

On P Colonies, a Simple Bio-Chemically Inspired Model of Computation

Jozef Kelemen^{1,2}, Alica Kelemenová^{1,3}

¹Institute of Computer Science, Silesian University
Opava, Czech Republic, and

²VSM School of Management, City University
Bratislava, Slovak Republic

³Department of Computer Science, Catholics University,
Ružomberok, Slovak Republic
{kelemen, kelemenova}@fpf.slu.cz

Abstract: The contribution presents the first overview of so called P colonies, a bio-chemically inspired formal model of a computing device, and the basic computational and language-theoretic properties derived from that model.

Keywords: P colony, membrane system, computation, formal model.

1 Introduction, Motivation, and an Intuitive Picture of P Colonies

The main reason leading to a proposal of the formal framework of so-called *P colony* (Kelemen et al., 2004) was to provide as simple as possible, purely symbolic and computationally relevant, theoretically naturally tractable formal model of phenomena appearing in real (bio-)chemical units like molecules and cells. In the sense of their required simplicity P colonies remind the so-called *colonies* of – in certain sense simplest – formal grammars defined in (Kelemen, Kelemenova, 1992), and presented e.g. in (Csuhaĵ-Varju et al., 1994).

Different types of elementary entities are used in the model. We list them and give a short motivation for their inclusion into the model.

The first type of elementary entities are the so-called *objects*, which correspond e.g. to chemical compounds of (bio-)chemistry (like ions, atoms, molecules, and macromolecules). In P colonies, all these compounds are modeled by objects with no internal structure. In this sense objects will be the simplest constituting elements in our model. In our formalism, we use letters of the Latin alphabet to denote objects.

Objects are supposed to be grouped into internally structured *active entities* (molecules, cells, or, more generally speaking, *agents*...) or they can appear in their completely passive *environment* in which these entities act. We assume that the environment contains several copies of the basic environmental objects denoted by *e*, as many as needed to perform a computation. Moreover the environment can contain also finite

multi-sets¹ of non-basic objects, and each entity contains a fixed (intuitively small) number of (possibly identical) objects.

The number of objects inside each active entity called an *agent* is same for each agent and it determines the *capacity* of the P colony. In the simplest case we suppose exactly two objects inside each agent at every moment of its activity.

The second type of elementary entities which are included into P colonies and which are associated with agents are *programs*. Programs correspond to the simplest (bio-chemical) reactions running on the objects and producing other objects.

Each program consists of one *rule* for each object inside the agent. So the number of rules in each program is determined by the capacity of the P system, too. The program associated with the agent with c objects uses all its rules in parallel to all actual objects, and each rule rewrites different object of the agent.

We distinguish two types of rules, namely the *evolution rules* acting inside agents, and the *communication rules* providing elementary interactions between the agents and the environment.

The *evolution rules* are again as simple as possible. Each of them is able to rewrite one object contained in the agent into another object which will remain inside this agent. This process of rewriting an object, say a , into another object, say b , will be denoted by $a \rightarrow b$.

The *communication rules* are able to influence the environment by an agent and vice versa, the agent by its environment. In as simple as possible way it consists in the mutual exchanging of one object inside the agent, and one object in its environment. Executing a communication rule, the object, say c , appearing in the agent will move into the environment, and another object, say d , appearing in the environment, will move into the agent. We will denote such a rule by $c \leftrightarrow d$, where the object appearing in the agent is written in the left side of the relation \leftrightarrow .

We extend the abilities of agents by following computationally non-trivial way: assume that communication rule can be chosen from two possibilities with the first one having higher priority. We get *checking rules* as a nontrivial extension of the *communication rules*. From two communication rules associated with the agent that one with the greater priority has to be active. The agent checks the possibility to execute the communication rule having higher priority. Otherwise, the second communication rule can be treated. So, suppose two communication rules in an agent: the rule $c \leftrightarrow d$, and the rule $c' \leftrightarrow d'$ and a checking rule being a pair $c \leftrightarrow d / c' \leftrightarrow d'$.

First, the agent checks the possibility to use rule $c \leftrightarrow d$, i.e. the appearance of c as an internal object of the agent and d in the environment. In positive case c must be replaced by d and vice versa. If this is not the case, the rule $c \leftrightarrow d$ cannot be realized, then the mutual exchange of the internal object c' with the outer object d' of the environment can be executed. A P colony with checking rules will be called also a P colony with priority.

The agent of a P colony can have several programs and the number of rules in each program is determined by the capacity of the P colony and corresponds to the number of objects in each agent, which will remain constant during the whole computation.

P colony starts a computation with the special basic objects only in the environment and in each agent. The number of objects e in the agent is given by the capacity of a P colony. The result of the computation is the number of distinguished symbols in the

¹ By a multi-set we mean a set in which multiple occurrences of elements are allowed and treated.

environment at the end of the computation, when no agent is able to realize any of its programs.

2 How P Colonies Compute – Simple Examples

We will demonstrate how a particular P colony looks like, and how it computes. We present a P colony which executes the arithmetic addition of the number 1, the operation “+1”.

Positive integer n will be represented in our P colony as n occurrences of a given object, denoted in our case by f , inside of the environment of the P colony. Our goal is to put the suitable agents into this environment, the activity of which will increase the number of objects f in the environment to $n+1$. Moreover we will assume that the requirement to solve our elementary problem +1 will be denoted by the object l_+ at the start of the computation in the environment and after the ending computation P colony halts.

Suppose our specific environment contains several copies of the object e , exactly n copies of the object f and one copy of the object l_+ which represents the additions. The P colony will contain exactly one agent (in the environment) with two basic objects e and two programs:

$$\langle e \rightarrow f; e \leftrightarrow l_+ \rangle, \langle l_+ \rightarrow e; f \leftrightarrow e \rangle$$

each of them with one evolution rule and one communication rule. We will represent such an agent in the following form:

$$AG_{+1} = (\{e, e\}, \langle e \rightarrow f; e \leftrightarrow l_+ \rangle, \langle l_+ \rightarrow e; f \leftrightarrow e \rangle).$$

So, our P colony is represented by the following items:

- the set of *objects* $\{e, f, l_+\}$, the *alphabet*
- two distinguished objects, e and f from the alphabet; e is called the *basic object*, and f is the *final object*, and
- one agent; in our specific case it consists of two basic objects e, e , and three *programs*, each composed from two rules, evolution rule (of type $a \rightarrow b$), and communication rule (of type $c \leftrightarrow d$).

We demonstrate the behavior of the P colony starting, as assumed above with n occurrences of the final object f and exactly one object l_+ in the environment. The computation will end and stop with one more occurrence of f in the environment, which indicates the result of the operation +1.

Let the P colony start to work. Only the program $\langle e \rightarrow f; e \leftrightarrow l_+ \rangle$ can be used for the pair of objects e of the agent AG_{+1} . It changes one of its internal symbols e to the internal symbol f , and executes the mutual exchange of the object e from the agent and the object l_+ from the environment. The objects l_+ and f are now in agent AG_{+1} and the environment contains n copies of the object f , several copies of the object e (and no other object).

In the next step, the program $\langle l_+ \rightarrow e; f \leftrightarrow e \rangle$ is used by the agent AG_{+1} with objects l_+ and f . The result gives a pair of objects e inside the AG_{+1} , and another f in the environment. Non of the programs of the agent AG_{+1} can be applied because of the absence of the object l_+ . The computation halts, and the outer environment contains the expected result.

We will continue with another example, a P colony which executes the arithmetic subtraction of the number 1, the operation “-1”.

Positive integer n will be again represented in our P colony as n occurrences of the object f , inside of the environment of the P colony. Our goal is to put the suitable agent into this environment, the activity of which will

- decrease the number of objects f in the environment to $n-1$ for each positive n and will produce the object l_p and move it to the environment, or
- it gives object l_z to the environment in the case there was not f in the environment.

So the agent leaves the information, represented by object l_p or l_z , in the environment whether there was any f in the environment or not. The requirement to perform subtraction of “-1” will be denoted by the object L in the environment at the beginning of a computation.

Suppose our specific environment contains several copies of the object e , exactly n copies of the object f and one copy of the object L , which represents the subtraction. Our P colony will contain exactly one agent (in the environment) with two basic objects e and five programs:

$$\begin{aligned} &\langle e \rightarrow e; e \leftrightarrow L \rangle, \langle L \rightarrow l_p; e \leftrightarrow f/e \leftrightarrow e \rangle, \\ &\langle f \rightarrow e; l_p \leftrightarrow e \rangle, \langle l_p \rightarrow l_z; e \leftrightarrow e \rangle, \langle e \rightarrow e; l_z \leftrightarrow e \rangle, \end{aligned}$$

each of them containing one evolution rule and one communication or checking rule. We will represent such an agent in the following form:

$$\text{AG}_{-1} = (\{e, e\}, \langle e \rightarrow e; e \leftrightarrow L \rangle, \langle L \rightarrow l_p; e \leftrightarrow f/e \leftrightarrow e \rangle, \langle f \rightarrow e; l_p \leftrightarrow e \rangle, \langle l_p \rightarrow l_z; e \leftrightarrow e \rangle, \langle e \rightarrow e; l_z \leftrightarrow e \rangle).$$

We demonstrate the behavior of the P colony starting, as assumed above with n occurrences of the final object f and exactly one object L in the environment. The computation will end and stop with one less occurrences of f in the environment, which indicates the result of the operation -1 and with the symbol l_p if there was any f in the environment, and with l_z otherwise.

Let the P colony start to work. Only the program $\langle e \rightarrow e; e \leftrightarrow L \rangle$ can be used for the pair e, e of objects of the agent AG_{-1} . It executes the mutual exchange of the object e from the agent and the object L from the environment. The objects L and e are now in the agent AG_{-1} and the environment contains n copies of the object f , several copies of the object e (and no other object).

In the next step, the checking program $\langle L \rightarrow l_p; e \leftrightarrow f/e \leftrightarrow e \rangle$ is used by the agent AG_{-1} with objects L and e . The program results with objects l_p and f or l_p and e , inside the agent AG_{-1} , depending on the presence or absence of f in the environment. In the first case we continue with the program $\langle f \rightarrow e; l_p \leftrightarrow e \rangle$, which sends l_p to the environment and computation stops. Otherwise, the program $\langle l_p \rightarrow l_z; e \leftrightarrow e \rangle$ produces object l_z inside the agent AG_{-1} , and in the next step it is sent to the environment by the program $\langle e \rightarrow e; l_z \leftrightarrow e \rangle$. No programs of AG_{-1} can be now applied, so the computation halts and the outer environment gives the expected result.

3 Formal Definition of P Colonies

- A *P colony* of the capacity c is a construct $\Pi = (A, e, f, B_1, \dots, B_n)$, where
- A is an alphabet of the colony (its elements are called *objects*),
 - $e \in A$ is the basic object of the colony,
 - $f \in A$ is the final object of the colony,
 - B_i for $1 \leq i \leq n$ are *agents*, with the structure $B_i = (O_i, P_i)$, where O_i is a multiset of c copies of the basic object e (the *initial state* of the agent), and $P_i = \{p_{i,1}, \dots, p_{i,k_i}\}$ is a finite set of *programs*; each program $p_{i,j}$

consists of c rules; evolution rules of the form $a \rightarrow b$, communication rules of the form $c \leftrightarrow d$, or checking rules of the form $c \leftrightarrow d / c' \leftrightarrow d'$.

In this paper we will consider colonies with capacity two or three, i.e. having two or three objects inside each agent, respectively and two or three rules in each program of agents, corresponding to the number of objects in agents.

In the above given definition of P colony no restrictions are formulated to the structure of programs. In the original paper [8] P colonies with capacity two and with special simple programs were considered. Namely the programs of one of the form

$$\langle a \rightarrow b; c \leftrightarrow d \rangle \text{ or } \langle a \rightarrow b; c \leftrightarrow d / c' \leftrightarrow d' \rangle.$$

We will refer to such colonies as *restricted* P colonies, as it was proposed in [3]. Each program of a restricted P colony realizes an evolution of one of the two objects of the agent and mutual exchange of another object and an object from the environment, indicated by the communication rule or by the checking rule.

At the beginning of the *computation* performed by a given P colony, the *environment* contains arbitrarily many copies of basic object e (and nothing else); moreover, as stated above, each agent contains c copies of e . Both sequential and parallel behavior of agents in of P colonies will be considered in present paper. Parallel behavior was treated in the original model in [8] in accordance with the membrane structure motivation. (Sequential P colonies were introduced later, in [5].)

At each step of the *parallel* computation, the content of the environment and the contents of agents change in the following manner: each agent, which can use any of its programs have to use one (chosen non-deterministically). When using the program, all rules are applied, each one to different object from the agent.

In the *sequential* computation one agent takes a part on the rewriting at each step. In this case the content of the chosen agent and the content of the environment can be changed in one step of computation.

Formally, a state (a configuration) of the P colony will be expressed as an $(n+1)$ -tuple of the objects appearing inside each agent and in the environment, written in the form $(w_1, \dots, w_n, w_E e^o)$, where $|w_i| = c$, $1 \leq i \leq n$, represents c objects placed inside the i -th agent, and $w_E \in (A - \{e\})^*$ represents the objects in the environment different from the basic object e ; the symbol e^o is included to denote the fact that the environment always contains arbitrarily many copies of e . An *initial state* of the computation is of the form (e^c, \dots, e^c, e^o) .

Computation will be a sequence of states (configurations) of the P colony, the first of which is the initial state, and each further state is a result of one step of the computation from its predecessor. The computation ends by halting, i.e. when it a state, in which no agent can use any of its programs. With a halting computation we associate the result, in the form of the number of copies of the special object f present in the environment.

Because of the non-determinism in choosing programs, we obtain several computations, hence we associate with a P colony II a set of numbers, $N(II)$, computed by all possible halting computations of II .

Given a P colony $II = (A, e, f, B_1, \dots, B_n)$ the number of agents n is called *the degree of II*, and *the height of II* is the maximal number of programs in agents of the P colony. The third parameter characterizing a P colony is already mentioned *capacity* of II , describing the number of objects in agents identical with the number of rules in programs.

The family of all sets of numbers $N(II)$, computed (in parallel or in sequential) by P colonies of capacity c , degree at most n and of height at most h , without using checking

rules in their programs is denoted by $NPCOL_{par}(c,n,h)$ (or by $NPCOL_{seq}(c,n,h)$). If any of the parameters n and h is not bounded in considered P colonies, we replace it with $*$. The case of restricted colonies will be indicated with R , i.e. write $NPCOL_mR$ instead of $NPCOL_m$ for m being seq or par . If checking rules are allowed, then we write $NPCOLK$ instead of $NPCOL$; thus, $NPCOL_{par}KR(2,n,h)$ will be the family of numbers computed by restricted P colonies (of the capacity 2) using checking rules, with at most n agents and with at most h programs in each agent.

In what follows we discuss the computation power of P colonies of given degree, height and capacity, comparing it with the standard etalon of the computation power of Turing machines. More formally, the families $NPCOL_m\alpha(c,n,h)$, for $\alpha \in \{\varepsilon, K, R, KR\}$, and $m \in \{par, seq\}$, are of the interest for various values of the parameters, especially values of parameters c, n, h , which guarantee the universal computational power of P colonies.

Proofs of all universality theorems throughout the paper are based on a simulation of the behavior of a register machine, which are known to be computationally complete, by a P colony. The reader can find some necessary comments on register machines in the Appendix.

4 P Colonies with One Agent

We start with investigation of P colonies with one agent. In this case sequential and parallel behaviors of P colonies coincide. So we will use $NPCOL(c,l,h)$ instead of $NPCOL_{seq}(c,l,h)$ or $NPCOL_{par}(c,l,h)$. Even restricted P colonies with one agent and with checking rules are computationally complete.

Theorem 4.1 [3,5]: $NPCOL KR(2,1,*) = NRE$.

Proof: Consider a set of numbers computed by a register machine $M=(m, H, l_0, l_h, P)$. In a P colony a content of the register i will be represented by the number of copies of a specific object a_i in the environment. Moreover, labels l from H together with their two copies denoted l', l'' will be objects for our colony.

We construct the P colony $\Pi = (A, f, e, B)$, with the alphabet $A = \{a_i \mid 1 \leq i \leq m\} \cup \{e, d, d'\} \cup H \cup H' \cup H''$ and $f = a_1$.

We consider an agent B . It starts to compute with producing primed labels l', l'' according the following programs:

$$\langle e \rightarrow \alpha; e \leftrightarrow e \rangle, \langle e \rightarrow \alpha; \beta \leftrightarrow e \rangle, \langle e \rightarrow l_0; \alpha \leftrightarrow e \rangle$$

for $\alpha, \beta \in \{l', l'' \mid l \in H\}$. It produces objects α, β and throw them in the environment.

The process stops after also producing the initial label l_0 and B starts to simulate the register machine. The object e and l_0 are now inside the agent.

For each instruction $l_1: (ADD(r), l_2, l_3)$ from P we introduce in B the programs P_{l_1} :

$$\langle l_1 \rightarrow l'_1; e \leftrightarrow e \rangle, \langle e \rightarrow a_r; l'_1 \leftrightarrow l'_1 \rangle, \langle l'_1 \rightarrow l_2; a_r \leftrightarrow e \rangle, \langle l'_1 \rightarrow l_3; a_r \leftrightarrow e \rangle$$

The first program produces pair of objects l'_1 and e , then by the second program the agent gets objects l'_1 and a_r inside and either the program $\langle l'_1 \rightarrow l_2; a_r \leftrightarrow e \rangle$ or $\langle l'_1 \rightarrow l_3; a_r \leftrightarrow e \rangle$ producing the pair e and l_1 or e and l_2 .

For each instruction $l_1: (SUB(r), l_2, l_3)$ from P we introduce in B the programs P_{l_1} :

$$\langle l_1 \rightarrow l'_1; e \leftrightarrow a_r/e \leftrightarrow e \rangle, \langle a_r \rightarrow e; l'_1 \leftrightarrow l''_1 \rangle, \langle l''_1 \rightarrow l_2; e \leftrightarrow e \rangle, \langle l'_1 \rightarrow l_3; e \leftrightarrow e \rangle.$$

The first checking program produces either the objects l_1, a_r , or l_1, e in the dependence on the presence of a_r in the environment. For the objects l_1, a_r the agent can use the second and third program to produce objects l_2 and e . For the objects l_1, e the agent uses the last program to obtain l_3 and e . In order to ensure that agent B has produced sufficient primed versions of labels so that the programs of B can exchange primed objects from inside with primed objects from the environment, the program $\langle l_1 \rightarrow l'_1; e \leftrightarrow e \rangle$ is also included. If in any moment, an exchange $l'_1 \leftrightarrow l_1$ or $l'_1 \leftrightarrow l''_1$ cannot be realized, then the program $\langle l_1 \rightarrow l'_1; e \leftrightarrow e \rangle$ can work forever, hence preventing the halting of the computation. Thus, we have $N(M)=N(\Pi)$. \square

The restriction to the P colonies with only one agent and no checking rule decreases the generative power P colonies. These colonies generate the set of numbers $NMAT$ characterized also by matrix grammars.

Theorem 4.2 [5]: $NPCOL(2,1,*) = NMAT$

We omit here the proof, which uses also the partially blind register machine.

5 Universality in Parallel P Colonies

As we demonstrated in Section 4 one agent is sufficient to grant the universal generative power of P colonies. However the number of programs in that agent is not limited in that case. Now we will deal with limitations of the number of programs in agents. Universal generative power can be achieved also in this case with low capacity and height of P colonies (the parameters of c, h) but with more agents in P colonies. We summarize results obtained in this direction. First consider the parallel P colonies.

5.1 Restricted P colonies

We start with the case of restricted P colonies, which have programs either of the form $\langle a \rightarrow b; c \leftrightarrow d \rangle$ or $\langle a \rightarrow b; c \leftrightarrow d / c' \leftrightarrow d' \rangle$. Each of such programs realizes an evolution of one of the two objects of an agent, and a mutual exchange of the another object of the agent and an object from the environment (eventually with checking possibility).

Theorem 5.1 [3,8]: $NPCOL_{par}KR(2,*,5) = NRE$.

Proof: Let us consider a register machine $M=(m,H,l_0,l_h,P)$. All the labels from H will be objects for our colony; moreover, the contents of a register i will be represented by the number of copies in the environment of a specific object a_i . We construct a P colony $\Pi = (A, f, e, B_1, \dots, B_s)$, with the alphabet $A=H \cup \{a_i \mid 1 \leq i \leq m\} \cup \{e,d,d'\}$, with $f = a_1$, and the following $s = card(P)+2$ agents.

We consider initial agents B_1, B_2 :

$$P_1 = \{ \langle e \rightarrow d; e \leftrightarrow e \rangle, \langle e \rightarrow l_0; d \leftrightarrow e \rangle, \langle e \rightarrow e; l_0 \leftrightarrow d' \rangle \}$$

$$P_2 = \{ \langle e \rightarrow d'; e \leftrightarrow e \rangle, \langle e \rightarrow e; d' \leftrightarrow e \rangle, \langle e \rightarrow e; e \leftrightarrow d \rangle \}.$$

The first agent sends to the environment one copy of d and one of l_0 , the initial label of M ; l_0 is sent out in exchange of d' and the work of B_1 stops. The second agent can produce several copies of d' , but it can stop by bringing inside the copy of d produced by the first agent. The possible additional copies of d' will be useless, important is that we have only one copy of l_0 , and its appearance in the environment triggers the beginning of the simulation of a computation in M .

For each instruction l_1 : (ADD(r), l_2, l_3) from P we consider an agent (the label l_1 precisely identifies the instruction) with programs

$$P_{l_1} = \{ \langle e \rightarrow a_r; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow l_2; a_r \leftrightarrow e \rangle, \langle e \rightarrow e; l_2 \leftrightarrow e \rangle, \\ \langle l_1 \rightarrow l_3; a_r \leftrightarrow e \rangle, \langle e \rightarrow e; l_3 \leftrightarrow e \rangle \}.$$

If the object l_1 is present in the environment, then this agent can bring it inside, at the same time changing its inner e with a_r ; in the next step, this a_r is released into the environment, while inside one transforms l_1 into l_2 or l_3 ; this label is then sent to the environment, thus completing the simulation of the ADD instruction and making possible the simulation of another instruction.

For each instruction l_i : ($SUB(r), l_2, l_3$) from P we consider an agent with programs

$$P_{l_i} = \{ \langle e \rightarrow e; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow l_2; e \leftrightarrow a_r / e \leftrightarrow e \rangle, \langle a_r \rightarrow e; l_2 \leftrightarrow e \rangle, \\ \langle l_2 \rightarrow l_3; e \leftrightarrow e \rangle, \langle e \rightarrow e; l_3 \leftrightarrow e \rangle \}.$$

The agent brings l_1 inside, then, in the second step of the simulation, it checks whether at least one copy of a_r is present in the environment; in the positive case, it brings one copy inside it (and it changes a_r here into e); in the negative case it does not change the inner object e . In the former case, the agent can then move l_2 into the environment, in the latter case it first transforms object l_2 into l_3 and this label is sent to the environment. The number of copies of a_r from the environment is decreased by one, if possible, and the next available label indicates the correct action.

From the previous explanations, it is easy to see that the computations in M are correctly simulated in Π , and that the computation in Π stops in the moment when the label l_h is sent to the environment (and at that time the number of copies of a_1 present in the environment is equal to the contents of the first register of M).

Consequently, $N(M) = N(\Pi)$, and, because each agent contains at most five programs, the proof is complete. \square

Note that each of the instructions ADD and SUB in the construction presented in the last proof are realized by exactly one individual agent.

The optimality of the previous result in the height of agents is not known, and also the power of P colonies with height 1, 2, 3 or 4 is an open question.

5.2 Elimination of checking rules

The checking rules directly remind the check for zero from the SUB instructions of register machines, that is why it is highly surprising to get universality even avoiding using this powerful feature (at the expense of using slightly more programs in each agent). It was proved in [3] that $NPCOL_{par}(2, *, 8) = NRE$ and $NPCOL_{par}(3, *, 7) = NRE$. These results were improved in [5], where it was proved that five programs in each agent are sufficient for restricted P colonies.

Theorem 5.2 [5]: $NPCOL_{par}R(2, *, 5) = NRE$

Proof: We consider a register machine $M = (m, H, l_0, l_h, P)$ and represent the content of register i by the number of copies of a specific object a_i in the environment. We construct a P colony $\Pi = (A, f, e, B_1, \dots, B_s)$ with alphabet $A = \{l_0, e, d, d'\} \cup \{a_i \mid 1 \leq i \leq m\} \cup \{l, l^1, l^2, l^3, l^4, l^5, l^6, l^7, l^8 \mid l \in H\}$. Each simulation of the instruction l starts with objects l and l^i in the environment.

To initialize the whole system, i.e. to produce l_0 and l^i_0 we take two agents with following programs

$$P_{0,i} = \{ \langle e \rightarrow l^3_0; e \leftrightarrow e \rangle, \langle e \rightarrow l^4_0; l^3_0 \leftrightarrow e \rangle, \\ \langle l^4_0 \rightarrow l_0; e \leftrightarrow e \rangle, \langle e \rightarrow l^5_0; l_0 \leftrightarrow e \rangle \}$$

$$P_{0,2} = \{ \langle e \rightarrow {}^1l_0; e \leftrightarrow {}^3l_0 \rangle, \langle {}^3l_0 \rightarrow {}^5l_0; {}^1l_0 \leftrightarrow e \rangle \}.$$

Both agents end with e and 3l_0 inside, and leave l_0 and 1l_0 in the environment.

For each instruction l_1 : (ADD(r), l_2 , l_3) from P we consider four agents (the label l_1 identifies the instruction)

$$\begin{aligned} P_{l_1,1} &= \{ \langle e \rightarrow {}^7l_1; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow a_r; {}^7l_1 \leftrightarrow e \rangle, \\ &\quad \langle e \rightarrow l_2; a_r \leftrightarrow e \rangle, \langle e \rightarrow e; l_2 \leftrightarrow e \rangle \}, \\ P_{l_1,2} &= \{ \langle e \rightarrow e; e \leftrightarrow {}^1l_1 \rangle, \langle {}^1l_1 \rightarrow e; e \leftrightarrow e \rangle, \\ &\quad \langle e \rightarrow {}^1l_2; e \leftrightarrow {}^7l_1 \rangle, \langle {}^7l_1 \rightarrow e; {}^1l_2 \leftrightarrow e \rangle \}, \\ P_{l_1,3} &= \{ \langle e \rightarrow {}^7l_1; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow a_r; {}^4l_1 \leftrightarrow e \rangle, \\ &\quad \langle e \rightarrow l_3; a_r \leftrightarrow e \rangle, \langle e \rightarrow e; l_3 \leftrightarrow e \rangle \}, \\ P_{l_1,4} &= \{ \langle e \rightarrow e; e \leftrightarrow {}^1l_1 \rangle, \langle {}^1l_1 \rightarrow e; e \leftrightarrow e \rangle, \\ &\quad \langle e \rightarrow {}^1l_3; e \leftrightarrow {}^4l_1 \rangle, \langle {}^4l_1 \rightarrow e; {}^1l_3 \leftrightarrow e \rangle \}. \end{aligned}$$

Agents $B_{l_1,1}$ and $B_{l_1,2}$ work together: $B_{l_1,1}$ takes l_1 from the environment, produces a_r and send out 7l_1 while at the same two steps $B_{l_1,2}$ simply consumes 1l_1 ending up with the multi-set e, e . In the subsequent steps $B_{l_1,1}$ sends out a_r and l_2 , whereas $B_{l_1,2}$ produces and sends out 1l_2 allowing the next instruction l_2 to be simulated.

The role of the agents $B_{l_1,3}$ and $B_{l_1,4}$ is similar but now 1l_3 and l_3 are generated and sent to the environment.

For each instruction l_1 : (SUB(r), l_2 , l_3) from P we will consider six agents $B_{l_1,1}, \dots, B_{l_1,6}$ with following programs:

$$\begin{aligned} P_{l_1,1} &= \{ \langle e \rightarrow e; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow {}^6l_1; e \leftrightarrow a_r \rangle, \langle a_r \rightarrow e; {}^6l_1 \leftrightarrow e \rangle, \\ &\quad \langle l_1 \rightarrow {}^3l_1; e \leftrightarrow {}^2l_1 \rangle, \langle {}^2l_1 \rightarrow e; {}^3l_1 \leftrightarrow e \rangle \}, \\ P_{l_1,2} &= \{ \langle e \rightarrow {}^2l_1; e \leftrightarrow {}^1l_1 \rangle, \langle {}^1l_1 \rightarrow e; {}^2l_1 \leftrightarrow e \rangle \} \end{aligned}$$

In the first step $B_{l_1,1}$ and $B_{l_1,2}$ takes in the labels l_1 and 1l_1 . In the second step $B_{l_1,2}$ sends out 2l_1 whereas $B_{l_1,1}$ in the case here is an object a_r in the environment, takes in a_r and then sends out 6l_1 . In the case there is no object a_r in the environment $B_{l_1,1}$ consumes 2l_1 after one waiting step and finally sends out 3l_1 .

With 6l_1 in the environment $B_{l_1,3}$ and $B_{l_1,4}$ become active and produce labels l_2 and 1l_2

$$\begin{aligned} P_{l_1,3} &= \{ \langle e \rightarrow {}^7l_1; e \leftrightarrow {}^6l_1 \rangle, \langle l_1 \rightarrow {}^8l_1; {}^7l_1 \leftrightarrow e \rangle, \\ &\quad \langle {}^8l_1 \rightarrow l_2; e \leftrightarrow {}^2l_1 \rangle, \langle {}^2l_1 \rightarrow e; l_2 \leftrightarrow e \rangle \}, \\ P_{l_1,4} &= \{ \langle e \rightarrow {}^1l_2; e \leftrightarrow {}^7l_1 \rangle, \langle {}^7l_1 \rightarrow e; {}^1l_2 \leftrightarrow e \rangle \}. \end{aligned}$$

On the other hand, if 3l_1 is in the environment then only $B_{l_1,5}$ and $B_{l_1,6}$ can work finally sending l_3 and 1l_3 to the environment

$$\begin{aligned} P_{l_1,5} &= \{ \langle e \rightarrow {}^4l_1; e \leftrightarrow {}^3l_1 \rangle, \langle l_1 \rightarrow {}^5l_1; {}^4l_1 \leftrightarrow e \rangle, \\ &\quad \langle {}^5l_1 \rightarrow l_3; e \leftrightarrow e \rangle, \langle e \rightarrow e; l_3 \leftrightarrow e \rangle \}, \\ P_{l_1,6} &= \{ \langle e \rightarrow {}^1l_3; e \leftrightarrow {}^4l_1 \rangle, \langle {}^4l_1 \rightarrow e; {}^1l_3 \leftrightarrow e \rangle \}. \end{aligned}$$

The computation stops when the label l_i appears in the environment. Each agent has at most five programs with no checking rule which prove the theorem. \square

5.3 Non-restricted P colonies

In the present sections the structure of rules allowed in programs is not fixed. This allows us to decrease the number of rules used in programs.

Theorem 5.3 [3]: $NPCOL_{par}K(2, *, 4) = NRE$.

Proof: Analogically as in the proof of Theorem 5.1 P colony will contain initial agents B_1, B_2 with programs

$$P_1 = \{ \langle e \rightarrow d; e \leftrightarrow e \rangle, \langle e \rightarrow l_0; d \leftrightarrow e \rangle, \langle e \rightarrow e; l_0 \leftrightarrow d' \rangle \}$$

$$P_2 = \{ \langle e \rightarrow d'; e \leftrightarrow e \rangle, \langle e \rightarrow e; d' \leftrightarrow e \rangle, \langle e \rightarrow e; e \leftrightarrow d \rangle \}.$$

and the agents B_{l_i} with programs:

For each instruction $l_i: (ADD(r), l_2, l_3)$ from P we take

$$P_{l_i} = \{ \langle e \rightarrow e; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow a_r; e \rightarrow l_2 \rangle, \langle l_1 \rightarrow a_r; e \rightarrow l_3 \rangle, \langle a_r \leftrightarrow e; l_2 \leftrightarrow e / l_3 \leftrightarrow e \rangle \}.$$

In the first step, the agent brings l_1 inside; in the next step l_1 is transformed in a_r and the other internal object e is non-deterministically changed into l_2 or l_3 ; a_r is sent out in the third step, together with the object l_2, l_3 present in the agent.

For each instruction $l_i: (SUB(r), l_2, l_3)$ from P we consider a program

$$P_{l_i} = \{ \langle e \leftrightarrow l_1; e \leftrightarrow a_r / e \rightarrow l_3 \rangle, \langle l_1 \rightarrow l_2; a_r \rightarrow e \rangle, \langle e \rightarrow e; l_2 \leftrightarrow e \rangle, \langle l_1 \rightarrow e; l_3 \leftrightarrow e \rangle \}.$$

In the first step, the agent brings l_1 inside, and at the same time it checks whether a_r is present in the environment; in the positive case it brings one copy inside the agent, in the negative case it changes the other object e in l_3 . In the former case, it changes l_1 in l_2 and a_r to e , then it sends l_2 out; in the latter case it sends out l_3 and at the same time transforms l_1 in e . The simulation of the SUB instruction is correct.

The equality $N(M) = N(II)$ is again easy provable. \square

Allowing one more object in each agent with one more rule in each program we can decrease the number of programs in each agent.

Theorem 5.4 [3]: $NPCOL_{par}K(3, *, 3) = NRE$.

Proof: Initial agents B_1, B_2 have following programs (note that they contain one more $e \leftrightarrow e$ rule in each program, otherwise they are similar as in the proof of the previous theorem)

$$P_1 = \{ \langle e \rightarrow d; e \leftrightarrow e; e \leftrightarrow e \rangle, \langle e \rightarrow l_0; d \leftrightarrow e; e \leftrightarrow e \rangle, \langle e \rightarrow e; l_0 \leftrightarrow d'; e \leftrightarrow e \rangle \}$$

$$P_2 = \{ \langle e \rightarrow d'; e \leftrightarrow e; e \leftrightarrow e \rangle, \langle e \rightarrow e; d' \leftrightarrow e; e \leftrightarrow e \rangle, \langle e \rightarrow e; e \leftrightarrow d; e \leftrightarrow e \rangle \}.$$

The agents B_{l_i} contain the following programs:

For each instruction $l_i: (ADD(r), l_2, l_3)$ from P we take

$$P_{l_i} = \{ \langle e \rightarrow a_r; e \rightarrow l_2; e \leftrightarrow l_1 \rangle, \langle e \rightarrow a_r; e \rightarrow l_3; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow e; a_r \leftrightarrow e; l_2 \leftrightarrow e / l_3 \leftrightarrow e \rangle \}.$$

In the first step, the agent brings l_1 inside and non-deterministically changes the other internal object e into l_2 or l_3 ; in the next step, l_1 is transformed in e and a_r is sent out, together with the label l_2 or l_3 present in the agent.

For each instruction l_1 : ($SUB(r), l_2, l_3$) from P we consider

$$P_{l_1} = \{ \langle e \leftrightarrow l_1; e \leftrightarrow a_r / e \rightarrow l_3; e \rightarrow l_2 \rangle, \langle l_1 \rightarrow e; a_r \rightarrow e; l_2 \leftrightarrow e \rangle, \\ \langle l_1 \rightarrow e; l_2 \rightarrow e; l_3 \leftrightarrow e \rangle \}.$$

In the first step, the agent brings l_1 inside, and at the same time checks whether a_r is present in the environment; in the positive case it brings one copy of it inside, in the negative case it changes the other object e in l_3 ; at the same time, the third copy of e is changed in l_2 . In the next step, the agent changes l_1 in e ; if a_r is present, then the agent applies the program which sends out the label l_2 ; otherwise, it changes l_2 to e and sends out l_3 .

The equality $N(M) = N(\Pi)$ easily follows. \square

6 Universality in Sequential P Colonies

Analogically to the parallel case, in sequential case programs with five rules are sufficient to generate all computable sets of numbers. But now, we cannot eliminate the checking rules.

Theorem 6.1 [5]: $NPCOL_{seq}KR(2, *, 5) = NRE$

Proof: We consider a register machine $M = (m, H, l_0, l_h, P)$. We construct a P colony $\Pi = (A, a_1, e, B_1, \dots, B_s)$ with alphabet $A = \{l_0', e, d, d'\} \cup \{a_i \mid 1 \leq i \leq m\} \cup \{l \mid l \in H\}$.

Starting agents B_1, B_2 have the programs

$$P_1 = \{ \langle e \rightarrow d; e \leftrightarrow e \rangle, \langle e \rightarrow l_0; d \leftrightarrow d' \rangle, \langle d' \rightarrow l_0'; l_0 \leftrightarrow e \rangle \} \\ P_2 = \{ \langle e \rightarrow d'; e \leftrightarrow e \rangle, \langle e \rightarrow l_0'; d' \leftrightarrow e \rangle \}$$

Both these agents stop with objects $l_0'; d'$ and leaving l_0 in the environment.

All further programs are same as in the proof of Theorem 5.1 so we list them with no additional comment.

ADD instruction l_1 : ($ADD(r), l_2, l_3$) is realized by one agent with rules

$$P_{l_1} = \{ \langle e \rightarrow a_r; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow l_2; a_r \leftrightarrow e \rangle, \langle e \rightarrow e; l_2 \leftrightarrow e \rangle, \\ \langle l_1 \rightarrow l_3; a_r \leftrightarrow e \rangle, \langle e \rightarrow e; l_3 \leftrightarrow e \rangle \}.$$

SUB instruction l_1 : ($SUB(r), l_2, l_3$)

$$P_{l_1} = \{ \langle e \rightarrow e; e \leftrightarrow l_1 \rangle, \langle l_1 \rightarrow l_2; e \leftrightarrow a_r / e \leftrightarrow e \rangle, \langle a_r \rightarrow e; l_2 \leftrightarrow e \rangle, \\ \langle l_2 \rightarrow l_3; e \leftrightarrow e \rangle, \langle e \rightarrow e; l_3 \leftrightarrow e \rangle \}.$$

STOP instruction l_h

$$B_{l_h} = \{ \langle e \rightarrow e; e \leftrightarrow l_h \rangle, \langle l_h \rightarrow e; e \leftrightarrow e \rangle \}. \square$$

Note that in the proofs of Theorems 5.3 and 5.4 the simulation of each ADD and SUB instruction is done by a single agent. So, starting with appropriate initial agents one can easily prove the following:

Theorem 6.2: $NPCOL_{seq}K(2, *, 4) = NRE$.

Theorem 6.3: $NPCOL_{seq}K(3, *, 3) = NRE$.

7 Note on the Initial State

We have introduced a computation in P colonies to start with the homogeneous state in the sense that it contains basic objects e only, inside all agents as well as in the environment. So (e^c, \dots, e^c, e^o) is the starting state of the P colony (with capacity c). We can consider also more general case when a starting state of a P colony is of the form $(w_{1,0}, \dots, w_{n,0}, w_{E,0} e^o)$, i.e. the case when computation starts with an arbitrary c -tuple of objects in agents and with finite number of non-basic objects in the environment. It follows from the universality of P colonies presented in the previous section that the generalization to the non uniform starting state has no influence to the generative power of the considered P colonies, both in sequential and parallel way of derivation. As the corollary of results we have

Theorem 7.1: Let $N(\Pi)$ be the set of numbers computed by a P colony Π starting with the configuration $(w_{1,0}, \dots, w_{n,0}, w_{E,0} e^o)$. Then there is a P colony Π' starting with the configuration (e^c, \dots, e^c, e^o) such that $N(\Pi) = N(\Pi')$.

8 Conclusions

The colonies we have considered here are composed of so weak cells, agents that the computational universality of the model is surprising, with the „explanation“ staying in the possibility to simulate a register machine in the environment, either using the „global information“, which can be achieved by the checking programs, or based on the cooperation of cells.

As we have seen, the investigation of P colonies concerns different models with uniform generative power. There are many topics which we have not discussed up to now for P colonies. Some of them are mathematical problems, for instance, concerning the optimality of the results obtained here, whether or not the parameters n and h induce, as expected, a double hierarchy of the families $PCOL(k,n,h)$, $NPCOLR(2,n,k)$, and so on, but other problems, although technical too, have a more general significance. For instance:

- Which further simplifications can be considered without losing the universality?
- What about cells containing only one object at a time, or with programs consisting of only one rule?
- What about having a bounded environment?
- On the other hand, what happens if the environment is populated at the beginning with different objects?
- Can this simplify the programs?
- What happens if the objects can act on the environment not only by exchanging objects, but also directly, by mutations of external objects; more generally, what about having environment evolution rules (thus coming closer to eco-grammar systems [4])?

We conclude with the belief that P colonies deserve further research efforts.

Recently, two generalizations of P colonies were introduced. EP colony introduced in [1] follows the idea of eco-system in research direction of P systems. The paper starts an investigation on the direction formulated in the last question. Motivation for an introduction and investigation of LP colony in [6] can be found in natural language evolution.

Acknowledgment: The authors' research on the topic is supported by the Czech Science Foundation Grant No. 201/04/0528.

References:

- [1] E. Csuhaj-Varju: EP-colonies: Micro-Organisms in a Cell-like Environment. Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla (Spain), January 31st - February 4th, 2005, pp. 123-130
- [2] E. Csuhaj-Varju, J. Dassow, J. Kelemen, Gh. Paun: *Grammar Systems - A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach, London, 1994
- [3] E. Csuhaj-Varju, J. Kelemen, A. Kelemenova, Gh. Paun, G. Vaszil: Cells in environment: P Colonies, *Journal of Multi-Valuated Logic*, 2005 (accepted) 13 pp.
- [4] E. Csuhaj-Varju, A. Kelemenova, J. Kelemen, Gh. Paun: Eco-grammar systems. A grammatical framework for life-like interactions. *Artificial Life* 3 (1997) 1-28
- [5] R. Freund, M. Oswald: P colonies working in the maximally parallel and in the sequential mode. In: *Pre-Proceedings of the 1st International Workshop on Theory and Application of P Systems* (G. Ciobanu, Gh. Paun, eds.). Timisoara, Romania, September 26-27, 2005, pp. 49-56
- [6] G. Bel-Enguix, M.D. Jimenez Lopez: LP Colonies for Language Evolution. A preview. In: *Pre-Proceedings of the 6th International Workshop on Membrane Computing (WMC6)* (R. Freund, G. Lojka, M. Oswald, Gh. Paun, eds.). Vienna, June 18-21, 2005, pp. 179-192
- [7] J. Kelemen, A. Kelemenová: A grammar-theoretic treatment of multiagent systems. *Cybernetics and Systems* 23 (1992) 621- 633
- [8] J. Kelemen, A. Kelemenová, G. Paun: The power of cooperation in a biochemically inspired computing model: P colonies. In: *Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems, ALIFE IX* (M. Bedau at al., eds.) Boston, Mass., 2004, pp. 82-86
- [9] M. L. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967
- [10] Gh. Paun: Computing with membranes. *Journal of Computer and System Sciences* 61 (2000) 108-143
- [11] Gh. Paun: *Computing with Membranes - An Introduction*. Springer-Verlag, Berlin, 2002
- [12] E. M. A. Ronald, M. Sipper, M. S. Capcarrère: Design, observation, surprise! A test of emergence. *Artificial Life* 5 (1999) 225-239

Appendix: The Register Machines

A *register machine* defined in [9] (and called also *Minsky-machine* in some publications) is a device consisting of a given number of registers, each of which can hold an arbitrarily large non-negative integer number, and a program, which is a sequence of labelled instructions, which specify how the numbers stored in registers can change and which instruction can be used in next step. There are three types of instructions considered:

- l_1 : (ADD(r), l_2 , l_3) (add 1 to register r and go to the instruction with label l_2 or l_3),
- l_1 : (SUB(r), l_2 , l_3) (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_2 , otherwise go to the instruction with label l_3),

l_h : HALT (the halt instruction).

Thus, formally, a *register machine* is a construct

$$M = (m, H, l_0, l_h, P),$$

where m is the number of registers,

H is the set of instruction labels,

l_0 is the start label,

l_h is the halt label (assigned to instruction HALT), and

P is the set of instructions (the program); each label from H labels only one instruction from P , thus precisely identifying it.

A register machine M computes a set $N(M)$ of numbers in the following way: we start with all registers empty (hence storing the number zero) with the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number stored at that time in register 1 is said to be computed by M and hence it is introduced in $N(M)$. It is known that in this way we can compute all sets of numbers which are Turing computable (see, e.g. [8]).