Evolution-Communication P Systems: Time-Freeness

Artiom Alhazov[1], Matteo Cavaliere[2]

[1] Institute of Mathematics and Computer Science
Academy of Science of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova

Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: artiom@math.md, artiome.alhazov@estudiants.urv.es

[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: martew@inwind.it

Summary. Membrane computing is a (biologically motivated) theoretical framework of distributed parallel computing. If symbol-objects are considered, then membrane systems (also called P systems) are distributed multiset processing systems. In evolution-communication (EC) P systems the computation is carried out with the use of non-cooperative rewriting rules and with (usually the minimally cooperative) transport rules.

The goal of this article is to improve the existing results on evolution-communication P systems. It is known that EC P systems with 2 membranes are universal, and so are time-free EC P systems with targets with 3 membranes. We prove that any recursively enumerable set of vectors of nonnegative integers can be generated by time-free EC P systems (without targets) with 2 membranes, thus improving both results.

1 Introduction

Membrane systems with symbol objects are a framework of distributed parallel multiset rewriting. One can see [10] for the comprehensive bibliography and [9] for the detailed introduction and a survey.

Evolution-communication P systems were first introduced in [4] as systems having two kinds of rules: rewriting-like (typically non-cooperative) rules without targets and (typically minimally cooperative) transport rules. The transport rules are called symport if they move objects in the same direction, and antiport if they move objects in different directions. The weight of a transport rule is the maximal number of objects it moves in either direction. The model was shown

to be universal with 3 membranes and symport/antiport rules of weight 1, and this result was improved in [1] by reducing the number of membranes to 2. A similar result was proved in [7]: universality with symport of weight at most 2 and 2 membranes.

In [2] it was shown that accepting EC P systems are universal with 3 membranes, using either symport/antiport of weight 1, or symport of weight at most 2. Yet another variant proved universal with 3 membranes is proton pumping P systems, see [3], which is a restricted variant of evolution-communication systems.

Timed and time-free P systems were first introduced in [6], the idea behind the first notion being that the reactions no longer take one step, but rather an arbitrary (defined by a mapping from the set of rules to the set of positive integers) number of steps, the rules still being applied non-deterministically in a maximally parallel manner. The time-freeness means that the result (set of numbers, vectors or words produced by all computations) should be independent of this mapping.

A generalized model of EC P systems was introduced in [7], allowing the rewriting-like rules to have targets. It was then shown in [5] that the time-free EC P systems with targets are universal with 3 membranes using either symport/antiport rules of weight 1 or symport rules of weight ≤ 2.

2 Preliminaries

First we recall from [4] the definition of evolution-communication P system, and from [6] the definition of timed and time-free P systems.

Definition 1. *An* evolution-communication P system *(in short,* an EC P system*), of degree $m \geq 1$, is defined as*

$$\Pi = (O, \mu, w_1, w_2, \cdots, w_m, R_1, \cdots, R_m, R'_1, \cdots, R'_m, i_0),$$

where:

- *O is the alphabet of objects;*
- *μ is a membrane structure with m membranes (and hence m regions) injectively labelled with $1, 2, \cdots, m$;*
- *w_i are strings which represent multisets over O associated with the regions $1, 2, \cdots, m$ of μ;*
- *R_i, $1 \leq i \leq m$, are finite sets of simple evolution rules over O; R_i is associated with the region i of μ; a simple evolution rule is of the form $u \to v$, where u and v are strings over the alphabet O;*
- *R'_i, $1 \leq i \leq m$, are finite sets of symport/antiport rules over O; R'_i is associated with the membrane i of μ;*
- *$i_0 \in \{0, 1, 2, \cdots, m\}$ is the output region; if $i_0 = 0$, then it is the environment, otherwise i_0 is the label of an elementary membrane of μ.*

Given a time-mapping

$$e : R_1 \cup R_2 \cup \cdots \cup R_m \cup R'_1 \cup R'_2 \cup \cdots \cup R'_m \longrightarrow \mathbb{N}$$

and an *EC* P system Π as defined above, it is possible to construct a *timed EC P system* $\Pi(e)$ as $(O, \mu, w_1, w_2, \ldots, w_m, R_1, \ldots, R_m, R'_1, \ldots, R'_m, i_0, e)$ working in the following way.

We suppose the existence of an external and global clock that ticks at uniform intervals of time. At each time in the regions of the system we have together rules (both evolution and transport) in execution and rules not in execution. At each time all the evolution and transport rules that can be applied (started) in each region, have to be applied. If a rule $r \in R_i, R'_i, 1 \leq i \leq m$, is applied, then all objects that can be processed by the rule have to evolve by this rule (a rule is applied in a maximally parallel manner as standard in P system area).

As usual, the rules from R_i are applied to objects in region i and the rules from R'_i govern the communication of objects through membrane i. There is no difference between evolution rules and communication rules: they are chosen and applied in the non-deterministic maximally parallel manner. When an evolution rule or a transport rule r is started at time j, its execution terminates at time $j + e(r)$. If two rules are started in the same time unit, then possible conflicts for using the occurrences of symbol-objects are solved assigning the objects in a non-deterministic way (again, in the way usually defined in P system area). Notice that when the execution of a rule r is started, the occurrences of objects used by this rule are not anymore available for other rules during the entire execution of r.

The computation stops when no rule can be applied in any region and there are no rules in execution: in this case the system has reached an *halting configuration*. The output of a halting computation is the vector of numbers representing the multiplicities of object presents in the output region in the halting configuration. (If $i_0 = 0$, then also the sequence of objects sent outside can be considered as the result; in this case, if some objects arrive into the environment simultaneously, then every permutation is considered.) Collecting all the vectors obtained, for any possible halting computation, we get the set of vectors of natural numbers generated by the system. (If we collect the sequences of objects, then we obtain a language.)

An EC P system $\Pi = (O, \mu, w_1, w_2, \ldots, w_m, R_1, \ldots, R_m, R'_1, \ldots, R'_m, i_0)$ *is* time-free *if and only if every system in the set*

$$\{\Pi(e) \mid e : R \longrightarrow \mathbb{N}\}$$

(where $R = R_1 \cup R_2 \cup \cdots \cup R_m \cup R'_1 \cup R'_2 \cup \cdots \cup R'_m$*) produces the same set of vectors of natural numbers (or the same language).*

Because there is no ambiguity, in this case the set of vectors of natural number generated by a time-free *EC* P system Π is indicated by $Ps(\Pi)$ (the corresponding language generated is denoted by $L(\Pi)$).

We use the notation $fPsECP_m(i, j)$ to denote the family of sets of vectors of natural numbers generated by *time-free EC* P systems with at most m membranes

(as usually, $m = *$ if such a number is unbounded), non-cooperative evolution rules, symport rules of weight at most i, and antiport rules of weight at most j. If languages are generated, then we replace Ps by L in the notation; when speaking of usual EC P systems (where rules are applied in one step), we omit f in the notation.

We also need to recall from [8] the definition of register machines. A non-deterministic *register machine* is a 5-tuple $M = (m, Q, q_0, q_f, P)$, where

- m is the number of registers (let us denote the jth register by c_j)
- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state,
- P is a finite set of instructions of the form
  $q_i : (c_j\gamma, q_k, q_l)$, with $q_i, q_k, q_l \in Q$, $q_i \neq q_f$, $1 \leq k \leq m$, $\gamma \in \{+, -\}$.

"Increment" instruction ($\gamma =$ "$+$"). The value of register c_j is increased by 1, and the system changes the state from q_i to q_k or q_l, non-deterministically.

"Decrement/zero test" instruction ($\gamma =$ "$-$"). If the value of register c_j is greater than zero, then this instruction decreases it by 1 and changes the state of the system from q_i to q_k. Otherwise (when the value of c_j is zero) the state of the system changes to q_l.

3 Results

We recall that every recursively enumerable set of vectors of k letters can be generated by a register machine with $k + 2$ registers without decrement and zero test instructions associated to the first k registers.

Theorem 1. $fPsECP_2(1, 1) = PsRE$.

Proof. Given a deterministic register machine $M = (k + 2, Q, q_0, q_f, P)$, where Q_+ is the set of states with increment instructions and Q_- is the set of states of decrement/zero test instructions, we construct the following time-free P system

$$
\begin{aligned}
\Pi &= (O, \mu = [_1\, [_2\ ]_2\,]_1, w_1 = q_0, w_2 = \lambda, R_1, R_2, R_1', R_2', i_0 = 0), \text{ where:}\\
O &= \{c_j \mid 1 \leq j \leq k+2\} \cup Q \cup \{p_i, r_i, s_i, t_i, u_i \mid i \in I_-\} \cup \{\#\},\\
R_1 &= \{q_i \to q_k c_j, q_i \to q_l c_j \mid q_i : (c_j+, q_k, q_l) \in P\}\\
&\quad \cup \{p_i \to q_k, s_i \to t_i, t_i \to \#, u_i \to q_l \mid q_i : (c_j-, q_k, q_l) \in P\}\\
&\quad \cup \{\# \to \#\},\\
R_2 &= \{q_i \to p_i, q_i \to \#, q_i \to r_i s_i, t_i \to u_i \mid q_i \in Q_-\}\\
&\quad \cup \{\# \to \#\},\\
R_1' &= \{(c_j, out) \mid 1 \leq j \leq k\},\\
R_2' &= \{(q_i, in), (s_i, out), (r_i, out; t_i, in), (u_i, out) \mid q_i \in Q_-\}\\
&\quad \cup \{(p_i, out; c_j, in) \mid q_i : (c_j-, q_k, q_l) \in P\}.
\end{aligned}
$$

The objects used are: those representing the values of the registers (c_j), those associated to the instructions (q_i), those associated to the decrement (p_i)/zero test (r_i, s_i, t_i, u_i) instructions, as well as the trap symbol (#). The computation starts with q_0 in the skin membrane, the instructions of M are simulated, and symbols c_j, $1 \leq j \leq k$, are ejected in the environment as the result. Addition instructions are simulated by $q_i \to q_k c_j$, $q_i \to q_l c_j$: changing the label of the current instruction and producing one more object associated to the corresponding counter.

The details of decrement/zero test instructions are shown in Figure 1. After q_i enters the elementary membrane, it "guesses" whether decrementing register j will be successful ($q_i \to p_i$) or not ($q_i \to r_i s_i$).

First case: p_i must come to region 1, removing one copy of c_j from there (otherwise the trap object will be produced and the computation will never finish). Then p_i evolves to q_k (label of the next instruction if the register had value zero).

Second case: object r_i will wait if and only if there are no objects c_j in the skin region (the register is empty). If it does, then eventually s_i will come to region 1, be rewritten to t_i, and then remove r_i from the inner region. Then t_i will change to u_i, which will come back to region 1 and be rewritten to q_l (label of the next instruction if the register was decremented). If r_i does not wait for t_i, then the latter will produce the trap object, causing an endless computation.

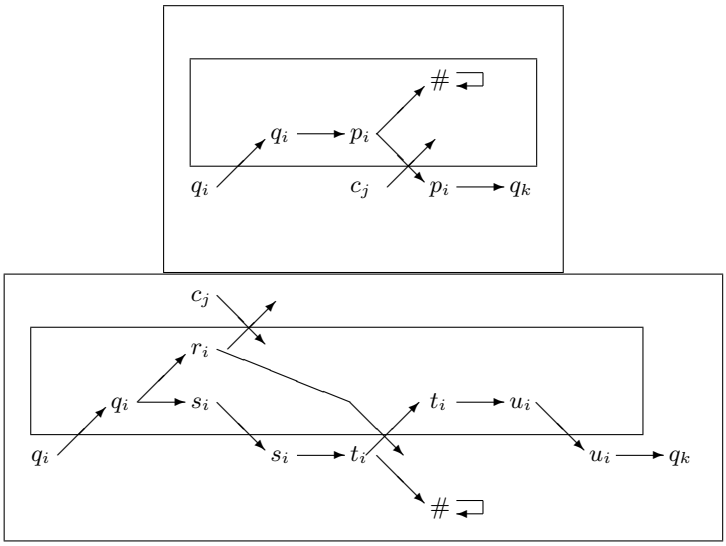

Fig. 1. Illustrating instruction $q_i : (c_j-, q_k, q_l)$.

We will now show a similar result to the one above: using rules moving two objects in the same direction (symport of weight 2) instead of using rules exchang-

ing two objects (antiport of weight 1) leads to time-free P systems with the same generative power.

Theorem 2. $fPsECP_2(2,0) = PsRE$.

Proof. Given a register machine $M = (k+2, Q, q_0, q_f, P)$, where Q_+ is the set of states with increment instructions and Q_- is the set of states with decrement/zero test instructions, we construct the following time-free P system

$$
\begin{aligned}
\Pi &= (O, \mu = [_1\, [_2\ ]_2\,]_1, w_1 = q_0, w_2 = \lambda, R_1, R_2, R'_1, R'_2, i_0 = 0), \text{ where:}\\
O &= \{c_j \mid 1 \le j \le k+2\} \cup Q \cup \{p_i, r_i, s_i, t_i \mid i \in I_-\} \cup \{\#\},\\
R_1 &= \{q_i \to q_k c_j, q_i \to q_l c_j \mid q_i : (c_j+, q_k, q_l) \in P\}\\
&\quad \cup \{q_i \to p_i, p_i \to \#, q_i \to r_i s_i, s_i \to t_i, t_i \to \# \mid q_i \in Q_-\}\\
&\quad \cup \{\# \to \#\},\\
R_2 &= \{p_i \to q_k, t_i \to q_l \mid : (c_j+, p_k, p_l)\},\\
R'_1 &= \{(c_j, out) \mid 1 \le j \le k\},\\
R'_2 &= \{(q_i, out) \mid i \in I\} \cup \{(r_i t_i, in) \mid q_i \in Q_-\}\\
&\quad \cup \{(p_i c_j, in), (r_i c_j, in) \mid q_i : (c_j-, q_k, q_l) \in P\}.
\end{aligned}
$$

Like in the previous theorem, we use the objects representing the values of the registers (c_j), those associated to the instructions (q_i), those associated to the decrement (p_i)/zero test (r_i, s_i, t_i) instructions, and the trap symbol (#). The computation starts with q_0 in the skin membrane, the instructions of M are simulated, and symbols c_j, $1 \le j \le k$, are ejected in the environment as the result. Addition instructions are simulated in one step: changing the label of the current instruction and producing one more object associated to the corresponding counter.

The details of decrement/zero test instructions are shown in Figure 2. q_i "guesses" whether decrementing register j will be successful ($q_i \to p_i$) or not ($q_i \to r_i s_i$).

First case: p_i must leave region 1, removing one copy of c_j from there (otherwise the trap object will be produced and the computation will never finish). Then p_i evolves to q_k (label of the next instruction if the register had value zero), which comes to region 1.

Second case: object r_i will wait if and only if there are no objects c_j in the skin region (the register is empty). If it does, then eventually s_i will be rewritten to t_i, which will come to region 1, and then remove r_i from the skin region. Then t_i will change to q_l (label of the next instruction if the register was decremented). If r_i does not wait for t_i, then the latter will produce the trap object, causing an endless computation.

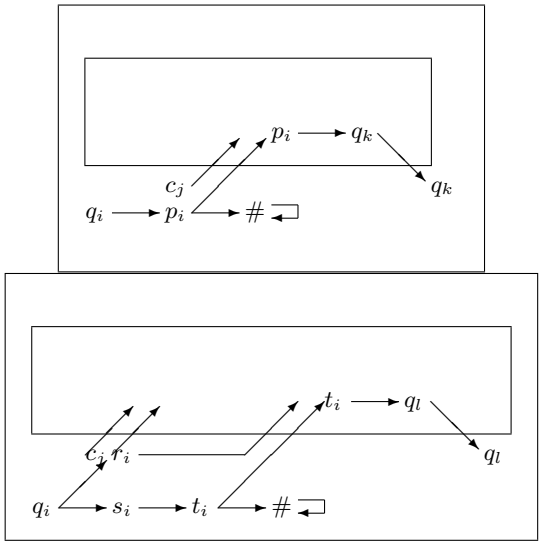

Fig. 2. Illustrating instruction $q_i : (c_j-, q_k, q_l)$.

4 Concluding Remarks

Recall that register machines, at a price of one more register, can generate recursively enumerable languages (assuming that incrementing counter j, $1 \leq j \leq k$, corresponds to "writing" c_j). Adding another register to the constructions of both theorems, one can notice that these systems generate recursively enumerable languages if we assume that all reactions happen in one step ($LECP_2(1,1) = LECP_2(2,0) = RE$).

In the time-free systems, we have no way of controlling the order in which objects c_j exit the system, and the order cannot be enforced because nothing else should be sent into the environment except the result. However, sending objects c_j directly to the environment (for output registers only) using targets will yield a similar result for the time-free EC P systems with targets ($fLEC_tP_2(1,1) = fLEC_tP_2(2,0) = RE$).

Finally, it would be interesting to combine the ideas of determinism and time-freeness, for example by considering time-free systems that are deterministic modulo the time when the rules are applied (i.e., the total number of applications of any rule does not depend on the times of rule execution).

Acknowledgements

The first author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The first author also acknowledges

the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034.

References

1. A. Alhazov: Minimizing evolution-communication P systems and EC P automata. In *Brainstorming Week on Membrane Computing*, Tarragona, 2003 (M. Cavaliere, C. Martín–Vide, Gh. Păun, eds.), GRLMC Report 26/03, Rovira i Virgili University, 2003, 23–31, and *New Generation Computing*, 22, 4 (2004), 299–310.
2. A. Alhazov: On determinism of evolution-communication P systems. In *Second Brainstorming Week on Membrane Computing*, Sevilla, 2004 (Gh. Păun, A. Riscos–Núñez, A. Romero–Jiménez, F. Sancho–Caparrini, eds.), GCN TR 01/2004, University of Sevilla, 2004, 11–15, and *Journal of Universal Computer Science*, 10, 5 (2004), 502–508.
3. A. Alhazov, M. Cavaliere: Proton pumping P systems. In *Preproceedings of the Workshop on Membrane Computing*, Tarragona, 2003 (A. Alhazov, C. Martín–Vide, Gh. Păun, eds.), GRLMC Report 28/03, Rovira i Virgili University, 2003, 1–16, and *Membrane Computing, International Workshop*, WMC 2003, Tarragona, 2003, Revised Papers (C. Martín–Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 2933, Springer-Verlag, Berlin, 2004, 1–18.
4. M. Cavaliere: Evolution-communication P systems. In *Membrane Computing. International Workshop*, WMC-CdeA 2002, Curtea de Argeş (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, Berlin, 2003, 134–145.
5. M. Cavaliere, V. Deufemia: Further results on time-free P systems. In *Cellular Computing. Complexity Aspects* (M.A. Gutiérrez–Naranjo, Gh. Păun, M.J. Pérez–Jiménez, eds.), Fénix Editora, Sevilla, 2005, 96–116.
6. M. Cavaliere, D. Sburlan: Time-independent P systems. In *Membrane Computing. International Workshop* WMC 2004, Milan, Italy, 2004, Revised Selected and Invited Papers (G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.), LNCS 3365, Springer-Verlag, Berlin, 2005, 239–258.
7. S.N. Krishna, A. Păun: Some universality results on evolution-communication P systems. In *Brainstorming Week on Membrane Computing*, Tarragona, 2003 (M. Cavaliere, C. Martín–Vide, Gh. Păun, eds.), GRLMC Report 26/03, Rovira i Virgili University, 2003, 207–215, and *New Generation Computing*, 22, 4 (2004), 377–394.
8. M.L. Minsky: *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
9. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
10. The P Systems Web Page: `http://psystems.disco.unimib.it`