Small Computationally Complete Symport/Antiport P Systems*

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

[1] Computer and Automation Research Institute
Hungarian Academy of Sciences
H-1111 Budapest, Kende u.13-17
{csuhaj,vaszil}@sztaki.hu
[2] Paul Verlaine University of Metz
UFR MIM, LITA, EA 3097
Ile du Saulcy, 57045 Metz Cédex
margens@univ-metz.fr
[3] LACL, University of Paris 12
UFR Sciences et Technologie
65, av. Général de Gaulle
94010 Créteil, France
verlan@univ-paris12.fr

Summary. It is known that P systems with symport/antiport rules simulate the register machines, i.e., they are computationally complete. Hence, due to the existence of universal register machines, there exist computationally complete subclasses of symport/antiport P systems with a number of rules limited by a constant. However, there was no estimation of this number in the literature. In this article, we first give a simple estimation of this constant, and then we show that the number can be reduced by grouping together several instructions of the simulated variant of the register machines. Finally, a universal P system with symport/antiport having only 44 rules is obtained.

1 Introduction

The theory of P systems is a recent vivid scientific field, on the borderline of bio-computing and theoretical computer science. P systems or membrane systems were introduced by Gheorghe Păun in 1998 (the full version of the first article

* Work supported by the Hungarian Foundation for Research and Technological Innovation (project no. TéT F-19/04) and the GRST in France (project no. Balaton 09000TC, year 2005) in the frame of the Hungarian-French Intergovernmental Scientific and Technological Cooperation. The work of E. Csuhaj-Varjú and Gy. Vaszil was also supported in part by the Hungarian Scientific Research Fund "OTKA" Grant no. T042529.

appeared in [8]) in order to introduce a computational concept which mimics the architecture and the functioning of the living cell and thus to provide us with a powerful unconventional computational device and a suitable tool for describing biological phenomena at the cellular (bio-molecular) level. Since then, the rapid development of the area and the obtained results have confirmed the expectations.

A P system or a membrane system is a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. The out-most membrane is called the skin membrane. The objects in the regions correspond to bio-chemical ingredients, the membranes to the membranes of the cell. During the functioning of the P system, the objects in the different regions may change and move across the membranes. The rules of the changes and the communication between the membranes can be defined in various manners, thus making possible to create and study different variants of P systems, with different motivations.

Especially important, biologically well-motivated variants of P systems are the so-called P systems with symport/antiport rules which realize models where the rules are purely communication rules, i.e., the objects do not change under the functioning of the system, they are only communicated (transported) from one region to some other one. The notion was introduced in [7]. Symport rules move objects across a membrane in one direction, while antiport rules move objects across the membranes in the two opposite directions. Similarly to other variants of P systems, these construct are computationally complete devices, even with only one membrane [2, 4, 3].

These constructs have been studied in details, with special emphasis on their size complexity. It has been shown that P systems with symport/antiport rules with bounded size parameters (the number of objects, the number of membranes, the number of rules per membrane, the number of objects carried together in one step, etc.) are as powerful as the general models, i.e., as powerful as the Turing machines. The interested reader is refereed to [1] for a survey on the results proved so far.

In [10] another important size complexity measure that was not investigated before, namely, the total number of rules in a P system was investigated. The authors showed that splicing tissue P systems with 8 rules are as powerful as the Turing machines. This result motivated the investigations to obtain a similar result for object-based P systems, in particular for symport/antiport P systems. The problem was also mentioned at the "6th Workshop on Membrane Computing", Vienna, July, 2005. In this paper, we give an answer to this question: we prove that any strongly universal register machine can be simulated by a P system with only one membrane and 44 antiport rules. The proof is based on a result by Ivan Korec [5], which shows that it is possible to construct a strongly universal register machine with not more than 32 instructions of three types (incrementing, decrementing, and zero-test).

2 Preliminaries

We assume that the reader is familiar with basic notions of formal language theory, computability, and membrane computing, i.e., the theory of P systems. For further details and unexplained notions we refer to [11], [6], [9]. For the notations and the results on register machines we use throughout the paper, the reader is advised to consult [5].

Let V be an alphabet, let V^* be the set of all words over V, and let $V^+ = V^* - \{\varepsilon\}$ where ε denotes the empty word. We denote by $\mathbb{N}$ the set of all non-negative integers.

Let V be a set – the universe – of objects. A multiset is a pair $M = (V, f)$, where $f : V \to \mathbb{N}$ is a mapping which assigns to each object $a \in V$ its multiplicity. The support of $M = (V, f)$ is the set $supp(M) = \{a \in V \mid f(a) \geq 1\}$. If $supp(M)$ is a finite set, then M is called a finite multiset. The set of all finite multisets over the set V is denoted by V°.

The number of objects in a finite multiset $M = (V, f)$, the cardinality of M, is defined by $card(M) = \sum_{a \in V} f(a)$. We say that $a \in M = (V, f)$ if $a \in supp(M)$. $M_1 = (V, f_1) \subseteq M_2 = (V, f_2)$ if $supp(M_1) \subseteq supp(M_2)$ and for all $a \in V$, $f_1(a) \leq f_2(a)$. The union of two multisets is defined as $(M_1 \cup M_2) = (V, f')$ where for all $a \in V$, $f'(a) = f_1(a) + f_2(a)$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) = (V, f'')$ where $f''(a) = f_1(a) - f_2(a)$ for all $a \in V$. We say that M is empty, if its support is empty, $supp(M) = \emptyset$.

A multiset M over the finite set of objects V can be represented as a string w over the alphabet V with $|w|_a = f(a)$ where $a \in V$ and where $|w|_a$ denotes the number of occurrences of the symbol a in the string w.

2.1 P systems with symport/antiport rules

In the following we briefly recall the basic notions concerning P systems with symport/antiport rules. For more details on these systems, we refer to [7], and for more information on P systems in general, the reader is advised to consult [9].

A P system is a structure of hierarchically embedded membranes, each having a label and enclosing a region containing a multiset of objects and possibly other membranes. The out-most membrane which is unique and usually labeled with 1, is called the skin membrane. The membrane structure is denoted by a sequence of matching parentheses where the matching pairs have the same label as the membranes they represent. If $x \in \{[_i,]_i \mid 1 \leq i \leq n\}^*$ is such a string of matching parentheses of length $2n$, denoting a structure where membrane i contains membrane j, then $x = x_1\ [_i\ x_2\ [_j\ x_3\ ]_j\ x_4\ ]_i\ x_5$ for some $x_k \in \{[_l,]_l \mid 1 \leq l \leq n,\ l \neq i, j\}^*,\ 1 \leq k \leq 5$. If membrane i contains membrane j, and there is no other membrane, k, such that k contains j and i contains k (x_2 and x_4 above are strings of matching parentheses themselves), then we say that membrane i is the parent membrane of j, and at the same time, membrane j is one of the child membranes of i.

By the contents of a region we mean the multiset of objects which is contained by the corresponding membrane excluding those objects which are contained by any of the other membranes which are included by the considered one.

The evolution of the contents of the regions of a P system is described by rules associated to the regions. Applying the rules synchronously in each region, the system performs a computation by passing from one configuration to another one. The rules are applied in the maximally parallel manner, i.e., at each step of the computation as many rules are applied in parallel in each region as possible.

The computation starts in the initial configuration (with initial contents of the regions). The end of the computation is defined by halting: a P system halts when no more rules can be applied in any of the regions. The result of the computation can be given in several ways, see [9] for more details. In the following we shall consider as the result of the computation the number of objects of a certain type that can be found in a certain region at halting.

In the sequel, we shall consider communication rules called antiport rules. An antiport rule is of the form $(x, in; y, out), x, y \in V^\circ$. If such a rule is associated with a membrane, then objects of x must enter from the parent region and at the same step, objects of y have to leave to the parent region. (Obviously, the successful application of the rule is only possible if the region contains x and its parent region contains y.) If the communication is one-way, then we speak of symport rules. A symport rule is of the form (x, in) or $(x, out), x \in V^\circ$. In this case either the objects of the multiset x must enter from the parent region or must leave to the parent region, $parent(i)$, respectively.

Formally, a P system with n membranes and antiport rules is a construct $\Pi = (V, O, \mu, E, (w_1, P_1), \ldots, (w_n, P_n,), i_0)$, where $n \geq 1$,

- V is a finite alphabet of objects, $O \subseteq V$ is the set of terminal objects, and $E \subseteq V$ is the set of objects appearing in the environment in arbitrarily many copies,
- μ is a membrane structure of n membranes with membrane 1 being the skin membrane, and for all $i, 1 \leq i \leq n$,
  - $w_i \in V^\circ$ is the initial contents (state) of region i, i.e., it is the finite multiset of objects contained by region i,
  - P_i is a finite set of antiport rules associated with membrane i,
  - $i_0 \in \{1, \ldots, n\}$ is a designated membrane, called the output membrane.

The result of a halting computation in Π is defined as the number of objects from O that can be found in membrane i_0 in the end of the computation.

We say that Π' is a scheme of P systems with antiport rules if no initial contents of the regions are specified, i.e., $\Pi' = (V, O, \mu, P_1, \ldots, P_n, i_0)$, where V, O, μ, $P_1, \ldots, P_n$ are defined as for Π above.

Π' is said to be *a universal antiport P system* if for any register machine M there exists a *simulating antiport P system* Π'' such that Π'' is obtained from Π' with adding axioms to the regions, i.e., $\Pi'' = (V, O, \mu, (w_1, P_1), \ldots, (w_n, P_n,), i_0)$, for some $w_i \in V^\circ$, $1 \leq i \leq n$.

It is known that P systems with symport/antiport rules are computationally complete, moreover, for any register machine with 3 registers we can construct a P system with symport/antiport rules with one membrane such that the contents of the first register at halting will be equal to the number of a certain symbol found in the unique region of the system at halting.

2.2 Register machines

In the following we recall the main concepts and results concerning register machines from [6] and [5].

A deterministic register machine consists of a finite control unit and uses a finite number of registers, $R_1, \ldots, R_k$. Each register may contain an arbitrary non-negative integer. These machines are deterministic (analogously to the Turing machines) and work in discrete time. Various classes of register machines exist, which differ from each other by the allowed one-step tests and/or operations, called instructions. We shall use the following five types of instructions from [5]:

1. $[RiP]$ – Add 1 to the content of register R_i;
2. $[RiM]$ – Subtract 1 from the content of register R_i if it is a positive value;
3. $\langle Ri \rangle$ – Check whether or not the contents of register R_i is zero; the next inner state depends on the result.
4. $\langle RiZM \rangle$ – Test whether the content of register R_i is positive or not and subtract 1 from the content of R_i in the first case. The new inner state of the machine depends on the result of the test.
5. $Halt$ – this is the halting operation, the machine stops working.

It is easy to see that $\langle RiZM \rangle$ joins $\langle Ri \rangle$ and $[RiM]$.

Formally, a deterministic register machine is the following construction:

$$M = (Q, R, q_1, q_0, P),$$

where Q is a set of states, $R = \{R_1, \ldots, R_k\}$ is the set of registers, $q_1 \in Q$ is the initial state, $q_0 \in Q$ is the final state and P is a set of transitions (called also rules or commands) of the form $(q, I, s_1, \ldots, s_m)$ where I is one of the instructions above ($[RiP]$, $[R_iM]$, $\langle Ri \rangle$, $\langle RiZM \rangle$) and $s_1, \ldots, s_m$ are states from Q (m depends on the type of instruction: 1 for $[RiP]$ and $[RiM]$, 2 for $\langle Ri \rangle$ and $\langle RiZM \rangle$, 0 for *Halt*).

We note that for each state q there is only one instruction of the type above.

A configuration of a register machine is given by the $k+1$-tuple $(q, n_1, \ldots, n_k)$ describing the current state of the machine as well as the contents of all registers. A transition of the register machine consists of updating/checking the value of a register according to an instruction of one of the types above and by changing the current state to some other one. More exactly, the machine works as follows. Depending on the state q, the corresponding instruction is performed and the new state from $s_1, \ldots, s_m$ is selected. By convention, if register R_i is not zero, then

after performing instruction $\langle Ri\rangle$, state s_1 is chosen, otherwise state s_2 is chosen. State q_1 is called the initial state and state q_0 is called the halting state.

We say that M computes a value $y \in \mathbb{N}$ on the *input* $x \in \mathbb{N}$ if starting from the initial configuration $(q_1, x, 0, \ldots, 0)$ it reaches the final configuration $(q_0, y, 0, \ldots, 0)$. If the machine halts in some other inner state, the value y is not defined. It is well-known that register machines compute all partial recursive functions and only them.

We say that M recognizes the set $S \subseteq \mathbb{N}$ if for any input $x \in S$ the machine stops and for any $y \notin S$ the machine performs an infinite computation. It is known that register machines recognize all recursively enumerable sets of numbers [6]. Moreover, only three registers are sufficient to obtain this power.

Register machines and partial recursive functions are strongly related, namely for every $n \in \mathbb{N}$, with every register machine M having n registers, an n-ary partial recursive function Φ_M^n is associated.

Let $\Phi_0, \Phi_1, \Phi_2, \ldots$, be a fixed admissible enumeration of the set of unary partial recursive functions. Then, a register machine M is said to be strongly universal if there exists a recursive function g such that for all $x, y \in \mathbb{N}$ it holds that $\Phi_x(y) = \Phi_M^2(g(x), y)$. A register machine M is called universal if there exist recursive functions f, g, h such that for all $x, y \in \mathbb{N}$ it holds that $\Phi_x(y) = f(\Phi_M^2(g(x), h(y))$.

We also note that the power and the efficiency of a register machine M depends on the set of instructions that are used. In [5] several sets of instructions are investigated. In particular, it is shown that there are strongly universal register machines with 32 instructions of form $[RiP]$, $\langle Ri\rangle$, and $[RiM]$, and there exist strongly universal register machines with 22 instructions of form $[RiP]$ and $\langle RiZM\rangle$. Moreover, these machines can be effectively constructed.

Since our result is based on the first construction, namely, the construction of the strongly universal register machine U_{32} from [5], we provide the reader with some necessary details. Firstly, we recall the construction of machine U_{32}.

We define $U_{32} = (Q, \{R_0, \ldots, R_7\}, q_1, q_0, P)$, where $Q = \{q_0, q_1, \ldots, q_{32}\}$, and P is defined as shown in Table 1.

The proof that U_{32} is a strongly universal register machine ([5]) is based on the following considerations.

The idea is to simulate any partial recursive function by register machines belonging to a very restricted class of machines. In [5], the considered machines are called $R3a$-machines. They consist of three register machines with additional constraints on the instructions which are restricted to three kinds always using the same registers and the same operations on the registers. Using standard tools, see [6] for instance, it is not difficult to prove that $R3a$-machines simulate any partial recursive function.

The idea of the simulation by the universal register machine of [5] is to encode the list of instructions into a finite sequence of numbers which can be extracted from a unique positive integer x thanks to an appropriate application of the Chinese remainder theorem. However, here, an additional property is used: we can

$(1) : (q_1, \langle R1\rangle, q_2, q_6)$	$(2) : (q_2, [R1M], q_3)$
$(3) : (q_3, [R7P], q_1)$	$(4) : (q_4, \langle R5\rangle, q_5, q_7)$
$(5) : (q_5, [R5M], q_6)$	$(6) : (q_6, [R6P], q_4)$
$(7) : (q_7, \langle R6\rangle, q_8, q_4)$	$(8) : (q_8, [R6M], q_9)$
$(9) : (q_9, [R5P], q_{10})$	$(10) : (q_{10}, \langle R7\rangle, q_{11}, q_{13})$
$(11) : (q_{11}, [R7M], q_{12})$	$(12) : (q_{12}, [R1P], q_7)$
$(13) : (q_{13}, \langle R6\rangle, q_{14}, q_1)$	$(14) : (q_{10}, \langle R4\rangle, q_{15}, q_{16})$
$(15) : (q_{15}, [R4M], q_1)$	$(16) : (q_{16}, \langle R5\rangle, q_{17}, q_{23})$
$(17) : (q_{17}, [R5M], q_{18})$	$(18) : (q_{18}, \langle R5\rangle, q_{19}, q_{27})$
$(19) : (q_{19}, [R5M], q_{20})$	$(20) : (q_{20}, \langle R5\rangle, q_{21}, q_{30})$
$(21) : (q_{21}, [R5M], q_{22})$	$(22) : (q_{22}, [R4P], q_{16})$
$(23) : (q_{23}, \langle R2\rangle, q_{24}, q_{25})$	$(24) : (q_{24}, [R2M], q_{32})$
$(25) : (q_{25}, \langle R0\rangle, q_{26}, q_{32})$	$(26) : (q_{26}, [R0M], q_1)$
$(27) : (q_{27}, \langle R3\rangle, q_{28}, q_{29})$	$(28) : (q_{28}, [R3M], q_{32})$
$(29) : (q_{29}, [R0P], q_1)$	$(30) : (q_{30}, [R2P], q_{31})$
$(31) : (q_{31}, [R3P], q_{32})$	$(32) : (q_{32}, \langle R4\rangle, q_{15}, q_0)$

Table 1. The instructions of the strongly universal machine U_{32}

require that the expected numbers are realized as the remainders of $x+1$ modulo non-divisors of $x+1$ whose ranks represent the instructions. If we denote by $F(x, y)$ the function given by $(x, y) \mapsto (x+1) \bmod \mathbf{nd}(x+1, y)$, where $\mathbf{nd}(z, j)$ is the j^{th} non-divisor of z, then it is not difficult to compute $F(x, i)$: we stop when the i^{th} non-zero test of the division of $x+1$ by k is positive, k being incremented at each test. Next, we code both the next state j of an instruction and the instruction I itself as $3j + \nu(I)$ where $\nu(I)$ ranges in $\{0, 1, 2\}$ as we use only three types of instructions.

The universal machine consists of three blocks of instructions. The first block extracts $F(x, i)$ and gives it as k. The second block extracts j and $\nu(I)$ from $k = 3j+\nu(I)$. Then, instruction I is executed in the third block, possibly transforming j into $j-1$, depending on I. And as the new label is known, we find the next code of an instruction by computing $F(x, j)$.

It is important to note that there are infinitely many representations of a given $R3a$-machine by a number x. The set of all the representations is not recursively enumerable but it contains recursive subsets in which there is at least one representative for each machine. This is enough for the validity of the proof.

The proof combines a few non-trivial results of prime number theory and combinatorial considerations on algorithms and register machines. It is very tricky and intricate.

We remark, that if instructions of type $\langle RiZM\rangle$ are used, then corresponding machine (U_{22}) has 22 instructions. This machine can be easily obtained from U_{32}. Indeed, we note that most of the $\langle Ri\rangle$ instructions are followed by $[RiM]$ instructions. Hence these two instructions can be combined into one $\langle RiZM\rangle$ instruction. However, this is not the case for state q_{13}, therefore an additional $[R6P]$ instruction shall be added. This gives a total of 22 rules.

To help the reader in the easier understanding how the machine functions, we recall on Figure 1 the flowchart of the computation by machine U_{32}, Fig. 1, page 269, from [5].

3 Constructing a Small Universal Antiport P System

We shall construct a universal P system with only antiport rules and one membrane which simulates register machine U_{32} (more exactly, register machine U_{22} derived from U_{32}). We describe the simulation in several steps.

3.1 A basic simulation technique

We first show how to simulate rules of an arbitrary register machine $M = (Q, R, q_1, q_0, P)$ using instructions $[RiP]$ and $\langle RiZM\rangle$ by a P system Π with only antiport rules and one membrane. For the sake of simplicity, we specify only the rules of Π, the alphabets of objects and terminal objects can be extracted easily from the presented rules.

It is easy to see that a rule $(p, [R_iP], q)$ of M can be simulated by an antiport rule $(p, out; qR_i, in)$ of Π, where p, R_i, q are objects of Π. Symbols p and q in Π correspond to states p and q of M, while the occurrences of symbols R_i represent the contents of register R_i. (The number of objects R_i found in the region of Π corresponds to the actual contents of register R_i of M.)

A rule $(p, \langle Ri\rangle, q, s)$ in M can be simulated by the following group of rules in Π:

$$(p, out; p'C_{p,i}, in),$$
$$(C_{p,i}R_i, out; C'_{p,i}, in),\ (p', out; p'', in),$$
$$(p''C'_{p,i}, out; q, in), \qquad (p''C_{p,i}, out; s, in).$$

As above, $p, p', p'', R_i, C_{p,i}, C'_{p,i}, q, s$ are objects of Π. The idea behind this simulation is very simple. Firstly, symbol p (which corresponds to state p of M) is replaced by two symbols $C_{p,i}$ and p'. The first symbol tries to decrease the number of R_is by one (in this case it becomes $C'_{p,i}$), while the second symbol is renamed to p''. Now, depending on the ability of $C_{p,i}$ to decrease the number of R_is, i.e., on the emptiness or non-emptiness of register R_i in M, the symbol corresponding to the new state of M is selected.

3.2 New commands

We note that using the basic simulation technique described above, one can translate machine U_{32} into an antiport P system with $13 \cdot 5 + 9 = 74$ rules. Now we show that this number can be decreased.

Firstly, we shall introduce new commands for the register machine U_{32} and then we shall show how to simulate these commands efficiently.

We introduce the following commands (see also Figure 2):

$(p, \text{ifplus}(R_i-, R_k+), q, s)$.

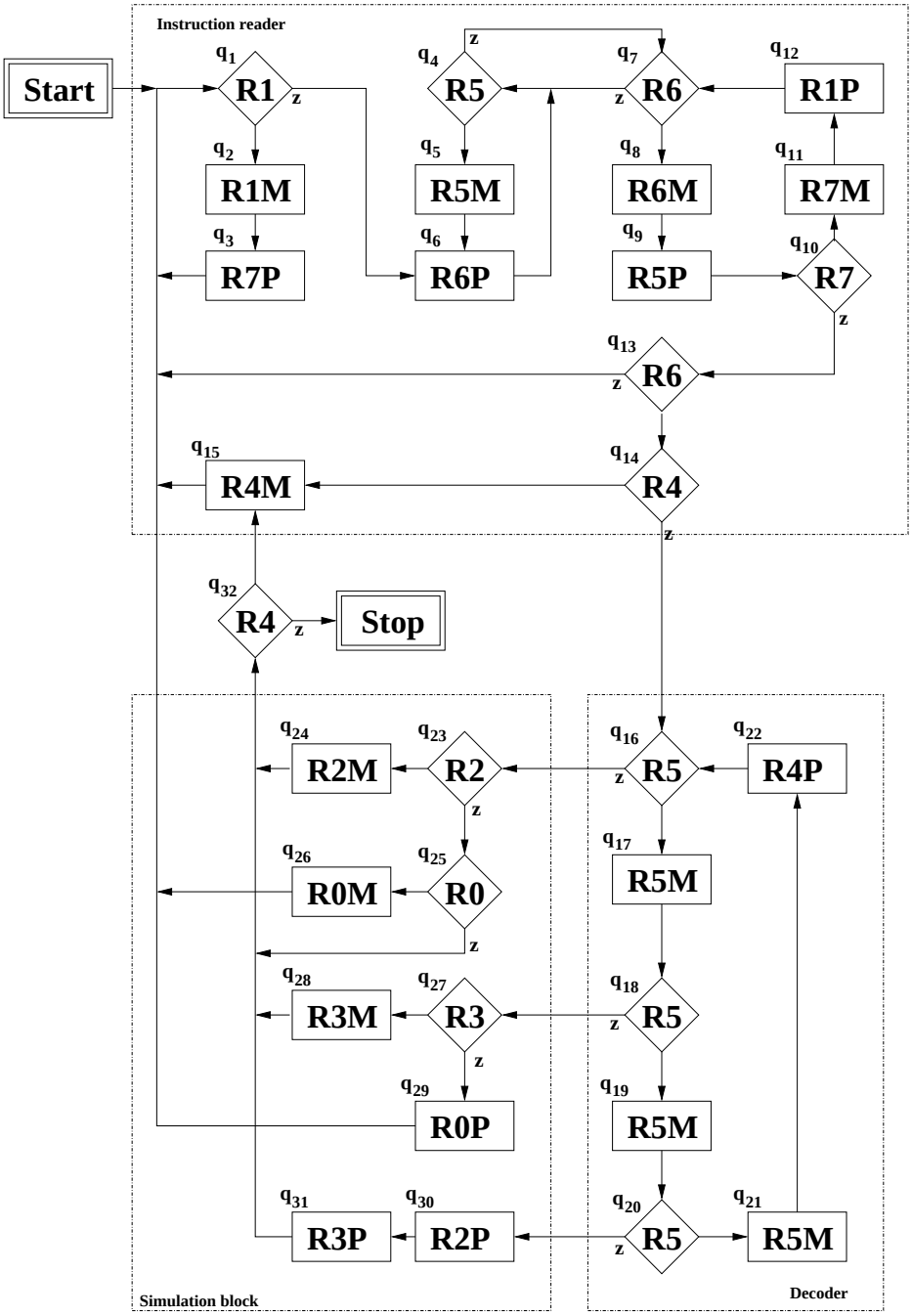

Fig. 1. Flowchart of strongly universal machine U_{32}

This command tries to subtract one from the contents of register R_i. If this is possible, then it increases the contents of register R_k by one and enters state q. Otherwise, it enters state s. The graphical representation of this command is depicted in Figure 2 (a). The path labeled by z is selected when the contents of R_i is zero, otherwise the other path is chosen.

$(p, \mathrm{mod3}(R_i, R_k), q_0, q_1, q_2)$.

This command adds to R_k the value R_i div 3. After that, if the value of R_i mod 3 is zero (resp. one, two) then the new state of the machine becomes q_0 (resp. q_1, q_2). Finally, register R_i becomes empty (its stored value is zero). In Figure 2 (b) the graphical representation of this command is given. The exiting paths are labeled by 0, 1, and 2 and they are selected depending on the value modulo 3 of register R_i.

$(p, \mathrm{2choice}(R_i, R_k), q_{00}, q_{10}, q_{01}, q_{11})$.

This command performs zero tests on registers R_i and R_k and decreases the corresponding register(s). Depending on these tests, a new state is selected (q_{00} if both registers are zero, q_{10} if the first register is non-zero and the second is zero and so on). If the contents of register R_i (resp. R_k) must not be decreased, then the corresponding states shall be marked by an overline (resp. underline). The graphical representation of this command is depicted in Figure 2 (c). This element has 4 exits labeled by a 2-bit binary string. The first bit of this string corresponds to the test result of register R_i (0 if R_i is zero, 1 otherwise), while the second bit corresponds to the test result of register R_k. A small rectangle may be placed over or under the outgoing edges. If a rectangle is placed over (resp. under) an edge, then register R_i (resp. R_k) will not be decreased (hence only a zero test for this register is performed). For example, the element depicted in Figure 2 (c) corresponds to command $(p, \mathrm{2choice}(R_i, R_k), q_{00}, \overline{q_{10}}, \underline{q_{01}}, \overline{q_{11}})$. This command only tests register R_i, and decrements register R_k only if both R_i and R_k are non-empty.

Now we translate the flow-chart from Figure 1 into a new one, which uses the new commands given above. This translation can be easily done and Figure 3 shows the new flow-chart. We only remark that the 2choice instruction combining states q_{27} and q_{32} is reused for states q_{20} and q_{23}, q_{25}. In order to achieve this, register R_3 is incremented, hence only R_4 is tested.

3.3 Efficient simulation of U_{32} with the new commands

We demonstrate that using the new commands we can describe U_{32} in a more economic manner than with the original instructions.

Let us start with the ifplus command. Using the basic simulation technique from Section 2.2., this command can be simulated using 6 instructions. But, it is

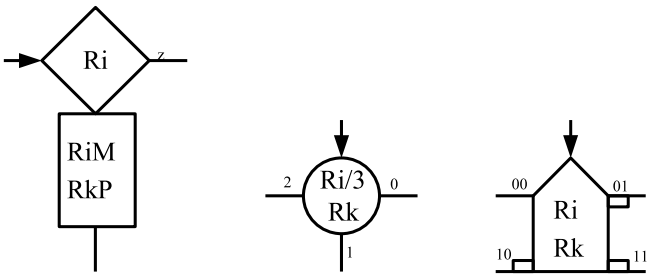

Fig. 2. The ifplus (a), mod3 (b) and 2choice (c) commands

easy to observe that the plus instruction can be performed during the last step of the simulation of the subtraction instruction. This gives us 5 rules:

$$\begin{array}{ll}(p, out; p'C_{p,i}, in), & \\ (C_{p,i}R_i, out; C'_{p,i}, in), & (p', out; p'', in), \\ (p''C'_{p,i}, out; qR_k, in), & (p''C_{p,i}, out; s, in).\end{array}$$

We note that several incrementing instructions can be performed by an appropriate modification of the fourth rule.

Now let us consider the mod3 instruction. This corresponds to three consecutive $\langle RiZM\rangle$ instructions and to one $[RkP]$ instruction, and thus by using the basic simulation technique we need 16 rules. However, 10 rules are sufficient for this purpose.

$$\begin{array}{lll}(p, out; p'X_{p,i}, in), & & \\ (X_{p,i}R_iR_iR_i, out; X'_{p,i}, in), & (p', out; p'', in), & \\ (p''X'_{p,i}, out; p'X_{p,i}R_k, in), & (p''X_{p,i}, out; S_pC_{p,i}C_{p,i}, in), & \\ (C_{p,i}R_i, out; C'_{p,i}, in), & (S_p, out; S'_p, in), & \\ (S'_pC'_{p,i}C'_{p,i}, out; q_2, in), & (S'_pC'_{p,i}C_{p,i}, out; q_1, in), & (S'_pC_{p,i}C_{p,i}, out; q_0, in).\end{array}$$

We act as in the previous case, but we simulate the decrement of register R_i not by one but by three. If this is successful, then the process is re-iterated (incrementing R_k at the same time). When the register cannot be decreased by three, then a new stage begins and two copies of symbol $C_{p,i}$, as well as symbol S_p are brought from outside. Symbol $C_{p,i}$ corresponds to symbol $C_{p,i}$ and symbol S_p corresponds to symbol p' from the basic simulation technique. At the end, there are three cases depending on value of register R_i which are considered by last three rules.

Finally, let us consider the 2choice instruction. Using the basic simulation technique, we can represent this instruction, which is essentially two $\langle RiZM\rangle$ instructions performed one after another, by 10 symport/antiport rules. We show that 8 rules suffice for this purpose:

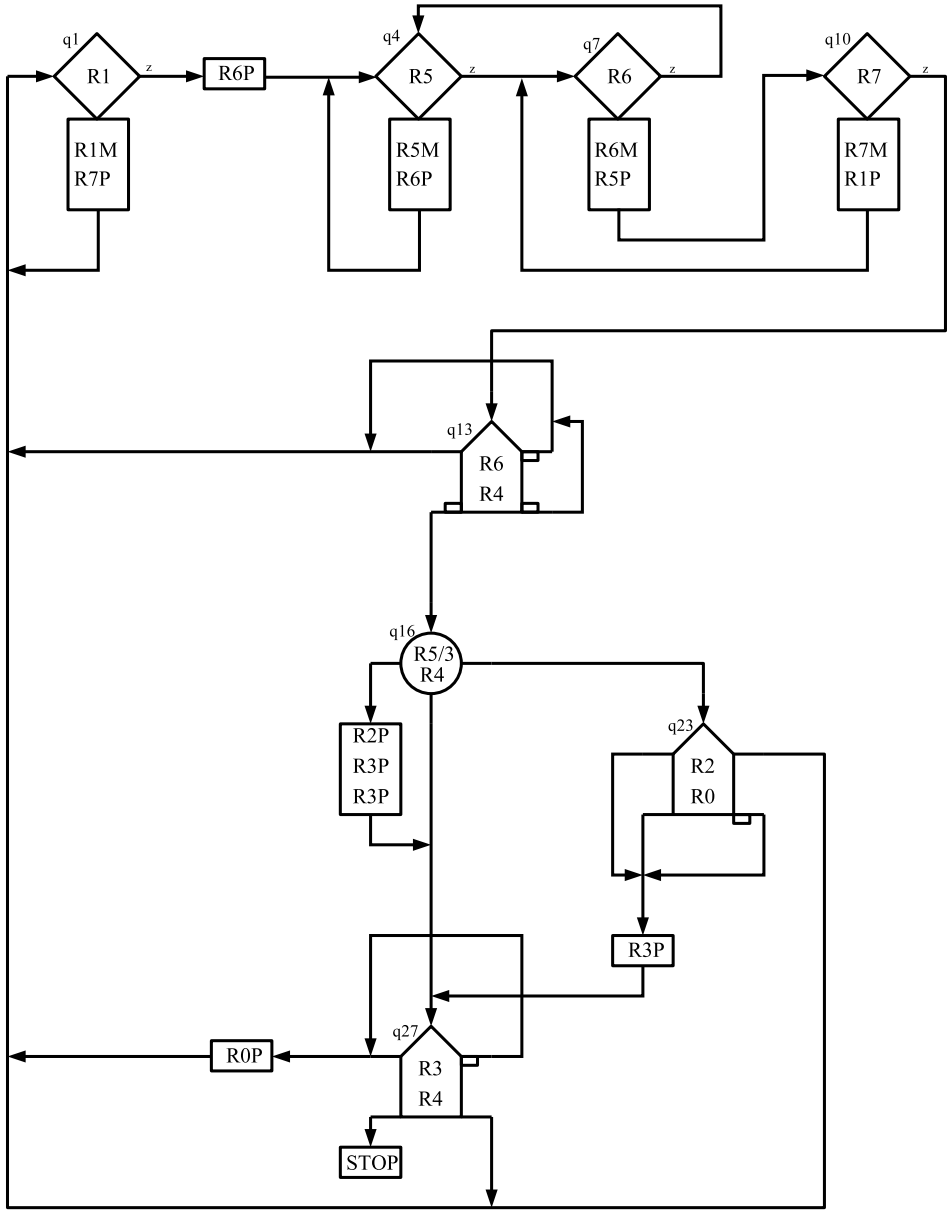

Fig. 3. Modified flowchart of strongly universal machine U_{32}

$(p, out; S_p C_{p,i} C_{p,k}, in)$,
$(C_{p,i} R_i, out; C'_{p,i}, in)$, $(C_{p,k} R_k, out; C'_{p,k}, in)$, $(S_p, out; S'_p, in)$,
$(S'_p C'_{p,i} C'_{p,k}, out; q_{11}, in)$, $(S'_p C'_{p,i} C_{p,k}, out; q_{10}, in)$,
$(S'_p C_{p,i} C'_{p,k}, out; q_{01}, in)$, $(S'_p C_{p,i} C_{p,k}, out; q_{00}, in)$.

If one of the registers must not be decremented, then we change the corresponding rule allowing to bring back one copy of the symbol corresponding to the register. For example, if R_i should not be decremented in the case when both register values are positive, then the fifth rule becomes $(S'_p C'_{p,i} C'_{p,k}, out; q_{11} R_i, in)$.

We described how to efficiently implement the above three instructions. We also note that all the remaining addition instructions from the original flowchart can be combined with the previous instructions during the simulation as it was shown in the case of the ifplus instruction. Hence, we need $4 \cdot 5 + 1 \cdot 10 + 3 \cdot 8 = 54$ rules.

This number can be further decreased by using the following technique. For each register R_i, $0 \leq i \leq 7$, there are "checker" symbols C_i and C'_i in the simulating P system Π. By adding rules $(C_i R_i, out; C'_i, in)$, we allow to check if R_i is not equal to zero and simultaneously to decrease its contents by one. In this case symbol C_i is replaced by C'_i. We note that several checks on different registers can be performed simultaneously. After that, the state of C_i can be checked by some other symbol S' and depending on this information the new state can be selected (combined with an addition instruction if needed). Finally, some initialization is necessary. As S' should not be present initially, a rule $(S, out; S', in)$ shall be added. As an example, we shall implement instruction $(p, 2\text{choice}(R_i, R_k), q_{00}, q_{10}, q_{01}, \underline{q_{11}})$. We obtain the following rules:

$(p, out; p' S C_i C_k, in)$,
$(S, out; S', in)$, $(p' S' C'_i C'_k, out; q_{11} R_k, in)$, $(p' S' C'_i C_k, out; q_{10}, in)$,
$(p' S' C_i C'_k, out; q_{01}, in)$, $(p' S' C_i C_k, out; q_{00}, in)$.

This gives us only 5 rules (comparing to the 8 rules which we had before) because we only need to add $(S, out; S', in)$ once. Moreover, this principle can be used in the case of instruction ifplus as well which brings it to 3 rules. Using the same idea in the case of instruction mod3 we can save two rules (hence it will need only 8 symport/antiport rules). Of course, the above idea also adds 9 rules, but at the end we gain more: only $4 \cdot 3 + 9 + 3 \cdot 5 + 8 = 44$ rules are needed for the simulation.

Summarizing the previous discussions, we can state the following statements.

Theorem 1.

1. *For any strongly universal register machine there exists a simulating universal symport/antiport P system with only one membrane and 44 antiport rules.*
2. *Any partial recursive function can be computed by a symport/antiport P system with only one membrane and 44 antiport rules.*
3. *Any recursively enumerable set of numbers can be computed with a symport/antiport P system with only one membrane and 46 rules.*

Proof. The first two statements follow from the discussion above, since any partial recursive function can be computed by the universal U_{32} machine where the code of the simulated machine is in register 1, the input in register 2 and the output in register 0. Thus, if we initialize the P system based on U_{32} with the appropriate number of R_1 and R_2 objects and make R_0 be the only terminal, then it computes the same value as U_{32}. To prove the third statement, consider the same P system built according to U_{32} and initialize it only with a number of R_1 symbols. If now we add a new initial state symbol q and two rules $(q, out; R_2q, in)$, $(q, out; q_0, in)$, then we can generate any number which is computed by the machine corresponding to the number of R_1 symbols which was used to initialize the system. □

4 Conclusion

In this paper we proved that antiport P systems with 44 rules are as powerful as the Turing machines. Since this is a first estimation of the upper bound of this size complexity parameter, it is a open question which number is the sharp bound if the system is supposed to have only one membrane. A further interesting research direction is to give an upper bound for a size description of these systems similar to the size complexity measure Symb in the case of Chomsky grammars, namely, to count the number of symbols (including the symbols describing the membrane architecture) that are necessary to describe a universal antiport P system. Since antiport P systems are only one of the different variants of object-based P systems, studying concise descriptions of other P system variants in this sense are of interest as well. Comparisons of size complexity measures of different types of P systems are of importance as well. The most challenging problem, raised by Gheorghe Păun, is the following: what is the size minimum of a universal P system irrelevant to its type. These and similar problems are topics of future research.

References

1. A. Alhazov, R. Freund, Y. Rogozhin: Computational power of symport/antiport: History, advances and open problems. In *Pre-Proceedings of the 6th International Workshop on Membrane Computing, WMC6* (R. Freund, G. Lojka, M. Oswald, Gh. Păun, eds.), Vienna, July 2005, 44–78.
2. R. Freund, M. Oswald: P systems with activated/prohibited membrane channels. In *Membrane Computing. International Workshop, WMC-CdeA'02* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, LNCS 2597, Springer, 2003, 261–268.
3. R. Freund, A. Păun: Membrane systems with symport/antiport: Universality results. In *Membrane Computing. International Workshop, WMC-CdeA'02* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, LNCS 2597, Springer, 2003, 270–287.

4. P. Frisco, H.J. Hoogeboom: Simulating counter automata by P systems with symport/antiport. In *Membrane Computing. International Workshop, WMC-CdeA'02* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), Curtea de Argeş, Romania, August 2002, LNCS 2597, Springer, 2003, 288–301.
5. I. Korec: Small universal register machines. *Theoretical Computer Science*, 168 (1996), 267–301.
6. M. Minsky: Computation – Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ, 1967.
7. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20 (2002), 295–305.
8. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
9. Gh. Păun: Membrane Computing – An Introduction. Springer, Berlin, 2002.
10. Y. Rogozhin, S. Verlan: On the rule complexity of universal tissue P systems. In *Pre-Proceedings of the 6th International Workshop on Membrane Computing, WMC6* (R. Freund, G. Lojka, M. Oswald, Gh. Păun, eds.), Vienna, July 2005, 510–516.
11. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Vol. I-III, Springer, Berlin, 1997.