Synchronization of rules in membrane computing

Péter Battyányi

Department of Computer Science,
Faculty of Informatics, University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
e-mail: battyanyi.peter@inf.unideb.hu

Summary. In this paper, computational models that are variants of membrane systems with synchronization of rules in the style of Aman and Ciobanu [1] are considered. We examine membrane system like computational models in different execution modes and with reusage and without reusage of objects in the same computational step. The weak cases are the ones without restrictions on the compound rules. We show in one of the cases that, as a computational model, it is strictly weaker than Turing machines when maximally parallel execution mode is omitted. Furthermore, we prove that the strong cases, when additional conditions are imposed on the compound rules, are computationally complete even without maximal parallelism. Finally, we give a more or less intuitive argument on why these computational systems with non-cooperative rules cannot be computationally complete even in the strong modes. **Keywords:** Membrane systems, Computational completeness, Rewriting systems

1 Introduction

Membrane systems, or P systems, are biologically inspired models of computation introduced by Gh. Păun in [9]. The original model is based on a tree-like structure of nested membranes. The computation proceeds separately in each region: the membranes or regions have their associated multisets that evolve in accordance with various rewriting rules specific to each membrane. In most of the cases, the whole process is synchronized by a global clock: each membrane is waiting for the other one to finish their computation before a new computational step begins. Originally, the computation in each membrane follows a maximally parallel mode, which means that a maximal multiset of rules is applied at the same time, that is, in each membrane, a multiset of rules is executed simultaneously which is such that no more rules could have been added to the multiset to maintain the simultaneous execution property. Several variants of P systems and application modes have been introduced and studied, we refer the interested reader to the monograph [10] for a thorough introduction, or the handbook [11] for a summary of notions and results of the area.

In this paper, we consider variants of the symbol object P system with several types of execution mode and forms of rules. Namely, besides the rules of the form $u \to v$, where u is the multiset to be replaced by the multiset v during a rule application, we consider synchronized rules in the sense introduced by Aman and Ciobanu [1, 2]. We pose the question of what is the computational power of these rules when we consider them with various execution modes and semantics. One of the execution modes is the unsynchronized or sequential one, where the rules can be applied one after the other. The other one is the synchronized mode where a (possibly empty) multiset of rules is executed simultaneously at the same time in a specific membrane computational step in each compartment. We remove, however, the requirement for the rule applications of being maximally parallel. Observe that the unsynchronized mode can be considered as rule executions when the newly obtained objects can be reused in the next computational step, while the synchronized mode is such execution of rules where the newly obtained objects coming from the right hand sides of the rules can only be reused in the next computational step. Regarding the semantics, we even distinguish two different interpretations concerning the synchronized rules. We term them weak and strong application modes. In total, we will talk about four different execution modes: unsynchronized and synchronized modes with the weak or with the strong application mode. It will turn out that the weak application modes are strictly weaker than the strong ones, the latter ones being equivalent to the computational power of the Turing machine model.

2 Preliminaries

2.1 Multisets

Let $\mathbb{N}$ and $\mathbb{N}_{>0}$ be the set of non-negative integers and the set of positive integers, respectively, and let O be a finite nonempty set (the set of object). A *multiset* M over O is a pair $M = (O, f)$, where $f : O \to \mathbb{N}$ is a mapping which gives the *multiplicity* of each object $a \in O$. If $f(a) = 0$ for every $a \in O$, then M is the empty multiset. If $f(a) = n > 0$, then $a \in M$, or $a \in^n M$.

Let $M_1 = (O, f_1), M_2 = (O, f_2)$. Then $(M_1 \sqcap M_2) = (O, f)$ where $f(a) = \min\{f_1(a), f_2(a)\}$; $(M_1 \sqcup M_2) = (O, f')$, where $f'(a) = \max\{f_1(a), f_2(a)\}$; $(M_1 \oplus M_2) = (O, f'')$, where $f''(a) = f_1(a) + f_2(a)$; $(M_1 \ominus M_2) = (O, f''')$ where $f'''(a) = \max\{f_1(a) - f_2(a), 0\}$; and $M_1 \sqsubseteq M_2$, if $f_1(a) \leq f_2(a)$ for all $a \in O$. We abbreviate $\underbrace{M \oplus M \oplus \ldots \oplus M}_{k}$ as $k \cdot M$.

The number of copies of objects in a finite multiset $M = (O, f)$ is its cardinality: $\mathrm{card}(M) = \Sigma_{\{a|f(a)>0\}} f(a)$. Such an M can be represented by any string w over O for which $|w| = \mathrm{card}(M)$, and $|w|_a = f(a)$ where $|w|$ denotes the length of the string, and $|w|_a$, or simply $w(a)$, denotes the number of occurrences of the symbol a in w.

We define the $\mathcal{MS}^n(O)$, $n \in \mathbb{N}$, to be the set of all multisets $M = (O, f)$ over O such that $f(a) \leq n$ for all $a \in O$, and we let $\mathcal{MS}^{<\infty}(O) = \bigcup_{n \geq 0} \mathcal{MS}^n(O)$. Moreover, if A is an arbitrary set, we define $A^{\geq k} = \cup_{n=k}^{\infty} A^n$, where $A^0 = \emptyset$, $A^1 = A$ and $A^n = \underbrace{A \times \ldots \times A}_{n}$ for $n \geq 2$.

If O is a set of objects and $u, v \in \mathcal{MS}^{<\infty}(O)$, we call the (u, v) a rule over O. In what follows, we write $\mathcal{MS}(O)$ in place of $\mathcal{MS}^{<\infty}(O)$ since we will exclusively deal with finite multisets.

2.2 Symbol object P systems of degree 1

Since the concepts that we will study in the sequel are in compliance with the construction of flattening of membrane systems [3], in the sequel, we restrict our attention to membrane systems of degree 1. Below, we provide the definition of a symbol object P system of degree 1.

A *P system* of degree 1 is a tuple $\Pi = (O, w_0, R)$ where O is an alphabet of objects, $w_0 \in MS(O)$ is the initial content of the region, R is the set of (evolution) rules associated with region 1. They are of the form $u \to v$, where $u, v \in \mathcal{MS}(O)$. We assume that the result of the computation is collected from membrane 1 in the form of a multiset of certain terminal objects. A computation gives a result when it comes to a halt. For a rule $r = u \to v \in R_i$, we write $lhs(r)$ for u and $rhs(r)$ for v. A configuration w is the actual multiset content of membrane 1.

2.3 P systems with synchronized rules

In this subsection we present the various versions of P systems with synchronized rules. A P system with with rule synchronization, or P system with synchronized rules, (of degree 1) is a tuple $\Pi = (O, w_0, (R, \rho))$, where $\Pi = (O, w_0, R)$ is a P system of degree 1 and $\rho \subseteq R^{\geq 2}$. For any element $(r_1, r_2, \ldots, r_k)$ of ρ, where $r_i \in R$ $(1 \leq i \leq k)$, we use the notation $r = r_1 \otimes r_2 \otimes \ldots \otimes r_k$ and we call r a synchronized rule, or a compound rule, and r_i its components $(1 \leq i \leq k)$. Let $comp(r)$ denote the set of components of r. We term $r \in R$ a single rule if it is not a synchronized one. In the sequel, we consider P systems with rule synchronization of degree 1. We say that a rule $r \in R$ is non-cooperative if it is of the form $a \to v$ for some $a \in O$. Otherwise, r is said to be cooperative. Similarly, a rule $\rho = r_1 \otimes \ldots \otimes r_k$ is non-cooperative if each of $r_1, \ldots, r_k$ is of the form $a \to v$ for some object $a \in O$.

We clarify how rule execution can be understood in P systems $\Pi = (O, w_0, (R, \rho))$ with rule synchronization. We distinguish two kinds of rule application modes-weak and strong application modes-, and two possibilities for dealing with objects appearing on the right hand side of rules: we can either allow reusing objects created by a rule application or prohibit this in the same computational step. The latter corresponds to the original interpretation in membrane computation. In total, we deal with four possible ways to interpret P systems with synchronized rules. Let w be a configuration of Π. In what follows, we describe how the

next configuration w' emerges from w. Let $r \in R$ be a single rule or let r be a component of a compound rule r'. Then we say that r is applicable in w if $lhs(r) \sqsubseteq w$. An application of a single rule involves the replacement of w with the multiset $w' = w \ominus lhs(r) \oplus rhs(r)$. The applicability and the result of application of compound rules will be clarified in the respective execution modes.

Definition 1.

(W1) *(Weak application mode with reusage of objects.) Reusage of objects means there is no synchronization in the P system, the rules are applied in a sequential manner. Let us clarify what application of a rule means in the specific cases. Let $r = u \to v \in R$. Then r is applicable if $u \sqsubseteq w$. In this case, the result of applying r to w is obtained by removing form w the objects of u and adding the objects of v to $w \ominus u$. Now, let $r = r_1 \otimes r_2 \otimes \ldots \otimes r_n \in \rho$, where $r_i = u_i \to v_i$ $(1 \le i \le n)$. Then r is applicable iff all of its components are applicable as single rules. When this is the case, an application of r consists of applications of the components of r an arbitrary number of times provided all of them are applied at least once. Moreover, an object emerging on the right hand side of a component can be reused even if the execution step of r has not finished yet. More formally, let $r = r_1 \otimes r_2 \otimes \ldots \otimes r_n \in \rho$. Then r is applicable if $u_1 \oplus \ldots \oplus u_n \sqsubseteq w$. In this case, the result of the application, w', is obtained through a sequence of intermediate configurations $w_1, w_2, \ldots, w_k$, where $w = w_1 \to^{r'_1} w_2 \to^{r'_2} \ldots \to^{r'_{k-1}} w_k = w'$ and $w_{j+1} = w_j \ominus lhs(r'_j) \oplus rhs(r'_j)$ $(1 \le j \le k)$ and each of $r_1, \ldots, r_n$ occurs in the sequence $r'_1, \ldots, r'_k$ at least once. We call the configurations w and w' proper and the configurations $w_2, \ldots, w_{k-1}$ intermediate ones. The transitions $w_j \to^{r'_j} w_{j+1}$ are small step transitions $(1 \le j \le k-1)$, while the transition yielding w' from w is a big step transition. In notation: $w \Rightarrow^r_{w,y} w'$.*

(W2) *(Weak application mode without reusage of objects.) In this case, a big step comprises the simultaneous application of several single or compound rules in the compartments. When all the membranes has finished working, only then can the next computational step begin. The objects coming from the right hand side of the rules can be used only in the next big computational step. Formally, let $\mathcal{R}$ be a multiset over the set of rules $R \cup \rho$. We define multisets, $sub(\mathcal{R})$ and $add(\mathcal{R})$ over O. Firstly, let $r = u \to v \in R$. Then $sub(r) = lhs(r) = u$ and $add(r) = rhs(r) = v$. On the other hand, if $r = r_1 \otimes \ldots \otimes r_n \in \rho$, let $r_i = u_i \to v_i$ $(1 \le i \le n)$. Then $sub(r) = \bigoplus_{i=1}^n k_i \cdot u_i$ and $add(r) = \bigoplus_{i=1}^n k_i \cdot v_i$, where $k_i \ge 1$ $(1 \le i \le n)$. We write $sub(\mathcal{R}) = \bigoplus\{sub(r) \mid r \in \mathcal{R}\}$ and $add(\mathcal{R}) = \bigoplus\{add(r) \mid r \in \mathcal{R}\}$ and we set $w' = (w \ominus sub(\mathcal{R})) \oplus add(\mathcal{R})$.*

(S1) *(Strong application mode with reusage of objects.) This application mode demands that the application of rules, being either single or compound ones, takes place in a sequential way. The only difference regarding application mode (W1) is the different interpretation of the compound rules. Let $r \in R$ be a single rule. Then r is applicable if $lhs(r) \sqsubseteq w$ and, in this case, $w' = w \ominus lhs(r) \oplus rhs(r)$. On the other hand, let $r = r_1 \otimes r_2 \otimes \ldots \otimes r_n \in \rho$. Then r is applicable to w_0 in the strong sense if at least one of r's components is applicable. When this holds,*

we apply r by searching from left to right the first applicable component r'. After executing r', we resume searching for an applicable component of r starting from the first component. More formally, r is applicable if $lhs(r_i) \sqsubseteq w_0$ for some $1 \leq i \leq n$. Then an application of r is described by the following sequence of intermediate configurations: $w_0 \to^{r'_1} w_1 \to^{r'_2} w_2 \ldots \to^{r'_k} w_k = w'$, where r'_i is of minimal index i among the components of r applicable to w_{i-1} $(1 \leq i \leq k)$. No component is applicable to $w' = w_k$. The transitions $w_{i-1} \to^{r'_i} w_i$ $(1 \leq i \leq k)$ are called small steps regarding the application of r in the strong sense with reusage of objects, while the process yielding w' form w is called a big step. In notation: $w \Rightarrow^r_{s,y} w'$. The configurations w and w' are proper.

(S2) *Strong application mode without reusage of objects.) It differs the above mode only in the treatment with the objects coming form the right hand side of the components during an execution of a compound rule. In plain words, a compound rule r is executed by searching for the first applicable component with the smallest index. Then the component is executed as a single rule, i.e., we subtract the multiset on the left hand side from the actual configuration and add the multiset on the right hand to the result of the subtraction. Then we start searching for the applicable component with the smallest index in the emerging new multiset. This process stops until there are no more applicable components. More precisely, let $r \in R$. Then let the result of the transition $w \Rightarrow^r_{s,n} w'$ be the multiset $w' = w \ominus lhs(r) \oplus rhs(r)$. Suppose $r \in \rho$. Then we define the multisets $lhs(r)$, $rhs(r)$ by giving three sequences of multisets $u_0, u_1, \ldots$, $v_0, v_1, \ldots$ and $w_0, w_1, \ldots$ simultaneously. Let $u_0 = \varepsilon$, the empty multiset, and let $v_0 = w_0 = w$. Assume u_i, w_i, v_i are already defined for some $i \in \mathbb{N}$. Let r' be the component of r that is the first one from left to right which is applicable to w_i. Then $u_{i+1} = u_i \oplus lhs(r')$ and $w_{i+1} = w \ominus u_{i+1}$, $v_{i+1} = v_i \oplus rhs(r')$. We continue if at least one component of r is applicable, that is, we calculate the $i + 2$-th elements of the sequences if $lhs(r'') \sqsubseteq w_{i+1}$ for some $r'' \in comp(r)$. Let m be the first index for which this is not case. Then we denote $lhs(r) = u_m$ and $rhs(r) = v_m$. We have $w' = w \ominus lhs(r) \oplus rhs(r)$, and we write $w \Rightarrow^r_{s,n} w'$.*

Let us illustrate the definition by demonstrating the operation of a P system with synchronization of rules in the strong application mode with reusage of objects (S2).

Example 1. Let $\Pi = (O, w_0, (R, \rho))$ be a P system with synchronization of rules, where $n = 1$, $O = \{a, b, d, e\}$, $w_0 = a^n b^m$, and $R = \{r_1 = ad^m \to e^m b^m, r_2 = ab \to da, r_3 = b \to \varepsilon\}$. Let $\rho = \{r = r_1 \otimes r_2 \otimes r_3\}$. Let us consider a terminating computation starting from $w_0 = a^n b^m$ in mode (S2). We assume $n, m \geq 1$. Let the subscripts of the arrows denote the components of r applied.

$$
\begin{aligned}
&a^n b^m \to^*_{r_2} a^n d^m \to_{r_1} a^{n-1} e^m b^m \to_{r_2} \\
&a^{n-1} e^m b^{m-1} d \to^*_{r_2} a^{n-1} e^m d^m \to_{r_1} a^{n-1} e^{2m} b^m \to_{r_2} \\
&\ldots \\
&a e^{(n-1)m} b^{m-1} d \to^*_{r_2} a e^{(n-1)m} d^m \to_{r_1} e^{nm} b^m \to^*_{r_3} e^{nm}
\end{aligned}
$$

In what follows, let w stand for the actual configuration of our membrane. In the example above, when the multiset w contains less than m d's, then the successive applications of the component r_2 remove one copy of b and add a d to w. The new copies of d add to the configuration. In the case when $d^m \sqsubseteq w$, r_1 is applicable and, hence, it must be applied. This means an erasure of one copy of a and introducing e^m and b^m to w. The computation proceeds in this way until all the a's are consumed. At this point, $w = e^{nm}b^m$. Now only r_3 can be applied, which yields the removal of b^m from w.

3 The power of synchronized rules in P systems

We turn to a brief discussion of the computational power of synchronization of rules in P systems with respect to the above application modes. We treat first the case of weak application modes. In this section, we present some results and provide their short justifications or we give hints on how they can be achieved.

3.1 Synchronized rules with the weak application mode

In this subsection we examine P systems with synchronization of rules using the weak application mode. It turns out that P systems with application mode (W1) are not computationally complete, and we formulate a conjecture for a similar statement on P systems with application mode (W2). In the case of application mode (W1), that is, the weak application mode with reusage of objects, we show that compound rules can be substituted for ordinary rules of a P system such that the multisets computed by the two P systems will be the same.

Proposition 1. *Let $\Pi = (O, w_0, (R, \rho))$ be a P system of degree 1 with synchronization of rules using execution mode (W1). Then there exists a P system Π' of degree 1 without synchronization of rules applying the sequential mode such that Π' and Π compute the same sets of vectors.*

Proof. Let $\Pi = (O, w_0, (R, \rho))$ be as above. Let us define Π' in the following way. Let us add to O a finite set of new objects as described below:

$$O' = O \cup \{\omega, \kappa, \vartheta_1^r, \ldots, \vartheta_{k_r}^r \mid r = r_1 \otimes \ldots \otimes r_{k_r} \in \rho\}.$$

Let $w_0' = w_0 \sqcup \{\omega, \vartheta_1^r \mid r \in \rho\}$. We obtain R' as follows. Firstly, we add the following single rules to R for any $r = r_1 \otimes \ldots \otimes r_k \in \rho$. Let $r_i = u_i \to v_i$ $(1 \leq i \leq k)$. We define the rules:

$$
\begin{aligned}
r'_1 &= \omega u_1 \vartheta_1^r \to \kappa v_1 \vartheta_1^r,\\
r''_1 &= \kappa u_1 \vartheta_1^r \to \kappa v_1 \vartheta_2^r,\\
r'_2 &= \kappa u_2 \vartheta_2^r \to \kappa v_2 \vartheta_2^r,\\
r''_2 &= \kappa u_2 \vartheta_2^r \to \kappa v_2 \vartheta_3^r,\\
&\dots\\
r'_k &= \kappa u_k \vartheta_k^r \to \kappa v_k \vartheta_k^r,\\
r''_k &= \kappa u_k \vartheta_k^r \to \omega v_k \vartheta_1^r.
\end{aligned}
$$

We add the rule $\kappa \to \kappa$ to R'. Moreover, if $r = u \to v \in R$, then we let $r' = \omega u \to \omega v \in R'$. It is easy to check that $\Pi' = (O', w'_0, R')$ produces the same set of multisets as Π. □

Regarding the case of (W2), i.e., weak application mode without reusage of objects we believe that the computational power is strictly weaker than that of Turing machines. We assert this as a conjecture. We think that a P system with application mode (W2) can be simulated with a forbidding context grammar with λ-rules.

Conjecture 1. Let $\Pi = (O, w_0, (R, \rho))$ be a P system of degree 1 with synchronization of rules applying execution mode (W2). Then there exists a forbidding context grammar G_Π with λ-rules such that $Ps(\Pi) = Ps(G_\Pi)$.

3.2 Synchronized rules with the strong application mode

Now we continue with our investigation with synchronization of rules in the strong application mode. We demonstrate that generalized P systems with synchronized rules in the strong application mode can compute any Turing computable, or in, other words, partial recursive function. To this end, we simulate register machines with zero-test subtraction with P systems applied with the strong application mode. We recall briefly the definition of such register machines.

A *register machine* is a tuple $W = (m, H, l_0, l_h, \text{Inst})$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halting label, and Inst is the set of instructions. There is a bijection between the labels of H and the instruction of Inst. The following types of instructions can be used. For $l_i, l_j, l_k \in H$ and $r \in \{1, \dots, m\}$ we have:

- l_i : (ADD(r), l_j, l_k) - *nondeterministic add*: Add 1 to register r and then go to one of the instructions with labels l_j or l_k, nondeterministically chosen.
- l_i : (SUB(r), l_j, l_k) - *zero check and subtract*: If register r is empty, then go to the instruction with label l_j, if r is non-empty, then subtract one from it and go to the instruction with label l_k.
- l_h : HALT - *halt*: Stop the machine.

A computation of a register machine starts with all registers empty except for some designated input registers. The control flow is regulated by the labels shown

in the actual instruction of the machine. The machine starts with the instruction labeled l_0. If the machine reaches the halt instruction l_h : `HALT`, then it stops, and the number stored in the first register, or in the output registers fixed in advance, is the result of the computation. Note that our register machine is a nondeterministic computing device. In this case, it is suitable for computing sets of natural numbers when we consider the outcomes of the various computations.

We consider the two computation modes, (S1) and (S2), separately. Firstly, we deal with (S1), that is, strong computation mode with reusage of objects.

Theorem 1. *Generalized P systems with synchronized rules in the strong application mode with reusage of objects can simulate arbitrary register machines with zero-test subtractions, even without using the maximally parallel rule execution. We may even assume that the P system is a purely catalyctic one with a three-state catalyst.*

Proof. Let $W = (m, H, l_0, l_h, \mathrm{Inst})$ be a register machine. Fpr the sake of simplicity, we assume that W has one output register, let this be register 1, W computes a number instead of a vector, and, in addition, W starts its computation with all registers initially empty. We construct a P system Π of degree 1 with synchronization of rules using execution mode (S1) such that, at every computational step, the number stored in register i of W is represented by the number of the object a_i in the only region of Π and, upon halting, Π provides the result by producing the same number of copies of object a_1 as the number stored in register R_1 of W.

We define Π as a purely catalyctic P system with a three-state catalyst. We have to take care that the execution mode allows us reusing elements created during a rule application, in contrast with common practice in membrane systems. Let $\Pi = (O, w_0, (R, \rho))$, where

$$\begin{aligned} O &= \{l \mid l \in H\} \cup \{a_r \mid 1 \le r \le m\} \cup \{c, c', c''\}, \\ w_0 &= l_0 c, \\ R &= R_{Add} \cup R_{Sub} \cup R_{Halt}, \end{aligned}$$

where

$$R_{Add} = \{r_{(i,1)} = cl_i \to cl_j a_r, r_{(i,2)} = cl_i \to cl_k a_r,$$
$$r''_{(i,1)} = c''l_i \to c''l_j a_r, r''_{(i,2)} = c''l_i \to c''l_k a_r \mid \text{for all}$$
$$l_i : (\texttt{ADD}(r), l_j, l_k) \in \text{Inst}\},$$

$$R_{Sub} = \{r^o_{(i,1)} = ca_r \to c', r^o_{(i,2)} = c'l_i \to c''l_j, r^o_{(i,3)} = cl_i \to c''l_k,$$
$$r^e_{(i,1)} = c''a_r \to c', r^e_{(i,2)} = c'l_i \to cl_j, r^e_{(i,3)} = c''l_i \to cl_k$$
$$\mid \text{for all } l_i : (\texttt{SUB}(r), l_j, l_k) \in \text{Inst}\},$$

$$R_{Halt} = \{cl_h \to c, c''l_h \to c''\},$$

$$\rho = \{\rho^o_i = r^o_{(i,1)} \otimes r^o_{(i,2)} \otimes r^o_{(i,3)}, \rho^e_i = r^e_{(i,1)} \otimes r^e_{(i,2)} \otimes r^e_{(i,3)} \mid l_i : (\texttt{SUB}(r), l_j, l_k)\}.$$

We give a brief explanation of how Π simulates a computation of W. The initial configuration corresponds to the initial configuration of W, since the first region contains l_0, the label of the starting instruction, and W commences its operation with the assumption that all registers are initially empty. The label of the next instruction, together with copies of the objects a_r ($1 \leq r \leq m$), are to be found in membrane 1, the only membrane of Π. There are two sets of rules with respect to the SUB instruction: they correspond to the odd and even turns of the applications of SUB. When in the instruction sequence the SUB has been called an even number of times, the next execution will be governed by a ρ-rule with superscript "o". Otherwise, it is the turn of the ρ-rules with superscript "e". This will be detailed below. We describe the next step of the simulation process by taking into account the different cases. Let $conf$ denote the actual configuration of Π, that is, the content of membrane 1.

- $l_i : (\texttt{ADD}(r), l_j, l_k)$. The applications of the corresponding rules of Π introduce the label representing the next instruction of W in Π, adding one copy of a_r to $conf$ at the same time. We distinguish the different cases regarding the actual number of SUB instructions applied before calling instruction l_i.
- $l_i : (\texttt{SUB}(r), l_j, l_k)$. Assume an even number of SUB has already been applied and it is the turn of the instruction l_i. Then $c \in conf$, and the rule ρ^o_i is executed. If $a_r \in conf$, then the object c' is introduced and, in the next step, l_i is transformed to l_k and c' is exchanged with c''. At this point, the operation of ρ^o_i halts and the simulation of the instruction labelled l_k can commence. If $a_r \notin conf$, then $r^o_{(i,1)}$ cannot be applied, instead, a copy of c'' and l_k is introduced using $r^o_{(i,3)}$. If l_i emerges as an even turn of an application of SUBs, then $c'' \in conf$ and a process similar to the above one can be executed.
- $l_h : \texttt{HALT}$. Then l_h is removed and the computation of Π comes to a halt.

Obviously, the P system Π halts if and only if the register machine W halts and, at this point, the number of copies of the object a_1 and the number stored in the first register of W are the same. □

We formulate a similar statement asserting the possibility of simulation register machines with P systems in application mode (S2). In this case, it is enough to consider a purely catalyctic P system with bistable catalyst for the simulation. Since the proof is very much like in the case of application mode (S1), we just state the result without proof.

Theorem 2. *Generalized P systems with synchronized rules in the strong application mode without reusage of objects (S2) can simulate, without using the maximally parallel rule execution, arbitrary register machines with zero-test subtractions, even when we consider purely catalyctic P systems with a bistable catalyst.*

Proof. Similar to that of the previous theorem. This time we do not need to consider alternating turns in the application of instruction SUB, hence, the proof even simplifies a little. □

The above proofs demonstrate the fact that P systems with synchronization of rules and the strong application mode are able to simulate register machines either with reusage or without reusage of objects. The question naturally arises how far can we go in the simplification of rules constituting the P system. In what follows, we assert a claim saying that non-cooperative rules are not enough for ensuring computational completeness. We omit the more or less intuitive argument for the claim, however, it already provides us with a string justification that the statement holds. We formulate our assertion for the case (S1) only, the case for (S2) being similar, mutatis mutandis.

Theorem 3. *Let Π be a P system of degree 1 with synchronization of rules, with non-cooperative rules of execution mode (S1). Then Π is not computationally complete.*

Proof. [Sketch] The intuitive argument relies on the fact that Π cannot compute the $\overline{sg}$ function, where

$$\overline{sg} = \begin{cases} 1 \text{ if } n = 0, \\ 0 \text{ otherwise.} \end{cases}$$

More precisely, we show that, if $O = \{a_1, \ldots, a_n\}$ is the object set and $c = (c(a_1), \ldots, c(a_n))$ is a corresponding configuration of Π, when we are given two configurations c' and c'' such that $c' \sqsubseteq c''$ as multisets, then Π cannot evolve on c' and c'' such that, upon halting, the reverse relation would hold. I.e., when $c'' \Rightarrow^* \underline{c}''$ and $\underline{c}''$ is a hlating configuration then there exists a halting configuration $\underline{c}'$ such that $c' \Rightarrow^* \underline{c}'$. Since Π computes a function, it cannot be the case that the result for the input c' is represented by a configuration c for which $c \sqsubseteq \underline{c}''$ does not hold. □

4 Concluding remarks

- The problems discussed in the paper stem from a specific membrane system defined by Aman et al. [1]. Our results partly deviate from the usual membrane

system notions by examining rule applications with the possibility of reusage of elements even in the same computational step. The execution modes (W1) and (S1) could have equally been formulated for computational models with this property, e.g., for Petri nets.

- In some programming languages, like SML [4], lines are processed from top to bottom. This imports lends some control facilities to program execution. The present results are in accordance with this experience of programmers: the weak application mode could be associated with a purely declarative philosophy of program execution, while the strong application mode can be related to an implementation where the order of instructions matter. Our result intimates that setting up an order of execution for the rules adds computational strength to the programming language implementation.
- The P systems constructed for the simulation of register machines look very much like generalized communicating P systems (GCPS) in appearance [5]. GCPSs posses a graph-like structure, where each node, called a cell, contains a multiset of objects which may move between the cells by the so-called communication rules. A communication rule has the form $(a, i)(b, j) \to (c, k)(d, l)$, where a, b, c, d are objects and i, j, k, l represent the input and output regions, respectively. Depending on the values of the identifiers i, j, k, l, several restricted forms of interaction rules can be specified. Generalized communicating P systems mostly obey the maximally parallel rule execution mode. It can be shown that, in most of the cases, GCPSs are Turing complete even with a set of restricted form of interaction rule and taking a relatively small, fixed number of cells [5]. Computational completeness is preserved when we consider an alphabet with one object and rules of restricted types or only as many as three cells together with rules of restricted types [6, 7]. It would be interesting to explore the similarities and differences between GCPSs and the computational model defined in this paper and obtain results of an analogous nature by restricting the form of the rules, the number of cells or the number of objects. We would emphasize the main difference between the two computational models: with synchronization of rules Turing completeness is achieved without additional control facilities in the strong application mode, namely, without imposing the necessity of maximally parallel execution mode. Hence, results of somehow different types should be expected in our case.

References

1. Aman, B., Ciobanu, G., Synchronization of rules in membrane computing. Journal of Membrane Computing **1**, 233–240 (2019). https://doi.org/10.1007/s41965-019-00022-1
2. Aman, B., Ciobanu, G., The power of synchronizing rules in membrane computing, Information Sciences **594**, 360–370 (2022)
3. Agrigoroaiei, O., Ciobanu, G., Flattening the transition P systems with dissolution In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds)

Membrane Computing. CMC 2010. LNCS, vol 6501. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-18123-8_7
4. Blume, M.,: The SML/NJ Compilation and Library Manager. Lucent Technologies, Bell Labs (2002). https://www.smlnj.org/doc/CM/new.pdf
5. Csuhaj-Varjú, E., Verlan, S.: On generalized communicating P systems with minimal interaction rules. Theoretical Computer Science **412**(1-2), 124–135 (2011), https://doi.org/10.1016/j.tcs.2010.08.020.
6. Csuhaj-Varjú, E., Verlan, S.: Computationally Complete Generalized Communicating P Systems with Three Cells. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing. CMC 2017. LNCS, vol 10725. Springer, Cham. https://doi.org/10.1007/978-3-319-73359-3_8
7. Csuhaj-Varjú, E., Vaszil, G., Verlan, S.: On Generalized Communicating P Systems with One Symbol. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds) Membrane Computing. CMC 2010. Lecture Notes in Computer Science, vol 6501. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-18123-8_14
8. Alhazov, A., Belingheri, O., Freund, R., Ivanov, S., Porreca, A. E., and Zandron, C. (2016). Semilinear Sets, Register Machines, and Integer Vector Addition (P) Systems. In: Proceedings of 17th International Conference on Membrane Computing (CMC17), 39–56 http://hdl.handle.net/20.500.12708/56909
9. Păun, G.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000)
10. Păun, G.: Membrane Computing: An Introduction. Springer-Verlag, Berlin, Heidelberg (2002)
11. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)