P Colonies Working in the Maximally Parallel and in the Sequential Mode

Rudolf FREUND and Marion OSWALD
Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9-11, A-1040 Wien, AUSTRIA
E-mail: {rudi,marion}@emcc.at

Abstract

We consider P colonies as introduced in [4] and investigate their computational power when working in the maximally parallel and in the sequential mode. It turns out that there is a trade-off between maximal parallelism and checking programs: Using checking programs (i.e., priorities on the communicaton rules in the programs of the agents), P colonies working in the sequential mode with height at most 5 are computationally complete, whereas when working in the maximally parallel mode, P colonies (again with height 5) already obtain the same computational power without using checking programs. Moreover, when allowing an arbitrary number of programs for each agent, we can prove that P colonies with only one agent (thus these P colonies are working in the sequential mode) are already computationally complete. Finally, P colonies with an arbitrary number of agents working in the sequential mode as well as even P colonies with only one agent using an arbitrary number of non-checking programs characterize the family of languages generated by matrix grammars without appearance checking.

1 Introduction

In [4], a class of membrane systems (introduced in [6] and currently called P systems; see [7] for a comprehensive overview and [9] for recent developments) similar to so-called colonies of simple formal grammars (introduced in [3]; see also [1] for basic elements of grammar systems), was introduced as *P colonies.*

Taking the idea of components that are as simple as possible and act in a shared environment, independent cells are the basic computing *agents* in the formal model of P colonies. Each agent is associated with a multiset of objects and with a set of rules. In the specific model introduced in [4], at each moment, only two objects are allowed to be inside any agent.

Each program assigned to an agent consists of an evolution rule of the form $a \to b$, transforming an internal object a into b, as well as a communication rule of the form $c \leftrightarrow d$, exchanging the internal object c with the object d taken from the environment. Hence, for each agent, a pair of rules $\langle a \to b, c \leftrightarrow d \rangle$ is called a *program*. Thus, if the agent contains the objects a, c, after applying this program it will contain the objects b, d. In one time unit both rules have to be applied, one for the evolution and one for communication with the environment.

At the beginning of the computation, all objects from the environment (that are supposed to be present in an unbounded number) as well as the two objects initially present inside of each agent are identical with the same generic object e.

In [4], a non-trivial capability is added to the agents: they can check for the appearance of a given object in the environment by programs of the form $\langle a \to b, c \leftrightarrow d/c' \leftrightarrow d' \rangle$ that are called *checking programs*. Here again, one evolution rule and one communication rule are applied at the same time, but the communication rule can be chosen from two possibilities, with the first one having a higher priority, thus, P colonies using checking programs will also be called P colonies with priorities in this paper.

Hence, a P colony consists of a finite number of agents (identified by their sets of programs) placed in a common environment where arbitrarily many copies of e are present.

In [4], P colonies were shown to be computationally complete when using an unbounded number of agents not containing more than five programs each when working in a maximally parallel way, i.e., in each moment, each agent which can apply any of its programs has to non-deterministically choose one and apply it, until no agent can apply any of its programs anymore (with such a halting computation, a result is associated in the form of the number of copies of a distinguished object in the environment). On the other hand, two agents were shown to be sufficient for obtaining computational completeness when using an unbounded number of programs.

After some preliminary definitions, we here show that P colonies even when working in the sequential mode are computationally complete when using an unbounded number of agents with no more that five programs each. Without using checking programs (i.e., without priorities) we obtain a similar result for P colonies working in the maximally parallel way. P colonies with only one agent working with an arbitrary number of checking programs are shown to be computationally complete, too. On the other hand, P colonies without priorities (with one agent or with an arbitrary number of agents working in the sequential mode) characterize the family of languages generated by matrix grammars without appearance checking.

2 Prerequisites

In this section, we give some preliminary definitions, consider basic facts of register machines and matrix grammars.

2.1 Preliminary Definitions

The set of non-negative integers is denoted by $\mathbf{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* - \{\lambda\}$ is denoted by V^+. By $|x|$ we denote the length of the string x over V. The family of recursively enumerable languages is denoted by RE.

Let $\{a_1, ..., a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $|x|_{a_i}$; the *Parikh vector* associated with x with respect to $a_1, ..., a_n$ is $\left(|x|_{a_1}, ..., |x|_{a_n}\right)$. The *Parikh image* of a language L over $\{a_1, ..., a_n\}$ is the set of all Parikh vectors of strings in L. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$. A (finite) multiset $\langle m_1, a_1\rangle ... \langle m_n, a_n\rangle$ with $m_i \in \mathbf{N}$, $1 \leq i \leq n$, is represented as any string x the Parikh vector of which with respect to $a_1, ..., a_n$ is $(m_1, ..., m_n)$.

In the following we will not distinguish between a vector $(m_1, ..., m_n)$, its representation by a multiset $\langle m_1, a_1\rangle ... \langle m_n, a_n\rangle$ or its representation by a string x with Parikh vector $\left(|x|_{a_1}, ..., |x|_{a_n}\right) = (m_1, ..., m_n)$.

For more notions as well as basic results from the theory of formal languages, the reader is referred to [2] and [8].

2.2 Register machines

A *register machine* is a construct $M = (m, P, l_0, l_h)$, where m is the number of registers, P is a finite set of instructions injectively labelled with elements from a given set $lab(M)$, l_0 is the initial/start label, and l_h is the final label.

The instructions are of the following forms:

- $l_1 : (A(r), l_2, l_3)$
  Add 1 to the contents of register r and proceed to the instruction (labelled with) l_2 or l_3. (We say that we have an ADD instruction.)
- $l_1 : (S(r), l_2, l_3)$
  If register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3. (We say that we have a conditional SUB instruction.)
- $l_h : halt$
  Stop the machine. The final label l_h is only assigned to this instruction.

Without loss of generality, one can assume that in each ADD instruction $l_1 : (A(r), l_2, l_3)$ and in each conditional SUB instruction $l_1 : (S(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct.

The following result already follows from the results proved in [5]:

Proposition 1. *Let $L \subseteq \mathbf{N}^\beta$ be a recursively enumerable set of (vectors of) non-negative integers. Then L can be generated by a register machine with at most $\beta + 2$ registers; moreover, at the beginning of a computation, all registers are empty; the results of a halting computation appear in the first β registers.*

Moreover, we call a register machine *partially blind*, if we interpret a subtract instruction in the following way:

> $l_1 : (S(r), l_2, l_3)$
>
> If register r is not empty, then subtract one from its contents and go to instruction l_2 or to instruction l_3. (We say that we have an unconditional SUB instruction; if register r is empty when attempting to decrement register r, then the program ends without yielding a result).

When the register machine reaches the final state, the results obtained in the first (β) register(s) are only taken into account if the remaining registers are empty (in some sense this means that although during the computation we cannot check a register for zero, but at the end of a computation we can check those registers for zero that do not contain a result). The family of sets of vectors of non-negative integers generated by partially blind register machines is denoted by NRM_{pb}.

2.3 Matrix grammars

A context-free *matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$ where N and T are sets of *non-terminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, M is a finite set of *matrices*, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices m_i are sequences of the form $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are context-free productions over (N, T). For $m_i = (m_{i,1}, \ldots, m_{i,n_i})$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$, w_j is the result of the application of $m_{i,j}$ to w_{j-1}. The language generated by G is

$$\begin{aligned} L(G) = \{w \in T^* \mid \quad & S \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w, \\ & w_j \in (N \cup T)^*,\ m_{i_j} \in M \ \text{ for } 1 \leq j \leq k, k \geq 1\big\}\,. \end{aligned}$$

The family of languages generated by matrix grammars without appearance checking is denoted by MAT^λ. It is known that $PsMAT^\lambda \subset PsRE$. Further details about matrix grammars can be found in [2] and in [8]. We only mention that the power of matrix grammars is not decreased if we only work with matrix grammars in the *f-binary normal form* where M contains rules of the following forms:

1. $(S \rightarrow XA)$, with $X \in N_1, A \in N_2$,

2. $(X \rightarrow Y, A \rightarrow x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$, $|x| \leq 2$,

3. $(X \rightarrow f, A \rightarrow x)$, with $X \in N_1, A \in N_2$, and $x \in T^*$, $|x| \leq 2$,

4. $(f \rightarrow \lambda)$.

Moreover, there is only one matrix of type 1 and only one matrix of type 4, which is only used in the last step of a derivation yielding a terminal result.

2.4 P colonies

A *P colony* is a construct $\Pi = (V, e, T, B_1, ..., B_n)$, where V is an alphabet (its elements are called objects), e is a distinguished object of V (the environmental object), $T \subseteq V - \{e\}$ is a set of final objects, and $B_1, ..., B_n$ are agents; each agent B_i is a pair $B_i = (O_i, P_i)$, where O_i is a multiset over V (the initial state of the agent), and P_i is a finite set $\{p_{i,1}, ..., p_{i,k_i}\}$ of programs; each program $p_{i,j}$ is either a non-checking program of the form $\langle a \to b, c \leftrightarrow d \rangle$, or a checking program of the form $\langle a \to b, c \leftrightarrow d/c' \leftrightarrow d' \rangle$. In what follows, we always assume that each O_i consists of two copies of e.

At the beginning of a computation performed by a given P colony, the environment contains arbitrarily many copies of e; moreover, each agent contains two copies of e. At each step of the computation, the contents of the environment and of the agents change in the following manner: In the maximally parallel derivation mode, each agent which can use any of its programs should use one (non-deterministically chosen), whereas in the sequential derivation mode, one agent uses one of its programs at a time (non-deterministically chosen). By using a program $\langle a \to b, c \leftrightarrow d \rangle$, an agent with objects ac inside and with d in the environment will get the objects bd inside and c will now be placed into the environment. Using a program $\langle a \to b, c \leftrightarrow d/c' \leftrightarrow d' \rangle$ means to pass from ac inside and d outside to bd inside and c outside, and this should be done whenever possible; if the interchange $c \leftrightarrow d$ cannot be done, then we pass from ac' inside and d' outside to ad' inside and c' outside. Note that the first rule is always applied, and that either the first or the second communication rule has to be applied, with priority for the first one.

Any copy of an object can be involved in only one rule. Using the programs in this way, with all agents acting simultaneously or sequentially, non-deterministically choosing the program(s) to be applied, we can pass from one configuration of the system (represented by the contents of the agents and of the environment) to another configuration. Formally, a configuration can be written as an $(n+1)$-tuple $(w_1, ..., w_n; w_E e^\omega)$, with $w_i \in V^2$ representing the objects from agent i, $1 \leq i \leq n$, and $w_E \in (V - \{e\})^*$, representing the objects from the environment different from the "background" object e; e^ω is used as notation for the arbitrarily many copies of the object e that are always contained in the environment; the initial configuration of a system is always $(ee, ..., ee; e^\omega)$. A sequence of transitions is called a *computation.* A computation is said to be *halting,* if a configuration is reached where no program can be applied anymore. With a halting computation we associate a *result* which is given as the number of copies of the objects from T present in the environment in the halting configuration.

Because of the non-determinism in choosing the programs, starting from the initial configuration we obtain several computations, hence, with a P colony we can associate a set of (vectors of) numbers, denoted by $N(\Pi)$, computed by all possible halting computations of Π.

The number of agents in a given P colony is called the *degree* of Π; the maximal number of programs of an agent of Π is called the *height* of Π. The family of all sets of (vectors of) numbers $N(\Pi)$ computed as above by P colonies of degree at most $n \geq 1$ and height at most $h \geq 1$ (using checking programs, i.e., priorities on the communication rules in the programs) working in the sequential mode is denoted by $NPCOL_{seq}(n, h, pri)$, whereas the corresponding families of P colonies working in the maximally parallel way are denoted by $NPCOL_{par}(n, h, pri)$. If one of the parameters n, h is not bounded, then we replace it with $*$. If only P colonies using programs without priorities are taken into account, we omit the parameter *pri*.

3 Results

In this section we will prove our results for P colonies. We start with showing that P colonies of height five but with an unbounded number of agents are computationally complete even when working in the sequential mode. To this aim, we can take over the proof of Theorem 1 given in [4].

Theorem 2. $NPCOL_{seq}(*, 5, pri) = PsRE$.

Proof. Let us consider a register machine $M = (m, P, l_0, l_h)$. All the labels from $lab(M)$ 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 assume the first k registers to contain the final results. More exactly, we construct the P colony

$$\begin{aligned} \Pi &= (V, e, T, B_1, ..., B_n), \\ V &= lab(M) \cup \{a_i \mid 1 \leq i \leq m\} \cup \{l'_0, e, d, d'\}, \\ T &= \{a_i \mid 1 \leq i \leq k\}, \end{aligned}$$

and the following $n = card(P) + 3$ agents and their corresponding programs:

1. We consider the starting agents B_1, B_2 with their sets of programs:
   $P_1 = \{\langle e \to d, e \leftrightarrow e\rangle, \langle e \to l_0, d \leftrightarrow d'\rangle, \langle d' \to l'_0, l_0 \leftrightarrow e\rangle\}$,
   $P_2 = \{\langle e \to d', e \leftrightarrow e\rangle, \langle e \to l'_0, d' \leftrightarrow e\rangle\}$.
   Each of the agents can start with its first program. Agent B_1 can only continue with its second program in step four, when d' is present in the environment after agent B_2 has expelled this symbol after having executed all its programs. Finally, B_1 can release l_0 into the environment to start the simulation of a computation in M. Note that at this moment, both agents have stopped their work (with B_1 and B_2 containing the multiset $\langle l'_0, e\rangle$).

2. For each ADD instruction $l_1 : (A(r), l_2, l_3)$ from P we consider an agent with the set of programs
$$\begin{aligned} P_{l_1} = \quad & \{\langle e \to a_r, e \leftrightarrow l_1\rangle, \\ & \langle l_1 \to l_2, a_r \leftrightarrow e\rangle, \langle e \to e, l_2 \leftrightarrow e\rangle, \\ & \langle l_1 \to l_3, a_r \leftrightarrow e\rangle, \langle e \to e, l_3 \leftrightarrow e\rangle\}. \end{aligned}$$
In the first step, we apply the program $\langle e \to a_r, e \leftrightarrow l_1\rangle$ thus obtaining the multiset $\langle a_r, l_1\rangle$ from the multiset $\langle e, e\rangle$ in the agent. By applying the programs $\langle l_1 \to l_i, a_r \leftrightarrow e\rangle$ and $\langle e \to e, l_i \leftrightarrow e\rangle$, $i \in \{2, 3\}$, we end up with the multiset $\langle e, e\rangle$ in the agent again, whereas in the environment the number of symbols a_r has been incremented and the label symbol l_1 has been replaced by l_i.

3. For each (conditional) SUB instruction $l_1 : (S(r), l_2, l_3)$ from P we consider the agent with the set of programs
$$\begin{aligned} P_{l_1} = \quad & \{\langle e \to e, e \leftrightarrow l_1\rangle, \langle l_1 \to l_2, e \leftrightarrow a_r/e \leftrightarrow e\rangle, \\ & \langle a_r \to e, l_2 \leftrightarrow e\rangle, \\ & \langle l_2 \to l_3, e \leftrightarrow e\rangle, \langle e \to e, l_3 \leftrightarrow e\rangle\}. \end{aligned}$$
After taking the label symbol l_1 inside by applying the rule $\langle e \to e, e \leftrightarrow l_1\rangle$, the application of the checking program $\langle l_1 \to l_2, e \leftrightarrow a_r/e \leftrightarrow e\rangle$ allows for checking whether the environment contains a symbol a_r or not. In case it is present, $\langle l_1 \to l_2, e \leftrightarrow a_r\rangle$ is applied taking one symbol a_r from the environment and yielding the multiset $\langle l_2, a_r\rangle$ inside the agent. Then the program $\langle a_r \to e, l_2 \leftrightarrow e\rangle$ has to be applied, yielding $\langle e, e\rangle$ in the agent again, whereas in the environment the number of symbols a_r has been decremented and the label symbol l_1 has been replaced by l_2. On the other hand, if no symbol a_r is present inside, then instead we execute the programs $\langle l_1 \to l_2, e \leftrightarrow e\rangle$ and afterwards we have to apply $\langle l_2 \to l_3, e \leftrightarrow e\rangle$ and $\langle e \to e, l_3 \leftrightarrow e\rangle$ thus yielding $\langle e, e\rangle$ in the agent again, whereas in the environment the label symbol l_1 has been replaced by l_3.

   As the simulations of the ADD as well as the conditional SUB instruction as defined above are exactly the same as given in the proof of Theorem 1 in [4], we refer to there for some further explanations; yet we have to emphasize the fact that the P colonies constructed there work in the maximally parallel mode, whereas here we work in the sequential mode instead, which in fact only makes a real difference for the work of the first two agents B_1, B_2 when starting the computation.

4. When the final label l_h appears in the environment, from all the agents defined above there is no program to be applied anymore; the final

agent B_n with the following two programs erases l_h and then stops with $\langle e,e\rangle$ in the agent:

$$P_n = \quad \{\langle e \to e, e \leftrightarrow l_h\rangle, \langle l_h \to e, e \leftrightarrow e\rangle\}$$

Thus, the P colony Π halts with with the result consisting of the objects a_r present in the environment, with B_1 and B_2 containing the multiset $\langle l_0', e\rangle$ and all other agents containing $\langle e, e\rangle$. As the environment only contains symbols from T, we could also regard Π as a non-extended system where the result of a halting computation is constituted by the symbols different from e in the environment.

We finally observe that to each agent at most five agents are assigned, which observation completes the proof. $\square$

On the other hand, when we omit the checking programs, i.e., the priorities on the communication rules, but use the maximally parallel derivation mode instead of the sequential mode, we get a similar result by using the idea of "paired" programs: P colonies of height 5 working in the maximally parallel mode but without checking programs are computationally complete.

Theorem 3. $NPCOL_{par}(*,5) = PsRE$.

Proof (sketch). Again we consider a register machine $M = (m, P, l_0, l_h)$, and represent the contents of a register i by the number of copies in the environment of a specific object a_i. Then we construct the P colony

$$\begin{array}{rcl}
\Pi & = & (V, e, T, B_1, ..., B_n) \\
V & = & \left\{ l, \hat{l}, \hat{l}', \tilde{l}, \tilde{l}', \tilde{l}'', \bar{l}, \bar{l}' \mid l \in lab(M) \right\} \\
 & \cup & \{a_i \mid 1 \le i \le m\} \cup \{l_0', e, d, d'\}, \\
T & = & \{a_i \mid 1 \le i \le k\},
\end{array}$$

and the following agents and their corresponding programs (to make the interplay of the agents more visible, we here use a vertical notation of the respective programs):

Each simulation of instruction l starts with l and $\hat{l}$ in the environment; to initialize the whole system, i.e. to produce l_0 and $\hat{l}_0$, we take the following two agents $P_{0,1}$ and $P_{0,2}$:

$$\begin{array}{llll}
P_{0,1} = \{ & \left\langle e \to \tilde{l}_0, e \leftrightarrow e \right\rangle, & P_{0,2} = \{ & \\
 & \left\langle e \to \tilde{l}_0', \tilde{l}_0 \leftrightarrow e \right\rangle, & & \\
 & \left\langle \tilde{l}_0' \to l_0, e \leftrightarrow e \right\rangle, & & \left\langle e \to \hat{l}_0, e \leftrightarrow \tilde{l}_0 \right\rangle, \\
 & \left\langle e \to \tilde{l}_0'', l_0 \leftrightarrow e \right\rangle\} & & \left\langle \tilde{l}_0 \to \tilde{l}_0'', \hat{l}_0 \leftrightarrow e \right\rangle\}
\end{array}$$

Both agents end up with the multiset $\left\langle e, \tilde{l}''_0 \right\rangle$ inside and stop forever; at the same time, i.e., after four steps, they have released l_0 and $\hat{l}_0$ into the environment.

An ADD instruction $l_1 : (ADD\,(r)\,, l_2, l_3)$ can be simulated by the following four agents $P_{l_1,i}$, $i \in \{1, 2, 3, 4\}$:

$$\begin{array}{llll}
P_{l_1,1} = \{ & \left\langle e \to \bar{l}'_1, e \leftrightarrow l_1 \right\rangle, & P_{l_1,2} = \{ & \left\langle e \to e, e \leftrightarrow \hat{l}_1 \right\rangle, \\
& \left\langle l_1 \to a_r, \bar{l}'_1 \leftrightarrow e \right\rangle, & & \left\langle \hat{l}_1 \to e, e \leftrightarrow e \right\rangle, \\
& \left\langle e \to l_2, a_r \leftrightarrow e \right\rangle, & & \left\langle e \to \hat{l}_2, e \leftrightarrow \bar{l}'_1 \right\rangle, \\
& \left\langle e \to e, l_2 \leftrightarrow e \right\rangle \quad \} & & \left\langle \bar{l}'_1 \to e, \hat{l}_2 \leftrightarrow e \right\rangle \quad \} \\
P_{l_1,3} = \{ & \left\langle e \to \tilde{l}'_1, e \leftrightarrow l_1 \right\rangle, & P_{l_1,4} = \{ & \left\langle e \to e, e \leftrightarrow \hat{l}_1 \right\rangle, \\
& \left\langle l_1 \to a_r, \tilde{l}'_1 \leftrightarrow e \right\rangle, & & \left\langle \hat{l}_1 \to e, e \leftrightarrow e \right\rangle, \\
& \left\langle e \to l_3, a_r \leftrightarrow e \right\rangle, & & \left\langle e \to \hat{l}_3, e \leftrightarrow \tilde{l}'_1 \right\rangle, \\
& \left\langle e \to e, l_3 \leftrightarrow e \right\rangle \quad \} & & \left\langle \tilde{l}'_1 \to e, \hat{l}_3 \leftrightarrow e \right\rangle \quad \}
\end{array}$$

Agents $B_{l_1,1}$ and $B_{l_1,2}$ ($B_{l_1,3}$ and $B_{l_1,4}$, respectively) work together in the following way. $B_{l_1,1}$ ($B_{l_1,3}$) takes l_1 from the environment, produces a_r and sends out $\bar{l}'_1$ ($\tilde{l}'_1$) while in the same two steps either $B_{l_1,2}$ or $B_{l_1,4}$ simply “consumes” $\hat{l}_1$ ending up with the multiset $\langle e, e \rangle$ again (due to the maximal parallelism, $B_{l_1,2}$ or $B_{l_1,4}$ have to work in parallel with $B_{l_1,1}$ or $B_{l_1,3}$, respectively). In the subsequent steps, $B_{l_1,1}$ ($B_{l_1,3}$) sends out a_r and l_2 (l_3), whereas now $B_{l_1,3}$ ($B_{l_1,4}$) produces and sends out $\hat{l}_2$ ($\hat{l}_3$), allowing for the next instruction l_2 or l_3, respectively, to be simulated.

A conditional SUB operation $l_1 : (SUB\,(r)\,, l_2, l_3)$ can be simulated by the following six agents $P_{l_1,i}$, $i \in \{1, 2, 3, 4, 5, 6\}$:

$$\begin{array}{llll}
P_{l_1,1} = \{ & \left\langle e \to e, e \leftrightarrow l_1 \right\rangle, & P_{l_1,2} = \{ & \left\langle e \to \hat{l}'_1, e \leftrightarrow \hat{l}_1 \right\rangle, \\
& \left\langle l_1 \to \bar{l}_1, e \leftrightarrow a_r \right\rangle, & & \left\langle \hat{l}_1 \to e, \hat{l}'_1 \leftrightarrow e \right\rangle \quad \} \\
& \left\langle a_r \to e, \bar{l}_1 \leftrightarrow e \right\rangle, & & \\
& \left\langle l_1 \to \tilde{l}_1, e \leftrightarrow \hat{l}'_1 \right\rangle, & & \\
& \left\langle \hat{l}'_1 \to e, \tilde{l}_1 \leftrightarrow e \right\rangle \quad \} & &
\end{array}$$

In the first step, $B_{l_1,1}$ and $B_{l_1,2}$ take in the labels l_1 and $\hat{l}_1$. In the second step, $B_{l_1,2}$ sends out $\hat{l}'_1$, whereas $B_{l_1,1}$,

1. in case there is an object a_r in the environment, takes in a_r and then sends out $\bar{l}_1$;
2. in case there is no object a_r present outside, consumes (after having waited one step) $\hat{l}'_1$ and finally sends out $\tilde{l}_1$.

With $\bar{l}_1$ in the environment, $B_{l_1,3}$ and, two steps later, $B_{l_1,4}$ become active; they produce the corresponding labels l_2 and $\hat{l}_2$:

$$\begin{array}{llll} P_{l_1,3} = \{ & \langle e \to \bar{l}'_1, e \leftrightarrow \bar{l}_1 \rangle, & P_{l_1,4} = \{ & \\ & \langle \bar{l}_1 \to \bar{l}''_1, \bar{l}'_1 \leftrightarrow e \rangle, & & \\ & \left\langle \bar{l}''_1 \to l_2, e \leftrightarrow \hat{l}'_1 \right\rangle, & & \left\langle e \to \hat{l}_2, e \leftrightarrow \bar{l}'_1 \right\rangle, \\ & \left\langle \hat{l}'_1 \to e, l_2 \leftrightarrow e \right\rangle \quad \} & & \left\langle \hat{l}'_1 \to e, \hat{l}_2 \leftrightarrow e \right\rangle \quad \} \end{array}$$

If, on the other hand, $\tilde{l}_1$ was expelled by $B_{l_1,1}$, then only $B_{l_1,5}$ and $B_{l_1,6}$ can apply their programs in a similar way as $B_{l_1,3}$ and $B_{l_1,4}$, finally sending out l_3 and $\hat{l}_3$:

$$\begin{array}{llll} P_{l_1,5} = \{ & \left\langle e \to \tilde{l}'_1, e \leftrightarrow \tilde{l}_1 \right\rangle, & P_{l_1,6} = \{ & \\ & \left\langle \tilde{l}_1 \to \tilde{l}''_1, \tilde{l}'_1 \leftrightarrow e \right\rangle, & & \\ & \left\langle \tilde{l}''_1 \to l_3, e \leftrightarrow e \right\rangle, & & \left\langle e \to \hat{l}_3, e \leftrightarrow \tilde{l}'_1 \right\rangle \\ & \langle e \to e, l_3 \leftrightarrow e \rangle \quad \} & & \left\langle \tilde{l}'_1 \to e, \hat{l}_3 \leftrightarrow e \right\rangle \quad \} \end{array}$$

When the final label l_h appears in the environment, from all the agents defined above there is no program to be applied anymore; the final agent B_h with the following two programs erases l_h and then stops with $\langle e, e \rangle$ in the agent:

$$P_h = \quad \{ \langle e \to e, e \leftrightarrow l_h \rangle, \langle l_h \to e, e \leftrightarrow e \rangle \}$$

Thus, the P colony Π halts with the result consisting of the objects a_r present in the environment, with B_1 and B_2 containing the multiset $\left\langle e, \tilde{l}''_0 \right\rangle$ and all other agents containing $\langle e, e \rangle$.

Finally, we again observe that each agent has at most five programs not containing a checking program, which observation concludes the proof. $\square$

If instead of bounding the height the number of agents is bounded, two agents were shown to be sufficient to obtain computational completeness in

[4]. We here can show that one agent with checking programs is already enough in this case (observe that one agent obviously by definition works in the sequential mode).

Theorem 4. *$NPCOL_{seq}(1, *, pri) = PsRE$.*

Proof (sketch). We take the idea given in the proof of Theorem 2 in [4], yet now there is only one agent instead of two. Once more we consider a register machine $M = (m, P, l_0, l_h)$, and represent the contents of a register i by the number of copies in the environment of a specific object a_i. We now construct the P colony

$$\begin{array}{rcl} \Pi & = & (V, e, T, B_1) \\ V & = & \{l, l', l'' \mid l \in lab(M)\} \cup \{a_i \mid 1 \leq i \leq m\} \cup \{\sigma, \sigma'\}, \\ T & = & \{a_i \mid 1 \leq i \leq k\}, \end{array}$$

and the following (possibly checking) programs in P_1 for the single agent B_1:

To start the simulation, the agent B_1 has to perform the following programs:

$$\langle e \to \sigma, e \leftrightarrow e \rangle, \langle e \to \sigma', \sigma \leftrightarrow e \rangle, \langle \sigma' \to \alpha, e \leftrightarrow \sigma \rangle, \langle \sigma \to \sigma, \alpha \leftrightarrow e \rangle,$$
$$\langle \sigma \to l_0, \alpha \leftrightarrow e \rangle.$$

By the first four programs, objects $\alpha \in \{l', l'' \mid l \in lab(M)\}$ are produced and sent to the environment. If at some moment, the program $\langle \sigma \to l_0, \alpha \leftrightarrow e \rangle$ is used, then the "dummy" object σ disappears forever and the simulation of the computation in M can start with the agent containing the multiset $\langle l_0, e \rangle$.

To simulate an ADD instruction $l_1 : (A(r), l_2, l_3)$ from P we have to include the following programs to P_1 :

$$\langle l_1 \to l_1', e \leftrightarrow e \rangle, \langle e \to a_r, l_1' \leftrightarrow l_1' \rangle, \langle l_1' \to l_2, a_r \leftrightarrow e \rangle, \langle l_1' \to l_3, a_r \leftrightarrow e \rangle$$

With these programs an object a_r is produced and sent to the environment while at the same time the corresponding label to proceed is generated so that after performing these programs, the agent contains either the multiset $\langle l_2, e \rangle$ or the multiset $\langle l_3, e \rangle$ and then can go on simulating the next instruction.

To simulate a conditional SUB instruction $l_1 : (S(r), l_2, l_3)$ from P we add the following programs to P_1 :

$$\langle l_1 \to l_1', e \leftrightarrow a_r / e \leftrightarrow e \rangle, \langle a_r \to e, l_1' \leftrightarrow l_1'' \rangle,$$
$$\langle l_1'' \to l_2, e \leftrightarrow e \rangle, \langle l_1' \to l_3, e \leftrightarrow e \rangle$$

In the case that at least one copy of a_r is present in the environment, it is brought into the agent and changed into e before producing label l_2 to proceed. Otherwise, after having checked that there is no copy of a_r present in the environment, label l_3 is produced. Hence, after simulating the conditional SUB instruction from P, the contents of the agent is either $\langle l_2, e \rangle$ or $\langle l_3, e \rangle$ allowing for the next instruction l_2 or l_3 to be simulated.

To make sure that sufficiently many primed versions of the corresponding labels have been produced before starting the simulation, we also have to add the program $\langle l_1' \to l_1', e \leftrightarrow e \rangle$. This program ensures the computation to go on forever in case the communication rules $l_1' \leftrightarrow l_1'$ or $l_1' \leftrightarrow l_1''$ cannot be used.

When the final label l_h appears in the environment, none of the programs defined above can be applied anymore; for eliminating all objects $\alpha \in \{l', l'' \mid l \in lab(M)\}$ in the environment, we use the following programs:

$$\langle l_h \to l_h, e \leftrightarrow \alpha \rangle, \langle \alpha \to l_h', l_h \leftrightarrow e \rangle, \langle l_h' \to e, e \leftrightarrow l_h \rangle$$

Thus, the P colony Π finally halts with the result consisting of the objects a_r present in the environment and with B_1 containing the multiset $\langle e, l_h \rangle$.

□

Without using checking programs, i.e., without priorities on the communication rules, P colonies working in the sequential mode only characterize the family of Parikh sets of languages generated by matrix grammars or equivalently, the family of sets of vectors of non-negative integers generated by partially blind register machines:

Theorem 5. $NRM_{pb} = NPCOL_{seq}(1,*) = NPCOL_{seq}(*,*) = PsMAT$.

Proof. We will prove the following sequence of inclusions:

$$NRM_{pb} \subseteq NPCOL_{seq}(1,*) \subseteq NPCOL_{seq}(*,*) \subseteq PsMAT \subseteq NRM_{pb}$$

The first inclusion

$$NRM_{pb} \subseteq NPCOL_{seq}(1,*)$$

is an immediate consequence of the preceding theorem - to simulate an unconditional SUB instruction $l_1 : (S(r), l_2, l_3)$ from P of a register machine $M = (m, P, l_0, l_h)$ we now use the following programs for B_1 (we have replaced the checking program $\langle l_1 \to l_1', e \leftrightarrow a_r / e \leftrightarrow e \rangle$ by the non-checking program $\langle l_1 \to l_1', e \leftrightarrow a_r \rangle$):

$$\langle l_1 \to l_1', e \leftrightarrow a_r \rangle, \langle a_r \to e, l_1' \leftrightarrow l_1'' \rangle,$$
$$\langle l_1'' \to l_2, e \leftrightarrow e \rangle, \langle l_1'' \to l_3, e \leftrightarrow e \rangle,$$

The only problem we have to face now is that the first program may not be applicable due to the absence of the symbol a_r in the environment. For that purpose, we add the program

$$\langle l_1 \to l_1', e \leftrightarrow e \rangle$$

After having executed this program, we end up in an infinite loop with the program $\langle l_1' \to l_1', e \leftrightarrow e \rangle$. Hence, a result will only be computed if we never have tried to subtract from zero, which is just the same idea as used in partially blind register machines. On the other hand, at the end of a simulation of a computation in M the condition of halting for the P colony Π allows us to check whether all the other registers are empty, i.e., to the programs

$$\langle l_h \to l_h, e \leftrightarrow \alpha \rangle, \langle \alpha \to l_h', l_h \leftrightarrow e \rangle, \langle l_h' \to e, e \leftrightarrow l_h \rangle$$

constructed in the proof of the preceding theorem for eliminating all objects $\alpha \in \{l', l'' \mid l \in lab(M)\}$ in the environment, we now also add the corresponding programs for $\alpha \in \{a_i \mid k+1 \leq i \leq m\}$, too, i.e.,

$$\langle l_h \to l_h, e \leftrightarrow \alpha \rangle, \langle \alpha \to l_h', l_h \leftrightarrow e \rangle.$$

The remaining elements of the P colony are just the same as those of the P colony constructed in the proof of the preceding theorem; we leave the remaining details to the reader.

The inclusion

$$NPCOL_{seq}(1, *) \subseteq NPCOL_{seq}(*, *)$$

is an immediate consequence of the definitions.

The main contribution in this proof now is to show the inclusion

$$NPCOL_{seq}(*,*) \subseteq PsMAT.$$

Let $\Pi = (V, e, T, B_1, ..., B_n)$ be a P colony without checking programs. We now (sketch how to) construct the matrix grammar $G = (N, T, S, M)$ simulating Π as follows:

The symbols $a \neq e$ in the agents and in the environment are represented by non-terminal symbols: For a symbol $a \in V$ in the agent i, $1 \leq i \leq n$, we use the non-terminal (a, i); for a symbol $a \in V - \{e\}$ (observe that e occurs in an unbounded number in the environment) in the environment, we use the non-terminal $(a, 0)$.

In G, we start with the initial matrix

$$\left(S \to E\,(e,1)^2 \ldots (e,n)^2\right).$$

For simulating the program in P_i for the agent B_i

$$\langle a \to b, c \leftrightarrow d \rangle$$

we then use the matrix

$$(E \to E, (a,i) \to (b,i), (c,i) \to h(c), (d,0) \to (d,i))$$

for $d \neq e$ and the matrix

$$(E \to E, (a,i) \to (b,i), (c,i) \to h(c), E \to (e,i))$$

in the case $d = e$, where
$h : (V \times \{1, ..., n\})^* \to (V \times \{0\})^*$
is the morphism with

$$h((c,i)) = (c,0) \text{ for } c \neq e \text{ and}$$
$$h((e,i)) = \lambda.$$

In that way, the execution of a program in the P colony Π can easily be simulated by the application of the corresponding matrix in the matrix grammar G.

The main difficulty arises when we non-deterministically have to guess when Π has reached a halting configuration where no program can be applied any more, hence, G now has to filter out the terminal symbols in the environment and to erase all the remaining non-terminal symbols. Therefore we have to consider all different possibilities for configurations to be halting ones; fortunately, these configurations can be described by a finite set of strings (lists of variables) of the form

$a_{1,1}a_{1,2}...a_{n,1}a_{n,2}a_{0,1}...a_{0,l}$

for some $l < card(V)$, the $a_{0,j}$, $1 \leq j \leq l$, being different symbols from $V - \{e\}$ representing all those symbols occurring in the form $(a_{0,j}, i)$ at least once in the environment, whereas $a_{i,1}, a_{i,2}$, $1 \leq i \leq n$, represent the multiset in the corresponding agent i appearing as $(a_{i,1}, i)$ and $(a_{i,2}, i)$ in the sentential form produced by the matrix grammar. According to our assumptions, each of these strings describes a situation where no program from the P colony Π can be applied any more, i.e., a halting configuration. Non-deterministically we now guess such a situation and define matrices which allow us to eliminate all those non-terminal symbols indicated by the corresponding string:

$(E \to (F, a_{1,1}a_{1,2}...a_{n,1}a_{n,2}a_{0,1}...a_{0,l}))$,

$((F, a_{1,1}a_{1,2}...a_{n,1}a_{n,2}a_{0,1}...a_{0,h}) \to$

$(F, a_{1,1}a_{1,2}...a_{n,1}a_{n,2}a_{0,1}...a_{0,h}), a_{0,h} \to \lambda)$,

for $1 \leq h \leq l$,

$((F, a_{1,1}a_{1,2}...a_{n,1}a_{n,2}a_{0,1}...a_{0,h}) \to (F, a_{1,1}a_{1,2}...a_{n,1}a_{n,2}a_{0,1}...a_{0,h-1}))$,

for $1 < h \leq l$,

$((F, a_{1,1}a_{1,2}...a_{n,1}a_{n,2}a_{0,1}) \to (F, a_{1,1}a_{1,2}...a_{n,1}a_{n,2}))$,

$((F, a_{1,1}...a_{i,f}) \to (F, a_{1,1}...a_{i,f}), a_{i,f} \to \lambda)$,

for $1 \leq i \leq n$ for $1 \leq f \leq 2$,

$((F, a_{1,1}...a_{i,2}) \to (F, a_{1,1}...a_{i,1}))$,

for $1 \leq i \leq n$,

$((F, a_{1,1}...a_{i,1}) \to (F, a_{1,1}...a_{i-1,2}))$,

for $1 < i \leq n$, and

$((F, a_{1,1}) \to \lambda)$.

In that way, we can eliminate all remaining non-terminal symbols and extract the terminal result provided we make the correct non-deterministic choices for the matrices to be applied. If at least one of the guesses is not correct, then not all the remaining non-terminal symbols are erased from the sentential form obtained after having simulated a computation in the P colony by the matrix grammar. From the explanations given so far it should have become clear that we can derive a terminal word w in the matrix grammar G if and only if the Parikh vector of w is the result of a halting computation in Π, which proves the inclusion $NPCOL_{seq}(*,*) \subseteq PsMAT$.

To complete the proof of the theorem, it only remains to prove the inclusion

$$PsMAT \subseteq NRM_{pb}.$$

Although the equality $PsMAT = NRM_{pb}$ is quite folklore, we sketch a proof of the inclusion $PsMAT \subseteq NRM_{pb}$ in order to have included complete proofs of all results stated in this paper:

Let $G = (N, T, S, M)$ be a matrix grammar $G = (N, T, S, M)$; without loss of generality we assume G to be in the f-binary normal form. We then construct a partially blind register machine $M' = (m, P, 1, n)$ with $Ps\left(L\left(G\right)\right) = L\left(M'\right)$ as follows:

We bijectively label the matrices in M with $1, ..., n$. For each symbol in $V = N \cup T$ we use a register named by this symbol.

1. The initial matrix $1 : (S \rightarrow XA)$, with $X \in N_1, A \in N_2$, is simulated by the instructions

   $1 : \left(S\left(S\right), \left(1', 1\right), 1'\right),\ \left(1', 1\right) : \left(A\left(A\right), l_1, \left(1, 2\right)\right),$

   $\left(1, 2\right) : \left(S\left(A\right), \left(1', 2\right), 1'\right),\ \left(1', 2\right) : \left(A\left(A\right), l_2, 1'\right),\ ...,$

   $\left(1, k\right) : \left(S\left(A\right), \left(1', k\right), 1'\right),\ \left(1', k\right) : \left(A\left(A\right), l_k, 1'\right),$

   where $l_1, ..., l_k$ are chosen in such a way that these are exactly all the labels of matrices which are of the form $(X \rightarrow Y, A \rightarrow x)$;

2. The matrices $l : (X \rightarrow Y, A \rightarrow x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$, are simulated depending on the length of x as follows:

   (a) $|x| = 0:$
   $l : \left(S\left(A\right), l_1, \left(1, 2\right)\right),$
   $\left(l, 2\right) : \left(A\left(A\right), \left(l', 2\right), l'\right),\ \left(l', 2\right) : \left(S\left(A\right), l_2, \left(l, 3\right)\right),\ ...,$
   $\left(l, k\right) : \left(A\left(A\right), \left(l', k\right), l'\right),\ \left(l', k\right) : \left(A\left(A\right), l_k, l'\right);$

   (b) $x = B:$
   $l : \left(S\left(A\right), \left(l', 1\right), l'\right),\ \left(l', 1\right) : \left(A\left(B\right), l_1, \left(l, 2\right)\right),$
   $\left(l, 2\right) : \left(S\left(B\right), \left(l', 2\right), l'\right),\ \left(l', 2\right) : \left(A\left(B\right), l_2, \left(l, 3\right)\right),\ ...,$
   $\left(l, k\right) : \left(S\left(B\right), \left(l', k\right), l'\right),\ \left(l', k\right) : \left(A\left(B\right), l_k, l'\right);$

(c) $x = CB$:
$l : (S(A), (l', 1), l')$, $(l', 1) : (A(C), (l'', 1), l')$,
$(l'', 1) : (A(B), l_1, (l, 2))$,
$(l, 2) : (S(B), (l', 2), l')$, $(l', 2) : (A(C), (l'', 2), l')$,
$(l'', 2) : (A(B), l_2, (l, 3))$,...,
$(l, k) : (S(B), (l', k), l')$, $(l', k) : (A(C), (l'', k), l')$,
$(l'', k) : (A(B), l_k, l')$;

in any case, $l_1, ..., l_k$ are are chosen in such a way that these are exactly all the labels of matrices which are of the form $(X \to Y, A \to x)$;

3. The matrices $l : (X \to f, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*, |x| \leq 2$, again are simulated depending on the length of x as follows:

   (a) $|x| = 0$:
   $l : (S(A), n, (1, 2))$;

   (b) $x = B$:
   $l : (S(A), (l', 1), l')$, $(l', 1) : (A(B), n, l')$;

   (c) $x = CB$:
   $l : (S(A), (l', 1), l')$, $(l', 1) : (A(C), (l'', 1), l')$,
   $(l'', 1) : (A(B), n, l')$;

4. The final matrix $n : (f \to \lambda)$ is simulated by the instruction
   $n : (S(f), l_h, l_h')$.

The whole program P for the partially blind register machine M' is obvious from the construction described above; moreover, when M' reaches the final label l_h, then all registers representing the numbers of non-terminal symbols are empty if and only if the simulated derivation in the matrix grammar G has yielded a terminal word, which completes the proof of the inclusion $PsMAT \subseteq NRM_{pb}$ and the proof of the theorem as well. □

4 Conclusion

We have shown that P colonies working in the sequential mode with checking programs of height at most 5 are computationally complete, whereas when working in the maximally parallel mode, P colonies using programs again with height 5 but without priorities on the communication rules (i.e., without checking programs) achieve the same computational power, too.

Already one agent using checking programs is enough to obtain computational completeness in P colonies working in the sequential mode, which is a quite surprising result optimal with respect to the number of agents. If we only allow non-checking programs, even P colonies with only one agent as well as P colonies with an arbitrary number of agents working in the sequential mode characterize the family of languages generated by matrix grammars without appearance checking.

References

[1] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun: *Grammar Systems: A Grammatical Approach to Distribution and Cooperation.* Gordon and Breach, London (1994)

[2] J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin (1989)

[3] J. Kelemen, A. Kelemenová: A grammar-theoretic treatment of multiagent systems. *Cybernetics and Systems* 23 (1992) 621-633

[4] J. Kelemen, A. Kelemenová, Gh. Păun: On the power of a biochemically inspired simple computing model: P colonies. Downloadable version at [9]

[5] M. Minsky: *Computation. Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ (1967)

[6] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences* **61,** 1 (2000) 108–143, and TUCS Research Report **208** (1998) (`http://www.tucs.fi`)

[7] Gh. Păun: *Membrane Computing: an Introduction.* Springer, Berlin (2002)

[8] A. Salomaa, G. Rozenberg (eds.): *Handbook of Formal Languages.* Springer-Verlag, Berlin (1997)

[9] The P systems webpage `http://psystems.disco.unimib.it`