Reversible P Systems with Symport/Antiport Rules

Taishin Y. Nishida

Department of Information Systems
Toyama Prefectural University
Kurokawa 5180, Imizu-shi, 939-0398 Toyama, Japan
nishida@pu-toyama.ac.jp

Summary. A deterministic computing system is reversible if every configuration of the system has just one previous configuration or it is an initial configuration. In this paper it is proved that for every reversible register machine there exists a reversible P system with symport/antiport rules such that the P system accepts the same set of integers as the register machine accepts. The result shows that the family of sets accepted by reversible P systems with symport/antiport rules is the family of recursively enumerable sets of integers.

1 Introduction

E. Schrödinger has pointed out, in his famous essay [8], that living organisms incorporate energy of low entropy, maintain systematic activity (including self-reproduction, which is a kind of information processing), and emit energy of high entropy. That is, life uses "negative entropy", to keep its structure, which inevitably generates heat.

On the other hand, it has been also shown that a reversible information processing system is physically possible [4]. Because such a system is reversible, computation is performed without entropy generation, without information loss, or equivalently, without heat generation. More specifically, an external force (e.g., mechanical, electrical, etc) proceeds a reversible computing system to the "forward" direction. The force does some work on the system and hence consumes some energy. The energy is stored in the reversible system and is restored from the system to the source of the external force during the reverse computation. Thus reversible computation is performed without energy dissipation or heat generation. This property, in turn, may help us to resolve the contemporary problem of huge heat emission in a large data processing centre.

There are two researches on reversible P systems [1, 2, 5]. In [5], simulation of the Fredkin gate is focused. Backward dynamics of a P system with object rewriting rules are considered in [1] by introducing dual P systems. The notion of dual P systems with determinism leads reversibility. In [2], reversible P systems with symport/antiport rules are considered. It is shown that such systems with one membrane and with control of priorities or inhibitors are universal. In this paper, we proceed studies on reversible P systems with symport/antiport rules and without any control on rules. The family of reversible P systems with 4 membranes is proved to be computationally universal. The result is shown by constructing a reversible P system which simulates a reversible register machine. Reversible register machines which are suitable for reversible P systems are explored in the next section, although a general study of them are already done by K. Morita [6].

2 Reversible register machine

Because nondeterministic change cannot be reversible, we consider deterministic systems only. First we introduce reversible register machine.

Let $M = (n, B, l_0, l_h, R)$ be a deterministic register machine, where n is the number of registers, B is the set of instruction labels, l_0 is the start label, l_h is the halt label, and R is the set of instructions. An instruction is one of the forms $l_i : (\mathrm{ADD}(r), l_j)$ or $l_i : (\mathrm{SUB}(r), l_j, l_k)$. The former adds 1 to register r and then executes the instruction labelled by l_j. The latter subtract 1 from register r if register r has a positive integer then executes the instruction labelled by l_j; or, if register r is 0, executes the instruction labelled by l_k. For every instruction label but the halt label, there is exactly one instruction. There are no instructions which are labelled by the halt label.

An n+1-tuple $(l, i_i, \ldots, i_n)$ is said to be a configuration of M if $l \in B$ and $i_j \in \mathbb{N}$ for every $1 \leq j \leq n$ where i_j is the number which is stored in register r_j and $\mathbb{N}$ is the set of nonnegative integers. An instruction $\iota \in R$ maps a configuration $(l, i_1, \ldots, i_n)$ to a configuration $(l', i'_1, \ldots, i'_n)$, denoted by $\iota((l, i_1, \ldots, i_n)) = (l', i'_1, \ldots, i'_n)$, if one of the following three conditions holds:

1. $\iota = l : (\mathrm{ADD}(r_j), l')$, $i'_j = i_j + 1$, and $i'_k = i_k$ for $k \neq j$.
2. $\iota = l : (\mathrm{SUB}(r_j), l', l_k)$, $i_j > 0$, $i'_j = i_j - 1$, and $i'_p = i_p$ for $p \neq j$.
3. $\iota = l : (\mathrm{SUB}(r_j), l_k, l')$, $i_j = 0$, and $i'_p = i_p$ for every p.

For two configurations c and c', if there is an instruction ι of M such that $\iota(c) = c'$, then c' is said to be directly derived from c and is denoted by $c \vdash_M c'$. As usual, the reflective and transitive closure of $\vdash_M$ is denoted by $\vdash_M^*$. We write $\vdash$ and $\vdash^*$ instead of $\vdash_M$ and $\vdash_M^*$ if M is understood.

A configuration $(l, i_1, \ldots, i_n)$ is called an initial (resp. halting) configuration if $l = l_0$ (resp. $l = l_h$). A sequence of configurations $c_0, c_1, \ldots, c_k$ where c_0 is an initial configuration is said to be a computation of M if for every $i = 0, 1, \ldots, k-1$ $c_i \vdash c_{i+1}$. A computation is said to be successful if the last configuration c_k is a halting configuration.

Let $M = (n, B, l_0, l_h, R)$ be a register machine. Let $c_0 = (l_0, i, 0, \ldots, 0)$ be an initial configuration of M where $i \in \mathbb{N}$. If there is a successful computation $c_0, c_1, \ldots, c_k$, then the input i is accepted by M. The set

$$N(M) = \{i \mid i \in \mathbb{N} \text{ is accepted by } M\}$$

is the set of integers which is accepted by M.

Definition 1 (reversible register machine). *A deterministic register machine M is* reversible *if every configuration of M which is reachable from an initial configuration has just one previous configuration or the configuration is an initial configuration.*

In order to have a deep insight into the meaning of the notion of a reversible register machine, let us consider a computation $c_0, c_1, \ldots, c_n$ of a non-reversible deterministic register machine M. Let us assume that configuration c in the computation is not reversible, that is there are two configurations c' and c'' such that $c' \neq c''$, $c' \vdash c$, and $c'' \vdash c$. If c' appears earlier than c'' in the sequence $c_0, c_1, \ldots, c_n$, then c'' appears after c, that is, $c \vdash^* c''$. This means that the computation is cyclic and hence cannot be successful. In other words, for every successful computation $c_0, \ldots, c_n$ of a non-reversible deterministic register machine, every configuration c has at most one configuration c' such that $c' \vdash c$. The "non-reversibility" emerges in the situation that two different initial configurations c_0 and c_0' derive the same configuration c. By keeping the inputs, which are the non-zero values of registers in the initial configuration, in all subsequent configurations, the situation may be avoided, that is, a non-reversible register machine may be converted to a reversible machine. But the next property must be took into account.

Property 1. Let M be a reversible register machine. Then M does not contain a SUB instructions of the type $l_i : \ (\mathrm{SUB}(r), l_i, l_k)$ which is executable from two different initial configurations and register r has non-zero initial value in at least one initial configuration.

Proof. The instruction $l_i : \ (\mathrm{SUB}(r), l_i, l_k)$ clears register r, in other words, loses information of the initial configuration. Reversible computation cannot contain such an instruction. □

We note that an instruction $\iota = l_i : \ (\mathrm{SUB}(r), l_i, l_k)$ in a non-reversible register machine can be removed by introducing a new register r' with initially 0. That is, the previous instruction of ι is modified to go directly to the instruction labelled by l_k and successive instructions of l_k which operate on register r are modified to operate on register r'.

A sequence of instructions $\iota_1, \iota_2, \ldots, \iota_p$ such that $\iota_1 = l_{i_1} : (\mathrm{SUB}(r), l_{j_1}, l_{k_1})$, the next instruction label of ι_p is l_{i_1}, and that $\iota_1, \iota_2, \ldots, \iota_p$ are executable in this order may cause the same situation as Property 1, i.e., register r may be cleared. In order to avoid losing information in register r, a new register r' which contains initially 0 and a new instruction $\iota_n = l_n : (\mathrm{ADD}(r'), l_{j_1})$ are introduced and ι_1 is modified to $\iota_1' = l_{i_1} : (\mathrm{SUB}(r), l_n, l_{k_1})$. Then information in register r is copied to register r' and is kept for the reverse computation.

We also note that an ADD instruction which adds an initially 0 register is treated specially. The reverse of an ADD instruction is a SUB instruction. If a reverse SUB instruction encounters 0 in the register, then it is the initial state and there are no previous configurations, that is, the reverse computation should halt at this point.

The next theorem follows from the definition of reversible register machines and Property 1 and its notes. Under slightly different notations, the same theorem is described in [6] with more detailed and sophisticated proof.

Theorem 1. *The family of sets accepted by reversible register machines is NRE, where NRE is the family of Turing computable sets of nonnegative integers.*

Now let us consider reverse instructions of a register machine. First, it should be noted that an instruction of a register machine is not reversible — a SUB instruction has two possible successors. By associating informations of register contents, it may be possible to make reverse transformation of configurations of a register machine. We modify, however, instructions to fit reverse operations, which corresponds to the transition functions of a reversible Turing machine [3].

Let G be the set of register names, let $O_p = \{-, 0, +\}$, and let $S = \{0, 1, *\}$. An instruction ι is an element in $(B \times G \times S) \times O_p \times (G \times S \times B)$ which has one of the forms:

$$(l_i, r, *) + (r, 1, l_j) \tag{1}$$

$$(l_i, r, 0)\ 0\ (r, 0, l_k) \tag{2}$$

$$(l_i, r, 1) - (r, *, l_j) \tag{3}$$

The left triplet shows the instruction label, register of the instruction, and the content of the register, where 0 means that the register is 0, 1 means that the register has a positive integer, and $*$ means that there may be both cases 0 and positive. The middle symbol, $+$, 0, or $-$, is the operation of the instruction, in which $+$ means that the register is added by one, 0 represents that the register is unchanged, and $-$ represents that the register is subtracted by one. The right triplet shows the register and its content after the operation and the label of the next instruction. An instruction of the form (1) is an add instruction. Instructions (2) and (3) form a subtract instruction. That is, an instruction of type (3) is accompanied by an instruction of type (2) which has the same instruction label at the first position in the left triplet. If register r is 0, then the instruction of type (2) is executed. Otherwise, type (3) is executed. We note that a single instruction of type (2) may be used as a goto instruction.

Reverse instructions are summarised by Table 1.

The next example shows a construction of a reverse machine from a reversible register machine.

Example. Let us consider a function $f(x, y) = x \dot{-} y$ which is defined by

$$x \dot{-} y = \begin{cases} x - y & \text{if } x > y \\ 0 & \text{if } x \leq y \end{cases}.$$

Table 1. Reverse instructions

forward	reverse	notes
$(l_i, r, *) + (r, 1, l_j)$	$(l_j, r, 1) - (r, *, l_i)$	If register r is initially 0, then instruction $(l_j, r, 0)\ 0\ (r, 0, l_0)$ is added where l_0 is the halt label of the reverse machine.
$(l_i, r, 0)\ 0\ (r, 0, l_k)$	$(l_k, r, 0)\ 0\ (r, 0, l_i)$	
$(l_i, r, 1) - (r, *, l_j)$	$(l_j, r, *) + (r, 1, l_i)$	

Because $f(x, y)$ is not logically reversible, we modify it to a function f' which maps a triple of nonnegative integers $(x, y, 0)$ to a triple $(f(x, y), f(y, x), \min(x, y))$. The register machine with 3 registers and instructions

$$\begin{gathered}(1, r_2, 1) - (r_2, *, 2)\\(1, r_2, 0)\ 0\ (r_2, 0, l_h)\\(2, r_1, 1) - (r_1, *, 3)\\(2, r_1, 0)\ 0\ (r_1, 0, 4)\\(3, r_3, *) + (r_3, 1, 1)\\(4, r_2, *) + (r_2, 1, l_h)\end{gathered}$$

computes the function f'. Registers have initial values $r_1 = x$, $r_2 = y$, and $r_3 = 0$.

The reverse instructions are

$$\begin{gathered}(2, r_2, *) + (r_2, 1, 1)\\(l_h, r_2, 0)\ 0\ (r_2, 0, 1)\\(3, r_1, *) + (r_1, 1, 2)\\(4, r_1, 0)\ 0\ (r_1, 0, 2)\\(1, r_3, 1) - (r_3, *, 3)\\(1, r_3, 0)\ 0\ (r_3, 0, l_0)\\(l_h, r_2, 1) - (r_2, *, 4)\end{gathered}$$

Figure 1 shows the flows of instructions of the forward machine (left) and the reverse machine (right).

3 Reversible P system with symport/antiport rules

In this section a reversible P system with symport/antiport rules is defined and the family of such systems is proved to be computationally universal. We first briefly summarise the notion of a P system with symport/antiport rules and with accepting mode; for details the reader is referred to [7].

A P system with symport/antiport rules of degree $n \geq 1$ is a construct of the form

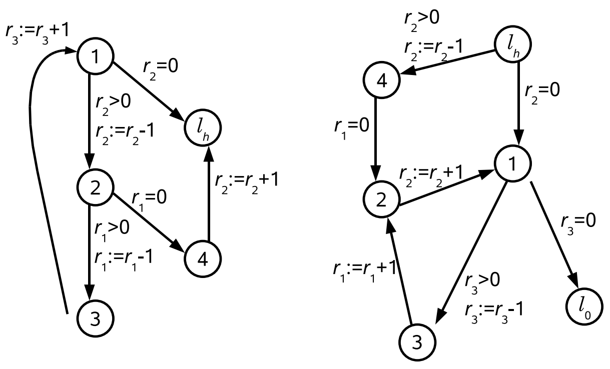

Fig. 1. Flow graphs of a register machine (left) and its reverse machine (right).

$$\Pi = (O, \mu, w_1, \ldots, w_n, E, R_1, \ldots, R_n, i_I),$$

where:

1. O is the alphabet of objects.
2. μ is a membrane structure consisting of n membranes. Membranes are injectively labelled with $1, \ldots, n$.
3. $w_1, \ldots, w_n$ are strings of objects which represent the multiset over O initially associated with the regions $1, \ldots, n$ of μ.
4. $E \subseteq O$ is the set of objects which are supposed to appear in the environment in arbitrary many copies.
5. $R_1, \ldots, R_n$ are finite sets of symport/antiport rules over O associated with the membranes labelled $1, \ldots, n$.
6. i_I is the input region.

Rules are applied in the usual nondeterministic maximally parallel manner. An input of a P system with symport/antiport rules Π is the multiplicity of objects initially associated with the input region, i.e., the multiplicity of w_{i_I}. The system Π accepts its input if and only if its computation halts. The set of numbers accepted by Π is the set of all inputs which are accepted by Π and is denoted by $N_{acc}(\Pi)$.

Definition 2 (reversible P system). *A P system with symport/antiport rules Π is said to be* reversible *if Π is deterministic and every configuration C of Π which is reachable from an initial configuration satisfies that C is an initial configuration or there is just one configuration C' such that C' is changed to C by Π.*

In [2], a notion of a strongly reversible P system, in which every (including non-reachable) configuration has at most one previous configuration, is defined in addition to the above reversible P system. But dynamics of configurations of a strongly reversible P system without inhibitors and without priorities in rules are extremely limited (Theorem 2 of [2]). Therefore, in this paper, we consider reversible P systems which are defined by Definition 2.

The next theorem is the main result of this paper.

Theorem 2. *For a reversible register machine $M = (n, B, l_0, l_h, R)$, there exists a P system with symport/antiport rules Π of degree 4 such that Π is reversible and that $N_{acc}(\Pi) = N(M)$.*

Proof. The P system

$$\Pi = (O, [_1\, [_2\, [_3\,]_3\,]_2\, [_4\,]_4\,]_1, w_1, w_2, w_3, w_4, E, R_1, R_2, R_3, R_4, 1)$$

is constructed by:

$$O = E = \{l, l', l'', l''', l^{iv}, l^v \mid l \in B\} \cup \{a_r \mid 1 \le r \le n\} \cup \{t_0, t_1, t_2\},$$

$$w_1 = a_1^k \quad \text{for an initial configuration } (l_0, k, 0, \ldots, 0) \text{ of } M,$$

$$w_2 = t_0,\ w_3 = t_1,\ w_4 = l_0'''t_2$$

$$R_2 = \{(t_0, out;\ t_1, in), (t_1, out;\ t_2, in), (t_2, out;\ t_0, in)\},$$

$$R_3 = \{(t_0, out;\ t_2, in), (t_1, out;\ t_0, in), (t_2, out;\ t_1, in)\},$$

$$R_4 = \{(l_0'''t_2, out)\}$$

and R_1 consists of the rules

$$(l_i, out;\ l_j' a_r, in), (l_j', out;\ l_j'', in), (l_j'', out;\ l_j''', in), (l_j''', out;\ l_j^{iv}, in),$$

$$(l_j^{iv}, out;\ l_j^v, in), (l_j^v, out;\ l_j, in)$$

for an add instruction $(l_i, r, *) + (r, 1, l_j)$ and the rules

$$(l_i a_r, out;\ l_j', in), (l_j', out;\ l_j'', in), (l_j'', out;\ l_j''', in), (l_j''', out;\ l_j^{iv}, in),$$

$$(l_j^{iv}, out;\ l_j^v, in), (l_j^v, out;\ l_j, in)$$

$$(l_i t_2, out;\ l_k''' t_2, in), (l_k''', out;\ l_k^{iv}, in), (l_k^{iv}, out;\ l_k^v, in), (l_k^v, out;\ l_k, in)$$

for subtract instructions $(l_i, r, +) - (r, *, l_j)$ and $(l_i, r, 0)\, 0\, (r, 0, l_k)$. R_1 also contains the rules

$$(l_0''', out;\ l_0^{iv}, in), (l_0^{iv}, out;\ l_0^v, in), (l_0^v, out;\ l_0, in).$$

The rule from R_4 is applied at an initial configuration only and brings $l_0'''t_2$ to region 1. At the same time, objects in regions 2 and 3 are exchanged by the rule $(t_1, out;\ t_0, in)$ from R_3. The successive computations are illustrated in the next table.

	rules used from R_i and objects in each region					
step	R_1	region 1	R_2	region 2	R_3	region 3
1	$(l_0''', out;\ l_0^{iv}, in)$	$a_1^k l_0''' t_2$	$(t_1, out;\ t_2, in)$	t_1	—	t_0
2	$(l_0^{iv}, out;\ l_0^{v}, in)$	$a_1^k l_0^{iv} t_1$	—	t_2	$(t_0, out;\ t_2, in)$	t_0
3	$(l_0^{v}, out;\ l_0, in)$	$a_1^k l_0^{v} t_1$	$(t_0, out;\ t_1, in)$	t_0	—	t_2
4		$a_1^k l_0 t_0$		t_1		t_2

From the last low of the above table, Π starts to simulate M.

An add instruction $(l_i, r, *) + (r, 1, l_j)$ is simulated by Π as follows:

	rules used from R_i and objects in each region					
step	R_1	region 1	R_2	region 2	R_3	region 3
1	$(l_i, out;\ l_j' a_r, in)$	$a_r^p l_i t_0$	—	t_1	$(t_2, out;\ t_1, in)$	t_2
2	$(l_j', out;\ l_j'', in)$	$a_r^{p+1} l_j' t_0$	$(t_2, out;\ t_0, in)$	t_2	—	t_1
3	$(l_j'', out;\ l_j''', in)$	$a_r^{p+1} l_j'' t_2$	—	t_0	$(t_1, out;\ t_0, in)$	t_1
4	$(l_j''', out;\ l_j^{iv}, in)$	$a_r^{p+1} l_j''' t_2$	$(t_1, out;\ t_2, in)$	t_1	—	t_0
5	$(l_j^{iv}, out;\ l_j^{v}, in)$	$a_r^{p+1} l_j^{iv} t_1$	—	t_2	$(t_0, out;\ t_2, in)$	t_0
6	$(l_j^{v}, out;\ l_j, in)$	$a_r^{p+1} l_j^{v} t_1$	$(t_0, out;\ t_1, in)$	t_0	—	t_2
7	next rule	$a_r^{p+1} l_j t_0$	—	t_1	$(t_2, out;\ t_1, in)$	t_2

where $p \geq 0$.

Subtract instructions $(l_i, r, 1) - (r, *, l_j)$ and $(l_i, r, 0)\, 0\, (r, 0, l_k)$ are simulated by Π as follows:

	rules used from R_i and objects in each region					
step	R_1	region 1	R_2	region 2	R_3	region 3
1	$(l_i a_r, out;\ l_j', in)$	$a_r^p l_i t_0$	—	t_1	$(t_2, out;\ t_1, in)$	t_2
2	$(l_j', out;\ l_j'', in)$	$a_r^{p-1} l_j' t_0$	$(t_2, out;\ t_0, in)$	t_2	—	t_1
3	the following steps are similar to the add case					

where $p \geq 1$ and

	rules used from R_i and objects in each region					
step	R_1	region 1	R_2	region 2	R_3	region 3
1	—	$l_i t_0$	—	t_1	$(t_2, out;\ t_1, in)$	t_2
2	—	$l_i t_0$	$(t_2, out;\ t_0, in)$	t_2	—	t_1 .
3	$(l_i t_2, out;\ l_k''' t_2, in)$	$l_i t_2$	—	t_0	$(t_1, out;\ t_0, in)$	t_1
4	$(l_k''', out;\ l_k^{iv}, in)$	$l_k''' t_2$	$(t_1, out;\ t_2, in)$	t_1	—	t_0
5	the following steps are similar to the add case					

Therefore Π accepts its input if and only if M accepts its input, that is, $N_{acc}(\Pi) = N(M)$.

By the construction of rules, Π is deterministic and every configuration but the initial configuration has just one previous configuration. Thus Π is reversible. □

Corollary 1. *The family of sets accepted by reversible P systems with symport/antiport rules is NRE.*

Remark. Theorem 2 and Corollary 1 give a negative answer to Conjecture 1 of [2]. The P system which is constructed in the proof of Theorem 2 uses nested membranes $[_1[_2[_3\ ]_3]_2]_1$ and three timing objects t_0, t_1, and t_2 in order to do a zero-test by try-and-wait-then-check strategy. It is a future work to reduce the numbers of membranes and objects in Theorem 2.

References

1. O. Agrigoroaiei and G. Ciobanu, Dual P systems, in: D. Corne et al. (Eds.) *Membrane Computing - 9th International Workshop*, LNCS 5391 (Springer, Berlin, 2009) 95–107.
2. A. Alhazov and K. Morita, A short note on reversibility in P system, in: Proc. of 7th Brainstorming Week on Membrane Computing, Sevilla, February 2009, Fenix Editora, Sevilla, 23–28.
3. C. H. Bennett, Logical reversibility of computation, *IBM Journal of Research and Development*, **17** (1973) 525–532.
4. R. Landauer, Irreversibility and heat generation in the computing process, *IBM Journal of Research and Development*, **5** (1961) 183–191.
5. A. Leporati, C. Zandron, and G. Mauri, Reversible P systems to simulate Fredkin circuits, *Fundamenta Informaticae* **74** (2006) 529–548.
6. K. Morita, Universality of a reversible two-counter machine, *Theoretical Computer Science* **168** (1996) 303–320.
7. Gh. Păun, *Membrane Computing* (Springer, Berlin Heidelberg, 2002).
8. E. Schrödinger, *What Is Life?*, (Cambridge University Press, Cambridge, 1944).