ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **16**, Number 1, 2013, 3–9

A Catalytic P System with Two Catalysts Generating a Non-Semilinear Set

Petr SOSÍK

Research Institute of the IT4Innovations Centre of Excellence,
Faculty of Philosophy and Science, Silesian University in Opava
74601 Opava, Czech Republic
and
Departamento de Inteligencia Artificial, Facultad de Informática,
Universidad Politécnica de Madrid, Campus de Montegancedo s/n,
Boadilla del Monte, 28660 Madrid, Spain
E-mail: `petr.sosik@fpf.slu.cz`

Abstract. Membrane computing is a relatively young but fast emerging bio-inspired computing paradigm, nowadays with many branches and applications. Its original computing model is the catalytic P system. Although it was proven already in 2005 that catalytic P systems with two catalysts are computationally universal [2], no simple example of such a P system generating a non-semilinear set was known. The present paper fills this gap and provides such an example with 54 rules. It is expected, however, that this number of rules can be reduced and the minimal number of rules to generate a non-semilinear set in a catalytic P system with two catalysts remains open.

Key words: Catalytic P System, Membrane Computing, Non-Semilinear Set.

1. Introduction

Membrane computing belongs to the family of computational models inspired by nature, or more precisely, by the structure of living cells and by the way the biochemical substances are processed within and moved between membrane-bounded regions. Since the first model of membrane computing was introduced in [4], many different models (called P systems) belonging to the membrane computing family arose. P systems principles include, *e.g.*, molecular synthesis within cells, selective particle recognition by membranes, controlled transport through protein channels, membrane division, membrane dissolution and many others. Their common feature is that they consist of membrane-bounded areas (called regions or cells) often hierarchically arranged. Multisets of objects serving as information medium are placed inside each cell. The processes are modeled in P systems by means of operations on multisets in separate cells.

Catalytic P systems represent the original model of P systems introduced in the seminal journal paper [4]. Objects in cells of catalytic P systems can evolve due to *evolution rules*, they can also pass through membranes. All objects evolve at the same time, in parallel; in turn, all membranes are active in parallel. If the objects evolve alone, then the system is said to be non-cooperative; if there are rules which specify the evolution of several objects at the same time, then the system is cooperative; an intermediate case is that where certain objects — *catalysts*, appear together with other objects in evolution rules and they are not modified by the use of the rules.

It was shown already in [6] that P systems with non-cooperative and catalytic rules are computationally universal. The proof used a construction with 8 catalysts. Later in [2] the minimal number of necessary catalysts to reach computational universality was improved to 2. It remains still an open problem whether the computational completeness can be reached with one catalysts only. The most recent discussion and results related to this topic can be found in [1]. Although the result [2] was published eight years ago, no simple example of a P system with two catalysts generating a non-semilinear set was known until now. In this note we present a P system with one membrane, two catalysts and 54 rules and we prove that the generated set is non-semilinear. It is supposed that the number of 54 rules is not minimal and could be possibly diminished.

2. Definitions

For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over V. The *empty string* is denoted by λ. A *multiset* M with underlying set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $M = (A, f)$ is a multiset then its *support* is defined as $supp(M) = \{x \in A \,|\, f(x) > 0\}$. A multiset is empty (resp. finite) if its support is the empty set (resp. a finite set). If $M = (A, f)$ is a finite multiset over A, and $supp(M) = \{a_1, \dots, a_k\}$ then it can also be represented by the string $a_1^{f(a_1)} \dots a_k^{f(a_k)}$ over the alphabet $\{a_1, \dots, a_k\}$. Nevertheless, all permutations of this string precisely identify the same multiset M. The next definition cites Def. 4.1 in Chapter 4 of [5].

Definition 1. An *extended catalytic P system* of degree $m \geq 1$ is a construct

$$\Pi = (O, C, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_0) \text{ where:}$$

1. O is the alphabet of objects;
2. $C \subseteq O$ is the alphabet of catalysts;
3. μ is a membrane structure of degree m with membranes labeled in a one-to-one manner with the natural numbers $1, 2, \dots, m$;

4. $w_1, \ldots, w_m \in O^*$ are the multisets of objects initially present in the m regions of μ;
5. R_i, $1 \leq i \leq m$, are finite sets of *evolution rules* over O associated with the regions $1, 2, \ldots, m$ of μ; these evolution rules are of the forms $ca \rightarrow cv$ or $a \rightarrow v$, where c is a catalyst, a is an object from $O \setminus C$, and v is a string from $((O \setminus C) \times \{here, out, in\})^*$;
6. $i_0 \in \{0, 1, \ldots, m\}$ indicates the output region of Π.

The membrane structure and the multisets in Π constitute the *initial configuration* of the system. A transition between configurations is governed by the application of the evolution rules, which is done in parallel: all objects, from all membranes, which can be the subject of local evolution rules, have to evolve simultaneously.

The application of a rule $u \rightarrow v$ in a region containing a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. The objects can be eventually transported through membranes due to targets *in* and *out*. We refer to [5] for further details and examples.

The system continues parallel steps until there remain no applicable rules in any region of Π. Then the system halts. We consider the number of objects from $O \setminus C$ contained in the output membrane i_0 at the moment when the system halts as the *result* of the underlying computation of Π. (The system is called *extended* since the catalytic objects in C are not counted to the result of computation.) The set of results of all computations possible in Π is called the set *generated by* Π and it is denoted by $N(\Pi)$.

A *register machine* (also called counter machine) is the computing model introduced by Minsky [3]. The machine uses a fixed number of registers for storing arbitrarily large nonnegative integers. Computation is performed by a program consisting of numbered instructions of several simple types. The basic instruction types we use here are:

$(\mathrm{ADD}(r), i, j)$ – add 1 to the contents of the register r and choose nondeterministically to continue with the instruction labeled i or j;

$(\mathrm{SUB}(r), i, j)$ – if the contents of the register r is nonzero, then subtract 1 from it and continue with instruction i, else continue with instruction j;

HALT – halt the machine.

A deterministic variant of the machine with $i = j$ for all instructions of type $(\mathrm{ADD}(r), i, j)$ can be used to compute any partial recursive function $f : \mathbb{N}^n \longrightarrow \mathbb{N}^m$. Starting with $(x_1, \ldots, x_n)$ in certain register(s), the machine M computes $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ if it reaches the final instruction HALT and $(y_1, \ldots, y_m)$ are contained in specified register(s). If the instruction HALT is never reached, then the output remains undefined.

In this paper we use the non-deterministic variant defined above. It starts with all registers empty. The set of all possible output m-tuples $(y_1, \ldots, y_m)$ contained in pre-defined registers when the machine halts is called the set *generated* by the machine.

3. Main result

In this paper we use extended catalytic P systems from Definition 1, where the terminal set of output objects is formed by all non-catalytic objects. No cooperative rules, bistable catalysts, priorities or similar enhancements are considered.

Theorem 1. *Extended catalytic P systems with a single membrane, two catalysts and 54 rules can generate non-semilinear sets of numbers.*

Proof. We construct an example of a catalytic P systems with two catalysts and 54 rules generating the set $\{2^n - 2n \mid n \geq 2\}$. This set is generated by a non-deterministic register machine with three registers, starting with all registers empty, running the following program which stores the result in register 3:

```
1: (ADD(1), 2, 8)
2: (SUB(1), 3, 5)
3: (ADD(2), 4, 4)
4: (ADD(2), 2, 2)
5: (SUB(2), 6, 1)
6: (ADD(1), 7, 7)
7: (ADD(3), 5, 5)
8: HALT
```

The register machine program described above works as follows: The first ADD instruction increments register 1 and then it decides non-deterministically whether the computation continues or whether it halts. In the former case the content of register 1 is emptied and duplicated to register 2 (instructions 2,3,4) and then the content of reg. 2 is copied to reg. 1 and added to reg. 3 (instructions 5,6,7). The whole cycle is repeated until a jump to instruction HALT is nondeterministically chosen in instruction 1.

We construct a catalytic P system Π following precisely the proof of Theorem 4.1 in Chapter 4 of [5]. The resulting catalytic P system Π simulates the non-deterministic register machine described above and it generates a representation of contents of its registers by the corresponding number of objects o_1, o_2 and o_3, respectively.

$$\begin{aligned}
\Pi &= (O, \{c_1, c_2\}, [_1\ ]_1, w, R, 1), \\
O &= \{\#\} \cup \{c_1, c_1', c_1'', c_2, c_2', c_2''\} \cup \{o_1, o_2, o_3\} \\
&\cup \{p_j, \tilde{p}_j, p_j', p_j'', \bar{p}_j, \bar{p}_j', \bar{p}_j'', \hat{p}_j, \hat{p}_j', \hat{p}_j'' \mid j = 2, 5\} \\
&\cup \{p_j, \tilde{p}_j \mid j = 1, 3, 4, 6, 7, 8\}, \\
R &= \{x \to \# \mid x \in \{p_j, \tilde{p}_j, p_j', p_j'', \bar{p}_j, \bar{p}_j'', \hat{p}_j, \hat{p}_j'' \mid j = 2, 5\}\} \\
&\cup \{x \to \# \mid x \in \{c_1', c_1'', c_2', c_2''\}\} \cup \{\# \to \#\}
\end{aligned}$$

$$\begin{array}{rcl}
 & \cup & \{c_1p_8 \to c_1,\ c_2\tilde{p}_8 \to c_2\} \\
 & \cup & \{c_1\tilde{p}_j \to c_1 \mid j = 1,3,4,6,7\} \\
 & \cup & \{c_2p_1 \to c_2p_2\tilde{p}_2o_1,\ c_2p_1 \to c_2p_8\tilde{p}_8o_1, \\
 & & c_2p_3 \to c_2p_4\tilde{p}_4o_2,\ c_2p_4 \to c_2p_2\tilde{p}_2o_2,\ c_2p_6 \to c_2p_7\tilde{p}_7o_1,\ c_2p_7 \to c_2p_5\tilde{p}_5o_3\} \\
 & \cup & \{c_rp_j \to c_r\hat{p}_j\hat{p}'_j,\ c_rp_j \to c_r\bar{p}_j\bar{p}'_j\bar{p}''_j, c_ro_r \to c_rc'_r,\ c_rc'_r \to c_rc''_r, \\
 & & c_{3-r}c''_r \to c_{3-r}, c_r\hat{p}'_j \to c_r\#,\ c_{3-r}\hat{p}'_j \to c_{3-r}\hat{p}''_j, c_r\hat{p}''_j \to c_rp_k\tilde{p}_k, \\
 & & c_r\bar{p}_j \to c_r,\ c_{3-r}\bar{p}''_j \to c_{3-r}p''_j,\ c_{3-r}p''_j \to c_{3-r}p'_j, \\
 & & c_rp'_j \to c_rp_l\tilde{p}_l \mid (j,r,k,l) = (2,1,3,5),(5,2,6,1)\} \\
 & \cup & \{c_2y \to c_2 \mid y \in \{\tilde{p}_2, \hat{p}_2, \bar{p}'_2\}\} \\
 & \cup & \{c_1y \to c_1 \mid y \in \{\tilde{p}_5, \hat{p}_5, \bar{p}'_5\}\}, \\
w & = & c_1c_2p_1\tilde{p}_1.
\end{array}$$

The above construction is correct by Theorem 4.1 in [5]. The total number of rules is 64. Notice that the P system Π automatically resets registers 1 and 2 when the instruction HALT is reached, removing all objects o_1 and o_2 by the rules $c_ro_r \to c_rc'_r$, $c_rc'_r \to c_rc''_r$ and $c_{3-r}c''_r \to c_{3-r}$, for $r = 1, 2$. Hence there must be another register 3 to collect the result of computation.

Observe now the construction provided in [5] is general for an arbitrary register machine program and it can be simplified in cases of specific sequences of instructions. Notice the following facts whose detailed justification can be found in [5]:

- the configuration of the P system Π at the beginning of simulation of any instruction labeled i is $c_1c_2p_i\tilde{p}_io_1^jo_2^ko_3^l$, where $j,k,l \geq 0$ are contents of the registers;
- simulations of subsequent instructions do not overlap in time;
- objects $p_i, \tilde{p}_i$ identifying the instruction i to be simulated are only produced in the last step of the previous instruction simulation.

Therefore, any instruction of the form i: $(\mathrm{ADD}(r), j, j)$ can be simulated in such a way that:

1. the last rule simulating the previously executed instruction produces the multiset $o_rp_j\tilde{p}_j$ instead of $p_i\tilde{p}_i$;
2. the rules originally simulating i: $(\mathrm{ADD}(r), j, j)$ are omitted.

Similarly, observe that the instruction HALT in the above program is simulated by the rules $c_1p_8 \to c_1$, $c_2\tilde{p}_8 \to c_2$, and then the above mentioned rules which eliminate all objects o_1 and o_2 follow until the system Π halts. If the last rule of the previously executed instruction 1: $(\mathrm{ADD}(1), 2, 8)$ just does not produce objects $p_8, \tilde{p}_8$, then the two rules simulating HALT can be omitted, replacing the computation

$$c_1c_2p_1\tilde{p}_1o_1^jo_2^ko_3^l \Rightarrow c_1c_2p_8\tilde{p}_8o_1^{j+1}o_2^ko_3^l \Rightarrow c_1c_2o_1^{j+1}o_2^ko_3^l \Rightarrow \dots$$

by

$$c_1c_2p_1\tilde{p}_1o_1^jo_2^ko_3^l \Rightarrow c_1c_2o_1^{j+1}o_2^ko_3^l \Rightarrow \dots$$

Any other application of rules of Π would lead to an introduction of the trap symbol #. Therefore, in the case of our program we can:

1. (a) remove the rules $c_1\tilde{p}_3 \to c_1$, $c_1\tilde{p}_4 \to c_1$, $c_2p_3 \to c_2p_4\tilde{p}_4o_2$, $c_2p_4 \to c_2p_2\tilde{p}_2o_2$, implementing instructions 3: (ADD(2), 4, 4), 4: (ADD(2), 2, 2), and

   (b) modify the rule $c_1\hat{p}_2'' \to c_1p_3\tilde{p}_3$ which is a part of implementation of 2: (SUB(1), 3, 5) to the form $c_1\hat{p}_2'' \to c_1p_2\tilde{p}_2o_2o_2$;

2. (a) remove the rules $c_1\tilde{p}_6 \to c_1$, $c_2p_6 \to c_2p_7\tilde{p}_7o_1$, $c_1\tilde{p}_7 \to c_1$, $c_2p_7 \to c_2p_5\tilde{p}_5o_3$, implementing instructions 6: (ADD(1), 7, 7), 7: (ADD(3), 5, 5), and

   (b) modify the rule $c_2\hat{p}_5'' \to c_2p_6\tilde{p}_6$ which is a part of implementation of 2: (SUB(2), 6, 1) to the form $c_2\hat{p}_5'' \to c_2p_5\tilde{p}_5o_1o_3$;

3. (a) remove the rules $c_1p_8 \to c_1$, $c_2\tilde{p}_8 \to c_2$, implementing 8: HALT, and

   (b) modify the rule $c_2p_1 \to c_2p_8\tilde{p}_8o_1$ which is a part of implementation of 1: (ADD(1), 2, 8) to the form $c_2p_1 \to c_2o_1$.

These modifications allow to save ten rules (and some objects), getting us to the final number of 54 rules. The resulting P systems gets the form:

$$\begin{array}{rcl}
\Pi & = & (O, \{c_1, c_2\}, [_1\ ]_1, w, R, 1), \\
O & = & \{\#\} \cup \{c_1, c_1', c_1'', c_2, c_2', c_2''\} \cup \{o_1, o_2\} \cup \{p_1, \tilde{p}_1\} \\
 & \cup & \{p_j, \tilde{p}_j, p_j', p_j'', \bar{p}_j, \bar{p}_j', \bar{p}_j'', \hat{p}_j, \hat{p}_j', \hat{p}_j'' \mid j = 2, 5\} \\
R & = & \{x \to \# \mid x \in \{p_j, \tilde{p}_j, p_j', p_j'', \bar{p}_j, \bar{p}_j'', \hat{p}_j, \hat{p}_j'' \mid j = 2, 5\}\} \\
 & \cup & \{x \to \# \mid x \in \{c_1', c_1'', c_2', c_2''\}\} \cup \{\# \to \#\} \\
 & \cup & \{c_1\tilde{p}_1 \to c_1, c_2p_1 \to c_2p_2\tilde{p}_2o_1,\ c_2p_1 \to c_2o_1\} \\
 & \cup & \{c_rp_j \to c_r\hat{p}_j\hat{p}_j',\ c_rp_j \to c_r\bar{p}_j\bar{p}_j'\bar{p}_j'', c_ro_r \to c_rc_r',\ c_rc_r' \to c_rc_r'', \\
 & & \ c_{3-r}c_r'' \to c_{3-r}, c_r\hat{p}_j' \to c_r\#,\ c_{3-r}\hat{p}_j' \to c_{3-r}\hat{p}_j'', \\
 & & \ c_r\bar{p}_j \to c_r,\ c_{3-r}\bar{p}_j'' \to c_{3-r}p_j'',\ c_{3-r}p_j'' \to c_{3-r}p_j', \\
 & & \ c_rp_j' \to c_rp_l\tilde{p}_l \mid\ (j, r, l) = (2, 1, 5), (5, 2, 1)\} \\
 & \cup & \{c_1\hat{p}_2'' \to c_1p_2\tilde{p}_2o_2o_2,\ c_2\hat{p}_5'' \to c_2p_5\tilde{p}_5o_1o_3\} \\
 & \cup & \{c_2y \to c_2 \mid y \in \{\tilde{p}_2, \hat{p}_2, \bar{p}_2'\}\} \\
 & \cup & \{c_1y \to c_1 \mid y \in \{\tilde{p}_5, \hat{p}_5, \bar{p}_5'\}\}, \\
w & = & c_1c_2p_1\tilde{p}_1.
\end{array}$$

$\square$

4. Conclusion

We have shown that an extended catalytic P systems with a single membrane, two catalysts and 54 rules can generate non-semilinear sets of numbers. However, this result is barely optimal in terms of simplicity of the P system, particularly the minimal number of necessary rules.

For example, we conjecture that a significant simplification of implementation of instructions SUB can be achieved using the fact that for each register there exists only one instruction SUB decrementing it. Another promising way is to abandon the constructions used in [2, 5] and to construct a catalytic P system generating a non-semilinear set directly "from the scratch".

Acknowledgements. I am indebted to Gheorghe Păun and Rudi Freund for their valuable comments to the final version of the paper.

This work was supported by the Ministerio de Ciencia e Innovación, Spain, under project TIN2012-36992, by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), and by the Silesian University in Opava under the Student Funding Scheme, project SGS/7/2011.

References

[1] Freund R., *Purely catalytic P systems: Two catalysts can be sufficient for computational completeness*, in A. Alhazov *et al.*, editor, *14th International Conference on Membrane Computing*, pp. 153–166, Chisinau, 2013. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova.

[2] Freund R., Kari L., Oswald M., Sosík P., *Computationally universal P systems without priorities: two catalysts are sufficient*, Theoretical Computer Science, **330**, pp. 251–266, 2005.

[3] Minsky M., *Computation – Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.

[4] Păun G., *Computing with membranes*, J. Comput. System Sci., **61**, pp. 108–143, 2000.

[5] Păun G., Rozenberg G., Salomaa A., editors, *The Oxford Handbook of Membrane Computing*, Oxford University Press, Oxford, 2010.

[6] Sosík P., Freund R., *P systems without priorities are computationally universal*, in G. Păun *et al.*, editor, *Membrane Computing. International Workshop, WMC-CdeA 2002*, volume **2597** of *Lecture Notes in Computer Science*, pp. 400–409. Springer, Berlin, 2003.