Sequential P Systems with Active Membranes Working on Sets*

Michal Kováč

Faculty of Mathematics, Physics and Informatics, Comenius University

Abstract. We study variants of P systems that are working in the sequential mode. Usually, they are not computationally complete, but there are possible extensions that can increase the computation power. Extensions that implement a notion of zero-checking are often computationally complete. P systems with an ability to create new membranes are a rare exception as they are known to be computationally complete even in the sequential mode without using a dedicated zero-check operation. Using sets instead of multiset was inspired by Reaction systems and we show how to use this relaxation in the context of active membranes. We present some results on computational power for this variant as well as for various notions of membrane creation.

1 Introduction

Membrane systems (P systems) [1] were introduced by Păun (see [2]) as distributed parallel computing devices inspired by the structure and functionality of cells. Starting from the observation that there is an obvious parallelism in the cell biochemistry and relying on the assumption that "if we wait enough, then all reactions which may take place will take place", a feature of the P systems is given by the maximal parallel way of using the rules. For various reasons, ranging from looking for more realistic models to just more mathematical challenge, the maximal parallelism was questioned, either simply criticized, or replaced with presumably less restrictive assumptions. In some cases, a sequential model may be a more reasonable assumption. In sequential P systems, only one rewriting rule is used in each step of computation. Without priorities, they are equivalent to Petri nets [3], hence not computationally complete. However, priorities, inhibitors and other modifications can increase the computation power. It seems that there is a link between universality and ability to zero-check [4].

Standard models of membrane systems have configurations, where any given compartment is represented as a multiset of objects and each computational action is represented by a multiset of simultaneously executed (multiple copies of) individual evolution rules.

Such strong reliance on counting (through multiple copies of objects and rules) may lead to potential problems in two respects. First, one may wonder how realistic is the counting (multiset) mechanism if one needs to represent huge

* Work supported by the grant VEGA 1/1333/12.

numbers of molecules and instances of biochemical reactions. Second, a membrane system would normally have an infinite state space, making the application of formal verifiation techniques impratical or indeed impossible (there exists a rich body of results proving Turing completeness of even very simple kinds of membrane systems).

A radical solution to the state space problems can be provided by reaction systems, which, however, model biochemial reactions in living cells using qualitative (based on presence and absence of entities) rather than quantitative rewriting rules.

Set membrane systems [5] are based on sets (of objects or rules) together with the associated set operations, rather than on multisets with multiset operations. An interesting property of maximal parallel steps in set membrane systems is that there is always exactly one maximal parallel set of simultaneously applicable rules, thus such system is deterministic.

Alhazov in [6] proposed P systems, where the multiplicities of objects are ignored, which is essentially the same as set membrane systems. He proved that with bounded number of membranes they have a very limited computing power, exactly the Parikh images of regular languages. On the other hand, allowing membrane creation or division implies the computational completeness.

The sequential mode was only mentioned in [5] under the notion of "min-enabled" computational step. As well as in the maximal parallel mode, the sequential set membrane systems can also generate only the regular languages [6].

In section 2 we recall some computer science basic notions that we will use through the work. Sequential P systems with active membranes working on sets instead of multisets (Sequential active set P systems) are formally presented in section 3.

In section 4 we show the computational completeness by simulating the register machine and in the last section 5 various modifications of membrane creation are investigated.

2 Preliminaries

Here we recall several notions from the classical theory of formal languages.

An **alphabet** is a finite nonempty set of symbols. Usually, it is denoted by Σ. A **string** over an alphabet is a finite sequence of symbols from the alphabet. We denote by Σ^* the set of all strings over an alphabet Σ. By $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ we denote the set of all nonempty strings over Σ. A **language** over the alphabet Σ is any subset of Σ^*.

The number of occurrences of a given symbol $a \in \Sigma$ in the string $w \in \Sigma^*$ is denoted by $|w|_a$. $\Psi_\Sigma(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$ is called a Parikh vector associated with the string $w \in \Sigma^*$, where $\Sigma = \{a_1, a_2, \ldots, a_n\}$. For a language $L \subseteq \Sigma^*$, $\Psi_\Sigma(L) = \{\Psi_\Sigma(w) | w \in L\}$ is the Parikh image of L. If FL is a family of languages, PsFL denotes the family of Parikh images of languages in FL.

Next, we recall notions from graph theory.

A **rooted tree** is a tree, in which a particular node is distinguished from the others and called the root node. Let T be a rooted tree. We will denote its root node by r_T. Let d be a node of $T \setminus \{r_T\}$. As T is a tree, there is a unique path from d to r_T. The node adjacent to d on that path is also unique and is called a **parent node** of d and is denoted by $parent_T(d)$. We will denote the set of nodes of T by $V(T)$ and set of its edges by $E(T)$. Let T_1, T_2 be rooted trees. A bijection $f : V(T_1) \to V(T_2)$ is an **isomorphism** iff $\{(f(u), f(v)) | (u, v) \in E(T_1)\} = E(T_2)$ and $f(r_{T_1}) = r_{T_2}$.

3 Sequential active set P systems

The fundamental ingredient of a P system is the **membrane structure** (see [7]). It is a hierarchically arranged set of membranes, all contained in the **skin membrane**. Each membrane determines a compartment, also called region, which is the space delimited from above by it and from below by the membranes placed directly inside, if any exists. Clearly, the correspondence between membranes and regions is one-to-one, that is why we sometimes use interchangeably these terms. The membrane structure can be also viewed as a rooted tree with the skin membrane as the root node.

A P system consists of a membrane structure, where each membrane is labeled with a number from 1 to m. Each membrane contains a set of objects. Objects can be transformed into other objects and sent through a membrane according to given rules defined for membrane labels. The rules are known from the beginning for each possible membrane, even for the ones that do not exist yet, or the ones that will never exist.

In this paper we work with sequential P systems with active membranes working on sets (Sequential active set P systems). The rules can modify the membrane structure by dissolving and creating new membranes. That is why we will define the configuration to include the membrane structure as well.

Let Σ be a set of objects. A **membrane configuration** is a tuple (T, l, c), where:

- T is a rooted tree,
- $l \in \mathbb{N}^{V(T)}$ is a mapping that assigns for each node of T a number (label), where $l(r_T) = 1$, so the skin membrane is always labeled with 1,
- $c \in (2^{\Sigma})^{V(T)}$ is a mapping that assigns for each node of T a set of objects from Σ, so it represents the contents of the membrane.

The common representation of a membrane structure in this paper is by a string, where a membrane is denoted by a pair of matching square brackets, e.g. $[_1[_2ab]_2ac]_1$.

A **sequential active set P system** is a tuple $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, where:

- Σ is a set of objects,
- C_0 is the initial membrane configuration,

- $R_1, R_2, \ldots, R_m$ are finite sets of rewriting rules associated with the labels $1, 2, \ldots, m$ and can be of forms:
  - $u \to w$, where $u \subseteq \Sigma$, $|u| \geq 1$, $w \subseteq (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})$ and $1 \leq j \leq m$,
  - a dissolving rule $u \to w\delta$, where $u \subseteq \Sigma$, $|u| \geq 1$, $w \subseteq (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})$ and $1 \leq j \leq m$,
  - a membrane creation $u \to [_j v_1]_j v_2$, where $u \subseteq \Sigma$, $|u| \geq 1$, $v_1, v_2 \subseteq \Sigma$ and $1 \leq j \leq m$.

For the first two forms, each rewriting rule may specify for each object on the right side, whether it stays in the current region (we will omit the symbol $\cdot$), moves through the membrane to the parent region ($\uparrow$) or to a specific child region ($\downarrow_j$, where j is a label of a membrane). We denote these transfers with an arrow immediately after the symbol. An example of such rule is the following: $ab \to ab \downarrow_2 c \uparrow c\delta$.

By applying the rule we mean the removal of objects specified on the left side and the addition of the objects on the right side with respect to set union semantics. Symbol $\delta \notin \Sigma$ does not represent an object. It may be present only at the end of the rule, which means that after the application of the rule, the membrane is dissolved and its contents (objects, child membranes) are propagated to the parent membrane.

Active P systems differ from classic (passive) P systems in their ability to create new membranes by rules of the third form. Such rule will create new child membrane with a given label j and a given set of objects v_1 as its contents, while the set v_2 is the set of products that stays in the current membrane. If the current membrane already contains a child membrane with label j, then such rule is not applicable.

For a sequential active set P system $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, configuration $C = (T, l, c)$, membrane $d \in V(T)$ the rule $r \in R_{l(d)}$ is **applicable** iff:

- $r = u \to w$ and $u \subseteq c(d)$ and for all $(a, \downarrow_k) \in w$ there exists $d_2 \in V(T)$ such that $l(d_2) = k \wedge parent(d_2) = d$,
- $r = u \to w\delta$ and $u \subseteq c(d)$ and for all $(a, \downarrow_k) \in w$ there exists $d_2 \in V(T)$ such that $l(d_2) = k \wedge parent(d_2) = d$ and $d \neq r_T$,
- $r = u \to [_j v_1]_j v_2$ and $u \subseteq c(d)$.

In this paper we assume only sequential systems, so in each step of the computation, there is one rule nondeterministically chosen among all applicable rules in all membranes to be applied.

A **computation step** of a sequential active P system is a relation $\Rightarrow$ on the set of membrane configurations such that $C_1 \Rightarrow C_2$ holds iff there is an applicable rule in a membrane in C_1, such that applying that rule can result in C_2.

The P system can work in generating or in accepting mode. For the generating mode we consider the concatenation of the objects which leave the system, in the order they are sent out of the skin membrane (if several symbols are expelled at the same time, then any ordering of them is considered). In this case we generate a language. The result of a single computation is clearly only one multiset or a string, but for one initial configuration there can be multiple possible computations. It follows from the fact that there can be more than one applicable rule in each configuration and they are chosen nondeterministically.

For the accepting mode the input word is encoded into a membrane structure by a given encoding and it is accepted if and only if a given accepting configuration can be reached[3].

4 Simulation of register machine

4.1 Register machine

Definition 1. *A* n**-register machine** *is a tuple* $M = (n, P, i, h)$*, where:*

- n *is the number of registers,*
- P *is a set of labeled instructions of the form* $j : (op(r), k, l)$*, where* $op(r)$ *is an operation on register* $r \leq n$*, and* j*,* k*,* l *are labels from the set* $Lab(M)$ *such that there are no two instructions with the same label* j*,*
- i *is the initial label, and*
- h *is the final label.*

The machine is capable of the following instructions:

- $(add(r), k, l)$: Add one to the contents of register r and proceed to instruction k or to instruction l; in the deterministic variants usually considered in the literature we demand $k = l$.
- $(sub(r), k, l)$: If register r is not empty, then subtract one from its contents and go to instruction k, otherwise proceed to instruction l.
- *halt* : This instruction stops the machine. This additional instruction can only be assigned to the final label h.

A deterministic m-register machine can analyze an input $(n_1, \ldots, n_m) \in N_0^m$ in registers 1 to m, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty (this last requirement is not necessary). If the machine does not halt, the analysis was not successful.

A configuration of a register machine is a tuple $(r_1, \ldots, r_m, ip)$, where r_i is value of the register i and ip (instruction pointer) is the label of current instruction to be executed.

4.2 Simple simulation

For a register machine with m registers we will construct a sequential active set P system $(\Sigma, C_0, R_1, \ldots R_{m+1})$, where $\Sigma = \{x_j, y_j$ for instructions with label $j\} \cup \{t_i$ for each register $i\}$. Skin membrane will be labeled with $m + 1$, other labels correspond to registers 1 to m. C_0 will be the input word for the register machine encoded into a membrane structure by the following encoding:

For a configuration of register machine $(r_1, r_2, \ldots r_m, ip)$ the membrane structure will consist of a skin membrane, which will contain m chains consisting of

r_i membranes embedded one into another like in a Matryoshka doll with label i. The innermost membranes will contain a single object t_i. If $r_i = 0$ then t_i is in the skin membrane and there is no membrane with label i. Object representing the label of the current instruction (x_{ip}) is in the skin membrane.

We will have following rules in the skin membrane:

- $y_j \to x_j$,
- $x_j \to x_j \downarrow_i$ for instruction $j : op(i)$,
- $x_j, t_i \to [_1 y_k, t_i]_1$ for instruction $j : (add(i), k, \_)$,
- $x_j, t_i \to l$ for instruction $j : (sub(i), \_, l)$

For the membrane i:

- $x_j \to x_j \downarrow_i$ for instruction $j : op(i)$,
- $x_j, t_i \to [_1 y_k, t_i]_1$ for instruction $j : (add(i), k, \_)$,
- $y_j \to y_j \uparrow$ for instruction $j : (op(i), \_, \_)$,
- $x_j, t_i \to y_k, t_i, \delta$ for instruction $j : (sub(i), k, l)$

Object x_j represents the instruction currently executed. It is sent down the chain of membranes and in the innermost membrane the creation of new membrane, or the dissolution is performed. Then the next instruction represented by object y_j is sent upwards all the way to the skin membrane. The object t_i is always present in the innermost membrane. The zero-test is implemented by the two rules in the skin membrane, which require the presence of t_i, meaning that the value of register i is zero.

If empty register halting is needed, we will consume t_i symbols with the label of a halting instruction in the skin membrane.

Theorem 1 *Sequential active set P systems are computationally complete.*

Proof. Computational completeness is proved by a direct simulation of a register machine, which is also computationally complete. □

The simulation was quite straightforward. We proved that the model is computationally complete. However, the simulation is not very effective. It uses alphabet of size $2 *$ number of instructions $+$ number of registers, and its number of membranes is linearly dependent on the sum of values of registers. The time needed for executing an instruction on register i is linearly dependent on r_i.

4.3 Optimalization of the simulation

In this subsection we address the inefficient usage of membranes in the previous simulation. New, optimized simulation will reduce it to logarithmic dependency.

For a register machine with m registers we will construct a seqeuntial active set P system, where $\Sigma = \{0, 1, p, s, t\} \cup \{x_j, y_j, z_j$ for instructions with label $j\}$. Skin membrane will be labeled with $m + 1$, other labels correspond to registers 1 to m.

Assume configuration of register machine $(r_1, r_2, \ldots r_m, ip)$. For each register i, let $b_1 b_2 \ldots b_k$ be a binary representation of r_i. The skin membrane will contain a chain of k membranes embedded one into another like in a Matryoshka doll with label i. The membrane in depth d will contain the object b_{k-d}, which is either 0 or 1. So the highest-order position in the binary number is represented by the innermost membrane and more-often incremented positions are in membranes closer to the skin membrane. Moreover, the innermost membranes contain a single object t. The skin membrane contains the label of the current instruction x_{ip}. Other membranes (not skin and not innermost) contain s. Object p will be in all membranes except the skin membrane and direct children of the skin membrane. It represents the fact that the membrane can be dissolved, while keeping at least one membrane for binary representation of the register value.

We will have following rules in the skin membrane:

- $y_j \to x_j$,
- $x_j \to x_j \downarrow_i$ for instruction $j : op(i)$

For the membrane i and instruction j:

- $y_j \to y_j \uparrow$ (return the next instruction to the skin membrane).

For the membrane i and instruction $j : add(i, k)$:

- $x_j 1 \to x_j \downarrow_i 0$ (we decremented lower position, so we must increment higher position (011 to 100, now at 1 to 0)),
- $x_j 0 \to y_k \uparrow 1$ (we incremented a position and can return and proceed to the next instruction),
- $x_j 1t \to [_i 1tp]_i y_k \uparrow 0s$ (incrementing 111 to 1000).

For the membrane i and instruction $j : sub(i, k, l)$:

- $x_j 1s \to y_k \uparrow 0s$ (we found position to decrement, proceed to the next instruction),
- $x_j 0 \to x_j \downarrow_i 1$ (1000 is decremented to 0111 and now we encountered a 0),
- $x_j 1tp \to z_k t\delta$ (decrementing the number of bits),
- $z_j st \to y_j t$ (after decremented the number of bits, remove s in the new highest-order position),
- $x_j 0t \to y_l \uparrow 0t$ (trying to decrement a zero)

4.4 Further optimizations

Could the simulation be optimized even more? Encoding the register value to a chain of membranes is not making full use of membrane structure. There are many options for a represention of an integer by a tree. For efficient implementation of the increment and decrement instructions, we need an encoding with a property that a local change in the value of the encoding of the entire tree corresponds to a local change in the value of the encodings of its child subtrees. Stein in 1999 [8] proposed a boustrophedonic variant of Cantor pairing function.

The implementation of a sequential active set P system simulating a register machine using this pairing function to encode child subtrees would be quite easy, but we would stick to the logarithmic time in the worst case (diagonal of the pairing function). Catalan pairing function [9] orders full binary trees by the number of nodes. The time would be logarithmic with a base 4, which is a slight improvement, but asymptotically still the same.

5 Modified membrane creation semantics

In this section we will investigate the effect of other semantics of membrane creation. The previous semantics assumed an explicit membrane creation rule. If the current membrane already contains child membrane with the same label as the membrane about to be created, then the rule is not applicable, and the membrane creation is aborted. Similar behavior is in the definition of sending objects to the child membrane. If such membrane does not exist, objects cannot be sent and the rule is not applicable.

These two behaviors are in fact complementary. It seems natural to join these two artificial rule abortions and provide a rule that will always be applicable if the precondition of left side inclusion is fulfilled.

5.1 Semantics inject-or-create

Therefore we will have no explicit membrane creation rule. Any rule which is sending some objects to child membrane labeled j will create child membrane j if it does not exist.

Formally, rules can be of form:

- $u \to w$, where $u \subseteq \Sigma$, $|u| \geq 1$, $w \subseteq (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})$ and $1 \leq j \leq m$,
- a dissolving rule $u \to w\delta$, where $u \subseteq \Sigma$, $|u| \geq 1$, $w \subseteq (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})$ and $1 \leq j \leq m$.

For a sequential active set P system $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, configuration $C = (T, l, c)$, membrane $d \in V(T)$ the rule $r \in R_{l(d)}$ is **applicable** iff:

- $r = u \to w$ and $u \subseteq c(d)$,
- $r = u \to w\delta$ and $u \subseteq c(d)$ and $d \neq r_T$.

TODO: example needed (with picture?)

We will now show how the computational completeness of this variant of membrane creation is achieved by simulating the register machine. The simulation is essentially the same as in section 4.3. All the rules which are sending objects into a child membrane are already assuming that the child membrane already exists. The only difference is in the rule for membrane creation: $x_j 1t \to [_i 1tp]_i y_k \uparrow 0s$. This rule is applied always in the innermost membrane with no child membranes. Modified simulation will therefore use rule $x_j 1t \to 1 \downarrow_i t \downarrow_i p \downarrow_i y_k \uparrow 0s$, which, when applied, creates a child membrane i, because no such child membrane exists.

Theorem 2 *Sequential active set P systems with inject-or-create semantics are computationally complete.*

Proof. Computational completeness is proved by a direct simulation of a register machine, which is also computationally complete. □

5.2 Semantics wrap-or-create

In this variant we stay with explicit membrane creation rule, but when the membrane with the same label is already contained in the current membrane, the rule remains applicable and the child membrane will be wrapped by a new membrane with the given contents. For example, applying the rule $a \to [_2 b]_2 c$ in the membrane 1 of membrane structure $[_1 a[_2 d]_2]_1$ would result in $[_1 c[_2 b[_2 d]_2]_2]_1$.

Again, we will show how to simulate the register machine. The simulation will be similar to the one defined in subsection 4.2, but with additional control objects similar to the second simulation 4.3.

For a register machine with m registers we will construct a sequential active set P system $(\Sigma, C_0, R_1, \dots R_{m+1})$, where

$$\Sigma = \{x_j \text{ for instructions with label } j\} \cup \{t_i, s_i \text{ for each register } \; i\}$$

. Skin membrane will be labeled with $m+1$, other labels correspond to registers 1 to m. C_0 will be the input word for the register machine encoded into a membrane structure by the following encoding:

For a configuration of register machine $(r_1, r_2, \dots r_m, ip)$ the membrane structure will consist of a skin membrane, which will contain m chains consisting of r_i membranes embedded one into another like in a Matryoshka doll with label i. Membranes with a child labeled i will contain a single object s_i. If the membrane has no child labeled i, it contains an object t_i. If $r_i = 0$ then t_i is in the skin membrane and there is no membrane with label i. Object representing the label of the current instruction (x_{ip}) is in the skin membrane.

We will have following rules in the skin membrane:

- $x_j s_i \to [_i s_i]_i s_i x_k$ for instruction $j : (add(i), k, \_)$,
- $x_j t_i \to [_i t_i]_i s_i x_k$ for instruction $j : (add(i), k, \_)$,
- $x_j t_i \to x_l t_i$ for instruction $j : (sub(i), k, l)$,
- $x_j s_i \to x_j \downarrow_i$ for instruction $j : (sub(i), k, l)$,

For the membrane i:

- $x_j \to x_k \delta$

For every *add* instruction there is just one rule applied in the simulation and for each *sub* instruction there is one or two instructions, depending on the register value. If $r_i > 0$ then the instruction enters the membrane labeled i and dissolves it, decreasing the number of stacked membranes with label i.

Theorem 3 *Sequential active set P systems with wrap-or-create semantics are computationally complete.*

Proof. Computational completeness is proved by a direct simulation of a register machine, which is also computationally complete. □

References

1. Paun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press, Inc., New York, NY, USA (2010)
2. Păun, G.: Computing with membranes. Journal of Computer and System Sciences **61**(1) (2000) 108 – 143
3. Ibarra, O., Woodworth, S., Yen, H.C., Dang, Z.: On sequential and 1-deterministic p systems. In Wang, L., ed.: Computing and Combinatorics. Volume 3595 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 905–914
4. Alhazov, A.: Properties of membrane systems. In Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., Verlan, S., eds.: Membrane Computing. Volume 7184 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 1–13
5. Kleijn, J., Koutny, M.: Membrane systems with qualitative evolution rules. Fundam. Inf. **110**(1-4) (jan 2011) 217–230
6. Alhazov, A.: P systems without multiplicities of symbol-objects. In: Information Processing Letters, accepted. (2005)
7. Păun, G.: Introduction to membrane computing. In Ciobanu, G., Păun, G., Pérez-Jiménez, M., eds.: Applications of Membrane Computing. Natural Computing Series. Springer Berlin Heidelberg (2006) 1–42
8. Stein, S.K.: Mathematics: the Man-made Universe. New York: McGraw-Hill (1999)
9. Stanley, R.P.: Enumerative Combinatorics. Wadsworth Publ. Co., Belmont, CA, USA (1986)