P Systems with One Membrane and Symport/Antiport Rules of Five Symbols Are Computationally Complete

Artiom Alhazov[1], Rudolf Freund[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] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
E-mail: `rudi@emcc.at`

Summary. We consider P systems with only one membrane using symport/antiport rules and prove that any recursively enumerable set of k-dimensional vectors of natural numbers can be generated (accepted) by using at most $k+4$ symbols; hence, any recursively enumerable set of natural numbers can be generated (accepted) by using at most five symbols.

1 Introduction

In the area of membrane computing there are two main classes of systems: P systems with a hierarchical (tree-like) structure as already introduced in the original paper of Gheorghe Păun (see [10]) and tissue P systems with cells arranged in an arbitrary graph structure (see [6], [5]). We here consider "classical" P systems using symport/antiport rules for the communication through membranes (these communication rules first were investigated in [9]).

It is well known that equipped with the maximally parallel derivation mode (tissue) P systems with only one membrane (one cell) already reach universal computational power, even with antiport rules of weight two (e.g., see [2] and [4]); yet on the other hand, in these P systems the number of symbols remains unbounded.

Considering the generation of recursively enumerable sets of natural numbers we may also ask the question how many symbols we need for obtaining computational completeness in a small number of membranes. In [12] the quite surprising result was proved that three symbols are enough in the case of P systems with symport/antiport rules. The specific type of maximally parallel application of at most one rule in each connection (link) between two cells or one cell and the environment, respectively, in tissue P systems allowed for an even more surprising result proved in [7]: The minimal number of one symbol is already sufficient to obtain computational completeness, e.g., it was shown that any recursively enumerable set of natural numbers can be generated by a tissue P system with at most seven cells using symport/antiport rules of only one symbol. The question remained open whether such a result for the minimal number of symbols can also be obtained for "classical" P systems with symport/antiport rules.

In this paper we follow another direction and consider "classical" P systems using symport/antiport rules with the minimal number of membranes, i.e., we are going to prove that computational completeness can already be obtained in one membrane by using only five symbols.

2 Preliminaries

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [1] and [14]. We just list a few notions and notations: $\mathbb{N}$ denotes the set of non-negative integers (natural numbers). V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; by RE $(RE\,(k))$ we denote the family of recursively enumerable languages (over a k-letter alphabet). By $\Psi_T\,(L)$ we denote the Parikh image of the language $L \subseteq T^*$, and by $PsFL$ we denote the set of Parikh images of languages from a given family FL. $PsRE\,(k)$ corresponds with the family of recursively enumerable sets of k-dimensional vectors of non-negative integers.

2.1 Register machines

The proofs of the main results established in this paper are based on the simulation of register machines; we refer to [8] for original definitions, and to [2] for definitions like that we use in this paper:

An *(non-deterministic) register machine* is a construct $M = (n, R, l_0, l_h)$, where n is the number of registers, R 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 one of the instructions (labelled with) l_2 and 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 SUB instruction.)
- $l_h : halt$
  Stop the machine. The final label l_h is only assigned to this instruction.

A register machine M is said to generate a vector $(s_1, \ldots, s_k)$ of natural numbers if, starting with the instruction with label l_0 and all registers containing the number 0, the machine stops (it reaches the instruction $l_h : halt$) with the first k registers containing the numbers $s_1, \ldots, s_k$ (and all other registers being empty).

Without loss of generality, in the succeeding proofs we will assume that in each ADD instruction $l_1 : (A(r), l_2, l_3)$ and in each SUB instruction $l_1 : (S(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct (for a short proof see [5]).

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate exactly the sets of vectors of natural numbers which can be generated by Turing machines, i.e., the family $PsRE$. Especially we know that $k+2$ registers are enough to generate/accept any recursively enumerable set of k-dimensional vectors of non-negative integers (see [2], [8]).

2.2 P Systems with Symport/Antiport Rules

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [11]; comprehensive information can be found on the P systems web page `http://psystems.disco.unimib.it`.

A P system (of degree $m \geq 1$) *with symport/antiport rules* (in the following we shall only speak of a *P system*) is a construct

$$\Pi = (O, T, E, \mu, w_\mu, R_\mu, i_0),$$

where

- O is the alphabet of *objects*,
- $T \subseteq O$ is the alphabet of *terminal* objects,
- $E \subseteq O$ is the set of objects present in arbitrarily many copies in the environment,
- μ is the *membrane structure* (it is assumed that we have m membranes, labelled with $1, 2, \ldots, m$, the skin membrane usually being labelled with 1),
- w_i, $1 \leq i \leq m$, are strings over O representing the *initial* multiset of *objects* present in the membranes of the system,
- R_i, $1 \leq i \leq m$, are finite sets of symport/antiport rules of the form x/y, for some $x, y \in O^*$, associated with membrane i (if $|x|$ or $|y|$ equals 0 then we speak of a symport rule, otherwise we call it an antiport rule),

- i_0 is the designated output membrane to collect the terminal symbols representing the result of a halting computation.

An antiport rule of the form $x/y \in R_i$ means moving the objects specified by x from membrane i to the surrounding membrane j (to the environment, if $i = 1$), at the same time moving the objects specified by y in the opposite direction. (The rules with one of x, y being empty are, in fact, symport rules, but in the following we do not explicitly consider this distinction here, as it is not relevant for what follows.) The objects from E are never exhausted, irrespective how many copies of each of them are brought into the system, an unbounded number of copies remains available in the environment.

The computation starts with the multisets specified by $w_1, \ldots, w_m$ in the m membranes; in each time unit, the rules assigned to each membrane are used in a maximally parallel way, i.e., we choose a multiset of rules at each membrane in such a way that, after identifying objects inside and outside the corresponding membranes to be affected by the selected multiset of rules, no objects remain to be subject to any additional rule at any membrane. The computation is successful if and only if it halts; the result is represented by the multiset of terminal objects from T in the output membrane i_0. The set of all k-dimensional vectors computed in this way by the system Π is denoted by $Ps(\Pi, k)$. The family of sets $Ps(\Pi, k)$ of vectors computed as above by systems with at most m cells and at most s symbols is denoted by $PsO_sP_m(k)$. When any of the parameters k, m, s is not bounded, it is replaced by $*$.

As in this paper we are dealing with systems using only one membrane, we consider the set O to consist of the symbols a_1 to a_s, the set of terminal symbols consists of the first k symbols a_1 to a_k, and, moreover, we shall always assume all symbols to be available in an unbounded number in the environment, i.e., $E = O$. Hence, in the following we use the following simplified notation:

Convention. For P systems with one membrane we shall only write

$$\Pi = (s, k, w_1, R_1),$$

where s is the number of symbols, k is the number of terminal symbols, the string w_1 describes the multiset of objects initially being present in the skin membrane, and R is the finite set of symport/antiport rules assigned to the skin membrane.

3 Results

We first prove our main result that for simulating a register machine with d registers in a P system with one membrane we only need $d + 2$ symbols.

Theorem 1. *Each register machine with d registers can be simulated by a P system with symport/antiport rules in only one membrane with at most $d + 2$ symbols.*

Proof. Let us consider a register machine $M = (n, R, l_0, l_h)$ with d registers; no matter what the goal of the computation of M is (the number k then designates which registers are to be interpreted as output registers), we can construct the P system (of degree 1)

$$\Pi = (d + 2, k, w_1, R_1)$$

in such a way that Π simulates the actions of M in such a way that Π starts with the multisets of symbols corresponding to the initial values in the registers of M and halts if and only if M halts, thereby representing the final contents of the registers of M by the corresponding multisets of symbols in the skin membrane (and no other symbols contained there). The P system Π therefore is constructed as follows:

The symbols a_1 to a_d represent the registers; their initial values i_1 to i_d are represented by

$$w_0 = {a_1}^{i_1} ... {a_d}^{i_d}$$

whereas the symbols a_{d+1} and a_{d+2} are needed for encoding the instructions of M; a_{d+2} also has the function of a trap symbol, i.e., in case of the wrong choice for a rule to be applied we take in so many symbols a_{d+2} that we can never again rid of them and therefore get "trapped" in an infinite loop.

Throughout the rest of the proof we shall write p for the "program symbol" a_{d+1} as well as q for the "trap symbol" a_{d+2}.

An important part of the proof is to define a suitable encoding c for the instructions of the register machine: Without loss of generality we assume the labels of M to be positive integers such that the labels assigned to ADD and SUB instructions have the values $3i - 2$ for $1 \leq i < t$, as well as $l_0 = 1$ and $l_h = 3t - 2$, for some $t \geq 1$.

We now define the encoding c on non-negative integers in such a way that $\mathbb{N} \to \mathbb{N}$ is a linear function that has to obey to the following additional conditions:

- For any i, j with $1 \leq i, j \leq 3t$, $c(i) + c(j) > c(3t)$, i.e., the sum of the codes of two instruction labels has to be larger than the largest code we will ever use for the given M.
- The distance g between any two codes $c(i)$ and $c(i+1)$ has to be larger than any of the multiplicities of the symbol p which appear besides codes in the rules defined below.

As we shall see in the construction of the rules below, we may take

$$g = 8(d + 2).$$

Moreover, we define

$$h = g/2 = 4(d + 2).$$

A function c fulfilling all the conditions stated above then, for example, is

$$c(x) = gx + 3tg = g(x + 3t) \text{ for } x \geq 0.$$

With $l_0 = 1$ we therefore obtain

$$c(l_0) = g(3t + 1) = 8(d + 2)(3t + 1).$$

Hence, the initial multiset now is

$$w_1 = w_0 p^{c(l_0)}.$$

Finally, we have to find a number f which is so large that after the introduction of f symbols we inevitably enter an infinite loop with the rule

q^f/q^{2f};

as we shall justify below, we can take

$f = (c(3t))^2$.

Equipped with this coding function and the constants defined above we are now able to define the following set of symport/antiport rules assigned to the skin membrane for simulating the actions of the given register machine M :

$$\begin{aligned} R_1 = & \left\{p^{c(l_1)}/p^{c(l_2)}a_r, p^{c(l_1)}/p^{c(l_3)}a_r \mid l_1 : (A(r), l_2, l_3) \in R\right\} \\ \cup & \left\{p^{c(l_1)}a_r/p^{c(l_2)}, p^{c(l_1)}/p^{c(l_1+1)}p^r q^{h-r},\right. \\ & p^{c(l_1+1)}/p^{c(l_1+2)}, p^r q^{h-r}a_r/q^{2f}, \\ & \left. p^{c(l_1+2)}p^r q^{h-r}/p^{c(l_3)} \mid l_1 : (S(r), l_2, l_3) \in R\right\} \\ \cup & \left\{p^{c(l_h)}/\lambda, p^h/q^{2f}, q^f/q^{2f}\right\}. \end{aligned}$$

The correct work of the rules in R_1 can be described as follows:

1. Throughout the whole computation in Π, it is directed by the code $p^{c(l)}$ for some $l \leq 3t-2$; in order to guarantee the correct sequence of encoded rules the trap is activated in case of a wrong choice, which in any case guarantees an infinite loop with the symbols q by the "trap rule"
   q^f/q^{2f}.
   The minimal number of superfluous number of symbols p to start the trap is h and causes the application of the rule p^h/q^{2f}.
2. For each ADD instruction $l_1 : (A(r), l_2, l_3)$ of M, we introduce the rules
   $p^{c(l_1)}/p^{c(l_2)}a_r$, and
   $p^{c(l_1)}/p^{c(l_3)}a_r$.
   In that way, the ADD instruction $l_1 : (A(r), l_2, l_3)$ of M is simulated in only one step: the number of symbols p representing the instruction of M labelled by l_1 is replaced by the number of symbols p representing the instruction of M labelled by l_2 or l_3, respectively, in the same moment also incrementing the number of symbols a_r.
   If we do not choose one of the correct rules, then the trap will be activated by the rule p^h/q^{2f}: The coding function has been chosen in such a way that instead of the correct rule for the label l_1 only rules for labels $l < l_1$ could be chosen, but on the other hand, as the number of symbols p is not large enough for allowing the remaining rest being interpreted as the code of another instruction label, at least $2h$ symbols p would remain and activate the trap by the rule p^h/q^{2f}, i.e., the computation would enter an infinite loop at this point.
3. For simulating the decrementing step of a SUB instruction $l_1 : (S(r), l_2, l_3)$ from R we introduce the rule

$p^{c(l_1)}a_r/p^{c(l_2)}$
for decrementing the contents of cell r (which represents register r).
In that way, the decrementing step of the SUB instruction $l_1 : (S(r), l_2, l_3)$ of M now is also simulated in one step: together with $p^{c(l_1)}$ we also send out one symbol a_r and take in $p^{c(l_2)}$, which encodes the label of the instruction that has to be executed after the successful decrementing of register r.
Again we notice that if we do not choose the correct rule, then the trap is activated by the rule p^h/q^{2f}.

4. For simulating the zero test, i.e., the case where the contents of register r is zero, of a SUB instruction $l_1 : (S(r), l_2, l_3)$ from R we now take the following rules:
   $p^{c(l_1)}/p^{c(l_1+1)}p^r q^{h-r}$,
   $p^{c(l_1+1)}/p^{c(l_1+2)}$,
   $p^r q^{h-r} a_r/q^{2f}$, and
   $p^{c(l_1+2)}p^r q^{h-r}/p^{c(l_3)}$.
   In that way, the zero test step of the SUB instruction $l_1 : (S(r), l_2, l_3)$ of M now is simulated in three steps: in the first step, together with $p^{c(l_1+1)}$ we also introduce a small additional number r of symbols p and a small number $h - r$ of symbols q (both less than the number necessary to activate the trap rule p^h/q^{2f} or q^f/q^{2f}, respectively, because $r \le d + 2 < 4(d + 2) = h$ and $h - r < h < f$).
   In the combination $p^r q^{h-r}$ the number of symbols q in fact uniquely determines which rule is applicable: for consuming a rule $p^{r'} q^{h-r'}$ with $r' < r$ we do not have enough symbols q, whereas for consuming a rule $p^{r'} q^{h-r'}$ with $r' > r$ we do not have enough symbols p or else, if choosing a rule involving a smaller code $c(l)$, instead too many symbols p would remain thus allowing the application of the trap rule p^h/q^{2f}, because $c(l_1 + 1) - c(l) \ge 2h$ and $r' < h$, i.e., $(c(l_1 + 1) - c(l)) - r' > h$.
   This unique combination $p^r q^{h-r}$ now checks whether the symbol a_r is present with the rule $p^r q^{h-r} a_r/q^{2f}$ (which, after having been applied, lets the computation never stop again), whereas in the meantime the rule $p^{c(l_1+1)}/p^{c(l_1+2)}$ can be applied in any case. If a_r has not been present, $p^r q^{h-r}$ are still there and exchanged together with $p^{c(l_1+2)}$ by the rule $p^{c(l_1+2)}p^r q^{h-r}/p^{c(l_3)}$ thus introducing $p^{c(l_3)}$, which encodes the label of the instruction that has to be executed after the successful zero test on register r.
   Once again we notice that if in any of these steps described above we do not choose the correct rule, then the trap is activated by the rule p^h/q^{2f}.
5. The number of symbols p never exceeds $c(l_h)$. By definition,
   $$f = (c(3t))^2 = (c(3t-2) + 2g)^2 = (c(l_h) + 4h)^2 > 8hc(l_h),$$
   therefore all the rules $p^r q^{h-r}$ consuming small numbers of symbols q in total can never eliminate more than $hc(l_h)$ symbols. As $2hc(l_h) < f$, from $2f$ symbols occurring in the skin membrane after the application of a trapping

rule, more than f symbols remain for the application of the rule q^f/q^{2f}, which guarantees that the computation will never stop.

6. Finally, for the halt label $l_h = 3t-2$ we only take the rule

   $p^{c(l_h)}/\lambda$,

   hence, the work of Π will stop exactly when the work of M stops (provided the trap has not been activated due to a wrong non-deterministic choice during the computation).

From the explanations given above we conclude that Π halts if and only if M halts, and moreover, the final configuration of Π represents the final contents of the registers in M. These observations conclude the proof. □

Looking carefully into the proof elaborated above we may realize that the final rule

$p^{c(l_h)}/\lambda$

is the only symport rules, all other rules are antiport rules. We could only avoid this by relaxing the condition that at the end of a computation in the P system only the symbols representing the contents of the registers of M are allowed to be present in the skin membrane; hence instead of the symport rule $p^{c(l_h)}/\lambda$ we might also take the antiport rule

$p^{c(l_h)}/p^1$,

i.e., one "garbage symbol" p would remain in the skin membrane.

The main result proved above immediately implies the following results for generating, accepting as well as for computing P systems, as it is well known that only two additional registers are needed for generating, accepting or computing any recursively enumerable set of k-dimensional vectors of non-negative integers by a register machine:

In the generating case, the first k symbols represent the output registers:

Corollary 1. *$PsRE\,(k) = PsO_sP_1(k)$ for all $k \geq 1$ and $s \geq k+4$.*

For $k = 1$ we therefore obtain the result that any recursively enumerable set of non-negative integers can be generated by a P system with one membrane and at most five symbols.

In the accepting case, we designate the first l cells as the input cells and start the computation by introducing multisets over the corresponding symbols a_1 to a_l in the skin membrane; these multisets are accepted if and only if the computation halts.

Corollary 2. *For every set L from $PsRE\,(l)$ we can construct a P system with one membrane and at most $l+4$ symbols which accepts L.*

For computing functions from $\mathbb{N}^\alpha$ to $\mathbb{N}^\beta$ we can use the first α symbols for representing the input values and the first β symbols for representing the output values; due to Theorem 1 we obtain the following result:

Corollary 3. *For every function f from $\mathbb{N}^\alpha$ to $\mathbb{N}^\beta$ we can construct a P system with one membrane and at most* $\max\{\alpha,\beta\}+4$ *symbols which computes f.*

4 Open Questions

The number of symbols needed in the proofs given in this paper possibly may be improved, but we conjecture that the results obtained in this paper are already optimal.

As was proved in [12], three symbols are enough to obtain computational completeness with four membranes. So there remain several interesting open questions, e.g., how many symbols are needed for two and three membranes and, on the other hand, how many membranes are needed for two and four symbols to obtain computational completeness.

References

1. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.
2. R. Freund, M. Oswald: P Systems with activated/prohibited membrane channels. In [13], 261–268.
3. R. Freund, M. Oswald: Tissue P systems with symport/antiport rules of one symbol are computationally complete. In *Cellular Computing. Complexity Aspects* (M.A. Gutiérrez–Naranjo, Gh. Păun, M.J. Pérez–Jiménez, eds.), Fénix Editora, Sevilla, 2005, 187–200.
4. R. Freund, A. Păun: Membrane systems with symport/antiport rules: universality results. In [13], 270–287.
5. R. Freund, Gh. Păun, M.J. Pérez–Jiménez: Tissue-like P systems with channel states. In *Brainstorming Week on Membrane Computing*, Sevilla, February 2004, TR 01/04 of Research Group on Natural Computing, Sevilla University (2004) 206–223, and *Theoretical Computer Science*, 330 (2005), 101–116.
6. C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodriguez–Patón: Tissue P systems. *Theoretical Computer Science*, 296, 2 (2003), 295–326.
7. R. Freund, M. Oswald: Tissue P systems with symport/antiport rules of one symbol are computationally complete. In *Cellular Computing. Complexity Aspects* (M.A. Gutiérrez–Naranjo, Gh. Păun, M.J. Pérez–Jiménez, eds.), Fénix Editora, Sevilla, 2005, 187–200.
8. M.L. Minsky: *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
9. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3 (2002), 295–306.
10. 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`).
11. Gh. Păun: *Computing with Membranes: An Introduction.* Springer-Verlag, Berlin, 2002.

12. Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodríguez-Patón: Symport/antiport P systems with three objects are universal. *Fundamenta Informaticae*, 64, 1–4 (2005), 353–367.
13. Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.: *Membrane Computing. International Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers*, LNCS 2597, Springer-Verlag, Berlin, 2003.
14. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages* (3 volumes). Springer-Verlag, Berlin, 1997.
15. The P Systems Web Page: `http://psystems.disco.unimib.it`