P Automata with Controlled Use of Minimal Communication Rules

Rudolf Freund[1], Marian Kogler[1], and Sergey Verlan[2]

[1] Institute for Computer Languages, Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
Email: {rudi,marian}@emcc.at

[2] LACL, Département Informatique
UFR Sciences et Technologie, Université Paris XII
61, av. Général de Gaulle, 94010 Créteil, France
Email: verlan@univ-paris12.fr

Abstract

In this paper, we introduce new variants of transition modes for P systems by adding rule control to the conventional transition modes used so far. This formalism of rule control allows us to specify which rules have to be applied together or not. We show that computational completeness can be obtained by using either minimal symport rules or minimal antiport rules together with uniport rules being applied in the maximally parallel transition mode with rule control.

1 Introduction

In this paper we consider some new special variants of P automata – a variation of P systems introduced in [3] which act as acceptors rather than producers; a similar concept of analyzing P systems was considered in [6]. A multiset is accepted by a P automaton if and only if the automaton halts.

In the original model of P systems introduced as membrane systems by Gh. Păun (see [4], [11]), the objects evolve within a hierarchical membrane structure in the maximally parallel transition mode. Here we extend this maximally parallel transition mode by imposing an additional condition on the applicable multisets of rules, i.e., we specify which rules have to be used together or not. We show some computational completeness results for P automata with only one membrane working in the maximally parallel transition mode with rule control using either minimal symport rules or minimal antiport rules together with

uniport rules. The same results also hold for the minimally parallel transition mode (see [1]) as well as for the k-bounded minimally parallel transition mode (see [8]) with rule control.

The rest of this paper is organized as follows: In the second section, we recall well-known definitions and notions. In the next section, we explain our model of P automata working in different conventional transition modes together with rule control and give an illustrative example. In the fourth section, by simulating deterministic register machines, we establish computational completeness results for P automata with only one membrane working in the maximally parallel transition mode with rule control using either minimal symport rules or minimal antiport rules together with uniport rules. A summary of the obtained results and an outlook to future research conclude the paper.

2 Preliminaries

We recall some of the notions and the notations we use (see [15] for elements of formal language theory) as in [8].

Let V be a (finite) alphabet; then V^* is the set of all strings (a language) over V, and $V^+ = V^* - \{\lambda\}$ where λ denotes the empty string. By $RE\,(RE\,(T))$ we denote the family of recursively enumerable languages (over the alphabet T). For any family of string languages F, PsF denotes the family of Parikh sets of languages from F and NF the family of Parikh sets of languages from F over a one-letter alphabet. By $\mathbb{N}$ we denote the set of all non-negative integers, by $\mathbb{N}^k$ the set of all vectors of non-negative integers of size k. In the following, we will not distinguish between NRE, which coincides with $PsRE\,(\{a\})$, and $RE\,(\{a\})$.

Let V be a (finite) set, $V = \{a_1, ..., a_k\}$. A *finite multiset* M over V is a mapping $M : V \longrightarrow \mathbb{N}$, i.e., for each $a \in V$, $M\,(a)$ specifies the number of occurrences of a in M. The size of the multiset M is $|M| = \sum_{a \in V} M\,(a)$. A multiset M over V can also be represented by any string x that contains exactly $M\,(a_i)$ symbols a_i for all $1 \leq i \leq k$, e.g., by $a_1^{M(a_1)}...a_k^{M(a_k)}$, or else by the set $\left\{a_i^{M(a_i)} \mid 1 \leq i \leq k\right\}$. The set of all finite multisets over the set V is denoted by $\langle V, \mathbb{N}\rangle$.

Throughout the rest of the paper, we will not distinguish between a multiset from $\langle V, \mathbb{N}\rangle$ and its representation by a string over V containing the corresponding number of each symbol. Moreover, when we speak of a partitioning of a set R into a set $\{R_i \mid 1 \leq i \leq h\}$ of (non-empty) subsets of R, i.e., $R_i \subseteq R$, $1 \leq i \leq h$, the R_i are not necessarily disjoint.

A *deterministic register machine* is a construct $M = (n, B, l_0, l_h, I)$, where n is the number of registers, B is a set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to `HALT` only), and I is a set of instructions of the following forms:

- l_i : (ADD(r), l_j) add 1 to register r, and then go to the instruction labeled by l_j;
- l_i : (SUB(r), l_j, l_k) if register r is non-empty (non-zero), then subtract 1 from it and go to the instruction labeled by l_j, otherwise go to the instruction labeled by l_k;
- l_h : HALT the halt instruction.

A register machine M accepts a set of (vectors of) natural numbers in the following way: start with the instruction labeled by l_0, with the first registers containing the input as well as all other registers being empty, and proceed to apply instructions as indicated by the labels and by the contents of the registers. If we reach the HALT instruction, then the input number (vector) is accepted. It is known (e.g., see [9]) that in this way we can accept all recursively enumerable sets of (vectors of) natural numbers.

3 P Automata with Communication Rules and Rule Control

For an introduction to the area of membrane computing we refer the interested reader to the monograph [12], the current state of the art can be seen in the web [16].

We first consider P automata using communication rules similar to the P automata with communication rules as described in [3] and the analyzing P systems introduced in [6], but utilize a different notation for the rules.

Definition 3.1. A *communication P automaton* is a construct

$$\Pi = (O, T, \mu, E, w_0, w_1, ..., w_d, i_0, R)$$

where

1. O is a finite alphabet of objects;
2. $T \subseteq O$ is the alphabet of terminal objects;
3. μ is a membrane structure of d membranes with labels i, $1 \leq i \leq d$, written as $[_i$ and $]_i$; the skin membrane always has the index 1; the environment is indicated by 0;
4. E is the alphabet of objects occurring in an arbitrary number in the environment;
5. $w_0, w_1, ..., w_d$ are finite multisets of objects over O representing the initial contents of the environment (w_0 only contains objects from $O - E$) as well as the membranes with labels $1, ..., d$;
6. i_0, $1 \leq i_0 \leq d$, is the input membrane;

7. R is a set of communication rules.

As special forms of communication rules, we consider symport rules and antiport rules:

Definition 3.2. *Symport* rules in R are of the form

$$x[_i \rightarrow [_i x$$

meaning that the multiset x from outside membrane i is moved into the region inside membrane i, or

$$[_i x \rightarrow x[_i$$

meaning that the multiset x from inside membrane i is moved into the region surrounding membrane i, with $x \in O^+$ and $1 \leq i \leq d$. An additional condition has to be added for symport rules that bring objects into a membrane from the environment – in such cases, at least one symbol from x in $x[_1 \rightarrow [_1 x$ has to be from $O - E$. The weight of a symport rule is defined as $|x|$. If we consider symport rules of any weight, we write sym_*; if we only consider symport rules with weight $\leq n$, we write sym_n; sym_2 rules are also called *minimal symport* rules; finally, sym_1 rules are called *uniport* rules.

Definition 3.3. *Antiport* rules in R are of the form

$$x[_i y \rightarrow y[_i x$$

with $x, y \in O^+$ and $1 \leq i \leq d$, meaning that the multiset x from outside membrane i is exchanged with the multiset y in the region inside membrane i. The weight of an antiport rule is defined as $max(|x|, |y|)$. If we consider antiport rules of any weight, we write $anti_*$; if we only consider antiport rules with weights $\leq n$, we write $anti_n$; $anti_1$ rules are also called *minimal antiport* rules.

A *configuration* C of Π is an $(n+1)$-tuple of multisets over O $(u_0, u_1 \ldots, u_n)$; the *initial configuration* of Π, C_0, is described by w_0,w_1, ..., w_d, i.e., $C_0 = (w_0, w_1, \ldots, w_d)$. The set of all multisets of rules from R *applicable* to C is denoted by $Appl\,(\Pi, C)$.

To narrow the possible set of multisets of rules that can be applied to a given configuration, we may apply different *transition modes* (for a formal definition of transition modes see [7]); for example, the *maximally parallel* transition mode was already introduced in the seeding paper for the P systems area, [11]. For the transition mode ϑ, the selection of multisets of rules applicable to a configuration C is denoted by $Appl\,(\Pi, C, \vartheta)$.

In the *maximally parallel* transition mode we only select multisets of rules R' that are not extensible, i.e., there is no other multiset of rules $R'' \supsetneqq R'$ applicable to C.

Definition 3.4. For the *maximally parallel* transition mode (max), we define

$$\begin{aligned} Appl\,(\Pi, C, max) \quad = \quad & \{R' \mid R' \in Appl\,(\Pi, C) \text{ and there is} \\ & \text{no } R'' \in Appl\,(\Pi, C) \text{ with } R'' \supsetneqq R'\}. \end{aligned}$$

For the *minimally parallel* mode, we need an additional feature for the set of rules R, i.e., we consider a partitioning of R into (not necessarily disjoint) subsets R_1 to R_h. Usually, this partitioning of R may coincide with a specific assignment of the rules to the membranes. There are several possible interpretations of this minimally parallel mode which in an informal way can be described as applying multisets such that from every set R_j, $1 \leq j \leq h$, at least one rule – if possible – has to be used (e.g., see [1] and [7]):

Definition 3.5. For the *minimally parallel* transition mode (min), we define

$$\begin{aligned} \mathit{Appl}\,(\Pi, C, min) \quad = \quad & \{R' \mid R' \in \mathit{Appl}\,(\Pi, C) \text{ and} \\ & \text{there is no } R'' \in \mathit{Appl}\,(\Pi, C) \\ & \text{with } R'' \supsetneqq R',\ (R'' - R') \cap R_j \neq \emptyset \\ & \text{and } R' \cap R_j = \emptyset \text{ for some } j,\ 1 \leq j \leq h\}\,. \end{aligned}$$

We also consider a restricted variant of the minimally parallel mode allowing only a bounded number of at most k rules to be taken from each set R_j, $1 \leq j \leq h$, of the partitioning into a multiset of rules applicable in the minimally parallel mode (see [8]):

Definition 3.6. For the *k-restricted minimally parallel* transition mode (min_k), we define

$$\begin{aligned} \mathit{Appl}\,(\Pi, C, min_k) \quad = \quad & \{R' \mid R' \in \mathit{Appl}\,(\Pi, C, min) \text{ and} \\ & |R' \cap R_j| \leq k \text{ for all } j,\ 1 \leq j \leq h\}\,. \end{aligned}$$

We finally introduce an additional control mechanism on the applicability of rules within a multiset of rules from R operating as an "overlay" on the transition modes.

Definition 3.7. A *communication P automaton with rule control* is a construct

$$\Pi' = (O, T, \mu, E, w_0, w_1, ..., w_d, i_0, R, R'_1, ..., R'_m, K)$$

where

1. $\Pi = (O, T, \mu, E, w_0, w_1, ..., w_d, i_0, R)$ is a communication P automaton with $O, T, \mu, E, w_0, w_1, ..., w_d, i_0$, and R being defined as before;
2. $R'_1, ..., R'_m$ is a partitioning of R into (non-empty, but not necessarily disjoint) subsets;
3. $K \subseteq \{0,1\}^m$ is a set of control vectors controlling the applicability of multisets of rules from R.

We now are able to define the multisets of rules from R applicable in Π' to a configuration C under a given transition mode ϑ using the control sets from K:

Definition 3.8. For some given transition mode ϑ,

$$\begin{aligned} Appl\,(\Pi', C, \vartheta) \;=\; & \{R' \mid R' \in Appl\,(\Pi, C, \vartheta) \text{ and} \\ & \text{there exists a vector } v \in K \text{ such that} \\ & \text{for all } j \text{ with } 1 \leq j \leq m \text{ it holds that} \\ & v\,(j) = 1 \text{ implies } R' \cap R_j \neq \emptyset \text{ and} \\ & v\,(j) = 0 \text{ implies } R' \cap R_j = \emptyset \;\}\,. \end{aligned}$$

The components with value 1 in a control vector v from K specify those rule sets from which at least one rule has to be taken into a multiset R' of rules from R to be applicable to a configuration C in Π', whereas from the components with value 0 in the control vector v no rule is allowed to be taken into R'.

A *computation* in Π' checking for the acceptance of (a multiset over O) w consists of a sequence of transitions starting from the initial configuration $C'_0 = (w'_0, w'_1, \ldots, w'_d)$ with $w'_i = w_i$ for $0 \leq i \leq m$ and $i \neq i_0$ as well as $w'_i = w_i + w$ for $i = i_0$; such a computation is called *successful* if and only if it halts, i.e., it has reached a configuration C_f to which no multiset is applicable anymore, $Appl\,(\Pi', C_f, \vartheta) = \emptyset$. A multiset w is accepted if and only if there exists a successful computation of Π' on C'_0.

The idea of this additional constraint for the applicability of multiset of rules imposed on a transition mode in some sense resembles the idea of prescribed teams of rules in grammar systems (e.g., see [13]) where the allowed combinations of rules taken from given rule sets are specified.

Example 3.1. Let

$$\Pi' = (O, T, [_1\ ]_1, E, w_0, w_1, 1, R, R_1, R_2, R_3, R_4, R_5, K)$$

be a communication P automaton with rule control with

$$\begin{aligned} O &= \{a, b, p_1, p_2\}\,, \\ T &= \{a\}\,, \\ E &= \{b\}\,, \\ w_0 &= \{\}\,, \\ w_1 &= \{p_1 p_2\}\,, \\ R &= R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5; \\ R_1 &= \{[_1 p_1 a \rightarrow p_1 a [_1\}\,, \\ R_2 &= \{[_1 p_2 a \rightarrow p_2 a [_1\}\,, \\ R_3 &= \{p_1 b [_1 \rightarrow [_1 p_1 b\}\,, \\ R_4 &= \{p_2 [_1 \rightarrow [_1 p_2\}\,, \\ R_5 &= \{p_1 [_1 p_2 \rightarrow p_2 [_1 p_1, p_2 [_1 p_1 \rightarrow p_1 [_1 p_2\}\,, \\ K &= \{(1,1,0,0,0)\,, (1,0,0,0,0)\,, (0,0,1,1,0)\,, (0,0,0,0,1)\}\,. \end{aligned}$$

In the maximally parallel mode, this automaton starts with the multiset $p_1 p_2 a^n$ for some $n \geq 0$ and first tries to apply the rules in R_1 and R_2 in parallel, thereby fulfilling the control conditions given by the control vector $(1, 1, 0, 0, 0)$, and exporting two symbols a from the skin membrane into the environment, if

at least two symbols a are available in the skin membrane; if only one symbol a is available, then only the first rule $[_1 p_1 a \to p_1 a [_1$ is applied thereby fulfilling the condition given by the control vector $(1,0,0,0,0)$. If both p_1 and p_2 have been moved out, then the rules in R_3 and R_4 are executed at the same time according to the control vector $(0,0,1,1,0)$; these two steps in fact have replaced two symbols a by one symbol b. The P automaton Π' repeats this process until at most one symbol a is left in the skin membrane. If no a is left, i.e., n has been an even number, the automaton terminates with having $p_1 p_2 b^{n/2}$ in the skin membrane. If one a is left, i.e., n has been an odd number, then only the rule in R_1 is executed, leading to a situation where p_1 is outside and p_2 is inside. In this case, the automaton never terminates (as the two rules $p_1 [_1 p_2 \to p_2 [_1 p_1$ and $p_2 [_1 p_1 \to p_1 [_1 p_2$ comprise an infinite loop). The automaton therefore exactly accepts a^{2m} for any $m \geq 0$.

Remark 3.1. We notice that the parallel application of the two rules $p_1 b [_1 \to [_1 p_1 b$ from R_3 and $p_2 [_1 \to [_1 p_2$ from R_4 in sum has the same effect as the sym_3 rule $p_1 p_2 b [_1 \to [_1 p_1 p_2 b$; in that way, it is possible to simulate symport rules of weight 3 using a combination of symport rules of weight 2 and 1 acting in parallel, i.e., with a vector in K that allows rules from the sets these rules come from to be applied in parallel. More general, consider a symport rule $abc [_i \to [_i abc$, where $a, c \in O - E$ are present in only one copy in the whole system and $b \in O$; then such a rule $abc [_i \to [_i abc$ can be simulated by a synchronized application of the two rules $ab [_i \to [_i ab$ and $c [_i \to [_i c$, which can be obtained by taking each of these two rules into different partitions and adding a vector to K that enables these two partitions. In a similar way, symport rules of any weight k can be simulated (provided that there are at least $k/2$ symbols present in only one copy).

As for symport rules, it is possible to simulate antiport rules of weight 2 using a combination of minimal antiport rules and uniport rules. More exactly, consider an antiport rule $a [_i bc \to bc [_i a$, where $a, c \in O - E$ are present in only one copy in the whole system and $b \in O$. The single application of such a rule can be simulated by a synchronized application of the two rules $a [_i b \to b [_i a$ and $[_i c \to c [_i$, which can be obtained by taking each of these two rules into different partitions and adding a vector to K that enables these two partitions. In a similar way antiport rules of any weight k can be simulated (provided that there are at least $k/2$ symbols present in one copy).

Definition 3.9. For some given transition mode ϑ, by

$$NO_l P_d K_m(\vartheta)\,[\text{rule types}] \quad (PsO_l P_d K_m(\vartheta)\,[\text{rule types}])$$

we define the sets of (vectors of) natural numbers accepted by communication P automata with rule control working in the transition mode ϑ in d membranes using l objects and a partitioning with m rule sets, allowing rules of the types specified in [rule types]; if any of the numbers d, m, l are unbounded, we write $*$ instead.

4 Computational Completeness

By simulating deterministic register machines, in this section we show that communication P automata with rule control using only minimal symport rules (sym_2 rules) or minimal antiport rules ($anti_1$ rules) together with uniport rules (sym_1 rules) in only one membrane are computationally complete.

4.1 Minimal Symport Rules

We first show that communication P automata with rule control using only minimal symport rules (sym_2 rules) can accept any recursively enumerable set of (vectors of) natural numbers.

Theorem 4.1. *For $X \in \{N, Ps\}$,*

$$XO_*P_1K_*\,(max)\,[sym_2] = XRE.$$

Proof. Let $M = (n, B, p_0, p_h, I)$ be a register machine with n registers accepting $L \in PsRE\,(T)$; then we construct a communication P automaton with rule control working with minimal symport rules in the maximally parallel transition mode

$$\Pi' = (O, T, [_1\ ]_1, E, w_0, w_1, 1, R, R'_1, ..., R'_m, K)$$

which accepts L in the following way:

1. $\begin{aligned} O &= \{a_i \mid 1 \le i \le n\} \\ &\cup \{p_i, p'_i, p''_i \mid p_i \in B, p_i : (\texttt{ADD}(r), p_j) \in I\} \\ &\cup \{p_i, p'_i, p''_i, p'''_i \mid p_i \in B, p_i : (\texttt{SUB}(r), p_j, p_k) \in I\} \\ &\cup \{p_h \mid p_h : \texttt{HALT} \in I\}; \end{aligned}$

2. $T = \{a_i \mid 1 \le i \le k\};$

3. $E = \{a_i \mid 1 \le i \le n\};$

4. $\begin{aligned} w_0 &= \{p_i \mid p_i \in B\} - \{p_0\}; \\ w_1 &= \{p_0\} \cup \{p'_i, p''_i \mid p_i \in B, p_i : (\texttt{ADD}(r), p_j) \in I\} \\ &\cup \{p'_i, p''_i, p'''_i \mid p_i \in B, p_i : (\texttt{SUB}(r), p_j, p_k) \in I\}; \end{aligned}$

5. the set of rules R consists of all the rules specified in the following which each of them represents a partition (containing one single rule) of the partitioning of R into sets of rules $R'_1, ..., R'_m$; instead of specifying K we specify which rules have to be used together, implicitly thereby assuming that all other rules are not allowed to be used at the same moment:

   - for $p_i \in B$, $p_i : (\texttt{ADD}(r), p_j) \in I$ we first take
     $[_1 p_i \to p_i[_1$ and $[_1 p'_i p''_i \to p'_i p''_i[_1$;
     these two rules applied in parallel prepare the simulation of the ADD instruction: as a_r is in E, a companion symbol is needed in order not to violate the additional condition for symport rules that import symbols from the environment;

- the simulation of the ADD instruction then is accomplished by applying in parallel the rules
  $p'_i p_j [_1 \rightarrow [_1 p'_i p_j$ and $p''_i a_r [_1 \rightarrow [_1 p''_i a_r$,
  i.e., the companion symbols p'_i, p''_i have returned into the skin membrane and an additional symbol a_r (the number of symbols a_r in the skin membrane represents the contents of register r) as well as the symbol p_j for the next instruction have been imported;
- for $p_i \in B$, $p_i : (\mathtt{SUB}(r), p_j, p_k) \in I$ we first take the two rules
  $[_1 p''_i p_i \rightarrow p''_i p_i [_1$ and $[_1 p'_i a_r \rightarrow p'_i a_r [_1$
  with the second rule being applicable if and only if the current contents of register r is not zero; after the execution of the first of these two rules, the instruction symbol p_i and the helper symbol p''_i are in the environment; if the register is non-empty, p'_i is in the environment, too, having been removed from the membrane together with one symbol a_r (which corresponds to having decremented register r); on the other hand, if the current contents of register r is zero, then only the first rule is applicable; hence, we have to include two vectors in K for these two cases, one of them enabling both rules and another one enabling the first rule alone;
- if the register has been non-empty, in which case p'_i has been moved into the environment, the simulation of the SUB instruction then is accomplished by applying in parallel the rules
  $p'_i p_j [_1 \rightarrow [_1 p'_i p_j$ and $p''_i [_1 \rightarrow [_1 p''_i$,
  which sends back the helper symbols p'_i, p''_i into the skin membrane together with the label of the next instruction p_j;
- if the register r has been empty, only the rule $[_1 p''_i p_i \rightarrow p''_i p_i [_1$ has been applied, whereas p'_i has remained in the skin membrane and then the two rules
  $p'''_i [_1 \rightarrow [_1 p''_i$ and $[_1 p'_i p'''_i \rightarrow p'_i p'''_i [_1$
  are enabled to be applied in parallel by a suitable vector in K, thereby bringing back the helper symbol p''_i into the skin membrane first; afterwards,
- the two rules
  $p'_i p_k [_1 \rightarrow [_1 p'_i p_k$ and $p'''_i [_1 \rightarrow [_1 p'''_i$
  are applied in parallel in order to bring back the helper symbols p'_i, p'''_i together with the symbol p_k for the next instruction.

When the P automaton Π' reaches the final label p_h (for $p_h : \mathtt{HALT} \in I$), the computation terminates and, by definition, the input is accepted. On the other hand, if the register machine never reaches the final label, the corresponding computation in Π' never terminates, too, i.e., the input is not accepted. □

We remark that the P automaton Π' is an analyzing (accepting) P system and not a generating P system, hence the "garbage" (the helper symbols

p'_i, p''_i, p'''_i) need not be removed from the skin membrane. Moreover, we could also interpret all pairs of rules described above as prescribed teams of size 2 (e.g., see [13]) with the extra condition of maximality; actually, except for the appearance checking executed by the pair

$$[_1 p''_i p_i \to p''_i p_i [_1 \text{ and } [_1 p'_i a_r \to p'_i a_r [_1$$

where eventually only the rule $[_1 p''_i p_i \to p''_i p_i [_1$ alone can be applied, in all other cases exactly two rules have to be applied in parallel.

4.2 Minimal Antiport Rules Together with Uniport Rules

We use a similar proof technique as in the preceding proof to show that minimal antiport rules together with uniport rules allow for computational completeness, too.

Theorem 4.2. *For $X \in \{N, Ps\}$,*

$$XO_* P_1 K_* \, (max) \, [anti_1, sym_1] = XRE.$$

Proof. Let $M = (n, B, p_0, p_h, I)$ be a register machine with n registers accepting $L \in PsRE\,(T)$; then we construct a communication P automaton with rule control working with minimal antiport rules and uniport rules in the maximally parallel transition mode

$$\Pi' = (O, T, [_1\ ]_1, E, w_0, w_1, 1, R, R'_1, ..., R'_m, K)$$

which accepts L in the following way:

1. $\begin{aligned} O &= \{a_i \mid 1 \le i \le n\} \\ &\cup \{p_i \mid p_i \in B, p_i : (\texttt{ADD}(r), p_j) \in I\} \\ &\cup \{p_i, p'_i, p''_i, p'''_i \mid p_i \in B, p_i : (\texttt{SUB}(r), p_j, p_k) \in I\} \\ &\cup \{p_h \mid p_h : \texttt{HALT} \in I\}\,; \end{aligned}$

2. $T = \{a_i \mid 1 \le i \le k\}\,;$

3. $E = \{a_i \mid 1 \le i \le n\}\,;$

4. $\begin{aligned} w_0 &= (B - \{p_0\}) \cup \{p'_i, p''_i, p'''_i \mid p_i \in B, p_i : (\texttt{SUB}(r), p_j, p_k) \in I\}\,, \\ w_1 &= \{p_0\}\,; \end{aligned}$

5. the set of rules R consists of all the rules specified in the following which each of them represents a partition (containing one single rule) of the partitioning of R into sets of rules $R'_1, ..., R'_m$; instead of specifying K we specify which rules have to be used together, implicitly thereby assuming that all other rules are not allowed to be used at the same moment:

- the simulation of the ADD instruction is accomplished by applying in parallel the two rules
  $a_r[_1 p_i \rightarrow p_i[_1 a_r$ and $p_j[_1 \rightarrow [_1 p_j$;
  the symbol for the next instruction p_j enters the skin membrane and the old instruction symbol p_i is exchanged for an additional symbol a_r;
- for $p_i \in B$, $p_i : (\mathtt{SUB}(r), p_j, p_k) \in I$ we first take
  $p_i''[_1 p_i \rightarrow p_i[_1 p_i''$ and $p_i'[_1 a_r \rightarrow a_r[_1 p_i'$;
  the rule $p_i'[_1 a_r \rightarrow a_r[_1 p_i'$ does the appearance checking – only if register r is not empty (i.e., there is at least one a_r in the skin membrane), p_i' enters the skin membrane; therefore, the first rule $p_i''[_1 p_i \rightarrow p_i[_1 p_i''$ is also allowed to be applied alone, i.e., we have two vectors to be included in K, the first one enabling both rules to be applied at the same moment, the second one enabling the first rule only;
- if p_i' is inside the skin membrane, we know that the register has been non-empty; hence, we exchange p_i'' with the label for the next instruction p_j and move out the helper symbol p_i' again by using in parallel the rules
  $p_j[_1 p_i'' \rightarrow p_i''[_1 p_j$ and $[_1 p_i' \rightarrow p_i'[_1$;
- in the case that p_i' is still in the environment, i.e., the register is empty, we exchange the helper symbols p_i''' and p_i'' and now send the helper symbol p_i' into the skin membrane by applying the pair of rules
  $p_i'''[_1 p_i'' \rightarrow p_i''[_1 p_i'''$ and $p_i'[_1 \rightarrow [_1 p_i'$;
- finally we bring in the symbol p_k for the next instruction and return back the helper symbols p_i' and p_i'''; this is accomplished by applying in parallel the pair of rules
  $p_k[_1 p_i''' \rightarrow p_i'''[_1 p_k$ and $[_1 p_i' \rightarrow p_i'[_1$.

The input is accepted if and only if the P automaton Π' reaches the final label p_h (for $p_h : \mathtt{HALT} \in I$), which observation completes the proof. □

We should like to mention that in the proof given above we accept with only p_h remaining in the skin membrane, which means that by adding the rule $[_1 p_h \rightarrow p_h[_1$ we even end up with an empty skin membrane. In other words, we could also consider these P automata with minimal antiport and uniport rules as generating mechanisms yielding their results as the numbers of objects in the skin membrane, without having any additional garbage.

4.3 Results Using (Restricted) Minimal Parallelism

Looking carefully into the proofs of the theorems elaborated in the preceding subsections we immediately observe that always at most one rule is taken from each partition of rules defined there. Hence, we can take the same partitioning of rules $R_1', ..., R_m'$ as used for the control of the rules to be applied together for the

partitioning of rules $R_1, ..., R_d$ needed for the definition of minimal parallelism as well as k-restricted minimal parallelism. Because all the rule sets in the partitioning $R'_1, ..., R'_m$ as constructed in the theorems above contain exactly one rule, computations carried out in the maximally parallel transition mode together with the control sets constructed there are also computations carried out in the (1-restricted) minimally parallel transition mode together with the same control sets. For the k-restricted minimally parallel transition mode with $k \geq 2$ we need a technical construction method known as the prolongation technique used in the area of grammar systems (e.g., see [5] and [10]); we leave a detailed proof to the interested reader.

5 Conclusion

We have explored the computational power of P automata working in the maximally parallel, the minimally parallel or the k-bounded minimally parallel transition mode with rule control and shown computational completeness with various minimal communication rules (minimal symport rules of weight at most two as well as minimal antiport rules of weight one together with uniport rules) working in only one membrane. For the case of analyzing models of P systems as for the P automata described in this paper, the garbage symbols remaining in the single membrane play no role, whereas for generating P systems the question remains which results can be obtained for various transition modes, especially with minimal symport rules, in the generating case. For sure, by adding an additional membrane for collecting the output, the rather simple proofs elaborated in the preceding section also work for the generating case; hence, for minimal symport rules, we either get a better result with respect to the number of membranes (we only need one instead of two) in case we do not care about garbage symbols or, if we take them into account, we at least get much simpler proofs with adding rule control to the maximally parallel transition mode than in the case of using the maximally parallel transition mode for its own, e.g., compare with the proofs given in [14].

Acknowledgements

Sergey Verlan acknowledges the Science and Technology Center in Ukraine, project 4032.

References

[1] G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez, P systems with minimal parallelism, *Theoretical Computer Science* **378** (1) (2007), 117–130.

[2] E. Csuhaj-Varjú, J. Dessow, J. Kelemen, Gh. Păun, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation, Topics in Com-*

puter Mathematics 5, Gordon and Breach Science Publishers, Amsterdam 1994.

[3] E. Csuhaj-Varjú, G. Vaszil, P automata or purely communicating accepting P systems, in: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), *Membrane Computing, International Workshop, WMC-CdeA 2002*, Curteă de Argeş, Romania, August 19–23, 2002, Revised Papers, Lecture Notes in Computer Science **2597**, Springer, 2003, 219–233.

[4] J. Dassow, Gh. Păun, On the power of membrane computing, *Journal of Universal Computer Science* **5** (2) (1999), 33–49.

[5] H. Fernau, R. Freund, M. Holzer, Hybrid modes in cooperating distributed grammar systems: internal versus external hybridization, *Theoretical Computer Science* **259** (2001), 405–426.

[6] R. Freund, M. Oswald, A short note on analysing P systems with antiport rules, Bulletin of the EATCS **78**, 2002, 231–236.

[7] R. Freund, S. Verlan, A formal framework for P systems, in: G. Eleftherakis, P. Kefalas, Gh. Păun (Eds.), *Pre-proceedings of Membrane Computing, International Workshop – WMC8*, Thessaloniki, Greece, 2007, 317–330.

[8] R. Freund, S. Verlan, (Tissue) P systems working in the k-restricted minimally parallel derivation mode, in: E. Csuhaj-Varjú, R. Freund, M. Oswald, K. Salomaa (Eds.), *Proceedings of the International Workshop on Computing with Biomolecules*, Österreichische Computer Gesellschaft, 2008, 43–52.

[9] M.L. Minsky, *Computation – Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, NJ, 1967.

[10] V. Mitrana, Hybrid cooperating/distributed grammar systems, *Comput. Artif. Intell.* **12** (1993), 83–88.

[11] Gh. Păun, Computing with membranes, *J. of Computer and System Sciences* **61**, 1 (2000), 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`).

[12] Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin 2002.

[13] Gh. Păun, G. Rozenberg, Prescribed teams of grammars, *Acta Inf.* **31** (6) (1994), 525–537.

[14] Y. Rogozhin, A. Alhazov, R. Freund, Computational power of symport/antiport: history, advances, and open problems, in: R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), *Membrane Computing. 6th Int. Workshop WMC 2005*, Lecture Notes in Computer Science **3850**, Springer-Verlag, 2006, 1–31.

[15] G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages* (3 volumes), Springer-Verlag, Berlin, 1997.

[16] The P Systems Web Page: `http://ppage.psystems.eu`.