Polymorphic P Systems

Artiom Alhazov[1,2], Sergiu Ivanov[1,3], Yurii Rogozhin[1]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E-mail: {artiom,rogozhin,sivanov}@math.md
[2] IEC, Department of Information Engineering
Graduate School of Engineering, Hiroshima University
Higashi-Hiroshima 739-8527 Japan
[3] Technical University of Moldova, Faculty of Computers,
Informatics and Microelectronics,
Ştefan cel Mare 168, Chişinău MD-2004 Moldova

Abstract. Membrane computing is a formal framework of distributed parallel computing. In this paper we introduce a variant of the multiset rewriting model where the rules of every region are defined by the contents of interior regions, rather than being explicitly specified in the description of the system. This idea is inspired by the von Neumann's concept of "program is data" and also related to the research direction proposed by Gh. Păun about the cell nucleus.

1 Introduction

Membrane computing is a fast growing research field opened by Gh. Păun in 1998. It presents a formal framework inspired from the structure and functioning of the living cells. In this paper we define yet another, relatively powerful, extension to the model, which allows the system to dynamically change the set of rules, not limited to some finite prescribed set of candidates. There are three motives for this extension. First, our experience shows that "practical" problems need "more" computing potential than just computational completeness. Second, we attempt to import a very important computational ingredients into P systems, this time from the conventional computer science. Third, this extension correlates with the biological idea that different actions are carried out by different objects, which too can be acted upon. (This last idea was also considered in, e.g., [6] and [1], but there one represented each rule by a single objects, therefore all rules were still prescribed, though not their multiplicities.) Let us first explain these motives.

Most papers of the field belong to the following categories: 1) introducing different models and variants, 2) studying the computational power of different models depending on what ingredients are allowed and on the descriptional complexity parameters, 3) studying the computational efficiency of solving intractable problems (supercomputing potential) depending on the ingredients,

4) using membrane computing to represent and model various processes and phenomena, including but not limited to biology, 5) other applications.

There is a surprisingly big gap between the sets of ingredients needed to fulfill requirements in directions 2, 3, and the sets of ingredients demanded by other applications. For instance, very weak forms of cooperation between objects are often enough for the computational completeness, but many "practical" problems cannot be solved in a satisfactory way under the same limitations. This leads to the following question.

1.1 What is implicitly required in most "practical" problems?

We will mention just a few of these requirements below.

- **Determinism** or at least confluence. Clearly, the end user wants to obtain the answer to the specified problem in a single run of a system instead of examining infinitely many computations. This is a strong constraint, e.g., catalytic P systems and P systems with minimal symport/antiport are universal, while in the deterministic case non-universality is published for the first ones and claimed for the latter ones. Informally speaking, less computational power is needed to just compute the result than it is to also enforce choice-free behavior of the system.
- **Input/output**. Most of the universality results are formulated as generating languages or accepting sets of vectors, or in an even more restricted setup. There is no need to deal with input in the first case, and in the latter case the final configuration itself is irrelevant (except yes or no in case of the efficiency research). On the other side, both input and output are critical for most applications.
- **Representation**. Clearly, any kind of discrete information can be encoded in a single integer in some consistent way. However, a much more transparent data representation is typically required; even the intermediate configurations in a computation are expected to reflect a state of the object in the problem area.
- **Efficiency**. Suppose numbers are represented by multiplicities of certain objects. The number of steps needed to multiply two numbers by plain (cooperative) multiset processing is proportional to the result. If the multiset processing can be controlled by promoters/inhibitors/priorities, then the number of steps needed for multiplication is proportional to one of the arguments. However, many applications would ask for a multiplication to be performed in a constant number of steps. Similar problems appear for string processing.
- **Data structures**. Membrane computing deals with multisets distributed over a graph, while conventional computers provide random memory access and pointer operations, allowing much more complex structures to be built.

Some of these implicit requirements originate because the user wants a solution which is at least as good as the one that can be provided by conventional computers. We hope that the explanations of the above list have convinced the reader that this is often a challenge.

1.2 Program is data. Cell nucleus

In this paper we try to introduce another feature into the membrane computing. This time the inspiration is not biological, but rather is from the area of conventional computing. Suppose we want to be able to manipulate the rules of the system during its computation. A number of papers has been written in this direction (see, e.g., GP systems [6], rule creation [3], activators [1], inhibiting/deinhibiting rules [4] and symport/antiport of rules [5]), but in most of them the rules are predefined in the description of the system.

The most natural way to manipulate the rules is to represent them as data, treat this data as rules, and manipulate it as usual in P systems, in the spirit of von Neumann's approach. In membrane systems, the data consists of multisets, so objects should be treated as description of the rules. Informally, a rule j in a region i can be represented by the contents of membranes jL and jR inside i.

For instance, $\boxed{\begin{array}{c}1 : ab \to ac \\ 2 : a \to d \\ abbb\end{array}}_s$ becomes $\boxed{\boxed{ab}_{1L}\ \boxed{ac}_{1R}\ \boxed{a}_{2L}\ \boxed{d}_{2R} \\ abbb}_s$.

Changing the contents of regions jL and jR results in the corresponding change of the rule j. The next section illustrates this effect in Figure 1 and gives the formal definitions. We call such P systems polymorphic, by analogy with polymorphic, or self-modifying computer programs.

At the same time, if a membrane system is an abstraction inspired by the biological cell, one can view inner regions as an abstraction inspired by the cell nucleus; their contents correspond to the genes encoding the enzymes performing the reactions of the system. The simplicity of the proposed model is that we consider the natural encoding, i.e., no encoding at all: the multisets describing the rules are represented by exactly themselves. Therefore, we are addressing a problem informally stated by Gh. Păun in Section "Where Is the Nucleus?" of [7] by proposing a computational variant based on one simple difference: the rules are taken from the current configuration rather than from the description of the P system itself.

The idea of a nucleus was also considered in [9], but such a presentation had the following drawbacks. First, one described the dynamics of the rules in a high-level programming language (good for simulators, but otherwise too powerful extension having the power of conventional computers directly built into the definition). Second, this dynamics of the rules did not depend on the actual configuration of the membrane system (no direct feedback from objects to rules). In the model presented in this paper, the dynamics of rules is defined by exactly the same mechanism as the standard dynamics of objects.

2 Definitions

We refer the reader to [8] for the standard preliminaries of membrane computing. We denote the family of recursively enumerable sets of non-negative integers by NRE.

We define a polymorphic P system as a tuple

$$\Pi = (O, T, \mu, w_s, w_{1L}, w_{1R}, \cdots, w_{mL}, w_{mR}, \varphi, i_{out}),$$

where O is a finite alphabet, μ is a tree structure consisting of $2m+1$ membranes, bijectively labeled by elements of $H = \{s\} \cup \{iL, iR \mid 1 \leq i \leq m\}$ (the skin membrane is labeled by s; we also require for $1 \leq i \leq m$ that the parent membrane of iL is the same as the parent membrane of iR), w_i is a string describing the contents of region i, $1 \leq i \leq m$, and φ is a mapping from $\{1, \cdots, m\}$ to the features of the rules described below. The set $T \subseteq O$ describes the output objects, while $i_{out} \in H \cup \{0\}$ is the output region (0 corresponds to the environment).

Notice that the rules of a P system are not explicitly given in its description. Essentially, such a system has m rules, and these rules change as the contents of regions other than skin changes. Initially, for $1 \leq i \leq m$ rule $i : w_{iL} \to (w_{iR}, \varphi(i))$ belongs to the region defined by the parent membrane of iL and iR. If w_{iL} is empty, then the rule is considered disabled. For every step of the computation each rule is defined in the same way, taking the current contents of iL and iR instead of initial ones.

In what follows we mainly consider a single feature, i.e., target indications. In this case, the range of φ is $Tar = \{in_i \mid i \in H\} \cup \{here, out\}$. We denote the class of all polymorphic P systems with cooperative rules and target indications and at most k membranes by

$$OP_k(polym_{+d}(coo), tar).$$

In the notation above, the number k is replaced by $*$ or omitted if no bound is specified. The subscript $+d$ means that the rules can be disabled; we write $-d$ instead, if w_{iL} is never empty for $1 \leq i \leq m$ during any computation. We prefix this notation with D if we restrict the class to the deterministic systems (for every input if it is specified, see below).

A computation is a sequence of configurations starting in the initial configuration, corresponding to the transitions induced by non-deterministic maximally parallel application of rules; it is called halting if no rules are applicable to the last configuration. In the latter case the multiset of objects from T in region i_{out} is called the result.

If we want to compute instead of generating, we extend the tuple Π by the description of the input as follows. In the definition of the P system, we insert the input alphabet $\Sigma \subset O$ after O and we insert the input region i_{in} after φ. In this case, the input multiset over Σ is added to $w_{i_{in}}$ before the computation starts. If we want to accept instead of computing, we remove T and i_{out} from the description of the P system; the input is considered accepted if and only if the system may halt. If we want to decide instead of computing, we construct a system that always halts with either `yes` or `no` in the output region, such that this answer uniquely depends on the input; the input is accepted if and only if the answer is `yes`. Speaking about the time complexity is more appropriate for deciding than for accepting.

The set of numbers or vectors generated by a P system Π is denoted by $N(\Pi)$ or $Ps(\Pi)$, respectively. In the accepting case, we write $N_a(\Pi)$ or $Ps_a(\Pi)$. In the deciding case, we write $N_d(\Pi)$ or $Ps_d(\Pi)$. If the computation of Π is deterministic for every input, then the partial function computed by Π is denoted by $f(\Pi)$. In this way, the entire class of polymorphic P systems with cooperative rules and target indications, allowing disabled rules, with at most k membranes, defines a family of sets of numbers, of sets of vectors or of functions, respectively denoted by

$$NOP_k(polym_{+d}(coo), tar),\ PsOP_k(polym_{+d}(coo), tar),$$
$$fDOP_k(polym_{+d}(coo), tar).$$

In a similar way it is possible to replace cooperative rules with a more restricted set, remove target indications or add more features to the polymorphic P systems, modifying the notation accordingly. It is even possible to consider completely different rules instead of rewriting, e.g., symport/antiport rules, but we do not address such a topic here.

We illustrate the definitions by the following example.

Example 1. A P system with a superexponential growth.

$$\Pi_1 = (\{a\}, \{a\}, \mu, a, a, a, a, a, a, aa, \varphi, 1),\ \text{where}$$
$$\mu = [\,[\ ]_{1L}[\ [\ ]_{2L}[\ [\ ]_{3L}[\ ]_{3R}\,]_{2R}\,]_{1R}\,]_s,$$
$$\varphi(i) = here,\ 1 \le i \le 3.$$

Naturally, contents of membranes $1L$, $2L$, $3L$ is never changed because they are elementary and no rules have the corresponding target indications, and their initial contents is a, so the system is non-cooperative, and the rules are never disabled. Since only one rule acts in each of the regions s, $1R$, $2R$, the system is deterministic. From all above we conclude that $\Pi_1 \in DOP_7(polym_{-d}(ncoo))$, a quite restricted class.

This system never halts. Its interesting aspect, however, is the growth of the number of objects in the skin. We claim that at step n the skin contains $2^{n(n-1)(n-2)/6}$ objects, so the growth function is an exponential of a polynomial. Indeed, this is not difficult to see by starting from the elementary membranes and going outside.

The contents of $3R$ is aa and it never changes. Region $2R$ initially contains a and undergoes rule $a \to aa$ every step, so its contents at step n is a^{2^n}. Region $1R$ initially contains a and undergoes rule $a \to a^{2^n}$ at step n, so its contents at step n is $a^{2^{n(n-1)/2}}$. The skin originally contains a and at step n rule $a \to a^{2^{n(n-1)/2}}$ is applied, so its contents at step n is $a^{2^{n(n-1)(n-2)/6}}$, see Figure 1 for the actual illustration of the computation and for the proof of the result.

This growth is faster than that of any non-polymorphic P systems, which is bounded by the exponential Ic^n, where I is the initial number of objects in the system and c is the maximum ratio for all rules of the right side size and its left side size. It is not difficult to see that the growth function of a polymorphic

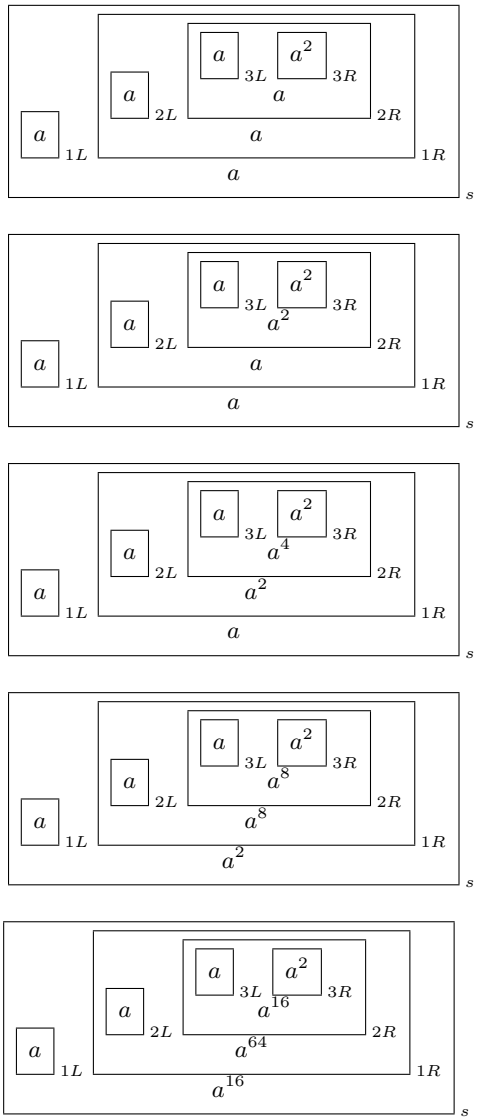


Fig. 1. The computation of Π_1 from Example 1. If the number of objects a in regions $3R$, $2R$ $1R$, s at step n is (x_n, y_n, z_n, t_n), respectively, then $(x_0, y_0, z_0, t_0) = (2, 1, 1, 1)$ and $(x_{n+1}, y_{n+1}, z_{n+1}, t_{n+1}) = (x_n, y_n x_n, z_n y_n, t_n z_n)$.
Following just this quadruple, the computation can be represented as $(2, 1, 1, 1) \Rightarrow (2, 2, 1, 1) \Rightarrow (2, 4, 2, 1) \Rightarrow (2, 8, 8, 2) \Rightarrow (2, 16, 64, 16) \Rightarrow (2, 32, 1024, 1024) \Rightarrow (2, 64, 32768, 1048576) \Rightarrow \cdots$.
The exponents of the closed form formula $(2, 2^n, 2^{n(n-1)/2}, 2^{n(n-1)(n-2)/6})$ can be verified as follows. $n + 1 = n + 1$, $(n + 1)n/2 = n(n - 1)/2 + n$, $(n + 1)n(n - 1)/6 = n(n - 1)(n - 2)/6 + n(n - 1)/2$.

P system without target indications is bounded by $Ic^p(n)$, where I and c are defined as above and p is a polynomial whose degree equals the depth of the membrane structure minus one.

3 Results

As long as full cooperation is allowed, the universality of polymorphic P system is not difficult to obtain, even without the actual polymorphism (i.e. without ever modifying rules) and without the use of target indications. The upper bound on the number of membranes needed is one plus twice the number of rules, because in the polymorphic P systems the rules can only be represented by pairs of membranes. We recall that in [2] one presents a strongly universal P system with 23 rules. Hence, the following theorem holds.

Theorem 1. $NOP_{47}(polym_{-d}(coo)) = NRE$.

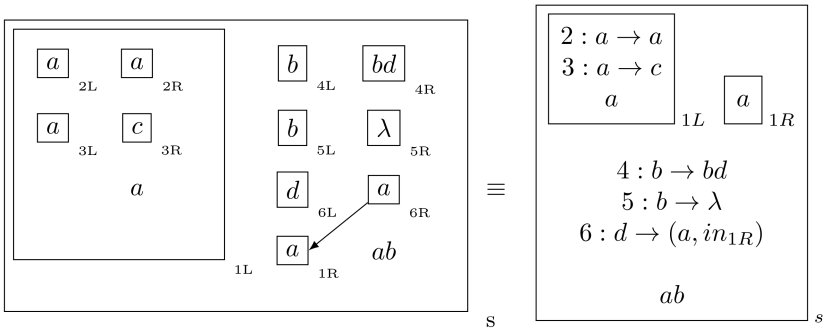
Proof. The claim is fulfilled by taking the one-membrane construction from the main result in [2] and replacing each of the 23 rules by two membranes containing the left-hand side and the right-hand side of that rule.

In the rest of the paper we focus on the efficiency of computations performed by polymorphic P systems, using the time complexity terms. We devote special attention to fast generating and deciding factorials, because they best illustrate constant-time multiplication where the factors are not known in advance and are even changing during the computation. First, we present a non-cooperative system generating "slightly" more than factorials, using target indications. It is a bit more complicated than Π_1 because, firstly, we need to multiply by numbers that grow linearly, and secondly, we want the system to halt.

Example 2. A polymorphic P system from $OP_{13}(polym_{-d}(ncoo), tar)$ which generates $\{n! \cdot n^k \mid n \geq 1, k \geq 0\}$.

$$\Pi_2 = (\{a, b, c, d\}, \{a\}, \mu, ab, a, a, a, a, a, c, b, bd, b, \lambda, d, a, \varphi, 1), \text{ where}$$
$$\mu = [\,[\,[\,]_{2L}[\,]_{2R}[\,]_{3L}[\,]_{3R}\,]_{1L}[\,]_{1R}[\,]_{4L}[\,]_{4R}[\,]_{5L}[\,]_{5R}[\,]_{6L}[\,]_{6R}\,]_s,$$
$$\varphi(i) = here,\ 1 \leq i \leq 5,\ \varphi(6) = in_{1R}.$$

The initial configuration can be graphically represented as shown below. In fact, such a graphical representation gives a complete description of Π_2 except the output alphabet and the output region. The target indication of a rule (here rule 6 in $1R$) may be indicated by an arrow, in this case from $6R$ to $1R$ (keeping in mind that the reactants of the rule are taken from the parent region of the membranes describing the rule, in this case, from region 1). At the right we give a simplified representation of the same system by replacing pairs of membranes with constant contents by the rules written explicitly (this is just a different representation, so-called "syntactic sugar", and we still count such rules as pairs of membranes). Rule 1 is not written with the rule syntax because the contents of both $1L$ and $1R$ will change.



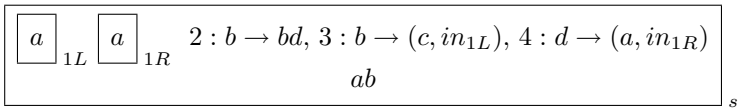
The essence of the functioning of Π_2 is the following. Rules 4 and 6 lead to incrementation of the number of copies of a in $1R$ (the number of copies of a in the skin does not change during the first two steps). The system will apply rule 4 for $n-1 \geq 0$ times and then rule 5 (applying rule 5 is necessary for the system to halt). Suppose that all this time rule 2 has been applied in region $1L$. Then, the number of objects in region $1R$ will grow linearly, and subsequent applications of a dynamic rule $1 : a \to a^i$, $1 \leq i \leq n$ will produce $a^{n!}$ in the skin. After that, the number of objects a in the skin will be multiplied by n until rule 3 is applied, because $1 : c \to a^n$ will be no longer applicable, halting with the skin only containing objects a their number being an arbitrary number of the form $n! \cdot n^k$. Now assume that rule 3 has been applied earlier, effectively stopping the multiplication of the number of objects a in the skin before the incrementation of objects a in $1R$ is finished. In that case the multiplicity of objects a in the skin will be just a factorial of a smaller number, and the system will evolve by application of rules 4, 6 until rule 5 is applied, without affecting the result. Notice that the time complexity (understood as the shortest computation producing the corresponding result) of generating $n! \cdot n^k$ is only $n + k + 1$.

To generate exactly $\{n! \mid n \geq 1\}$ we need to stop the multiplication when we stop the increment. This seems impossible without cooperative rules.

Example 3. A P system from $OP_9(polym_{-d}(coo), tar)$ generating $\{n! \mid n \geq 1\}$.

$$\Pi_3 = (\{a, b, c, d\}, \{a\}, \mu, ab, a, a, b, bd, b, c, d, a, \varphi, 1), \text{ where}$$
$$\mu = [\,[\,[\,\,]_{1L}[\,\,]_{1R}[\,\,]_{2L}[\,\,]_{2R}[\,\,]_{3L}[\,\,]_{3R}[\,\,]_{4L}[\,\,]_{4R}\,]_s,$$
$$\varphi(i) = here,\ 1 \leq i \leq 2,\ \varphi(3) = in_{1L},\ \varphi(4) = in_{1R}.$$

This system is very similar to Π_2. There are only the following differences. First, rules $a \to a$ and $a \to c$ are removed from region $1L$. Second, instead of erasing b in the skin, the corresponding rule sends object c to region $1L$, which stops both increment (b is erased) and multiplication ($1 : ac \to a^n$ is not applicable in the skin). Ironically, this system never applies any non-cooperative rule, but the non-cooperative feature seems unavoidable in order to stop the computation in the synchronized way. A compact graphical representation of Π_3 is given below.



Now we proceed to describing a P system generating $\{2^{2^n} \mid n \geq 0\}$ in $O(n)$ steps. Since the growth of polymorphic P systems without target indications is bounded by exponential of polynomials, the system below grows faster than any of them. Moreover, it produces the above mentioned result by halting.

It is also worth noting that even polymorphic P systems cannot grow faster than exponential of exponential in linear time, because if a system has $n+n+1 > 3$ objects at some step, then it cannot have more than n^2+n+1 objects in the next step. Indeed, consider that some rule r is applied for n times; let its left side contain x objects and let its right side contain y objects. Then, $x+y$ objects are needed to describe the rule and they transform nx other objects into ny objects. It is not difficult to see that the growth is maximal if $x = 1$ and $y = n$. Since n^2+n+1 is less than the square of $n+n+1$, and iterated squaring yields the growth which is exponential of exponential, it is not possible to grow faster. The system below grows three times slower than this bound.

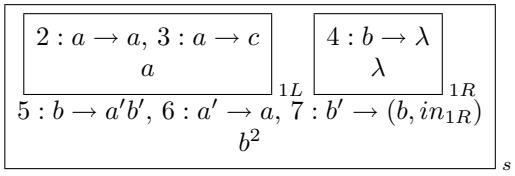
Example 4. A P system from $OP_{15}(polym_{-d}(ncoo), tar)$ generating numbers from $\{2^{2^n} \mid n \geq 0\}$ in $3n+2$ steps.

$$\Pi_4 = (\{a, b, a', b', c\}, \{a\}, \mu, b^2, a, \lambda, a, a, a, c, b, \lambda, b, a'b', a', a, b', b, \varphi, 1), \text{ where}$$

$$\mu = [\,[\,[\,]_{2L}[\,]_{2R}[\,]_{3L}[\,]_{3R}\,]_{1L}[\,[\,]_{4L}[\,]_{4R}\,]_{1R} \prod_{i=5}^{7} ([\,]_{iL}[\,]_{iR})\,]_s,$$

$$\varphi(i) = here,\ 1 \leq i \leq 6,\ \varphi(7) = in_{1R}.$$

The desired effect is obtained by iterated squaring. By rules 5, 6, 7, in two steps each copy of b in the skin changes into a and also sends a copy of b in region $1R$. In the next step, if region $1L$ still contains an a, each copy of a in the skin is replaced by the contents of region $1R$, and the process continues. Therefore, if we had b^k in the skin at some step, then in two steps we will have a^k in the skin and rule 1 will be of the form $a \to b^k$, yielding b^{k^2} in the third step. The iteration continues while rule 2 is being applied in region $1L$. When rule 3 is applied, the cycle stops because rule $1 : c \to b^k$ will not be applicable, and the result is given as the multiplicity of objects a in the skin. Clearly, $2 = 2^{2^0}$ and $2^{2^{n+1}} = (2^{2^n})^2$, so the systems generates 2^nth powers of 2. We underline that no cooperation was used in this case. A compact graphical representation of this system is shown below.



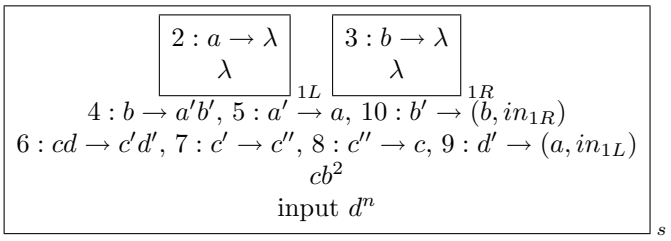
We remind the reader that the picture above represents a system with 15 membranes because the rules notation is simply a compact way to represent pairs of membranes. Note that one rule could have been saved if the right side of the rule were allowed to have objects with different target indications, but this issue does not affect the computational power, only the number of rules, whereas the definitions are much simpler. Another rule could be saved at the price of using a cooperative rule to stop the computation instead of rules 2 and 3, like in the previous example.

We now proceed to tasks which are more difficult than generating, namely, deciding a set of numbers or computing a function in a deterministic way. We illustrate the first case by modifying the previous example. We use an additional ingredient compared to the previous systems: we rely on disabling a rule by emptying the region describing its left side. Although we expect that this ingredient does not change the computational power of the systems, we use it in order to have smaller constructions.

Example 5. A deterministic P system from $OP_{15}(polym_{+d}(coo), tar)$ computing the function $n \longrightarrow 2^{2^n}$ in $3n+2$ steps.

$$\begin{aligned}
\Pi_5 = (&\{a, b, a', b', c, d, d'\}, \{d\}, \{a\}, \mu, cb^2, \lambda, \lambda, a, \lambda, b, \lambda,\\
&b, a'b', a', a, cd, c'd', c', c'', c'', c, d', a, b', b, \varphi, 1, 1), \text{ where}\\
\mu = [\,&[\,[\,[\,]_{2L}[\,]_{2R}\,]_{1L}[\,[\,]_{3L}[\,]_{3R}\,]_{1R} \prod_{i=4}^{10} ([\,]_{iL}[\,]_{iR})\,]_s,\\
&\varphi(i) = here,\ 1 \leq i \leq 8,\ \varphi(9) = in_{1L},\ \varphi(10) = in_{1R}.
\end{aligned}$$

This system works like Π_4 from the previous example. We only focus on the differences. The previous system used non-deterministic choice between rules 2 and 3 to continue the computation or to stop it. In this case, squaring stops by itself due to the rule $2 : a \to \lambda$, so producing object a in region $1L$ activates one squaring. The most important difference is that the number n is given as input into the skin, by the multiplicity of objects d. Moreover, besides two copies of b the skin initially contains an object c, responsible for counting until n by consuming objects d and activating the squaring routine the corresponding number of times. The cycle takes 3 steps, see rules 6, 7, 8, 9. When object c has no more copies of d to consume, the result is obtained as the multiplicity of objects a in the skin. We show a compact graphical representation of Π_5 below.



Note that this system uses cooperation for counting and disabling the rules for easier control. We leave it as an exercise for the reader to construct a P system Π_5' computing the same function without disabling rules. Hint: as long as objects a only appear in the skin every third step, there is no need to disable rule 1 while the computation is in progress. Object c can deterministically subtract d and perform its appearance checking. Finally, when there are no copies of d in the skin, moving c into $1L$ will make rule 1 inapplicable without the need to disable it by emptying its left side.

Now we give an example of a P system deciding a set of numbers. It works deterministically and produces an object `yes` or `no` in the skin, depending on whether the input number belongs to the specified set. We also emphasize its time complexity.

Example 6. A deterministic P system from $OP_{37}(polym_{-d}(coo), tar)$ deciding the set $\{n! \mid n \geq 1\}$. A number $k \leq n!$ is decided in at most $4n$ steps, i.e., in a sublogarithmic time with respect to k.

$$\begin{aligned}\Pi_6 = (&\{a, b, c_0, c_1, c_2, A, A', B, B', p_0, p_1, p_2, p_3, \texttt{yes}, \texttt{no}\}, \{a\}, \{\texttt{yes}, \texttt{no}\}, \mu,\\ &p_0c_0, a^2, b, b, a, c_1, c_2, c_2, \lambda, p_0, AABp_1, Aa, A'a, Bb, B'b, p_1, p_2,\\ &p_2B'AA, p_3d, p_2B'A'A, f\texttt{no}, p_2B'A'A', f\texttt{no}, p_2BAA, f\texttt{no},\\ &p_2BA'A, f\texttt{yes}, p_2BA'A', f\texttt{no}, p_3, p_0c_0, c_0, c_1, d, a, f, f, \varphi, 1, 1), \text{ where}\end{aligned}$$

$$\mu = [\,[\,]_{1L}[\,]_{3L}[\,]_{3R}[\,]_{4L}[\,]_{4R}]_{2L}[\,]_{2R} \prod_{i=5}^{18} ([\,]_{iL}[\,]_{iR}) \;]_s,$$

$$\varphi(i) = here,\ 1 \leq i \leq 15,\ \varphi(16) = in_{2L},\ \varphi(17) = \varphi(18) = in_{1L}.$$

The work of Π_6 consists of iterated division of the input a^k. Each cycle consists of 4 steps. The role of object c_0 is to enter into $2L$ by rule 16, thus preventing rule $2 : b \to a$ to work during the second and the third step of the cycle ($bc_1 \to a$ is not applicable, changing by rule 3 to $bc_2 \to a$, which is also not applicable, and then being restored by rule 4).

Object p_0 marks the steps and produces the necessary objects for checking some numbers, and finally produces symbols to increment the divisor or to modify the dividing rule to stop the computation, and give the answer, as follows. Suppose that the input is a^k. In the first step, p_0 changes into p_1, also producing checkers AAB. In the same time, the number k will be divided by n (initially $n = 2$) by rule $1 : a^n \to b$, changing a^k into $b^x a^y$, where x is the quotient and y is the remainder.

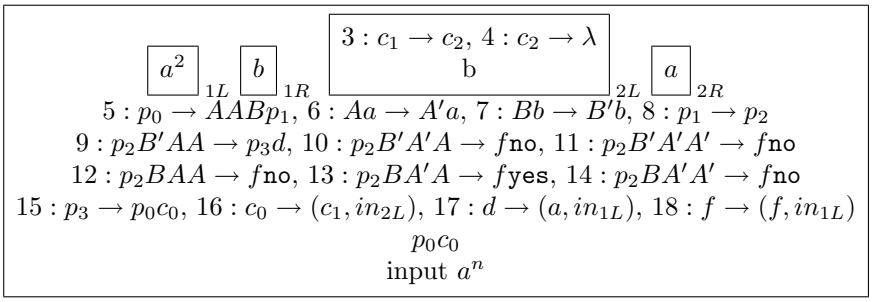
In the second step, p_1 changes into p_2, waiting for the checkers. The role of the checking rules $6 : Aa \to A'a$ and $7 : Bb \to B'b$ is to test the multiplicity of the remainder and the quotient, respectively. Hence, object B will be primed if $x > 0$. Notice that since there are two copies of A in the system, the number of symbols A that will be primed is $\min(y, 2)$. Thus, there are 6 combinations of symbols A and B, primed or not.

In the third step, we distinguish two special cases. If $x > 0$ and $y = 0$, then the input is a multiple of the currently computed factorial, and we proceed

to the next iteration by rule $9 : p_2B'AA \to p_3d$. If $x = 0$ and $y = 1$, then the input is equal to the previously computed factorial, and the system gives the positive answer by the rule $13 : p_2BA'A \to f\texttt{yes}$. Four other combinations correspond to detecting that the input is not equal to a factorial of any number (two cases correspond to non-zero quotient and non-zero remainder, the third case corresponds to the input being zero, and the last case corresponds to a multiple of some factorial which is smaller than the next factorial), so $f\texttt{no}$ is produced.

In the fourth step, rule $2 : b \to a$ is used, so the quotient is ready to be divided again. Object f is used to stop the computation by rule 18, since rule $1 : a^n f \to b$ is not applicable. In case we proceed to the next iteration, the role of object d is to increment the multiplicity n of objects a in region $1L$, and object p_3 changes back to p_0 and produces a new copy of c_0 for the next cycle.

Below is a compact graphical representation of Π_6.



We summarize some of the results we obtained as follows.

Theorem 2. *There exist*

- *A strongly universal P system from* $OP_{47}(polym_{-d}(coo))$*;*
- *A P system* $\Pi_1 \in DOP_7(polym_{-d}(ncoo))$ *with a superexponential growth;*
- *A P system* $\Pi_2 \in OP_{13}(polym_{-d}(ncoo), tar)$ *such that* $N(\Pi_2) = \{n! \cdot n^k \mid n \geq 1,\ k \geq 0\}$ *and the time complexity of generating* $n! \cdot n^k$ *is* $n + k + 1$*;*
- *A P system* $\Pi_3 \in OP_9(polym_{-d}(coo), tar)$ *such that* $N(\Pi_3) = \{n! \mid n \geq 1\}$ *and the time complexity of generating* $n!$ *is* $n + 1$*;*
- *A P system* $\Pi_4 \in OP_{15}(polym_{-d}(ncoo), tar)$ *such that* $N(\Pi_4) = \{2^{2^n} \mid n \geq 0\}$ *and the time complexity of generating* 2^{2^n} *is* $3n + 2$*;*
- *A P system* $\Pi'_5 \in DOP_*(polym_{-d}(coo), tar)$ *such that* $f(\Pi_5) = (n \longrightarrow 2^{2^n})$ *and the time complexity of computing* $n \longrightarrow 2^{2^n}$ *is* $O(n)$*;*
- *A P system* $\Pi_6 \in DOP_*(polym_{-d}(coo), tar)$ *such that* $N_d(\Pi_6) = \{n! \mid n \geq 1\}$ *and the complexity of deciding any number* k*,* $k \leq n!$ *does not exceed* $4n$*.*

Moreover, polymorphic P systems can grow faster than any non-polymorphic P systems, whereas even non-cooperative polymorphic P systems with targets can grow faster than any polymorphic P systems without targets.

4 Discussion

We proposed a variant of the rewriting model of P systems where the rules are represented by objects of the system itself and thus can dynamically change. This yields a mechanism whose idea is similar to the idea of the functioning of the cell nucleus (i.e., DNA represent the proteins performing certain functions on the objects including DNA), except our formalism is more elegant mathematically because of its simplicity and because we only used a trivial encoding (which is no encoding at all, except the left and right parts of the rule are given in dedicated membranes).

This variant also has a number of connections to the conventional computing, since the "program" can be changed by manipulating data (cf. von Neumann architecture vs Harvard architecture). A number of possible extensions is suggested in the Definition section of the paper.

Polymorphic P systems are universal (with 47 membranes) because non-polymorphic P systems are universal. While the growth of non-polymorphic P systems is bounded by exponential, polymorphic P systems without target indications can grow faster, bounded by an exponential of polynomials, and polymorphic P systems with target indications can grow even faster, bounded by an exponential of exponentials.

Non-cooperative polymorphic P systems can generate non-context-free sets of numbers. Cooperative polymorphic P systems can multiply numbers in constant time and generate factorials of n or exponentials of exponentials of n in time $O(n)$, which is a very important advantage over non-polymorphic P systems.

An especially interesting case is that of deciding if the input belongs to a given set, e.g., $\{n! \mid n \geq 1\}$. While non-polymorphic P systems cannot even grow with factorial speed, not to speak about halting or verifying the input, we have shown that polymorphic P systems can decide factorials in time $O(n)$. This implies that there exist infinite sets of numbers that are accepted in a time which is sublinear with respect to the size of the input in binary representation (without cheating by only examining a part of the input to accept).

Many questions are left open, we mention three questions here. First, we find it particularly interesting what is the exact characterization of the most restricted classes we defined, like $OP_*(polym_{-d}(ncoo))$. On the other hand, it seems interesting how the (general classes of) polymorphic P systems can solve the problems of real applications which non-polymorphic P system are not suitable for. Another question is whether the polymorphic P systems can effectively use superexponential growth and dynamics of rule description to solve intractable problems in polynomial time without dividing or creating membranes. Conjecture: no, because the total number of rules (counting rules in different regions as different) cannot grow.

Acknowledgments. The authors acknowledge the support by the Science and Technology Center in Ukraine, project 4032. Artiom Alhazov also acknowleges the support of the Japan Society for the Promotion of Science and the Grant-in-Aid for Scientific Research, project 20·08364.

References

1. Alhazov, A.: A Note on P Systems with Activators. In: Păun, Gh., Riscos-Núñez, A., Romero-Jimenez, A., Sancho-Caparrini, F. (eds.): Second Brainstorming Week on Membrane Computing, RGNC Report 01/2004, University of Sevilla, 16–19 (2004)
2. Alhazov, A., Verlan, S.: Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. Technical Report, 862, Turku Centre for Computer Science, Turku (2008), `http://tucs.fi` and Theoretical Computer Science (submitted)
3. Arroyo, F., Baranda, A., Castellanos, J., Păun, Gh.: Membrane Computing: The Power of (Rule) Creation. Journal of Universal Computer Science, vol.8, 3, 369–381 (2002)
4. Cavaliere, M., Ionescu, M., Ishdorj, T.-O.: Inhibiting/de-inhibiting Rules in P systems. In: Preproceedings of the Fifth Workshop on Membrane Computing (WMC5), Milano, 174–183 (2004)
5. Cavaliere, M., Genova, D.: P systems with Symport/Antiport of Rules. Journal of Universal Computer Science, vol. 10, 5, 540–558 (2004)
6. Freund, R.: Generalized P Systems. In: Fundamentals of Computation Theory, FCT'99, Iaşi. Ciobanu, G., Păun, Gh. (eds.), LNCS, vol. 1684, Springer, 281–292 (1999)
7. Păun, Gh.: 2006 Research Topics in Membrane Computing. In: Fourth Brainstorming Week on Membrane Computing, vol. II, Gutiérrez-Naranjo, M.A., Păun, Gh., Riscos-Núñez, A., Romero-Campero, F.J. (eds.), Fénix Editora, Sevilla, 235–252 (2006)
8. Păun, Gh.: Membrane Computing. An Introduction, Springer (2002)
9. Ştefănescu, G., Şerbănuţa, T., Chira, C., Roşu, G.: P Systems with Control Nuclei. In: Preproceedings of the Tenth Workshop on Membrane Computing (WMC10), Curtea de Argeş, 361–365 (2009)
10. P systems webpage. `http://ppage.psystems.eu/`