^{(1)} &= \#(p_1,c)dd\#p_1(r_1,d)d\#p_1r_1(s_1,d) \\ w^{(2)} &= \#(p_2,p_1)r_1s_1\#p_2(r_2,r_1)s_1\#p_2r_2(s_2,s_1) \\ w^{(3)} &= \#(p_3,p_2)r_2s_2\#p_3(r_3,r_2)s_2\#p_3r_3(s_3,s_2) \end{aligned}$$

Then,

$$\begin{aligned} &f(w^{(0)}w^{(1)}w^{(2)}w^{(3)}) = [\#,c'][c',d'][d',d'][d',\#][\#,(p_1,c)][(p_1,c),d][d,d][d,\#] \\ &[\#,p_1][p_1,(r_1,d)][(r_1,d),d][d,\#][\#,p_1][p_1,r_1][r_1,(s_1,d)][(s_1,d),\#][\#,(p_2,p_1)] \\ &[(p_2,p_1),r_1][r_1,s_1][s_1,\#][\#,p_2][p_2,(r_2,r_1)][(r_2,r_1),s_1][s_1,\#][\#,p_2][p_2,r_2] \\ &[r_2,(s_2,s_1)][(s_2,s_1),\#][\#,(p_3,p_2)][(p_3,p_2),r_2][r_2,s_2][s_2,\#][\#,p_3][p_3,(r_3,r_2)] \\ &[(r_3,r_2),s_2][s_2,\#][\#,p_3][p_3,r_3][r_3,(s_3,s_2)] \end{aligned}$$

is a valid computation of M.

The length of a valid computation can be easily calculated.

Lemma 8 *Let M be an OCA on input $w_1w_2\cdots w_n$ which is accepted after t time steps. Then the length of the corresponding valid computation is $n+(n+1)\cdot n\cdot t$.*

The next lemma is the key tool for the reductions.

Lemma 9 *Let M be an OCA. Then language*

$$VALC'(M) = \{\, \$_1x_1x_2\cdots x_m\$_2a^{2^0}bba^{2^1}bb\cdots bba^{2^{m-1}}bb\& \mid m \geq 1, x_1x_2\cdots x_m \in VALC(M)\,\}$$

belongs to $\mathscr{L}(O(\log(n))$-DRCPCFA$(4))$.

Proof In [1] a $O(n)$-DRCPCFA(3) is constructed that accepts VALC(M). Basically, the master component A_1 and component A_2 are used to verify that after every subconfiguration the correct successor subconfiguration is given, whereas component A_3 is used to check the correct format of the input. This construction can be implemented identically for the present construction if we interpret $\$_2$ as the right endmarker. Additionally, component A_4 is used in the same way as component A_3 in the construction of Lemma 4, that is, initially it reads $\$_1$ and x_1, stores x_1 in its state, and waits at position 2 until it is queried. After being reset to its initial state, it again reads the next input symbol, stores it, and waits.

When $x_1x_2\cdots x_m \in$ VALC(M) is tested, the master A_1 and component A_2 are both located at $\$_2$. The second part of the input is now tested along the line of the construction given in the proof of Lemma 4,

where the master plays the role of the master, component A_2 the role of component A_2, and component A_4 the role of component A_3.

The length of a word $w \in \mathrm{VALC}'(M)$ is $|w| = 3m+3+\sum_{i=0}^{m-1} 2^i = 2^m+3m+2$, for some $m \geq 1$. The test whether $x_1x_2\cdots x_m$ belongs to $\mathrm{VALC}(M)$ requires $O(m)$ communications. For the remaining tests additional $O(m)$ communications are necessary as the proof of Lemma 4 shows. So, altogether, $O(m)$ communications are sufficient which is of order $O(\log(|w|))$. □

The set of *invalid computations* $\mathrm{INVALC}'(M)$ is simply defined to be the complement of $\mathrm{VALC}'(M)$ with respect to the alphabet $\{a,b,\$_1,\$_2,\&\} \cup (A \times A)$.

Lemma 10 *Let M be an OCA. Then language $\mathrm{INVALC}'(M)$ belongs to $\mathscr{L}(O(\log(n))$-DRCPCFA$(4))$.*

Proof To accept the set of invalid computations $\mathrm{INVALC}'(M)$ almost the same construction as for Lemma 9 can be used. The only adaption concerns acceptance and rejection. Since the only possibility to accept is that the master halts in state *accept* while the other components are non-halting, accepting computations can be made rejecting by sending the master into a halting non-accepting state *reject* instead. In order to make rejecting computations accepting, it is now sufficient to send the components into some halting accepting state whenever they would halt rejecting. □

Theorem 11 *For any degree $k \geq 4$, emptiness, finiteness, infiniteness, universality, inclusion, equivalence, regularity, and context-freeness are not semi-decidable for $O(\log(n))$-DRCPCFA(k).*

Proof All these problems are known to be non-semi-decidable for OCA [11]. By standard techniques (cf., for example, [1]) the OCA problems are reduced to $O(\log(n))$-DRCPCFA(k) via the valid and invalid computations and Lemmas 9 and 10. □

4.2 Undecidability of Communication Boundedness

This subsection is devoted to questions concerning the decidability or computability of the communication bounds. In principle, we deal with three different types of problems. The first type is to decide for a given DRCPCFA(k) A and a given function f whether or not A is communication bounded by f. The next theorem solves this problem negatively for all non-trivial communication bounds and all degrees $k \geq 3$.

Theorem 12 *Let $k \geq 3$ be any degree, $f \in o(n)$, and A be a DRCPCFA(k). Then it is not semi-decidable whether A is communication bounded by f.*

Proof Let A be a DRCPCFA(k) with $k \geq 3$ accepting some language $L(A) \subseteq \Sigma^*$. We take two new symbols $\{a,\$\} \cap \Sigma = \emptyset$ and construct a DRCPCFA(k) A' accepting language $a^*\$L(A)$. The idea of the construction is that, initially, all components move synchronously across the leading a-block. During this phase, the master component queries one of the non-master components in every time step. When all components have read the separating symbol \$, they enter the initial state of the corresponding component of A. Subsequently, A is simulated, thus testing whether the remaining input belongs to $L(A)$. So, on input $a^n\$w$ with $n \geq 1$ and $w \in L(A)$, A' performs at least n communications. In particular, for $n \geq |w|$ we obtain words that show that A' is not communication bounded by any function $f \in o(n)$, unless $L(A)$ is empty. So, A' is a f-DRCPCFA(k) if and only if $L(A) = \emptyset$.

Since in [1] it has been shown that emptiness is not semi-decidable for DRCPCFA with at least three components, the theorem follows. □

Mitrana considers in [13] the *degree of communication* of parallel communicating finite automata systems. The degree of communication of an accepting computation is defined as the number of queries posed. The degree of communication $Comm(x)$ of a nondeterministic PCFA A on input x is defined as the minimal number of queries posed in accepting computations on x. The degree of communication $Comm(A)$ of a PCFA A is then defined as $\sup\{Comm(x) \mid x \in L(A)\}$. Here we have the second type of problems we are dealing with. Mitrana raised the question whether the degree of communication $Comm(A)$ is computable for a given nondeterministic PCFA(k) A. Since $Comm(A)$ is either finite or infinite, in our terms the question is to decide whether or not A is communication bounded by some function $f \in O(1)$ and, if it is, to compute the precise constant. The next theorem solves the problem.

Theorem 13 *Let $k \geq 3$ be an integer. Then the degree of communication $Comm(A)$ is not computable for DRCPCFA(k).*

Proof For a given DRCPCFA(k) A and new input symbols a and \$, we construct a DRCPCFA(k) A' accepting the language $a^*\$L(A)$ as in the proof of Theorem 12.

Now, we claim that $Comm(A') = 0$ if and only if $L(A) = \emptyset$. If $L(A)$ is empty, then A' accepts the empty set and, thus, $Comm(A') = 0$. On the other hand, if $L(A)$ is not empty, then $Comm(A') > 0$ by construction of A'. Since emptiness is not semi-decidable for DRCPCFA(k) with $k \geq 3$ [1], the theorem follows. □

Now we turn to the last type of problems we are dealing with in this section. The question is now whether the degree of communication is computable for the *language* accepted by a given nondeterministic PCFA(k) A. In [13] the degree of communication $Comm_X(L)$ of a language L is defined as $\inf\{Comm(A) \mid A \text{ is device of type } X \text{ and } L(A) = L\}$. Mitrana showed in [13] that $Comm_{CPCFA}(L(A))$ for some nondeterministic CPCFA A is not computable. He leaves as an open question whether the degree is computable for RCPCFA. Here we are going to show that the degree is not even computable for deterministic RCPCFA.

Lemma 14 *Let $k \geq 3$ be an integer. Then the degree of communication $Comm_{DRCPCFA(k)}(L(A))$ is not computable.*

Proof For a given DRCPCFA(k) A over alphabet Σ and new input symbols $b, 0, 1, \$_1, \$_2$, we construct a DRCPCFA(k) A' accepting the language

$$\{w_1 w_2 \cdots w_m b w_1 w_2 \cdots w_m \mid m \geq 1, w_i \in \{0,1\}, 1 \leq i \leq m\}\$_1\$_2 L(A).$$

We present the construction for $k = 3$. The generalization to larger k is straightforward.

The idea of the construction is that in a first phase master component A_1 and a non-master component A_2 check the correctness of the prefix $w_1 w_2 \cdots w_m b w_1 w_2 \cdots w_m$. This is done as in the construction of Lemma 5. Component A_3 checks the correct format of the input up to the separating symbol $\$_1$ and waits on symbol $\$_2$ until it is queried. At the end of this phase, the master is on the $\$_1$ and component A_2 stays on the symbol b.

In a second phase, the master component stays on $\$_1$ and repeatedly queries component A_2 until this one has read $\$_1$ and, thus, stays on $\$_2$. Now the master reads $\$_1$ and queries component A_2. After being reset to its initial state, component A_2 reads $\$_2$ and performs one λ-step. Then it changes to the initial state of A_2 in A. During this λ-step, the master component reads $\$_2$ and queries component A_3. Then it changes to the initial state of the master of A. Finally, after being reset to its initial state, component A_3 reads $\$_2$ and changes into the initial state of A_3 in A.

Now, all components are in their initial states on the first symbol of the input of A and in a third phase A is simulated. We claim that $Comm(L(A')) = 0$ if and only if $L(A) = \emptyset$. If $L(A)$ is empty, then A' accepts the empty set and $Comm(L(A')) = Comm(\emptyset) = 0$. If $L(A)$ is not empty, we fix some $x \in L(A)$. Assume contrarily that $Comm(L(A')) = 0$. Then there exists a DRCPCFA(k) B accepting $L(A')$ such that $Comm(B) = 0$. From B a DRCPCFA($k+1$) B' is constructed by providing an additional component which checks whether the suffix is precisely x, and halts non-accepting if an error is found. So, B' accepts the language

$$\{ w_1 w_2 \cdots w_m b w_1 w_2 \cdots w_m \mid m \geq 1, w_i \in \{0,1\}, 1 \leq i \leq m \} \$_1 \$_2 x$$

and we still have $Comm(B') = 0$. Similar as in the proof of Lemma 5, it follows by an incompressibility argument that this conclusion leads to a contradiction.

Since emptiness is not semi-decidable for DRCPCFA(k) with $k \geq 3$ [1], the degree of communication $Comm_{DRCPCFA(k)}(L(A))$ is not computable. □

5 An Infinite Hierarchy

In this section, we are going to show that there is an infinite strict hierarchy of language classes in between $\mathscr{L}(O(\log(n))$-DRCPCFA(k)) and $\mathscr{L}(O(\sqrt{n})$-DRCPCFA(k)), for any $k \geq 4$. To this end, we consider functions $f : \mathbb{N} \to \mathbb{N}$ that are time-computable by one-way cellular automata. That means, given any unary input of length $n \geq 1$, say a^n, the rightmost cell has to enter an accepting state exactly after $f(n)$ time steps and never before. Time-computable functions in OCA have been studied in [3], where it is shown that, for any $r \geq 1$, there exists an OCA-time-computable function $f \in \Theta(n^r)$. We will use this result in the sequel. So, let M_r be an OCA that time-computes $f \in \Theta(n^r)$, for $r \geq 1$. We will use

$$\begin{aligned} L_r = \{ \$_1 x_1 x_2 \cdots x_\ell \$_2 w_1' w_2' \cdots w_m' w_{m+1} \cdots w_\ell \$_3 w_1' w_2' \cdots w_m' w_{m+1} \cdots w_\ell \$_4 a^{2^0} bb a^{2^1} bb \cdots a^{2^{m-1}} bb \& \mid \\ m \geq 1, x_1 x_2 \cdots x_\ell \text{ is the valid computation of } M_r \text{ on input } a^m, \\ w_i' \in \{0', 1'\}, 1 \leq i \leq m, w_i \in \{0,1\}, m+1 \leq i \leq \ell \} \end{aligned}$$

as witness languages for the infinite hierarchy.

Lemma 15 *Let $r \geq 1$ be an integer. Then language L_r belongs to $\mathscr{L}(O(\log(n)^{r+2})$-DRCPCFA$(4))$.*

Proof An $O(\log(n)^{r+2})$-DRCPCFA(4)) A accepting L_r works in five phases.

As mentioned before, in [1] an $O(n)$-DRCPCFA(3) is constructed that accepts VALC(M), where the master component A_1 and component A_2 are used to verify the subconfigurations, and component A_3 is used to check the correct format of the input. In the first phase, A simulates this behavior where $\$_2$ plays the role of the endmarker. When $x_1 x_2 \cdots x_\ell \in$ VALC(M) has been tested, the master A_1 and component A_2 are both located on the symbol after $\$_2$, that is, on w_1'. Additionally, component A_4 initially reads $\$_1$ and waits on x_1 to be queried. The total number of communications in this phase is of order $O(\ell)$.

In the second phase, it is verified that there are as many symbols in between $\$_1$ and $\$_2$ as in between $\$_2$ and $\$_3$, that is, the length ℓ is matched. Furthermore, it is checked whether there are exactly m symbols of the second infix primed. Since $x_1 x_2 \cdots x_\ell$ describes an OCA computation on some unary input a^m, the initial configuration of the OCA is of the form $\#(a')^m$. Therefore, the valid computation begins with $[\#, a'][a', a']^{m-1}[a', \#]$ followed by symbols not containing primed versions of other symbols. As in the constructions before, the master A_1 moves to the right while querying component A_4 in every step. Whenever component A_4 is reset to its initial state, it reads the next input symbol, remembers it,

and waits. In this way, component A_4 is tracked over the valid computations. Moreover, the master A_1 receives information about the symbols read by A_4 and can check the number of primed symbols to be m. The phase ends successfully when A_1 has read $\$_3$ and receives the information that A_4 has read $\$_2$ in this moment, that is, both infixes have the same length ℓ. This phase takes $O(\ell)$ communications. At its end, the master A_1 is located on the symbol after $\$_3$ and components A_2 and A_4 are both located on the symbol after $\$_2$.

The third phase is used to compare the word in between $\$_2$ and $\$_3$ with the word in between $\$_3$ and $\$_4$. Similar as in the phase before, to this end, the master A_1 moves to the right while querying component A_2 in every step. Whenever component A_2 is reset to its initial state, it reads the next input symbol, remembers it, and waits. So, A_1 can check whether the currently read symbols are identical. The phase ends successfully when A_1 has read $\$_4$ and receives the information that A_2 has read $\$_3$ in this moment. Now, the master A_1 is located on the symbol after $\$_4$, A_2 is located on the symbol after $\$_3$, and A_4 still on the symbol after $\$_2$. The total number of communications in this phase is of order $O(\ell)$.

The fourth phase is used to track component A_2 to the position of A_1. So, the master A_1 loops on its position while it queries A_2 in every step. In this way, A_2 moves to the right. The phase ends when A_1 receives the information that A_2 has read $\$_4$. At this time step the master A_1 and component A_2 are located on the symbol after $\$_4$ and A_4 still on the symbol after $\$_2$. During this phase $O(\ell)$ communications take place.

The fifth and final phase is to check the suffix. The master knows that this phase starts and changes into some appropriate state in a λ-step. The situation is similar for component A_2. It is in its initial state on a symbol a for the first time. So, both synchronously start the phase. Basically, here we can use again the construction of the proof of Lemma 9. That is, the master component and component A_2 check that the lengths of a-blocks are doubling. Communication takes place at both symbols b. Reading the first b, component A_4 is queried and forced to proceed one input symbol in order to check the correct number m of a-blocks. Since component A_4 is tracked over an infix whose first m symbols are primed this can be done almost as before. Reading the second b, the master queries component A_2 to ensure that the a-blocks ended correctly. The total number of communications in this phase is of order $O(m)$. This concludes the construction of A.

The length ℓ of the valid computation of M_r on input a^m is of order $\Theta(m^2 \cdot m^r) = \Theta(m^{r+2})$ by Lemma 8. The length of an input is $n = 3\ell + 2^m - 1 + 2m + 5 \in \Theta(2^m)$. The total number of communications is of order $O(\ell) + O(\ell) + O(\ell) + O(\ell) + O(m) = O(m^{r+2})$. So, the number of communications is of order $O(\log(n)^{r+2})$. □

Lemma 16 *Let $r \geq 1$ be an integer. Then language L_r does not belong to $\mathscr{L}(O(\log(n)^r)$-DRCPCFA$(4))$.*

Proof The proof is along the line of the proof of Lemma 6. By way of contradiction, we assume that L_r is accepted by some $O(\log(n)^r)$-DRCPCFA(4).

Let z be a word in L_r whose infix $x = x_1 x_2 \cdots x_\ell$ is the valid computation of M_r on input a^m. Then $|z|$ is of order $\Theta(2^m)$ and ℓ is of order $\Theta(m^{r+2})$. We will use an incompressibility argument and choose a string $w = w_1 w_2 \cdots w_\ell \in \{0,1\}^*$ so that the Kolmogorov complexity is $C(w) \geq |w| = \ell \in \Theta(m^{r+2})$. Then the word $z' = \$_1 x \$_2 w'_1 w'_2 \cdots w'_m w_{m+1} \cdots w_\ell \$_3 w'_1 w'_2 \cdots w'_m w_{m+1} \cdots w_\ell \$_4 a^{2^0} bb a^{2^1} bb \cdots a^{2^{m-1}} bb\&$ belongs to L_r as well.

With the help of the accepting computation on z' we write down a program that uniquely reconstructs w. The order of magnitude of the size of the program is given by the product of the size of one snapshot and the number of all snapshots. Since one snapshot can be described by $O(m)$ bits and the number of snapshots is bounded by $O(m^r)$, we derive that $C(w)$ is of order $O(m^{r+1})$, a contradiction. □

Combining Lemma 15 and Lemma 16 the desired infinite hierarchy of the next theorem follows.

Theorem 17 *Let $r \geq 1$ be an integer. Then the class $\mathscr{L}(O(\log(n)^r)$-DRCPCFA$(4))$ is properly included in the class $\mathscr{L}(O(\log(n)^{r+2})$-DRCPCFA$(4))$.*

Since the proofs of Lemma 15 and Lemma 16 do not rely on a specific number of components as long as at least four components are provided, the hierarchy follows for any number of components $k \geq 4$.

Corollary 18 *Let $k \geq 4$ and $r \geq 1$ be two integers. Then the class $\mathscr{L}(O(\log(n)^r)$-DRCPCFA$(k))$ is properly included in the class $\mathscr{L}(O(\log(n)^{r+2})$-DRCPCFA$(k))$.*

References

[1] Henning Bordihn, Martin Kutrib & Andreas Malcher (2011): *Undecidability and Hierarchy Results for Parallel Communicating Finite Automata*. *Int. J. Found. Comput. Sci.* 22, pp. 1577–1592, doi:10.1142/S0129054111008891.

[2] Henning Bordihn, Martin Kutrib & Andreas Malcher (2012): *On the Computational Capacity of Parallel Communicating Finite Automata*. *Int. J. Found. Comput. Sci.* 23, pp. 713–732, doi:10.1142/S0129054112500062.

[3] Thomas Buchholz & Martin Kutrib (1998): *On time computability of functions in one-way cellular automata*. *Acta Inform.* 35, pp. 329–352, doi:10.1007/s002360050123.

[4] Ashish Choudhary, Kamala Krithivasan & Victor Mitrana (2007): *Returning and non-returning parallel communicating finite automata are equivalent*. *RAIRO Inform. Théor.* 41, pp. 137–145, doi:10.1051/ita:2007014.

[5] John E. Hopcroft & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

[6] Oscar H. Ibarra (1973): *On Two-way Multihead Automata*. *J. Comput. System Sci.* 7, pp. 28–36, doi:10.1016/S0022-0000(73)80048-0.

[7] Martin Kutrib (2008): *Cellular Automata – A Computational Point of View*. In: *New Developments in Formal Languages and Applications*, chapter 6, Springer, pp. 183–227, doi:10.1007/978-3-540-78291-9_6.

[8] Martin Kutrib (2009): *Cellular Automata and Language Theory*. In: *Encyclopedia of Complexity and System Science*, Springer, pp. 800–823, doi:10.1007/978-0-387-30440-3_54.

[9] Martin Kutrib & Andreas Malcher (2011): *Two-Party Watson-Crick Computations*. In: *Implementation and Application of Automata (CIAA 2010), LNCS* 6482, Springer, pp. 191–200, doi:10.1007/978-3-642-18098-9_21.

[10] Ming Li & Paul M. B. Vitányi (1993): *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, doi:10.1007/978-1-4757-3860-5

[11] Andreas Malcher (2002): *Descriptional Complexity of Cellular Automata and Decidability Questions*. *J. Autom., Lang. Comb.* 7, pp. 549–560.

[12] Carlos Martín-Vide, Alexandru Mateescu & Victor Mitrana (2002): *Parallel Finite Automata Systems Communicating by States*. *Int. J. Found. Comput. Sci.* 13, pp. 733–749, doi:10.1142/S0129054102001424.

[13] Victor Mitrana (2000): *On the Degree of Communication in Parallel Communicating Finite Automata Systems*. *J. Autom., Lang. Comb.* 5, pp. 301–314.

[14] Friedrich Otto (2013): *Asynchronous PC systems of pushdown automata*. In: *Language and Automata Theory and Applications (LATA 2013), LNCS* 7810, Springer, pp. 456–467, doi:10.1007/978-3-642-37064-9_40.

[15] Marcel Vollweiler (2013): *Asynchronous systems of parallel communicating finite automata*. In: *Fifth Workshop on Non-Classical Models for Automata and Applications (NCMA 2013), books@ocg.at* 294, Austrian Computer Society, Vienna, pp. 243–257.