
Two Notes on APCol Systems

Lucie Cencialová^{1,2} and Luděk Cenciala^{1,2}

¹ Research Institute of the IT4Innovations Centre of Excellence,
Faculty of Philosophy and Science, Silesian University in Opava, Czech Republic

² Institute of Computer Science
Faculty of Philosophy and Science, Silesian University in Opava, Czech Republic
lucie.cencialova@fpf.slu.cz ludek.cenciala@fpf.slu.cz

Summary. In this work, we continue our research in the field of string processing membrane systems - APCol systems. We focus on a relation of APCol systems with PM colonies - colonies whose agents can only perform point mutation transformations of the common string, in a vicinity of the agent. The second part is devoted to a connection of APCol systems and logic circuits using AND, OR and NOT gates.

1 Introduction

There are many different theoretical computational models, where are independent agents engage in a shared environment or interact with it directly. In this paper, we continue our research in the examination of connections between such models. In [26] our research started with the comparison of eco-colonies – composed from components (grammars) acting in a string, the string is also evolved by environmental rules (0L scheme), P colonies and eco-P colonies – membrane systems with one-membrane agents and a unordered environment. In this paper, we continue our study with APCol systems – the model related closely to P colonies – PM colonies and logic circuits.

APCol systems are formal models of a computing device combining properties of membrane systems and colonies - parallel distributed systems of formal grammars. Membrane systems (called P systems, [24]) are biologically inspired distributed multiset rewriting systems that are characterised by massive parallelism and simple rules. APCol systems were introduced in [4] The reader can find more information about membrane systems in [23] and in [17] can a reader find details on grammar systems theory.

An APCol system is formed from agents - collections of objects embedded in a membrane - and shared environment - string. Agents are equipped with programs composed of rules that allow agents to interact with objects that are placed inside them and they form a string. The number of objects inside each agent is set by definition and it is usually very low up to 3. The string is processed by agents

and it is used as a communication channel for agents too. Through the string the agents are able to affect the behaviour of another agent. There is a special object defined in the APCol system it is denoted by e . It has a special role: whenever it is introduced in the string, the corresponding input symbol is erased.

The activity of agents is based on rules that can be rewriting, communication or checking; these three types was introduced in [18]. Rewriting rule $a \rightarrow b$ allow agent to rewrite (evolve) one object a to object b . Both objects are placed inside the agent. Communication rule $a \leftrightarrow b$ gives to the agent the possibility to exchange object c placed inside the agent for object d from the string. A checking rule is formed from two rules r_1, r_2 of type rewriting or communication. It sets a kind of priority between rules r_1, r_2 . The agent tries to apply the first rule and if it cannot be performed, the agent executes the second rule. The rules are combined into programs in such a way that all object are affected by the execution of the rules. Consequently, the number of rules in the program is the same as the number of objects inside the agent.

The computation in APCol systems starts with an input string, representing the environment, and with each agents having only symbols e in its state. Every computational step means a maximally parallel action of the active agents: an agent is active if it is able to perform at least one of its programs, and the joint action of the agents is maximally parallel if no more active agent can be added to the synchronously acting agents. The computation ends if the input string is reduced to the empty word, there are no more applicable programs in the system, and meantime at least one of the agents is in so-called final state.

We start the paper with the necessary definitions not only of APCol systems, PM-colonies and logic circuits. We continue with the construction of an APCol system simulating a PM-colony and with the construction of APCol system suitable for simulation of logic circuits. We conclude the paper with final remarks about future work and open problems.

2 Definitions

Throughout the paper the reader is assumed to be familiar with the basics of formal language theory and membrane computing. For further details we refer to [16] and [23].

For an alphabet Σ , the set of all words over Σ (including the empty word, ε), is denoted by Σ^* . We denote the length of a word $w \in \Sigma^*$ by $|w|$ and the number of occurrences of the symbol $a \in \Sigma$ in w by $|w|_a$. For a language $L \subseteq \Sigma^*$, the set $length(L) = \{|w| \mid w \in L\}$ is called the length set of L . For a family of languages FL , the family of length sets of languages in FL is denoted by NFL .

A multiset of objects M is a pair $M = (V, f)$, where V is an arbitrary (not necessarily finite) set of objects and f is a mapping $f : V \rightarrow N$; f assigns to each object in V its multiplicity in M . The set of all multisets with the set of objects V is denoted by V° . The set V' is called the support of M and denoted by $supp(M)$.

The cardinality of M , denoted by $|M|$, is defined by $|M| = \sum_{a \in V} f(a)$. Any multiset of objects M with the set of objects $V' = \{a_1, \dots, a_n\}$ can be represented as a string w over alphabet V' with $|w|_{a_i} = f(a_i)$; $1 \leq i \leq n$. Obviously, all words obtained from w by permuting the letters can also represent the same multiset M , and ε represents the empty multiset.

2.1 APCol Systems

In the following we recall the concept of an APCol system [4].

As in the case of standard P colonies, agents of APCol systems contain objects, each being an element of a finite alphabet. With every agent, a set of programs is associated. There are two types of rules in the programs. The first one, called an evolution rule, is of the form $a \rightarrow b$. It means that object a inside of the agent is rewritten (evolved) to the object b . The second type of rules, called a communication rule, is in the form $c \leftrightarrow d$. When this rule is performed, the object c inside the agent and a symbol d in the string are exchanged, so, we can say that the agent rewrites symbol d to symbol c in the input string. If $c = \varepsilon$, then the agent erases d from the input string and if $d = \varepsilon$, symbol c is inserted into the string.

An APCol system works successfully, if it is able to reduce the given string to ε , i.e., to enter a configuration where at least one agent is in accepting state and the processed string is the empty word.

Definition 1. [4] *An APCol system is a construct*

$$\Pi = (O, e, A_1, \dots, A_n), \text{ where}$$

- O is an alphabet; its elements are called the objects,
- $e \in O$, called the basic object,
- A_i , $1 \leq i \leq n$, are agents. Each agent is a triplet $A_i = (\omega_i, P_i, F_i)$, where
 - ω_i is a multiset over O , describing the initial state (content) of the agent, $|\omega_i| = 2$,
 - $P_i = \{p_{i,1}, \dots, p_{i,k_i}\}$ is a finite set of programs associated with the agent, where each program is a pair of rules. Each rule is in one of the following forms:
 - $a \rightarrow b$, where $a, b \in O$, called an evolution rule,
 - $c \leftrightarrow d$, where $c, d \in O$, called a communication rule,
 - $F_i \subseteq O^*$ is a finite set of final states (contents) of agent A_i .

During the work of the APCol system, the agents perform programs. Since both rules in a program can be communication rules, an agent can work with two objects in the string in one step of the computation. In the case of program $\langle a \leftrightarrow b; c \leftrightarrow d \rangle$, a sub-string bd of the input string is replaced by string ac . If the program is of the form $\langle c \leftrightarrow d; a \leftrightarrow b \rangle$, then a sub-string db of the input string is replaced by string ca . That is, the agent can act only in one place in a computation step and the change of the string depends both on the order of the rules in the

program and on the interacting objects. In particular, we have the following types of programs with two communication rules:

- $\langle a \leftrightarrow b; c \leftrightarrow e \rangle$ - b in the string is replaced by ac ,
- $\langle c \leftrightarrow e; a \leftrightarrow b \rangle$ - b in the string is replaced by ca ,
- $\langle a \leftrightarrow e; c \leftrightarrow e \rangle$ - ac is inserted in a non-deterministically chosen place in the string,
- $\langle e \leftrightarrow b; e \leftrightarrow d \rangle$ - bd is erased from the string,
- $\langle e \leftrightarrow d; e \leftrightarrow b \rangle$ - db is erased from the string,
- $\langle e \leftrightarrow e; e \leftrightarrow d \rangle; \langle e \leftrightarrow e; c \leftrightarrow d \rangle, \dots$ - these programs can be replaced by programs of type $\langle e \rightarrow e; c \leftrightarrow d \rangle$.

The program is said to be *restricted* if it is formed from one rewriting and one communication rule. The APCol system is restricted if all the programs the agents have are restricted.

At the beginning of the work of the APCol system (at the beginning of the computation), there is an input string placed in the environment, more precisely, the environment is given by a string ω of objects which are different from e . This string represents the initial state of the environment. Consequently, an initial configuration of the automaton-like P colony is an $(n+1)$ -tuple $c = (\omega; \omega_1, \dots, \omega_n)$ where ω is the initial state of the environment and the other n components are multisets of strings of objects, given in the form of strings, the initial states the of agents.

A configuration of an APCoL system Π is given by $(w; w_1, \dots, w_n)$, where $|w_i| = 2$, $1 \leq i \leq n$, w_i represents all the objects placed inside the i -th agent and $w \in (O - \{e\})^*$ is the string to be processed.

At each step of the computation every agent attempts to find one of its programs to use. If the number of applicable programs is higher than one, the agent non-deterministically chooses one of them. At every step of computation, the maximal possible number of agents have to perform a program.

By applying programs, the automaton-like P colony passes from one configuration to another configuration. A sequence of configurations starting from the initial configuration is called a computation. A configuration is halting if the APCol system has no applicable program.

The result of computation depends on the mode in which the APCol system works. In the case of accepting mode, a computation is called accepting if and only if at least one agent is in final state and the string obtained is ε . Hence, the string ω is accepted by the automaton-like P colony Π if there exists a computation by Π such that it starts in the initial configuration $(\omega; \omega_1, \dots, \omega_n)$ and the computation ends by halting in the configuration $(\varepsilon; w_1, \dots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$.

The situation is different when the APCol system works in the generating mode. A computation is called successful if only if it is halting and at least one agent is in final state. The string w_F is generated by Π iff there exists computation starting in an initial configuration $(\varepsilon; \omega_1, \dots, \omega_n)$ and the computation ends by halting in the configuration $(w_F; w_1, \dots, w_n)$, where at least one of $w_i \in F_i$ for $1 \leq i \leq n$.

We denote by $APCol_{acc}R(n)$ (or $APCol_{acc}(n)$) the family of languages accepted by APCol system having at most n agents with restricted programs only (or without this restriction). Similarly we denote by $APCol_{gen}R(n)$ the family of languages generated by APCol systems having at most n agents with restricted programs only.

APCol system Π can generate or accept set of numbers $|L(\Pi)|$.

By $NAPCol_xR(n), x \in \{acc, gen\}$, the family of sets of natural numbers accepted or generated by APCol systems with at most n agents is denoted.

In [4] the authors proved that the family of languages accepted by jumping finite automata (introduced in [21]) is properly included in the family of languages accepted by APCol systems with one agent, and it is proved that any recursively enumerable language can be obtained as a projection of a language accepted by an automaton-like P colony with two agents.

In [4] the reader can find following theorems about accepting power of APCol systems:

- The family of languages accepted by APCol system with one agent properly includes the family of languages accepted by jumping finite automata.
- Any recursively enumerable language can be obtained as a projection of a language accepted by an APCol system with two agents.

The results about generative power of APCol systems are shown in [2]:

- Restricted APCol systems with only two agents working in generating mode can accept any recursively set of natural numbers. $NAPCol_{gen}R(2) = NRE$
- A family of sets of natural numbers acceptable by partially blind register machine can be generated by an APCol system with one agent with restricted programs. $NRM_{PB} \subseteq NAPCol_{gen}R(1)$

2.2 PM-colonies

In this part we recall the definition of PM-colonies that was introduced in [20].

Definition 2. A PM-colony (of degree $n; n \geq 1$) is a construct

$$\pi = (E; \#; N; >; R_1; \dots; R_n);$$

where E is an alphabet (of the environment), $\#$ is a special symbol not in E (the boundary marker of the environment), $N = \{A_1; \dots; A_n\}$ is the alphabet of agents names, $>$ is a partial order relation over N (the priority relation among agents), and $R_1; \dots; R_n$ are finite sets of action rules of the agents. The action rules can be of the following forms:

$$\begin{array}{ll}
\text{Deletion:} & (a; A_i; b) \rightarrow (\varepsilon; A_i; b), \quad \text{for } a \in E \cup N; b \in E \cup N \cup \{\#\}, \\
& (a; A_i; b) \rightarrow (a; A_i; \varepsilon), \quad \text{for } a \in E \cup N \cup \{\#\}; b \in E \cup N, \\
\text{Insertion:} & (a; A_i; b) \rightarrow (a; c; A_i; b), \quad \text{for } a; b \in E \cup N \cup \{\#\}; c \in E \cup N, \\
& (a; A_i; b) \rightarrow (a; A_i; c; b), \quad \text{for } a; b \in E \cup N \cup \{\#\}; c \in E \cup N, \\
\text{Substitution:} & (a; A_i; b) \rightarrow (c; A_i; b), \quad \text{for } b \in E \cup N \cup \{\#\}; a; c \in E, \\
& (a; A_i; b) \rightarrow (a; A_i; c), \quad \text{for } a \in E \cup N \cup \{\#\}; b; c \in E, \\
\text{Move:} & (a; A_i; b) \rightarrow (A_i; a; b), \quad \text{for } a \in E \cup N; b \in E \cup N \cup \{\#\}, \\
& (a; A_i; b) \rightarrow (a; b; A_i), \quad \text{for } a \in E \cup N \cup \{\#\}; b \in E \cup N, \\
\text{Death:} & (a; A_i; b) \rightarrow (a; \varepsilon; b), \quad \text{for } a; b \in E \cup N \cup \{\#\}.
\end{array}$$

Each action has a premise of the form $(a; A_i; b)$, it specifies the agent and its neighbouring symbols, and has a consequence where the agent is still present, with only one exception, when it disappears; the boundary marker cannot be changed or overpassed.

The state of the environment is described by a string of the form $\#w\#$, where $w \in (E \cup N)^*$. The string $pr_E(w)$ describes the environment without the agent population, $pr_N(w)$ describes the agent population. One agent can appear in several copies in the environment.

The agents act on the environment symbols as well as on other agents, in a local manner, according to the action rules; the agents can also change their position in the environment, according to the move rules, but they cannot step outside the boundary markers.

Agents in PM-colony work in parallel manner. Similarly as in the other parallel working systems, conflicts can occur between agents. If in a word $w \in (E \cup N)^*$ context overlay of two agents A_i and A_j happens or if agent A_j takes part in context of agent A_i , we call it direct conflict between agents. If in w the pairs of agents $(A_1, A_2), (A_2, A_3), \dots, (A_n, A_{n+1})$ are in direct conflict then the whole set of agents $A_1, A_2, A_3, \dots, A_n, A_{n+1}$ are in conflict. The conflict of agents $A_1, A_2, A_3, \dots, A_n, A_{n+1}$ in PM-colony can be solved by the agent with the greatest priority, which takes action. So, to solve the conflict, conflicting agents have to be ordered in such a way, that there is an agent with priority higher than all other agents in the conflict. Moreover the agent with the greatest priority occurs in the conflict set only once.

Let A be agent of PM-colony and $\#w\# = xaAby$ be a configuration of PM-colony, where $a, b \in (E \cup N) \cup \{\#\}$. This occurrence of agent A is active with respect to configuration $\#w\#$, if (1) an action rule exists, whose left side is in the form (a, A, b) , and (2) A is not conflicting with any other agent occurrence, or A has the highest priority from all agents from those in conflict. An agent occurrence is inactive, if it is not active.

A derivation step in a PM-colony denoted as \Rightarrow is a binary relation on a set of configurations. We write $\#w\# \Rightarrow \#z\#$ if and only if each active agent A in the string w replaces its context in w by corresponding rule and the resultant string is $\#z\#$. Derivation \Rightarrow^* is the reflexive and transitive closure of relation \Rightarrow . A language associated with PM-colony $\pi = (E; \#; N; >; R_1; \dots; R_n)$ and a starting state w_0 of its environment is defined as

$$L(\pi; w_0) = \{x \in E^* \mid w_0 \Rightarrow^* w; x = pr_E(w)\}$$

2.3 Logic gates and boolean circuits

Boolean circuits are a formal model of the combinational logic circuits. Circuits consist of wires able to carry one bit, and logical gates connecting such wires and computing the elementary logical functions, NOT, AND, OR. To simplify circuits used in practice, more logical gates were introduced: NOR, XOR and NAND. Value tables and graphical symbols are shown in Table 1.


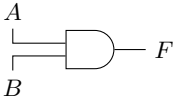
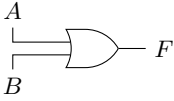
Name of logic gate	Graphical symbol	Value table															
NOT		<table border="1"> <thead> <tr> <th>A</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	F	0	1	1	0									
A	F																
0	1																
1	0																
AND		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
OR		<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F															
0	0	0															
0	1	1															
1	0	1															
1	1	1															

Table 1. Graphical symbols and value tables for logic gates

Definition 3. A Boolean circuit $\alpha = (V, E, \lambda)$ is a finite directed acyclic graph (V, E) , with set of vertices V , and set of directed edges E , and $\lambda : V \rightarrow \{I\} \cup \{AND, OR, NOT\}$ a vertex labelling, where I is a special symbol. A vertex $x \in V$ with $\lambda(x) = I$ has indegree 0 and is called an input. A vertex $y \in V$ with outdegree 0 is called an output. Vertices with labels AND and OR have indegree 2 and outdegree 1, while vertices with the label NOT have indegree and outdegree 1.

3 APCol Systems Versus PM-colonies

In this section we construct APCol system simulating derivation steps in PM-colony. The idea is to perform one derivation step of PM-colony in four phases of computation in APCol system. In the first phase one agent rewrite all occurrences of PM-colony agents into new symbols of a type $_a A_b$ where a, b is a context of current copy of agent A . In the second phase the agent check whether there is any conflict.

If the answer is positive, it tries to solve collision. The third phase is performing phase, when APCol system agent rewrite active and inactive agents into objects in the way corresponding to executed rules. If there is more than one applicable rule (PM-colony is non-deterministic) executed rule is chosen non-deterministically. In the last phase the APCol system agent non-deterministically chooses whether computation will end and erase all occurrences of PM-colony agents or the system will continue with the next derivation step - to ensure that APCol system can halt after every simulated derivation step of PM-colony.

Theorem 1. *To every PM-colony π there exist APCol system that can generate all states of PM-colony.*

Proof. Let $\pi = (E; \#; N; >; R_1; \dots; R_n)$ be a PM-colony operating on a string $w_0 \in (E \cup N)^*$. We construct APCol system $\Pi = (O, e, A_1, \dots, A_m)$, where m equals to number of rules in π plus 6, as follows:

The agent A_1 has initial configuration ee and the subset of programs to initialize simulation.

- $$\begin{array}{c} \overline{A_1} \\ 1. \langle e \rightarrow \#'; e \rightarrow T \rangle \\ 2. \langle \#' \leftrightarrow \#; T \leftrightarrow a \rangle \quad \text{for all } a \in E \cup N \cup \{\#\} \\ 3. \langle \# \rightarrow P; a \rightarrow a \rangle \\ 4. \langle P \leftrightarrow e; a \leftrightarrow T \rangle \quad \text{for all } a \in E \cup N \cup \{\#\} \\ 5. \langle T \rightarrow \uparrow; e \rightarrow e \rangle \end{array}$$

APCol system starts computation on the string $\#w_0\# = \#aw_1\#$. Agent A_1 is in the initial configuration ee . In the first step it rewrites its contents to $\#T$ using the program 1. At the second step The agent replace the first two symbols from the string by its contents. After second step the contents of the agent A_1 $\#a$ and the string in the environment is $\#Tw_1\#$. In the next step agent rewrites $\#$ to P and exchange its contents by T from a string. The configuration of the agent is eT and the string has a form $\#Paw_1\#$.

The agent A_2 is supporting agent; It generates the auxiliary objects consumed by agent A_1 . Set P_2 contains following programs:

- $$\begin{array}{c} \overline{A_2} \\ A. \langle e \rightarrow \downarrow; e \rightarrow e \rangle \\ B. \langle \downarrow \leftrightarrow P; e \rightarrow e \rangle \end{array}$$

step	string	agent A_1	agent A_2	applicable programs of A_1	applicable programs of A_2
0.	$\#aw_1\#$	ee	ee	1.	$A.$
1.	$\#aw_1\#$	$\#T$	$\downarrow e$	2.	–
2.	$\#Tw_1\#$	$\#a$	$\downarrow e$	3.	–
3.	$\#Tw_1\#$	Pa	$\downarrow e$	4.	–
4.	$\#Paw_1\#$	eT	$\downarrow e$	5.	B
5.	$\# \downarrow aw_1\#$	$\uparrow e$	Pe	6.	C

Phase 1. - Context detection

After symbol \downarrow appears in the string $\#'\downarrow aw_1\#$, agent A_1 goes through the string from the left to the right end and rewrites occurrences of PM-colony agents A by symbols ${}_aA_b$ where a, b are neighbouring symbols (aAB is substring of the input string).

A_1	A_2
6. $\langle \uparrow \leftrightarrow \downarrow; e \leftrightarrow a \rangle$ for all $a \in E \cup N$	C. $\langle P \rightarrow L'; e \rightarrow e \rangle$
7. $\langle a \leftrightarrow \uparrow; \downarrow \leftrightarrow e \rangle$ for all $a \in E$	D. $\langle L' \leftrightarrow b; e \leftrightarrow L \rangle$ for all $b \in E \cup \bar{N}_j$ $i, j \in E \cup N \cup \{\#\}$
8. $\langle a \rightarrow \bar{A}; \downarrow \rightarrow L \rangle$ for all $a \in N$	E. $\langle b \rightarrow b; L \rightarrow L_b \rangle$
9. $\langle \bar{A} \rightarrow \bar{A}; L \leftrightarrow \uparrow \rangle$	F. $\langle b \leftrightarrow L'; L_b \leftrightarrow e \rangle$
10. $\langle \bar{A} \rightarrow \bar{A}; \uparrow \leftrightarrow L' \rangle$	G. $\langle L'' \rightarrow R'; e \rightarrow e \rangle$
11. $\langle \bar{A} \rightarrow \bar{A}; L' \rightarrow L'' \rangle$	H. $\langle R' \leftrightarrow R; e \leftrightarrow c \rangle$ $c \in E \cup N$
12. $\langle \bar{A} \rightarrow \bar{A}; L'' \leftrightarrow \uparrow \rangle$	I. $\langle R \rightarrow R_c; c \rightarrow c \rangle$
13. $\langle \bar{A} \rightarrow \bar{A}; \uparrow \leftrightarrow L_{b'} \rangle$	J. $\langle R_c \leftrightarrow R''; c \leftrightarrow e \rangle$
14. $\langle \bar{A} \rightarrow {}_{b'}A; L_{b'} \rightarrow R \rangle$	K. $\langle R'' \rightarrow L'; e \rightarrow e \rangle$
15. $\langle {}_{b'}A \rightarrow {}_{b'}A; R \leftrightarrow \uparrow \rangle$	
16. $\langle {}_{b'}A \rightarrow {}_{b'}A; \uparrow \leftrightarrow R' \rangle$	
17. $\langle {}_{b'}A \rightarrow {}_{b'}A; R' \rightarrow R'' \rangle$	
18. $\langle {}_{b'}A \rightarrow {}_{b'}A; R'' \leftrightarrow \uparrow \rangle$	
19. $\langle {}_{b'}A \rightarrow {}_{b'}A; \uparrow \leftrightarrow R_{c'} \rangle$ where $b, c \in E \cup N$	
20. $\langle {}_{b'}A \rightarrow {}_{b'}A_{c'}; R_{c'} \rightarrow \downarrow \rangle$ where $b, c \in E \cup N$	
21. $\langle {}_{b'}A_{c'} \leftrightarrow e; \downarrow \leftrightarrow \uparrow \rangle$ where $b, c \in E \cup N$	
22. $\langle \downarrow \rightarrow E_x; \# \rightarrow \# \rangle$	
23. $\langle E_x \leftrightarrow \uparrow; \# \leftrightarrow e \rangle$	

Agents A_1 and A_2 helps each other to change all occurrences of PM-colony agents to more complex symbols capturing agent neighbouring symbols. In following table, $a, b \in E$ and $A \in N$.

step	string	agent A_1	agent A_2	applicable programs of A_1	applicable programs of A_2
5.	$\#'\downarrow aAbw_1\#$	$\uparrow e$	Pe	6.	$C.$
6.	$\#'\uparrow Abw_1\#$	$\downarrow a$	$L'e$	7.	–
7.	$\#'a\downarrow Abw_1\#$	$\uparrow e$	$L'e$	6.	–
8.	$\#'a\uparrow bw_1\#$	$\downarrow A$	$L'e$	8.	–
9.	$\#'a\uparrow bw_1\#$	$\overline{A}L$	$L'e$	9.	–
10.	$\#'aLbw_1\#$	$\overline{A}\uparrow$	$L'e$	–	$D.$
11.	$\#'L'bw_1\#$	$\overline{A}\uparrow$	aL	10.	$E.$
12.	$\#\uparrow bw_1\#$	$\overline{A}L'$	aL_a	11.	–
13.	$\#\uparrow bw_1\#$	$\overline{A}L''$	aL_a	12.	–
14.	$\#'L''bw_1\#$	$\overline{A}\uparrow$	aL_a	–	$F.$
15.	$\#'aL_abw_1\#$	$\overline{A}\uparrow$	$L'e$	13.	$G.$
16.	$\#'a\uparrow bw_1\#$	$\overline{A}L_a$	$R'e$	14.	–
17.	$\#'a\uparrow bw_1\#$	$_aAR$	$R'e$	15.	–
18.	$\#'aRbw_1\#$	$_aA\uparrow$	$R'e$	–	$H.$
19.	$\#'aR'w_1\#$	$_aA\uparrow$	Rb	16.	$I.$
20.	$\#'a\uparrow w_1\#$	$_aAR'$	R_bb	17.	–
21.	$\#'a\uparrow w_1\#$	$_aAR''$	R_bb	18.	–
step	string	agent A_1	agent A_2	applicable programs of A_1	applicable programs of A_2
22.	$\#'aR''w_1\#$	$_aA\uparrow$	R_bb	–	$J.$
23.	$\#'aR_bbw_1\#$	$_aA\uparrow$	$R''e$	19.	$K.$
24.	$\#'a\uparrow bw_1\#$	$_aAR_b$	$L'e$	20.	–
25.	$\#'a\uparrow bw_1\#$	$_aA_b\downarrow$	$L'e$	21.	–
26.	$\#'a\ _aA_b\downarrow bw_1\#$	\uparrow	$L'e$	6.	–

Phase 2. - Determining agents as active or inactive

In the second phase agents A_1 and A_3 read the string from right to the left and they search for conflicts. Inactive agents are marked by $_y\overline{X}_z$, active agents has the label in a form $_yX_z$.

A_1	A_3
24. $\langle \uparrow \rightarrow \uparrow; e \rightarrow \uparrow \rangle$	i. $\langle e \rightarrow \uparrow; e \rightarrow e \rangle$
25. $\langle \uparrow \rightarrow \downarrow; \uparrow \leftrightarrow \# \rangle$	ii. $\langle \uparrow \leftrightarrow E_x; e \rightarrow e \rangle$
26. $\langle \downarrow \leftrightarrow e; \# \leftrightarrow \uparrow \rangle$	iii. $\langle E_x \rightarrow \uparrow; e \rightarrow e \rangle$
	iv. $\langle \uparrow \leftrightarrow \downarrow; e \rightarrow e \rangle$
	v. $\langle \downarrow \rightarrow e; e \rightarrow e \rangle$
A_1	
To skip $a \in E$	
27. $\langle \uparrow \leftrightarrow a; e \leftrightarrow \uparrow \rangle$	
28. $\langle \uparrow \leftrightarrow e; a \leftrightarrow \uparrow \rangle$	

-
- A_1
- If A_1 consumes ${}_bA_c, A \in N; b, c \in E \cup N \cup \{\#\}$
29. $\langle \uparrow \leftrightarrow {}_bA_c; e \leftrightarrow \Downarrow \rangle$
30. $\langle {}_bA_c \rightarrow {}_bA'_c; \Downarrow \rightarrow \Downarrow_A \rangle$

$${}_bA'_c = \begin{cases} {}_bA_c & \text{for } b, c \in E \text{ or } b, c \in N \text{ and } b < A, c < A \\ {}_b\bar{A}_c & \text{otherwise} \end{cases}$$
31. $\langle \Downarrow_A \leftrightarrow \uparrow; {}_bA'_c \leftrightarrow e \rangle$
-
- A_1
- If \Downarrow_A is sent to the string and next symbol is $a \in E$
32. $\langle \uparrow \leftrightarrow a; e \leftrightarrow \Downarrow_A \rangle$
33. $\langle \Downarrow_A \rightarrow \Downarrow_A; a \rightarrow a \rangle$
34. $\langle \Downarrow_A \leftrightarrow e; a \leftrightarrow \uparrow \rangle$
-
- A_1
- If \Downarrow_A is sent to the string and next symbol is ${}_bB_c$
35. $\langle \uparrow \leftrightarrow {}_bB_c; e \leftrightarrow \Downarrow_A \rangle$
36. $\langle {}_bB_c \rightarrow {}_bB'_c; \Downarrow_A \rightarrow Y \rangle;$

$${}_bB'_c = \begin{cases} {}_bB_c & \text{for } B > A \text{ and } \{b, c \in E \text{ or } b, c \in N \text{ and } b < A, c < A\} \\ {}_b\bar{B}_c & \text{otherwise} \end{cases}$$

$$Y = \begin{cases} \Downarrow_B & \text{for } B > A \\ \Downarrow_A & \text{for } A > B \\ \Downarrow_{\{A, B\}} & \text{otherwise} \end{cases}$$
37. $\langle X \leftrightarrow \uparrow; {}_bB'_c \leftrightarrow e \rangle$
-
- A_1
- If \Downarrow_M (M is a subset of N) is sent to the string and next symbol is ${}_bB_c$
- 35'. $\langle \uparrow \leftrightarrow {}_bB_c; e \leftrightarrow \Downarrow_M \rangle$
- 36'. $\langle {}_bB_c \rightarrow {}_bB'_c; \Downarrow_M \rightarrow Y \rangle;$

$${}_bB'_c = \begin{cases} {}_bB_c & \text{if } B \text{ is greatest element of } M \cup \{B\} \text{ and} \\ & b, c \in E \text{ or } b, c \in N \text{ and} \\ & b, c \text{ is not greatest element of } M \cup \{b, c\} \\ {}_b\bar{B}_c & \text{otherwise} \end{cases}$$

$$Y = \begin{cases} \Downarrow_B & \text{for } B > A \\ \Downarrow_A & \text{for } A > B \\ \Downarrow_{\{A, B\}} & \text{otherwise} \end{cases}$$
- 37'. $\langle X \leftrightarrow \uparrow; {}_bB'_c \leftrightarrow e \rangle$
-
- A_1
- If \Downarrow_A is sent to the string and next symbol is $a \in E$
38. $\langle \uparrow \leftrightarrow a; e \leftrightarrow \Downarrow_A \rangle$
39. $\langle a \rightarrow a; \Downarrow_A \rightarrow \Downarrow \rangle;$
40. $\langle \Downarrow \leftrightarrow \uparrow; a \leftrightarrow e \rangle$

A_1

 If \Downarrow_A is sent to the string and next symbol is ${}_bB_c$
41. $\langle \Uparrow \leftrightarrow {}_bB_c; e \leftrightarrow \Downarrow_A \rangle$ 42. $\langle {}_bB_c \rightarrow {}_bB'_c; \Downarrow_A \rightarrow X \rangle;$

$${}_bB'_c = \begin{cases} {}_bB_c & \text{for } B > A \text{ and } \{b, c \in E \text{ or } b, c \cup N \text{ and } b < A, c < A\} \\ {}_b\overline{B}_c & \text{otherwise} \end{cases}$$

$$X = \begin{cases} \Downarrow_A & \text{for } A > B \\ B_A & \text{for } B > A \\ e_A & \text{otherwise} \end{cases}$$

43. $\langle X \leftrightarrow \Uparrow; {}_bB'_c \leftrightarrow e \rangle$

If the next symbol to be checked is boundary object $\#'$ the agent A_4 finishes the second phase.

 A_5

 AA. $\langle e \rightarrow D; e \rightarrow e \rangle$
AB. $\langle D \leftrightarrow \#'; e \leftrightarrow X \rangle; X \in \{\Downarrow; \Downarrow_A; \Downarrow_A\}$ AC. $\langle X \rightarrow C; \#' \rightarrow \#' \rangle$ AD. $\langle \#' \leftrightarrow D; C \leftrightarrow e \rangle$

When symbol B_A or symbol e_A appears in the string agent A_5 starts to work by consuming the symbol and replacing it by two symbols: \Uparrow_A for agent A_6 that moves right to set PM-colony agent A as inactive and the second symbol for agent A_1 to continue computation.

 A_5 a. $\langle e \rightarrow Z; e \rightarrow e \rangle$ b. $\langle Z \leftrightarrow B_A; e \rightarrow e \rangle$ c. $\langle Z \leftrightarrow e_A; e \rightarrow e \rangle$ d. $\langle B_A \rightarrow \Downarrow_B; e \rightarrow \Uparrow_A \rangle$ e. $\langle e_A \rightarrow \Downarrow; e \rightarrow \Uparrow_A \rangle$ f. $\langle \Downarrow_B \leftrightarrow Z; \Uparrow_A \leftrightarrow e \rangle$ g. $\langle \Downarrow \leftrightarrow Z; \Uparrow_A \leftrightarrow e \rangle$ A_6

 I. $\langle e \rightarrow B; e \rightarrow e \rangle$
II. $\langle B \leftrightarrow \Uparrow_A; e \leftrightarrow X \rangle;$

$$X \in E \cup \{ {}_bB_c \mid b, c \in E \cup N \cup \{\#, \#'\}, B \in N - \{A\} \}$$

III. $\langle X \leftrightarrow B; \Uparrow_A \leftrightarrow e \rangle$ IV. $\langle B \leftrightarrow \Uparrow_A; e \rightarrow {}_bA_c \rangle$ V. $\langle \Uparrow_A \rightarrow e; {}_bA_c \rightarrow {}_b\overline{A}_c \rangle$ VI. $\langle {}_b\overline{A}_c \leftrightarrow B; e \rightarrow e \rangle$

Phase 3. - Simulation of execution of the rules

This phase starts with arrows change. Instead of \Uparrow and \Downarrow the agent A_1 uses \Uparrow and \Downarrow . The change is done by agent A_1 and A_5 .

A_1	A_5
44. $\langle \uparrow \leftrightarrow C; e \rightarrow e \rangle$	h. $\langle Z \leftrightarrow \#'; e \leftrightarrow \uparrow \rangle$
45. $\langle C \rightarrow \downarrow; e \rightarrow e \rangle$	i. $\langle \#' \rightarrow \#'; \uparrow \rightarrow \uparrow \rangle$
46. $\langle \downarrow \leftrightarrow \uparrow; e \rightarrow e \rangle$	j. $\langle \#' \leftrightarrow Z; \uparrow \leftrightarrow e \rangle$

The agent A_1 reads the string from the left to the right, it skips symbols from $E \cup N$, changes ${}_b\bar{A}_c$ to A and sends messages to performing agent to add, move, rewrite or delete symbols.

A_1
47. $\langle \uparrow \leftrightarrow \downarrow; e \leftrightarrow a \rangle \quad a \in E \cup N$
48. $\langle \uparrow \leftrightarrow \downarrow; e \rightarrow {}_b\bar{A}_c \rangle$
49. $\langle a \leftrightarrow \uparrow; \downarrow \leftrightarrow e \rangle$
50. $\langle \downarrow \rightarrow \downarrow; {}_b\bar{A}_c \rightarrow A \rangle$
51. $\langle A \leftrightarrow \uparrow; \downarrow \leftrightarrow e \rangle$
52. $\langle \uparrow \leftrightarrow \downarrow; e \rightarrow {}_bA_c \rangle$

Using the program 52 agent A_1 consumes symbol corresponding to active agent. For every rule of PM-colony there exists sets of programs of agent A_1 and one agent performing the action.

If the rule is deletion of the form

$$(a; A_i; b) \rightarrow (\varepsilon; A_i; b), \text{ for } a \in E \cup N; b \in E \cup N \cup \{\#\},$$

$$(a; A_i; b) \rightarrow (a; A_i; \varepsilon), \text{ for } a \in E \cup N \cup \{\#\}; b \in E \cup N,$$

the programs are following:

A_1
53. $\langle {}_aA_b \rightarrow A; \downarrow \rightarrow D_{xy} \rangle$ xy is La or Rb depending on which side the symbol have to be deleted
54. $\langle A \leftrightarrow e; D_{xy} \leftrightarrow \uparrow \uparrow \rangle$

A_{del}
A1. $\langle e \rightarrow J; e \rightarrow e \rangle$
A2. $\langle J \leftrightarrow D_{xy}; e \rightarrow e \rangle;$
A3. $\langle D_{La} \rightarrow d_{La}; e \leftrightarrow \downarrow \rangle$
A4. $\langle d_{La} \leftrightarrow A; \downarrow \leftrightarrow J \rangle$
A5. $\langle A \rightarrow A; J \rightarrow J_0 \rangle$
A6. $\langle J_0 \leftrightarrow d_{La}; A \leftrightarrow e \rangle$
A7. $\langle d_{La} \leftrightarrow a'; e \leftrightarrow J_0 \rangle \quad a'$ is a if $a \in E$ or a' is ${}_a\bar{a}_A$
A8. $\langle J_0 \rightarrow J_1; a' \rightarrow e \rangle$
A9. $\langle J_1 \rightarrow J_2; e \leftrightarrow d_{La} \rangle$
A10. $\langle J_2 \rightarrow J; d_{La} \rightarrow e \rangle$
A11. $\langle D_{Rb} \rightarrow d_{Rb}; e \leftrightarrow \downarrow \rangle$
A12. $\langle \downarrow \leftrightarrow J; d_{Rb} \leftrightarrow b' \rangle \quad b'$ is b if $b \in E$ or b' is ${}_A\bar{b}_d$
A13. $\langle b' \rightarrow e; J \rightarrow J_0 \rangle$
A14. $\langle J_0 \rightarrow J_1; e \leftrightarrow d_{Rb} \rangle$
A15. $\langle J_1 \rightarrow J; d_{Rb} \rightarrow e \rangle$

If the rule is insertion of the form

$$(a; A_i; b) \rightarrow (a; c; A_i; b), \text{ for } a; b \in E \cup N \cup \{\#\}; c \in E \cup N,$$

$$(a; A_i; b) \rightarrow (a; A_i; c; b), \text{ for } a; b \in E \cup N \cup \{\#\}; c \in E \cup N,$$

the programs are following:

A_1 55. $\langle {}_a A_b \rightarrow A; \Downarrow \rightarrow I_{xy} \rangle$ xy is Lc or Rc depending on which side the symbol have to be inserted56. $\langle A \leftrightarrow e; I_{xy} \leftrightarrow \Uparrow \rangle$ A_{ins} A1. $\langle e \rightarrow J; e \rightarrow K \rangle$ A2. $\langle J \leftrightarrow A; K \leftrightarrow I_{Lc} \rangle$; A13. $\langle J \leftrightarrow I_{Rc}; K \rightarrow c \rangle$;A3. $\langle I_{Lc} \rightarrow I_{Lc}^0; A \rightarrow A \rangle$ A14. $\langle I_{Rc} \rightarrow I_{Rc}^0; c \rightarrow c \rangle$ A4. $\langle I_{Lc}^0 \leftrightarrow J; A \leftrightarrow K \rangle$ A15. $\langle c \leftrightarrow J; I_{Rc}^0 \leftrightarrow e \rangle$ A5. $\langle J \leftrightarrow I_{Lc}^0; K \rightarrow c \rangle$ A16. $\langle J \leftrightarrow I_{Rc}^0; e \rightarrow \Downarrow \rangle$ A6. $\langle I_{Lc}^0 \rightarrow I_{Lc}^1; c \rightarrow c \rangle$ A17. $\langle I_{Rc}^0 \rightarrow K; \Downarrow \leftrightarrow J \rangle$ A7. $\langle c \leftrightarrow J; I_{Lc}^1 \leftrightarrow e \rangle$ A8. $\langle J \leftrightarrow I_{Lc}^1; e \leftrightarrow A \rangle$ A9. $\langle I_{Lc}^1 \rightarrow I_{Lc}^2; A \rightarrow A \rangle$ A10. $\langle A \leftrightarrow J; I_{Lc}^2 \leftrightarrow e \rangle$ A11. $\langle J \leftrightarrow I_{Lc}^2; e \rightarrow \Downarrow \rangle$ A12. $\langle I_{Lc}^2 \rightarrow K; \Downarrow \leftrightarrow J \rangle$

If the rule is substitution of the form

 $(a; A_i; b) \rightarrow (c; A_i; b)$, for $b \in E \cup N \cup \{\#\}$; $a; c \in E$, $(a; A_i; b) \rightarrow (a; A_i; c)$, for $a \in E \cup N \cup \{\#\}$; $b; c \in E$,

the programs are following:

 A_1 57. $\langle {}_a A_b \rightarrow A; \Downarrow \rightarrow S_{xy} \rangle$ xy is Lc or Rc depending on which side the symbol have to be replaced58. $\langle A \leftrightarrow e; S_{xy} \leftrightarrow \Uparrow \rangle$ A_{subs} A1. $\langle e \rightarrow J; e \rightarrow e \rangle$ A2. $\langle J \leftrightarrow S_{Lc}; e \rightarrow e \rangle$; A11. $\langle J \leftrightarrow S_{Rc}; e \rightarrow e \rangle$;A3. $\langle S_{Lc} \rightarrow s_{Lc}; e \rightarrow \Downarrow \rangle$ A12. $\langle S_{Rc} \rightarrow s_{Rc}; e \rightarrow c \rangle$ A4. $\langle s_{Lc} \leftrightarrow A; \Downarrow \leftrightarrow J \rangle$ A13. $\langle c \leftrightarrow J; s_{Rc} \leftrightarrow b \rangle$ A5. $\langle A \rightarrow A; J \rightarrow J_0 \rangle$ A14. $\langle J \rightarrow J^0; b \rightarrow \Downarrow \rangle$ A6. $\langle J_0 \leftrightarrow a; A \leftrightarrow s_{Lc} \rangle$ A15. $\langle J^0 \rightarrow J^1; \Downarrow \leftrightarrow s_{Rc} \rangle$ A7. $\langle s_{Lc} \rightarrow s_{Lc}^0; a \rightarrow c \rangle$ A16. $\langle J^1 \rightarrow J; s_{Rc} \rightarrow e \rangle$ A8. $\langle s_{Lc}^0 \leftrightarrow J_0; c \leftrightarrow e \rangle$ A9. $\langle J^0 \rightarrow J^1; e \leftrightarrow s_{Lc}^0 \rangle$ A10. $\langle J_1 \rightarrow J; s_{Lc}^0 \rightarrow e \rangle$

If the rule is move of the form

 $(a; A_i; b) \rightarrow (A_i; a; b)$, for $a \in E \cup N; b \in E \cup N \cup \{\#\}$, $(a; A_i; b) \rightarrow (a; b; A_i)$, for $a \in E \cup N \cup \{\#\}; b \in E \cup N$,

the programs are following:

 A_1 59. $\langle {}_a A_b \rightarrow A; \Downarrow \rightarrow M_{xy} \rangle$ xy is LA or RA depending on which side the symbol have to be moved60. $\langle A \leftrightarrow e; M_{xy} \leftrightarrow \Uparrow \rangle$

A_{mov}

-
- A1. $\langle e \rightarrow J; e \rightarrow e \rangle$
A2. $\langle J \leftrightarrow M_{LA}; e \rightarrow e \rangle$; A10. $\langle J \leftrightarrow M_{RA}; e \rightarrow e \rangle$;
A3. $\langle M_{LA} \rightarrow m_{LA}; e \rightarrow \Downarrow \rangle$ A11. $\langle M_{RA} \rightarrow m_{RA}; e \rightarrow e \rangle$
A4. $\langle m_{LA} \leftrightarrow A; \Downarrow \leftrightarrow J \rangle$ A12. $\langle e \leftrightarrow J; m_{RA} \leftrightarrow b' \rangle$ b' is b if $b \in E$ or b' is $A\bar{b}_d$
A5. $\langle A \rightarrow A; J \rightarrow J_0 \rangle$ A13. $\langle J \rightarrow J^0; b' \rightarrow O \rangle$ O is b if $b \in E$ or O is B
A6. $\langle A \leftrightarrow a; J_0 \leftrightarrow m_{LA} \rangle$ A14. $\langle O \leftrightarrow A; J_0 \leftrightarrow m_{RA} \rangle$
A7. $\langle m_{LA} \rightarrow m_{LA}^0; a \rightarrow a \rangle$ A15. $\langle A \rightarrow A; m_{RA} \rightarrow \Downarrow \rangle$
A8. $\langle a \leftrightarrow J_0; m_{LA}^0 \rightarrow e \rangle$ A16. $\langle A \leftrightarrow e; \Downarrow \leftrightarrow J_0 \rangle$
A9. $\langle J^0 \rightarrow J; e \rightarrow e \rangle$ A17. $\langle J_0 \rightarrow J; e \rightarrow e \rangle$

If the rule is death of the form $(a; A_i; b) \rightarrow (a; \varepsilon; b)$, for $a; b \in E \cup N \cup \{\#\}$ the programs are following:

 A_1

-
61. $\langle {}_a A_b \rightarrow e; \Downarrow \leftrightarrow \Uparrow \rangle$

In the last phase agent A_1 detects that the whole string is rewritten and it is time to non-deterministically choose if computation will be halted or will continue by simulating of following derivation step.

 A_1

-
62. $\langle \Uparrow \leftrightarrow \Downarrow; e \leftrightarrow \# \rangle$
63. $\langle \Downarrow \rightarrow U; \# \leftrightarrow \Uparrow \rangle$
64. $\langle U \rightarrow V; \Uparrow \rightarrow \Downarrow \rangle$
65. $\langle U \rightarrow F; \Uparrow \rightarrow F \rangle$
66. $\langle V \leftrightarrow \#'; \Downarrow \leftrightarrow e \rangle$
67. $\langle \#' \leftrightarrow V; e \rightarrow \uparrow \rangle$
68. $\langle V \rightarrow e; \uparrow \rightarrow \uparrow \rangle$

The computation of the APCol system can halt only when in corresponding state of the PM-colony there is no applicable rule or in the case that program 56. is executed.

4 Boolean Circuits and APCol Systems

In this part, we show how APCol systems can simulate the functioning of logic gates. The input is always inserted into the string. Because there is no inner structure in the APCol systems, we use direct addressing. It means that every input symbol obtains index determining which agent will consume it. The result will be find on the string at the end of computation.

Theorem 2. *The functioning of logic gates NOT, OR and AND can be simulated by APCol systems with only one agent.*

Proof. The first we show how APCol system for logic gate NOT is constructed. Let NOT gate has an input with index i . We can simulate NOT gate with only one agent with following restricted programs:

1. $\langle e \leftrightarrow 0_i; e \rightarrow 1_{out} \rangle$
2. $\langle 0_i \rightarrow e; 1_{out} \leftrightarrow e \rangle$
3. $\langle e \leftrightarrow 1_i; e \rightarrow 0_{out} \rangle$
4. $\langle 1_i \rightarrow e; 0_{out} \leftrightarrow e \rangle$

Formally, we define APCol system $\Pi_{NOT} = (O, e, A_{NOT})$, where $O = \{e, 1_i, 0_i, 1_{out}, 0_{out}\}$, $A_{NOT} = (ee, P_{AND})$, the set of programs is described above and initial string is formed from only one symbol 1_i or 0_i . The result is obtained after computation halts and it is placed in the string.

Simulation of the AND gate is done by APCol system with only one agent and following programs:

5. $\langle e \leftrightarrow 0_i; e \leftrightarrow 0_i \rangle$
6. $\langle e \leftrightarrow 1_i; e \leftrightarrow 0_i \rangle$
7. $\langle e \leftrightarrow 1_i; e \leftrightarrow 0_i \rangle$
8. $\langle e \leftrightarrow 1_i; e \leftrightarrow 1_i \rangle$
9. $\langle 0_i \rightarrow 0_{out}; 0_i \rightarrow e \rangle$
10. $\langle 1_i \rightarrow 0_{out}; 0_i \rightarrow e \rangle$
11. $\langle 1_i \rightarrow 1_{out}; 1_i \rightarrow e \rangle$

These programs can work only for boolean circuit with only one gate, because we expect that symbols of input are neighbouring. If we change the programs in such a way that they contains at most one communication rule. In this case they do not use context (one agent changes only one symbol on the string in one step).

12. $\langle e \leftrightarrow 0_i; e \rightarrow e \rangle$
13. $\langle e \leftrightarrow 1_i; e \rightarrow e \rangle$
14. $\langle e \leftrightarrow 0_i; 0_i \rightarrow 0 \rangle$
15. $\langle e \leftrightarrow 1_i; 1_i \rightarrow 1 \rangle$
16. $\langle e \leftrightarrow 1_i; 0_i \rightarrow 0 \rangle$
17. $\langle e \leftrightarrow 0_i; 1_i \rightarrow 1 \rangle$
18. $\langle 0_i \rightarrow 0_{out}; 0 \rightarrow e \rangle$
19. $\langle 1_i \rightarrow 1_{out}; 1 \rightarrow e \rangle$
20. $\langle 1_i \rightarrow 0_{out}; 0 \rightarrow e \rangle$
21. $\langle 0_i \rightarrow 0_{out}; 1 \rightarrow e \rangle$
22. $\langle 0_{out} \leftrightarrow e; e \rightarrow e \rangle$
23. $\langle 1_{out} \leftrightarrow e; e \rightarrow e \rangle$

Formally, we define APCol system $\Pi_{AND} = (O, e, A_{AND})$, where $O = \{e, 1_i, 0_i, 1_{out}, 0_{out}\}$, $A_{AND} = (ee, P_{AND})$, the set of programs is formed from programs 12.-23. and initial string contains only one symbol 1_i or 0_i . The result is obtained after computation halts and it is placed in the string.

Simulation of the OR gate is done with programs very similar to programs of agent simulating AND gate. It is done by APCol system with only one agent and following programs:

24. $\langle e \leftrightarrow 0_i; e \rightarrow e \rangle$
25. $\langle e \leftrightarrow 1_i; e \rightarrow e \rangle$
26. $\langle e \leftrightarrow 0_i; 0_i \rightarrow 0 \rangle$
27. $\langle e \leftrightarrow 1_i; 1_i \rightarrow 1 \rangle$
28. $\langle e \leftrightarrow 1_i; 0_i \rightarrow 0 \rangle$
29. $\langle e \leftrightarrow 0_i; 1_i \rightarrow 1 \rangle$
30. $\langle 0_i \rightarrow 0_{out}; 0 \rightarrow e \rangle$
31. $\langle 1_i \rightarrow 1_{out}; 1 \rightarrow e \rangle$
32. $\langle 1_i \rightarrow 1_{out}; 0 \rightarrow e \rangle$
33. $\langle 0_i \rightarrow 1_{out}; 1 \rightarrow e \rangle$
34. $\langle 0_{out} \leftrightarrow e; e \rightarrow e \rangle$
35. $\langle 1_{out} \leftrightarrow e; e \rightarrow e \rangle$

Formally, we define APCol system $\Pi_{OR} = (O, e, A_{OR})$, where $O = \{e, 1_i, 0_i, 1_{out}, 0_{out}\}$, $A_{OR} = (ee, P_{AND})$, the set of programs is formed from programs 24.-35. and initial string contains only one symbol 1_i or 0_i . The result is also obtained after computation halts and it is placed in the string.

We can proceed to formulate a Boolean circuit simulation theorem:

Theorem 3. *For every Boolean circuit there exists an APCol system that can simulate its functioning and for input string formed from input signals of Boolean circuit APCol system generates correct output.*

Let $\alpha = (V, E, \lambda)$ be a boolean circuit. We associate with each gate a label from a set $\{1, \dots, |V|\}$. Let (x_1, \dots, x_p) be the vector of input signals for the boolean circuit. The corresponding initial string of APCol system is formed from objects $x_{0j}; 1 \leq j \leq p$. For simulation of the whole boolean circuit, we cannot only put the corresponding agents together. The output of one agent can be the input for

more agents. To obtain more input symbols we modify input objects consumed by agents (we use X'_i instead of X_i), output objects are in a form X_i – output from the agent i . We also add one more agent generating inputs for those agents which are connected with output of i -th agent. We deal with three cases: i -th agent is connected with one agent, with two agents or with m agents, $m > 2$. Let y be the index of agent with output and z if for $1 \leq i \leq m$ are indices of agents with input connected to agent y or corresponding to the initial input; $X \in \{0, 1\}$. We construct programs for agent $A_{|V|+1} = (ee, P_{|V|+1})$ generating inputs.

Programs for three variants of generated number of inputs

One input	Two inputs
36. $\langle e \leftrightarrow X_y; e \rightarrow X'_{z1} \rangle$	38. $\langle e \leftrightarrow X_y; e \rightarrow X'_{z1} \rangle$
37. $\langle X'_{z1} \leftrightarrow e; X_y \rightarrow e \rangle$	39. $\langle X'_{z1} \leftrightarrow e; X_y \rightarrow X'_{z2} \rangle$
	40. $\langle X'_{z2} \leftrightarrow e; e \rightarrow e \rangle$

m inputs; $m > 2$

41. $\langle e \leftrightarrow X_y; e \rightarrow X'_{z1} \rangle$
42. $\langle X'_{z1} \leftrightarrow e; X_y \rightarrow X_{2y} \rangle$
43. $\langle e \rightarrow X'_{z2}; X_{2y} \rightarrow X_{2y} \rangle$
44. $\langle X'_{z2} \leftrightarrow e; X_{2y} \rightarrow X_{3y} \rangle$
45. $\langle e \rightarrow X'_{zk}; X_{ky} \rightarrow X_{ky} \rangle$ for $2 < k \leq m$
46. $\langle X'_{zk} \leftrightarrow e; X_{ky} \rightarrow X_{(k+1)y} \rangle$ for $2 < k \leq m - 1$
47. $\langle X'_{zm} \leftrightarrow e; X_{my} \rightarrow e \rangle$

The agents associated to logic gates can consume only input symbols of a type X'_i , where X is 0 or 1 and i is label of gate. At the beginning of computation there are only symbols of type X_i present in the input string of APCol system. The only agent with applicable program is agent $A_{|V|+1}$ – program 36., 38. or 41. In the next step agent $A_{|V|+1}$ put the first input symbol (X'_zk), where zk is label of corresponding gate. From this step agents associated with gates started to do their work – rewriting their own inputs into outputs – and agent $A_{|V|+1}$ is rewriting their output to inputs. If the boolean circuit is defined correctly, at the end of computation there are only symbols X'_{out} in the string corresponding to the output of boolean circuit.

5 Conclusion

In this paper we continue with our research devoted to relationship of APCol systems which are devices derived from P colonies and P systems in general with classical formal models used not only in theoretical computer science. For this paper we have chosen PM-colonies and boolean circuits. We showed that APCol systems with two agents can simulate PM-colonies with strict conflict solving. In the second part we showed that even APCol systems are without inner structure they can substitute boolean circuits.

Acknowledgements.

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602, and by the Silesian University in Opava under the Student Funding Scheme, project SGS/13/2016.

References

1. Ceterchi, R., Sburlan, D.: Simulating Boolean Circuits with P Systems, pp. 104–122. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-24619-0_8
2. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: A class of restricted p colonies with string environment. *Natural Computing* 15(4), 541–549 (2016), <http://dx.doi.org/10.1007/s11047-016-9564-3>
3. Cienciala, L., Ciencialová, L., Kelemenová, A.: On the number of agents in P colonies. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *Workshop on Membrane Computing. Lecture Notes in Computer Science*, vol. 4860, pp. 193–208. Springer (2007)
4. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E.: Towards on P colonies processing strings. In: *Proc. BWMC 2014, Sevilla, 2014*. pp. 102–118. Fénix Editora, Sevilla, Spain (2014)
5. Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E., Vaszil, G.: Pcol automata: Recognizing strings with P colonies. In: *Proc. BWMC 2010, Sevilla, 2010*. pp. 65–76. Fénix Editora, Sevilla, Spain (2010)
6. Cienciala, L., Ciencialová, L., Kelemenová, A.: Homogeneous P colonies. *Computing and Informatics* 27(3+), 481–496 (2008)
7. Ciencialová, L.: Variation on the theme: P colonies. In: *Proceedings of the International Workshop on Formal Models Prerov, the Czech Republic, April 25–27, 2006*. pp. 27–34. CEUR Workshop Proceedings, Ostrava (2006)
8. Ciencialová, L., Csuhaj-Varjú, E., Kelemenová, A., Vaszil, G.: Variants of P colonies with very simple cell structure. *International Journal of Computers, Communication and Control* 4(3), 224–233 (2009)
9. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A.: Computing with cells in environment: P colonies. *Multiple-Valued Logic and Soft Computing* 12(3-4), 201–215 (2006)
10. Csuhaj-Varjú, E., Margenstern, M., Vaszil, G.: P colonies with a bounded number of cells and programs. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) *Workshop on Membrane Computing. Lecture Notes in Computer Science*, vol. 4361, pp. 352–366. Springer (2006)
11. Dassow, J., Păun, Gh.: *Regulated Rewriting in Formal Language Theory*, EATCS Monographs in Theoretical Computer Science, vol. 18. Springer-Verlag Berlin (1989)
12. Freund, R., Oswald, M.: P colonies working in the maximally parallel and in the sequential mode. In: in G. Ciobanu, Gh. Păun, *Pre-Proc. of First International Workshop on Theory and Application of P Systems*, Timisoara, Romania, September 26–27. pp. 49–56 (2005)
13. Freund, R., Oswald, M.: P colonies and prescribed teams. *Int. J. Comput. Math.* 83(7), 569–592 (2006)

14. Gheorghe, M., Konur, S., Ipate, F.: Kernel P Systems and Stochastic P Systems for Modelling and Formal Verification of Genetic Logic Gates, pp. 661–675. Springer International Publishing, Cham (2017), [http://dx.doi.org/ 10.1007/978-3-319-33924-5_25](http://dx.doi.org/10.1007/978-3-319-33924-5_25)
15. Greibach, S.: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science* 7(3), 311 – 324 (1978)
16. Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation* (3rd Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2006)
17. Kelemen, J., Kelemenová, A.: A grammar-theoretic treatment of multiagent systems. *Cybern. Syst.* 23(6), 621–633 (November 1992)
18. Kelemen, J., Kelemenová, A., Păun, Gh.: Preview of P colonies: A biochemically inspired computing model. In: *Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX)*. pp. 82–86. Boston, Massachusetts, USA (September 12-15 2004)
19. Kelemenová, A.: P colonies. In: Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) *The Oxford Handbook of Membrane Computing*, pp. 584–593. Oxford University Press (2010)
20. Martín-Vide, C., Păun, G.: Pm-colonies. *Computers and Artificial Intelligence* 17(6) (1998)
21. Meduna, A., Zemek, P.: Jumping finite automata. *Int. J. Found. Comput. Sci.* 23(7), 1555–1578 (2012)
22. Minsky, M.L.: *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1967)
23. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA (2010)
24. Păun, G.: Computing with membranes. *Journal of Computer and System Sciences* 61(1), 108–143 (2000), <http://www.sciencedirect.com/science/article/pii/S0022000099916938>
25. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Springer-Verlag New York, Inc., New York, NY, USA (1997)
26. Vavrecková, Š., Cienciala, L., Ciencialová, L.: *About Models Derived from Colonies*, pp. 369–386. Springer International Publishing, Cham (2015), http://dx.doi.org/10.1007/978-3-319-28475-0_25