P Colonies with a Bounded Number of Cells and Programs*

Erzsébet Csuhaj-Varjú[1,2], Maurice Margenstern[3], and György Vaszil[1]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences
Kende utca 13–17, H-1111 Budapest, Hungary
{csuhaj,vaszil}@sztaki.hu
[2] Department of Algorithms and Their Applications, Loránd Eötvös University
Pázmány Péter sétány 1/c, H-1117 Budapest, Hungary
[3] Université Paul Verlaine - Metz, LITA, EA 3097
Île du Saulcy, 57045 Metz Cedex 1, France
margens@univ-metz.fr

Abstract. We continue the investigation of P colonies, a class of abstract computing devices composed of very simple agents, acting and evolving in a shared environment. We show that if P colonies are initialized by placing a number of copies of a certain object in the environment, then they can generate any recursively enumerable set of numbers with a bounded number of cells, each cell containing a bounded number of programs (of bounded length).

1 Introduction

P colonies were introduced in [4] as a class of membrane systems similar to the so called colonies of simple formal grammars ([3]). A P colony intends to model a community of very simple cells living together in a shared environment. The cells are represented by a collection of objects and rules for processing these objects, they are the basic computing agents in this formal model of computing.

To restrict the capabilities of the agents, only two (or three) objects are allowed to be inside any cell. Moreover, the rules of the cells are either of the form $a \to b$, specifying that an internal object a is transformed into an internal object b, or of the form $c \leftrightarrow d$, specifying the fact that an internal object c is sent out of the cell, to the environment, in exchange of the object d, which was present in the environment and is now brought inside the cell. Thus, a cell containing the objects a, c will contain the objects b, d after applying these rules. Two such rules can also be combined into so called "checking" rules of the from $c \leftrightarrow d/c' \leftrightarrow d'$ or $c \leftrightarrow d/a \to b$ which specifies two possible actions: if the first rule is not applicable then the second one should be applied.

* This publication was supported in part by the Hungarian Foundation for Research and Technological Innovation (project no. TéT F-19/04) and the EGIDE in France (project no. Balaton 09000TC, year 2005) in the frame of the Hungarian-French Intergovernmental Scientific and Technological Cooperation, and by the Hungarian Scientific Research Fund "OTKA" grant no. T 042529.

With each cell we associate a set of programs composed of rules as above. In the case of systems consisting of cells with only two objects inside, each program will have two rules; when considering cells with three objects inside, then the programs will have three rules.

The cells of a P colony execute a computation by synchronously applying their programs to objects inside the cells and outside in the environment. When a halting configuration is reached, that is, when no more rules can be applied, the result of the computation is read as the number of certain types of objects present in the environment.

For more information on membrane computing, see the monograph [7], for more on grammar systems and colonies in particular, see [1] and [3], respectively.

It was shown in [4] and [2] that P colonies are able to compute any recursively enumerable set of numbers, even in the situation when the starting configuration can only contain an infinite number of copies of one object in the environment, and two or three copies of the same one object inside the cells. The number of necessary cells or the number of necessary programs, however, has to be unbounded to reach this power.

In the present paper we show, that if the environment is allowed to be initialized by a finite multiset of objects before the computation begins, then P colonies generate any recursively enumerable set of natural numbers with a bounded number of cells and a bounded number of programs in each cell. The values of the bounds presented in our results depend on the type of rules the P colonies are allowed to use and seem to suggest a trade-off between the number of necessary cells and the number of necessary programs in each cell. Our results demonstrate that one cell with a bounded, but fairly large amount of programs might possess the power of Turing machines, which power can also be reached by several cells and a significantly lower number of programs in each cell.

2 Preliminaries and Definitions

Let V be an alphabet, let V^* be the set of all words over V, and let ε denote the empty word. We denote the length of a word $w \in V^*$ by $|w|$, and the number of occurrences of a symbol $a \in V$ in w by $|w|_a$. The set of non-negative integers is denoted by $\mathbb{N}$.

A multiset over an arbitrary (not necessarily finite) set V is a mapping $M : V \to \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The support of M is the set $supp(M) = \{a \mid M(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. The set of all finite multisets over the finite set V is denoted by V°. We say that $a \in M$ if $M(a) \geq 1$, the cardinality of M, $card(M)$ is defined as $card(M) = \Sigma_{a \in M} M(a)$. For two multisets $M_1, M_2 : V \to \mathbb{N}$, $M_1 \subseteq M_2$ holds, if for all $a \in V$, $M_1(a) \leq M_2(a)$. The union of M_1 and M_2 is defined as $(M_1 \cup M_2) : V \to \mathbb{N}$ with $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$ for all $a \in V$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) : V \to \mathbb{N}$ with $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$. A multiset M is empty if its support is empty, $supp(M) = \emptyset$.

We will represent a finite multiset M over V by a string w over the alphabet V with $|w|_a = M(a)$, $a \in V$, and ε will represent the empty multiset.

Now we recall the definition of a P colony. A *P colony* is a construct

$$\Pi = (V, e, o_f, I_E, C_1, \ldots, C_n),\ n \geq 1,$$

where V is an alphabet (its elements are called *objects*), e (the environmental object) and o_f (the final object) are two distinguished objects of V, $I_E \in (V - \{e\})^\circ$ is a finite multiset of objects initially present in the environment besides the infinitely many copies of e, and $C_1, \ldots, C_n$ are the *cells* of the colony. In [2] and in [4], the multiset I_E is empty, that is, initially only an infinite supply of e objects are present in the environment. Here we define I_E to be a finite non-empty multiset, because we would like to allow the initialization of the colony by placing objects different from e in the environment.

Each cell C_i, $1 \leq i \leq n$, is a pair $C_i = (O_i, P_i)$, where O_i is a multiset over $\{e\}$ having the same cardinality for all i, $1 \leq i \leq n$ (the initial state of the cell), and P_i is a finite set of *programs*; each program being a set of rules of the forms $a \to b$ (internal point mutation), $c \leftrightarrow d$ (one object exchange with the environment), $c \leftrightarrow d/c' \leftrightarrow d'$ (checking rule for one object exchange with the environment), or $c \leftrightarrow d/a \to b$ (checking rule for one object exchange with the environment or internal point mutation), where $a, b, c, d, c', d' \in V$. The programs contain one rule for each element of O_i, thus, the number of rules of a program coincides with the cardinality of O_i, $1 \leq i \leq n$.

A program is called restricted if it contains one point mutation rule of the form $a \to b$, and either one exchange rule of the form $c \leftrightarrow d$, or one checking rule of the form $c \leftrightarrow d/c' \leftrightarrow d'$. A P colony is called restricted if it contains two objects in each cell and has only restricted programs. Two object colonies with non-restricted programs, or three object colonies are called non-restricted.

The programs of the cells are used in the non-deterministic maximally parallel way usual in membrane computing: in each time unit, each cell which can use one of its programs should use one. When using a program, each of its rules must be applied to distinct objects of the cell. In this way, we get transitions among the configurations of the colony. A sequence of transitions is a *computation*. A computation is halting if it reaches a configuration where no cell can use any program. The result of a halting computation is the number of copies of the object o_f present in the environment in the halting configuration. Initially, the environment contains I_E, a finite number of copies of objects from $V - \{e\}$, plus arbitrarily many copies of the environmental object e. Moreover, as stated above, the cells also contain two or three copies of e inside.

Because of the non-determinism in choosing the programs, several computations can be obtained from a given initial configuration, hence with a P colony Π we can associate a set of numbers computed by all possible halting computations of Π.

For a P colony $\Pi = (V, e, o_f, I_E, C_1, \ldots, C_n)$ as above, a configuration can be formally written as an $(n+1)$-tuple

$$(w_1, \ldots, w_n; w_E),$$

where w_i represents the multiset of objects from cell C_i, $1 \leq i \leq n$ ($w_i \in V^2$ in two-objects colonies and $w_i \in V^3$ in three-objects colonies), and $w_E \in (V - \{e\})^*$ represents the multiset of objects from the environment different from the "background" object e.

The initial configuration is $(ee, \ldots, ee; I_E)$ in the case of two-objects colonies and $(eee, \ldots, eee; I_E)$ in the case of three-objects colonies where $I_E \in (V - \{e\})^*$.

Let the programs of each P_i be labeled in a one-to-one manner by labels in the set $lab(P_i)$ in such a way that $lab(P_i) \cap lab(P_j) = \emptyset$ for $i \neq j$, $1 \leq i, j \leq n$. For a rule r, and a multiset $w \in V^\circ$, let $left(r, w) = a$ and $right(r, w) = b$ if r is a point mutation rule $r = (a \to b)$, or a checking rule $r = (c \leftrightarrow d/a \to b)$ with $d \notin w$, and let $left(r, w) = right(r, w) = \varepsilon$ otherwise. Let also, for a rule r and a multiset $w \in V^\circ$ $export(r, w) = c$ and $import(r, w) = d$ if r is an exchange rule $r = (c \leftrightarrow d)$, or a checking rule $r = (c \leftrightarrow d/c' \leftrightarrow d')$ with $d \in w$. If r is a checking rule as above with $d \notin w$ but $d' \in w$, then let $export(r, w) = c'$, $import(r, w) = d'$. Let $export(r, w) = import(r, w) = \varepsilon$ in all other cases. For a program p and any $\alpha \in \{left, right, export, import\}$, let $\alpha(p, w) = \bigcup_{r \in p} \alpha(r)$.

A transition from a configuration to another is denoted as

$$(w_1, \ldots, w_n; w_E) \Rightarrow (w'_1, \ldots, w'_n; w'_E)$$

where the following conditions are satisfied: There is a set of program labels P with $|P| \leq n$, such that $p, p' \in P$, $p \neq p'$, $p \in lab(P_j)$ implies $p' \notin lab(P_j)$, and for each $p \in P$, $p \in lab(P_j)$, $left(p, w_E) \cup export(p, w_E) = w_j$, and $\bigcup_{p \in P} import(p, w_E) \subseteq w_E$. Furthermore, the chosen set P is maximal, that is, if any other program $r \in \bigcup_{1 \leq i \leq n} lab(P_i)$, $r \notin P$, is added to P, then the conditions above are not satisfied.

Now, for each j, $1 \leq j \leq n$, for which there exists a $p \in P$ with $p \in lab(P_j)$, let

$$w'_j = right(p, w_E) \cup import(p, w_E).$$

If there is no $p \in P$ with $p \in lab(P_j)$ for some j, $1 \leq j \leq n$, then let

$$w'_j = w_j,$$

and moreover, let

$$w'_E = w_E - \bigcup_{p \in P} import(p, w_E) \cup \bigcup_{p \in P} export(p, w_E).$$

A configuration is halting if the set of program labels P satisfying the conditions above cannot be chosen to be other than the empty set, $\emptyset$.

The set of numbers computed by a P colony Π is defined as

$$N(\Pi) = \{|v_E|_{o_f} \mid (w_1, \ldots, w_n, I_E) \Rightarrow^* (v_1, \ldots, v_n, v_E)\}$$

where $(w_1, \ldots, w_n, I_E)$ is the initial configuration, $(v_1, \ldots, v_n, v_E)$ is a halting configuration, and $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

The family of all sets of numbers computed as above by initialized environment P colonies with k-objects ($k = 2, 3$) of degree at most $n \geq 1$ having

at most $h \geq 1$ programs in the cells without using checking rules, is denoted by $IPCol\_k(n,h)$. In the case of two-objects colonies, if we use only restricted programs, then we write $IPCol\_2R$ instead of $IPCol\_2$. If checking rules are allowed, then we write $IPCol\_Ch\_k$ instead of $IPCol\_k$; thus, for instance, $IPCol\_Ch\_2R(n,h)$ will be the family of numbers computed by restricted two-objects P colonies with at most n cells, each having at most h programs where the use of checking rules is allowed.

In the following we compare the families $IPCol\_\alpha(n,h)$, where $\alpha \in \{2,3,2R,Ch\_2,Ch\_3,Ch\_2R\}$, with $\mathbb{N}RE$, the family of sets of numbers computed by Turing machines, and for this we need the notion of a *register machine*. A register machine consists of a given number of registers each of which can hold an arbitrarily large non-negative integer number (we say that the register is empty if it holds the value zero), and a set of labeled instructions which specify how the numbers stored in registers can be manipulated. There are several types of instructions which can be used.

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ - add 1 to register r and then go to one of the instructions with labels l_j or l_k, non-deterministically chosen,
- $l_i : (\texttt{SUB}(r), l_j)$ - if register r is non-empty, then subtract 1 from it, otherwise leave it unchanged, and go to the instruction with label l_j in both cases,
- $l_i : (\texttt{CHECK}(\texttt{r}), l_j, l_k)$ - if the value of register r is zero, go to instruction l_j, otherwise go to l_k,
- $l_i : (\texttt{CHECKSUB}(r), l_j, l_k)$ - if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k,
- $l_h : \texttt{HALT}$ - halt the machine.

Thus, formally, a *register machine* is a construct $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halting label, and R is the set of instructions; each label from H labels only one instruction from R. A register machine M computes a set $N(M)$ of numbers in the following way: it starts with empty registers by executing the instruction with label l_0 and proceeds by applying instructions as indicated by the labels (and made possible by the contents of the registers); if the halt instruction is reached, then the number stored at that time in register 1 is said to be computed by M. Because of the non-determinism in choosing the continuation of the computation in the case of ADD instructions, $N(M)$ can be an infinite set.

It is known (see, e.g., [6]) that in this way we can compute all sets of numbers which are Turing computable by using instructions of type ADD, CHECKSUB, and HALT. It is also known, that there exist universal register machines with a small number of registers and a small number of instructions, the exact numbers depending on the chosen set of instructions and the chosen notion of universality.

In [5] several results on small universal register machines are presented. The register machines in this framework are used to compute functions of non-negative integers by having the argument of the function in one of the registers before the computation starts, and obtaining the result of the function in an other register after a halting computation. The universal machines have eight

registers, and they can simulate the computation of any register machine M with the help of a "program", an integer $code(M) \in \mathbb{N}$ coding the particular machine M. If $code(M)$ is placed in the second register and an argument $x \in \mathbb{N}$ is placed in the third register, then the universal machine simulates the computation of M by halting if and only if M halts, and by producing the same result in its first register as M produces in its output register after a halting computation.

Since these machines are defined to compute functions of non-negative integers and work in such a way that the argument of the function is initially present in the third register, we need to modify them in order to conform to the number generating definition we have given above. Thus, we add a new start label l_0' and a separate non-deterministic ADD instruction, $l_0' : (\texttt{ADD}(r_3), l_0', l_0)$, to produce an argument $x \in \mathbb{N}$ in the third register before the actual computation begins, that is, to make the resulting universal machine generate any value from the range of the function computed by the simulated register machine. We summarize the results we use from [5] in the following theorem.

Theorem 1. [5] *Let $\mathbb{M}$ be the set of register machines. Then, there are register machines U_1, U_2, U_3 with eight registers and a recursive function $g : \mathbb{M} \to \mathbb{N}$ such that for each $M \in \mathbb{M}$, $N(M) = N(U_i(g(M)))$, where $N(U_i(g(M)))$ denotes the set of numbers computed by U_i, $1 \leq i \leq 3$, with initially containing $g(M)$ in the second register.*

All these machines have one HALT *instruction labeled by l_h, one instruction of the type* ADD *labeled by l_0, and*

- *U_1 has $8 + 11 + 13 = 32$ instructions of the types* ADD, SUB, *and* CHECK, *respectively,*
- *U_2 has $9 + 13 = 22$ instructions of the types* ADD, *and* CHECKSUB, *respectively,*
- *U_3 has $8 + 1 + 12 = 21$ instructions of the types* ADD, CHECK, *and* CHECKSUB, *respectively.*

Moreover, these machines either halt using the HALT *instruction and having the result of the computation in the first register, or their computation goes on infinitely.*

3 The universality of P colonies with a bounded number of cells

The proofs of the following results are based on techniques from [2] which are combined with Theorem 1.

3.1 Two object P colonies with checking rules

First we consider restricted two object P colonies with checking rules.

Theorem 2.

$$IPCol\_Ch\_2R(23, 5) = IPCol\_Ch\_2R(22, 6) = \mathbb{N}RE.$$

Proof. Let $L \in \mathbb{N}RE$ and let M be a register machine with $L = N(M)$. Consider the universal register machines from Theorem 1. By placing the code of M, the value $g(M)$, in the second register, they compute $N(M)$, producing the result in their first register. We show how to construct P colonies which simulate the computation of U_2 and U_3.

Consider a P colony $(V, e, a_1, I_E, C_1, \ldots, C_n)$ where V contains the special object e, two symbols l_i and l'_i for each instruction label l_i of the universal machine, and one symbol a_j, $1 \leq j \leq 8$, for each register. The number of a_j symbols in the environment corresponds to the value of register j. Thus, the initial contents of the environment, I_E is $g(M)$ copies of the object a_2 plus the label of the initial instruction l_0. And so, the result of the computation can be read as the number of a_1 objects corresponding to the value of the first register in a halting configuration.

The P colony we are going to construct contains one cell for each instruction.

An instruction $l_i : (\texttt{ADD}(r), l_{i,1}, l_{i,2})$ is simulated by increasing the number of objects corresponding to the value of register r by one and by changing the instruction label l_i present in the environment to $l_{i,1}$ or $l_{i,2}$. This can be done with five programs as follows.

$$P_i = \{\langle e \to a_r; e \leftrightarrow l_i\rangle,\ \langle l_i \to l_{i,1}; a_r \leftrightarrow e\rangle,\ \langle e \to e; l_{i,1} \leftrightarrow e\rangle,$$
$$\langle l_i \to l_{i,2}; a_r \leftrightarrow e\rangle,\ \langle e \to e; l_{i,2} \leftrightarrow e\rangle\}.$$

An instruction $l_i : (\texttt{CHEKSUB}(r), l_{i,1}, l_{i,2})$ is simulated with five programs by exchanging the label l_i to $l_{i,1}$ and decrease the number of a_r objects by one, or if the number of a_r objects is zero, then exchange l_i to $l_{i,2}$.

$$P_i = \{\langle e \to e; e \leftrightarrow l_i\rangle,\ \langle l_i \to l_{i,1}; e \leftrightarrow a_r / e \leftrightarrow e\rangle,\ \langle a_r \to e; l_{i,1} \leftrightarrow e\rangle,$$
$$\langle l_{i,1} \to l_{i,2}; e \leftrightarrow e\rangle,\ \langle e \to e; l_{i,2} \leftrightarrow e\rangle\}.$$

An instruction of type $l_i : (\texttt{CHECK}(r), l_{i,1}, l_{i,2})$ is simulated by six programs as follows.

$$P_i = \{\langle e \to e; e \leftrightarrow l_i\rangle,\ \langle l_i \to l'_{i,1}; e \leftrightarrow a_r / e \leftrightarrow e\rangle,\ \langle l'_{i,1} \to l_{i,1}; a_r \leftrightarrow e\rangle,$$
$$\langle e \to e; l_{i,1} \leftrightarrow e\rangle,\ \langle l'_{i,1} \to l_{i,2}; e \leftrightarrow e\rangle,\ \langle e \to e; l_{i,2} \leftrightarrow e\rangle\}.$$

There are no programs for the halting label l_h, thus, its appearance ends the computation which otherwise never stops.

For the simulation of U_2, consider $\Pi_2 = (V, e, a_1, I_E, C_0, \ldots, C_{22})$ with V and I_E as above, and one cell with the programs P_0 for the initial `ADD` instruction labeled with l_0 which fills the input register, nine cells with P_i, $1 \leq i \leq 9$, for the simulation of the rest of the `ADD` instructions, and thirteen cells with P_i, $10 \leq i \leq 22$, for simulating the `CHECKSUB` instructions. This gives us $1 + 9 + 13 = 23$ cells with at most 5 programs.

For the simulation of U_3, consider $\Pi_3 = (V, e, a_1, I_E, C_0, \ldots, C_{21})$ with V, I_E and P_0 as above, eight cells with P_i, $1 \leq i \leq 8$, for the simulation of the `ADD` instructions, one cell with P_9 for the `CHECK` instruction, and twelve cells

with P_i, $10 \leq i \leq 21$, for simulating the CHECKSUB instructions. This gives us $1 + 8 + 1 + 12 = 22$ cells, this time with at most 6 programs.

Thus, we have shown that U_2 can be simulated with 23 cells having at most 5 programs, and that U_3 can be simulated with 22 cells having at most 6 programs. This proves our statement. □

The number of programs in one cell can be decreased if we use arbitrary, not necessarily restricted programs.

Theorem 3.

$$IPCol\_Ch\_2(22, 5) = \mathbb{N}RE.$$

Proof. Consider U_3 from Theorem 1 and $\Pi_3 = (V, e, a_1, I_E, C_0, \ldots, C_{21})$ from the proof of Theorem 2 where cell C_9 corresponds to the instruction l_9 : (CHECK$(r), l_{9,1}, l_{9,2}$) which is simulated in P_9 with six restricted programs as described above. If we allow non-restricted programs, we can replace P_9 with

$$P'_9 = \{\langle e \to e; e \leftrightarrow l_9\rangle,\ \langle l_9 \to l_{9,1}; e \leftrightarrow a_r / e \to l_{9,2}\rangle,\ \langle l_{9,1} \leftrightarrow e; a_r \leftrightarrow e\rangle,$$
$$\langle l_{9,1} \to e; l_{9,2} \leftrightarrow e\rangle\}$$

which achieve the same effect as P_9 with four programs. Thus, we can simulate U_3 with $\Pi'_3 = (V, e, a_1, I_E, C_0, \ldots, C_8, C'_9, C_{10}, \ldots, C_{21})$, $C'_9 = (O_9, P'_9)$, which gives us 22 cells with at most 5 programs. As 5 programs are necessary for the simulation of the ADD and the CHECKSUB instructions, at most 5 programs are needed for each P_i. □

Now we show that by increasing the number of programs, one cell is sufficient to generate any set in $\mathbb{N}RE$, even with restricted programs.

Theorem 4.

$$IPCol\_Ch\_2R(1, 142) = \mathbb{N}RE.$$

Proof. Similarly to the proof of Theorem 2, we show how to construct a P colony Π which simulates the computation of the universal register machine U_3 from Theorem 1.

Let the nine ADD instructions be labeled by $l_0, \ldots, l_8$, the one CHECK instruction be labeled by l_9, and the twelve CHECKSUB instructions be labeled by $l_{10}, \ldots, l_{21}$.

Let $\Pi = (V, e, a_1, I_E, C)$ where V contains the special objects e, t; the symbol l_i for each instruction of U_3, that is, for $0 \leq i \leq 21$; the symbols l'_i for the ADD instructions, that is, for $0 \leq i \leq 8$; the symbols l''_i for the CHECK and CHECKSUB instructions, that is, for $9 \leq i \leq 21$; and one symbol a_j for each register j, $1 \leq j \leq 8$, of U_3. The initial contents of the environment, I_E is a number of copies of the object a_2 for initializing the contents of the second register, plus one copy of the symbols l'_i for $0 \leq i \leq 8$. The result of a computation can be read in a halting configuration as the number of a_1 objects in the environment.

The computation starts by producing an arbitrary number of copies of the double primed instruction labels $l_i'',\ 9 \leq i \leq 21$, and the introduction of a copy of the initial instruction label l_0. This is achieved with the following programs.

$$P_{ini} = \{\langle e \to l_9''; e \leftrightarrow e\rangle,\ \langle e \to l_{21}''; l_{21}'' \leftrightarrow e\rangle,\ \langle e \to l_0; l_{21}'' \leftrightarrow e\rangle\} \cup \{\langle e \to l_i''; l_i'' \leftrightarrow e\rangle,\ \langle e \to l_{i+1}''; l_i'' \leftrightarrow e\rangle, \mid 9 \leq i \leq 20\}.$$

Now, for each instruction $l_i : (\texttt{ADD}(r), l_{i,1}, l_{i,2}),\ 0 \leq i \leq 8$, we have the following programs

$$P_i = \{\langle l_i \to l_i'; e \leftrightarrow e\rangle,\ \langle e \to a_r; l_i' \leftrightarrow l_i'\rangle,\ \langle l_i' \to l_{i,1}; a_r \leftrightarrow e\rangle, \langle l_i' \to l_{i,2}; a_r \leftrightarrow e\rangle\}.$$

For the instructions $l_i : (\texttt{CHECKSUB}(r), l_{i,1}, l_{i,2}),\ 10 \leq i \leq 21$, we have the programs

$$P_i = \{\langle l_i \to l_i'; e \leftrightarrow a_r / e \leftrightarrow e\rangle, \langle a_r \to e; l_i' \leftrightarrow l_i''\rangle, \langle l_i'' \to l_{i,1}; e \leftrightarrow e\rangle, \langle l_i' \to l_{i,2}; e \leftrightarrow e\rangle\}.$$

For the instruction $l_9 : (\texttt{CHECK}(r), l_{9,1}, l_{9,2})$ we also add these four programs, but with the second and the third one modified to avoid decreasing the value stored in the register. Thus,

$$P_9 = \{\langle l_9 \to l_9'; e \leftrightarrow a_r / e \leftrightarrow e\rangle, \langle a_r \to a_r; l_9' \leftrightarrow l_9''\rangle, \langle l_9'' \to l_{9,1}; a_r \leftrightarrow e\rangle, \langle l_9' \to l_{9,2}; e \leftrightarrow e\rangle\}.$$

Moreover, in order to ensure that the cell exchanges primed objects from inside with double primed objects from the environment, we also consider the programs

$$P_{trap} = \{\langle l_i' \to t; e \leftrightarrow e\rangle,\ \langle l_i' \to t; a_r \leftrightarrow e\rangle,\ \langle t \to t; e \leftrightarrow e\rangle \mid 9 \leq i \leq 21\}$$

where t is a trap-object. If an exchange cannot be realized because the number of double primed symbols produced in the initial phase is insufficient, then the trap-object is introduced and the program $\langle t \to t; e \leftrightarrow e\rangle$ works forever, preventing the halting of the computation.

Considering the P colony above with $C = (O, P)$ with $P = \bigcup_{i=0}^{21} P_i \cup P_{ini} \cup P_{trap}$, we need 27 programs for the initial phase, 36 programs for the ADD instructions, 48 programs for the CHECKSUB instructions, 4 programs for the CHECK instruction, and 27 programs in P_{trap}. This gives us $27 + 36 + 4 + 48 + 27 = 142$, thus our statement is proved. □

3.2 P colonies without checking rules

Now we show how the use of checking rules can be avoided. First we consider the case of two objects P colonies with restricted and with unrestricted programs.

Theorem 5.

$$IPCol\_2(35, 8) = IPCol\_2R(57, 8) = \mathbb{N}RE.$$

Proof. We show how the universal register machines U_1 and U_3 from Theorem 1 can be simulated. Let U_3 have nine `ADD` instructions labeled by $l_0, \ldots, l_8$, one `CHECK` instruction labeled by l_9, and twelve `CHECKSUB` instructions labeled by $l_{10}, \ldots, l_{21}$, as in the previous theorem.

Consider $\Pi_3 = (V, e, a_1, I_E, C_0, \ldots, C_{21})$ from the proof of Theorem 2 and let $\Pi_3' = (V', e, a_1, I_E, C_0', \ldots C_{34}')$ where $V' = V \cup \{l_i', l_i'' \mid 0 \leq i \leq 21\}$. Now we construct the cells $C_i',\ 0 \leq i \leq 34$, which achieve the same effect as the 21 cells of Π_3. Note that the `ADD` instructions are simulated in Π_3 by $P_i,\ 0 \leq i \leq 8$, without checking rules, thus we can take

$$C_i' = C_i,\ 0 \leq i \leq 8,$$

without any change at all.

For the instruction $l_9 : (\texttt{CHECK}(r), l_{9,1}, l_{9,2})$ we need two cells in Π_3', having at most eight restricted programs as follows.

$$\begin{aligned} P_9' = \{&\langle e \to l_{9,2}'; e \leftrightarrow l_9\rangle,\ \langle l_9 \to l_9'; l_{9,2}' \leftrightarrow e\rangle,\ \langle l_9' \to l_9''; e \leftrightarrow a_r\rangle,\\ &\langle l_9'' \to l_{9,1}; a_r \leftrightarrow e\rangle,\ \langle l_{9,1} \to l_{9,1}; e \leftrightarrow l_{9,2}''\rangle,\ \langle l_9' \to l_{9,2}; e \leftrightarrow l_{9,2}''\rangle,\\ &\langle l_{9,2}'' \to e; l_{9,1} \leftrightarrow e\rangle,\ \langle l_{9,2}'' \to e; l_{9,2} \leftrightarrow e\rangle\}, \end{aligned}$$

and

$$P_{22}' = \{\langle e \to l_{9,2}''; e \leftrightarrow l_{9,2}'\rangle,\ \langle l_{9,2}' \to e; l_{9,2}'' \leftrightarrow e\rangle\}.$$

The interplay of these two cells produces the desired "checking" effect, they exchange the symbol l_9 to $l_{9,1}$ if there is at least one a_r symbol in the environment, otherwise $l_{9,2}$ is released.

For the instructions $l_i : (\texttt{CHECKSUB}(r), l_{i,1}, l_{i,2}),\ 10 \leq i \leq 21$, we also need two cells with at most eight programs as follows. Let for each $i,\ 10 \leq i \leq 21$,

$$\begin{aligned} P_i' = \{&\langle e \to l_{i,2}'; e \leftrightarrow l_i\rangle,\ \langle l_i \to l_i'; l_{i,2}' \leftrightarrow e\rangle,\ \langle l_i' \to l_i''; e \leftrightarrow a_r\rangle,\\ &\langle a_r \to e; l_i'' \to l_{i,1}\rangle,\ \langle l_{i,1} \to l_{i,1}; e \leftrightarrow l_{i,2}''\rangle,\ \langle l_i' \to l_{i,2}; e \leftrightarrow l_{i,2}''\rangle,\\ &\langle l_{i,2}'' \to e; l_{i,1} \leftrightarrow e\rangle,\ \langle l_{i,2}'' \to e; l_{i,2} \leftrightarrow e\rangle\}, \end{aligned}$$

and

$$P_{13+i}' = \{\langle e \to l_{i,2}''; e \leftrightarrow l_{i,2}'\rangle,\ \langle l_{i,2}' \to e; l_{i,2}'' \leftrightarrow e\rangle\}.$$

These pairs work together very similarly to the ones above. However, they not only check, but if possible, also decrease the number of a_r symbols in the environment.

Thus, if we have

$$C_i' = (ee, P_i'),\ 9 \leq i \leq 34,$$

in Π_3', then we can simulate U_3 with 35 cells, each having at most eight non-restricted programs, but no checking rules.

For the proof of the statement concerning 2 object P colonies with restricted rules, we simulate the universal machine U_1 from Theorem 1. U_1 has nine ADD instructions labeled by $l_0, \ldots, l_8$, thirteen CHECK instructions labeled by $l_9, \ldots, l_{21}$, and eleven SUB instructions labeled by $l_{22}, \ldots, l_{32}$.

Let $\Pi''_3 = (V'', e, a_1, I_E, C''_0, \ldots C''_{56})$ with $V'' = \{e, l_i, l'_i, l''_i, a_j \mid 0 \leq i \leq 32,\ 1 \leq j \leq 8\}$, and let

$$C''_i = C'_i,\ 0 \leq i \leq 8, \text{ where } C'_i \text{ is as above.}$$

For each instruction $l_i : (\texttt{CHECK}(r), l_{i,1}, l_{i,2}),\ 9 \leq i \leq 21$, we need two cells C''_i and C''_{24+i}, as described above for P'_9 and P'_{22}.

For each instruction $l_i : (\texttt{SUB}(r), l_{i,1}),\ 22 \leq i \leq 32$, we need two cells in the P colony, having at most seven restricted programs as follows. For $i,\ 22 \leq i \leq 32$, let

$$\begin{aligned} P_i = \{&\langle e \to l'_{i,1}; e \leftrightarrow l_i\rangle,\ \langle l_i \to l'_i; l'_{i,1} \leftrightarrow e\rangle,\ \langle l'_i \to l''_i; e \leftrightarrow a_r\rangle,\\ &\langle l''_i \to l_{i,1}; a_r \leftrightarrow e\rangle,\ \langle l_{i,1} \to l_{i,1}; e \leftrightarrow l''_{i,1}\rangle,\ \langle l'_i \to l_{i,1}; e \leftrightarrow l''_{i,1}\rangle,\\ &\langle l''_{i,1} \to e; l_{i,1} \leftrightarrow e\rangle\}, \end{aligned}$$

and

$$P_{24+i} = \{\langle e \to l''_{i,1}; e \leftrightarrow l'_{i,1}\rangle,\ \langle l'_{i,1} \to e; l''_{i,1} \leftrightarrow e\rangle\}.$$

The interplay of these pairs results in the decrease of the number of a_r symbols in all cases when this number is non-zero.

Now, if we have

$$C''_i = (ee, P''_i),\ 9 \leq i \leq 56,$$

in Π''_3, then we can simulate U_1 with 57 cells, each having at most seven restricted program, without checking rules. □

The number of programs in the cells can be decreased if, instead of two, we allow three objects in each cell, and thus, three instructions in each program.

Theorem 6.

$$IPCol\_3(35, 7) = \mathbb{N}RE.$$

Proof. Consider $\Pi'_3 = (V', e, a_1, I_E, C'_0, \ldots C'_{34})$ simulating U_3 from Theorem 1 as described in the first part of the proof of the previous theorem. Now define $C''_i = (eee, P''_i),\ 0 \leq i \leq 35$, where for $i,\ 0 \leq i \leq 8,\ P''_i$ is obtained from P'_i by adding a rule $e \to e$ to each program.

For $l_9 : (\texttt{CHECK}(r), l_{9,1}, l_{9,2})$, we have

$$\begin{aligned} P''_9 = \{&\langle e \to l'_{9,2}; e \to l'_{9,1}; e \leftrightarrow l_9\rangle,\ \langle l_9 \to e; l'_{9,2} \leftrightarrow e; l'_{9,1} \to l'_{9,1}\rangle,\\ &\langle l'_{9,1} \to l''_{9,1}; e \leftrightarrow a_r; a_r \leftrightarrow e\rangle,\ \langle e \to e; l''_{9,1} \to l_{9,1}; e \leftrightarrow l''_{9,2}\rangle,\\ &\langle l'_{9,1} \to e; e \leftrightarrow l''_{9,2}; e \to l_{9,2}\rangle,\ \langle l''_{9,2} \to e; l_{9,1} \leftrightarrow e, e \to e\rangle,\\ &\langle l''_{9,2} \to e; l_{9,2} \leftrightarrow e; e \to e\rangle\}, \end{aligned}$$

and

$$P''_{22} = \{\langle e \to l''_{9,2}; e \leftrightarrow l'_{9,2}; e \to e\rangle,\ \langle l'_{9,2} \to e; l''_{9,2} \leftrightarrow e; e \to e\rangle\},$$

and for the rules $l_i : (\texttt{CHECKSUB}(r), l_{i,1}, l_{i,2}),\ 10 \leq i \leq 21$, let

$$\begin{aligned} P''_i = \{&\langle e \to l'_{i,2}; e \to l'_{i,1}; e \leftrightarrow l_i\rangle,\ \langle l_i \to e; l'_{i,2} \leftrightarrow e; l'_{i,1} \to l'_{i,1}\rangle,\\ &\langle l'_{i,1} \to l''_{i,1}; e \leftrightarrow a_r; e \to e\rangle,\ \langle a_r \to e; l''_{i,1} \to l_{i,1}; e \leftrightarrow l''_{i,2}\rangle,\\ &\langle l'_{i,1} \to e; e \leftrightarrow l''_{i,2}; e \to l_{i,2}\rangle,\ \langle l''_{i,2} \to e; l_{i,1} \leftrightarrow e, e \to e\rangle,\\ &\langle l''_{i,2} \to e; l_{i,2} \leftrightarrow e; e \to e\rangle\}, \end{aligned}$$

and

$$P''_{13+i} = \{\langle e \to l''_{i,2}; e \leftrightarrow l'_{i,2}; e \to e\rangle,\ \langle l'_{i,2} \to e; l''_{i,2} \leftrightarrow e; e \to e\rangle\}.$$

These cells have the same checking effect with seven programs instead of eight. Thus, if we consider the P colony $\Pi''_3 = (V', e, a_1, I_E, C''_0, \ldots C''_{34})$, then our statement is proved. □

4 Conclusion

We have shown that P colonies are able to simulate universal register machines, provided they are initialized as follows: besides the environmental object, a finite number of objects are placed in the environment. Thus, P colonies are able to generate any recursively enumerable set of nonnegative integers with a bounded number of cells, each containing a bounded number of programs of a bounded length.

We have considered different types of universal machines and different types of P colonies, but still, our results are only a first approximation of the number of necessary cells and programs. To decrease the present bounds, or to prove that they are sharp ones, remains the topic of further research.

References

1. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, Gh.: Grammar Systems – A Grammatical Approach to Distribution and Cooperation. Gordon and Breach, London, 1994
2. Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., Vaszil, Gy.: Computing with cells in environment: P colonies. Journal of Multiple Valued Logic and Soft Computing (to appear)
3. Kelemen, J., Kelemenová, A.: A grammar-theoretic treatment of multi-agent systems. Cybernetics and Systems **23** (1992) 621–633
4. Kelemen, J., Kelemenová, A., Păun, Gh.: Preview of P colonies: A biochemically inspired computing model. In: Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX) (M. Bedau et al., eds.) Boston Mass. (2004) 82–86
5. Korec, I.: Small universal register machines. Theoretical Computer Science **168** (1996) 267-301
6. Minsky, M.: Computation – Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ, 1967
7. Păun, Gh.: Membrane Computing – An Introduction. Springer-Verlag, Berlin, 2002