Event–Related Outputs of Computations in P Systems

Matteo Cavaliere[1], Rudolf Freund[2], Alexander Leitsch[2], Gheorghe Păun[1,3]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: `martew@inwind.it, gpaun@us.es`

[2] Department of Computer Science, Technische Universität Wien
Favoritenstraße 9, A-1040 Wien, Austria
E-mail: `rudi@emcc.at, leitsch@logic.at`

[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
E-mail: `george.paun@imar.ro`

Summary. We briefly investigate the idea to consider as the result of a computation in a P system the number of steps elapsed between two events produced during the computation. Specifically, we first consider the case when the result of a computation is defined in terms of events related to using rules, introducing objects, or meeting objects. Universality is easily obtained in each case for symport/antiport P systems. Then, we address the case when the number computed by a system is the length of a computation itself. We obtain a few results both for catalytic multiset-rewriting and for symport/antiport systems (in each case, also with using membrane dissolution) showing that non-semilinear sets of vectors can be computed in this way. A general non-universality result is proved for this case – no system, of any type, can have as the length of its halting computations all sets of numbers computable by Turing machines. The general problem, of characterizing the sets of numbers computed in this way, remains open.

1 Introduction

Input and output of computations in a P system usually are defined in terms of multiplicities of sequences of objects, entering or leaving the system during a computation. Recently, in [5], "signals" were considered for defining the input in a P system used in the recognizing mode: the number of steps in between the moments when two specified objects are introduced in the system is taken as the number to be computed (accepted) by the system.

We here address a "dual" question of defining the results of a computation in a similar way. Actually, we deal with a more general problem, stated in [13] as

Problem W. The main idea is not to use a *support* for information, such as the multiplicity of objects, but to take certain *events* and to relate the information we compute (e.g., numbers) with the occurrence of these events. "For instance, let us distinguish some special objects and count the number of times when they meet each other in any membrane of the system. Precise pairs of objects can be given in advance. Then, we can consider as events the use of certain rules – which leads to a sort of control word, or Szilard word, associated with a computation. Another idea is not to count the events themselves, but to look for some parameters related to events. One possibility is to count the number of steps elapsed in between two specified events. For instance, the very length of a computation (the number of steps from an initial configuration to a halting one) can be taken as the number computed by the system."

Several of these questions (the problem from [13] also contains other questions) will be investigated in this paper, and some further – sometimes, more technical – related questions will be investigated, too. The considerations given below are preliminary ones, but we already get two general conclusions:

(i) If the events can be "freely chosen" (such as the use of certain rules, the entering/exiting of certain objects into/from the system, the number of times two objects meet), then it is easy to obtain universality results, by adapting known universality proofs.

(ii) If we take the length of a computation as the result of the computation, then the question seems to be much more difficult. Anyway, all semilinear sets of numbers can easily be computed by simple P systems, and both with multiset-rewriting rules (and with catalysts) and with symport/antiport rules we can also compute non-semilinear sets of numbers. However, no class of systems can be universal in this second case, because of the simple observation that the space created during a computation of length n is at most of the order of k^n, for a constant k depending on the type of P system under consideration. A more precise characterization of the sets of numbers computable in this way remains to be found.

2 Prerequisites

The reader is assumed to be familiar with membrane computing, for instance, from [12], as well as with basic elements of formal language theory, e.g., from [16]. We only mention here that by REG, CF, CS, RE we denote the families of regular, context-free, context-sensitive, and recursively enumerable languages, respectively. The length set of a language $L \subseteq V^*$ is denoted by $length(L)$.

A *register machine* will be denoted by $M = (n, H, l_0, j_h, R)$, where n is the number of registers, H is the set of labels, l_0 is the start label, l_h is the halt label, and R is the set of labelled instructions. We here consider deterministic register machines working in the accepting mode: the computation starts with a number m being introduced in the first register, with all other registers being empty, using the instruction labelled with l_0; if the instruction l_h : `halt` is reached, then the number m is accepted.

In turn, a *P system with symport/antiport rules* is given in the form $\Pi = (n, O, \mu, w_1, \ldots, w_n, E, R_1, \ldots, R_n, i_o)$, where $n \geq 1$ is the degree of the system, O is the set of objects, μ is the membrane structure (of degree n, represented by a sequence of matching labelled parentheses), $w_1, \ldots, w_n$ are the multisets of objects initially present in the regions of μ, E is the set of objects present in the environment, $R_1, \ldots, R_n$ are the sets of rules associated with the n membranes, and i_o is the output region. The rules are written in the form $(x, in), (x, out)$ (symport rules) and $(x, out; y, out)$ (antiport rules), where x, y are strings over O representing multisets of objects. When the rules have promoters or inhibitors (always as objects), we write, e.g., $(x, out; y, in)|_a$ to denote that a is a promoter of the antiport rule $(x, out; y, in)$, and $(x, out; y, in)|_{\neg a}$ to denote that a is an inhibitor of the antiport rule $(x, out; y, in)$; a similar notation is used for symport rules.

In the case of multiset-rewriting P systems, the component E is no longer necessary, but, if the system is catalytic, then a component $C \subseteq O$, of catalysts, is added. Thus, such a system is given in the form $\Pi = (n, O, C, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_o)$, with the rules of the forms $a \to v$ or $ca \to cv$, where $c \in C$, $a \in O$, and $v \in (O \times \{here, in, out\})^*$. If the membrane dissolving action is used, then we write $a \to v\delta,\ ca \to cv\delta$, respectively.

In this paper we also consider the membrane dissolution for symport and antiport rules, and we present such rules in the form $(x, in)\delta$, $(x, out)\delta$, $(x, out; y, in)\delta$; after using the respective rules, the membrane with which they are associated is dissolved – with the standard effect in membrane computing (and with the restriction of never dissolving the skin membrane).

Below, because we do not need an output region, the respective component of systems is not mentioned.

3 Symport/Antiport Systems and "Freely Chosen" Events

Let us start in a sort of "reverse" manner: instead of giving first definitions and then examples/results, we first consider a specific symport/antiport P system, and then we examine it in order to see which events can be considered in order to obtain the desired results.

Let $M = (n, H, l_0, l_h, R)$ be a register machine accepting the set of numbers $N(M)$, and construct the P system

$$\Pi = (1, O, [_1\ ]_1, c_0, E, R_1)$$

with the following rules:

$$\begin{aligned} r_1 &: (c_0, out; ca_1, in), \\ r_2 &: (c, out; ca_1, in), \\ r_3 &: (c, out, l_0, in), \end{aligned}$$

as well as rules for simulating the register machine M, e.g., as in [4] or [5] (one-membrane systems were proved to be universal in these papers). Thus, it is understood that the alphabet O of objects contains all objects used for simulating the register machine (always, the contents of a register r is represented by the number of copies of a specific object a_r, and the labels are also objects). Similarly, in the environment (set E) we have all objects necessary for the computation.

The system Π works as follows. Starting from the initial configuration, containing only the object c_0, one first introduces a number $n \geq 1$ of copies of a_1 in the unique membrane of the system, then one also introduces l_0. Each computation starts by using the rule $r_1 : (c_0, out; ca_1, in)$, and continues as above until using the rule $r_3 : (c, out; l_0, in)$. Now, from the configuration $[_1 a_1^n l_0]_1$ we start the simulation of the register machine, hence we stop if and only if $n \in N(M)$ (and we stop by introducing the object l_h in the system, and, in addition, we will assume that in that moment no other object is present any more, i.e., the register machine only stops with all registers being empty).

Let us now consider any pair (ev_1, ev_2) of events, for

1. $ev_1 \in$ {start the computation,
   use the rule r_1,
   send the object c_0 out,
   bring the first a_1 in the system,
   bring c in the system for the first time},
2. $ev_2 \in$ {use the rule r_3,
   introduce the object l_0 in the system,
   send c out for the last time}.

Clearly, the number of steps elapsed between any ev_1 and any ev_2 as above is equal to the number of copies of a_1 introduced in the system, and the computation stops if and only if this number is recognized by M. Consequently, any recursively enumerable set of numbers (which does not contain the number 0) can be computed in this way.

In order to compute also the number 0 we can introduce the rule r_0 : $(c_0, out; l_0, in)$, taking as events, e.g., sending c_0 out and introducing l_0 in.

In a similarly easy way, universality can be obtained if we count the number of times two distinguished objects meet in a given region of the system.

We again start by taking a register machine $M = (n, H, l_0, l_h, R)$ and construct the system

$$\Pi = (1, O, [_1\ ]_1, cd, E, R_1)$$

with the following rules:

$$(c, out; c'a_1, in),$$
$$(c', out; cb, in),$$
$$(c', out, bl_0, in),$$
$$(b, out),$$

as well as rules for simulating the register machine M. The objects whose meetings are counted are b and d. We start again by introducing $n \geq 1$ copies of a_1 in the system, but this time we also introduce a copy of b for each copy of a_1. The copy of b exits immediately – but already we have the pair (b, d) present in the unique membrane of the system. After introducing l_0, the computation simulates the work of M for recognizing the number n. Thus, Π computes m if and only if $m \in N(M)$.

As shown in [3], P systems with multiset rewriting rules are able to simulate register machines when using at least two catalysts in one membrane. For such systems, similar events as above for P systems with symport/antiport rules can be defined, e.g., we start with $c_1c_2d_1d_2$ (c_1, c_2 are the two catalysts) and use the following rules:

$$\begin{aligned}
r_1 &: c_1d_1 \to c_1d_2a_1,\\
r_2 &: c_2d_2 \to c_2d_1,\\
r_3 &: c_1d_1 \to c_1,\\
r_4 &: c_2d_2 \to c_2w_1w_2.
\end{aligned}$$

The simulation of the register machine then starts with c_1 using w_1 and c_2 using w_2. Suitable trap rules like $d_2 \to \#$ and $x \to \#$ for symbols contained in w_1w_2 then guarantee the correct synchronization between the two catalysts c_1, c_2 when ending this initial phase. Like above, a suitable pair of events is the first use of rule r_1 (r_2) and the use of rule r_3 (r_4).

Thus, these cases do not look very interesting. To make the problem more difficult, we have to restrict either the complexity of the system (for instance, using only minimal symport/antiport rules, like in [1], or bounding the number of used objects, like in [14]), or the freedom in choosing the events. Both these ideas look very restrictive, in the latter case at least when we take the length of a computation as the number computed by the derivation. In the next section we deal with this possibility, both for symport/antiport systems and for catalytic P systems.

4 The Length of a Computation

In some sense, the most natural (and restrictive) events to consider are the start and the end of a computation, thus considering the length of a computation as the computed number.

For a system Π, let us denote by $lg(\Pi)$ the set of numbers of steps (lengths) of halting computations in Π. Then, let us denote by $N_{lg}OP_m(\mathit{features})$ the family of sets of numbers $lg(\Pi)$ computed by systems with at most $m \geq 1$ membranes, using the features specified by *features*. These features can be, as usual, $sym_k, anti_r, ncoo, cat_s, \delta$, where k, r are the maximal weights of symport and antiport rules, and s is the maximal number of catalysts which are allowed. When promoters or inhibitors are used for symport or antiport rules, we write $psym, isym$ and $panti, ianti$, respectively.

We start the study of these families with the easy observation that *the length set of any regular language can be computed in this way, both by symport/antiport systems and by non-cooperative multiset-rewriting P systems.* Indeed, for any regular grammar $G = (N, T, S, P)$ we can consider the one-membrane system with rules

$(A, out; B, in)$, for $A \to aB \in R$,
(A, out), for $A \to a \in R$,

in the symport/antiport case, and

$A \to B$, for $A \to aB \in R$,
$A \to a$, for $A \to a \in R$,

in the multiset-rewriting case (the obvious details are omitted).

Therefore, the families $N_{lg}OP_1(sym_1, anti_1)$, $N_{lg}OP_1(ncoo)$ contain all semilinear sets of numbers. We *conjecture* that also the converse is true, that is, no further sets of numbers can be computed by non-cooperative multiset-rewriting P systems, or by a symport/antiport P system – in both cases, without additional features, such as membrane dissolution, promoters/inhibitors, etc.

Actually, we do not know whether membrane dissolution helps also in the case of non-cooperative multiset-rewriting systems, but for symport/antiport systems both the use of promoters (or inhibitors), and the use of the dissolution operation seems to help. We start with the easy case, of using promoters or inhibitors.

Theorem 1. *The families $N_{lg}OP_1(sym_0, panti_2)$, $N_{lg}OP_1(sym_0, ianti_2)$ contain non-semilinear sets of numbers.*

Proof. We consider the following symport/antiport P system:

$$\begin{aligned} \Pi &= (1, \{a, b, c, d\}, [_1\ ]_1, acd, \{a, b, d\}, R_1), \\ R_1 &= \{(a, out; aa, in)|_c, \\ &\qquad (ca, out; ab, in), \\ &\qquad (da, out; d, in)|_b\}. \end{aligned}$$

In the presence of promoter c, in n steps we can bring into the system $2^n - 1$ copies of a, for $n \geq 1$; in the last step, c exits the system and b enters, thus promoting the rule $(da, out; d, in)$; this rule will be used for $2^n - 1$ steps, and then the computation halts. Therefore, the length of the computation is $n + 2^n - 1$, that is, $lg(\Pi) = \{n + 2^n - 1 \mid n \geq 1\}$. The objects c, b can be used as inhibitors, cross-wise (b inhibiting the first rule and c inhibiting the third one) and the result is the same. □

In the case of using the dissolution operation, the construction is more complicated.

Theorem 2. *The family $N_{lg}OP_3(sym_1, anti_2, \delta)$ contains non-semilinear sets of numbers.*

Proof. We consider the symport/antiport P system

$$\Pi = (3, O, \mu, w_1, w_2, w_3, E, R_1, R_2, R_3),$$

with the following components:

$$\begin{aligned}
O &= \{a, b, c, d, d', e, f, g\},\\
\mu &= [_1 [_2 [_3\ ]_3]_2]_1,\\
w_1 &= fbe,\\
w_2 &= db,\\
w_3 &= b^3 d,\\
E &= \{a, b, c, d', f, g\},\\
R_1 &= \{(e, out; eb, in),\ (de, out; d'a, in),\ (d', out; cg, in),\\
&\qquad (d, out; f, in),\ (ff, out; ff, in),\\
&\qquad (cg, out; cg, in),\ (g, out),\ (a, out; aa, in)\},\\
R_2 &= \{(b, in),\ (d, out),\ (c, in)\delta\},\\
R_3 &= \{(d, out; db, in),\ (b, out; a, in),\ (b, out; f, in)\delta\}.
\end{aligned}$$

The computation starts by using the rule $(e, out; eb, in) \in R_1$ and the rule $(d, out; db, in) \in R_3$. Assume that this latter rule is used for $n \geq 0$ steps; this means that in membrane 3 we accumulate $n + 3$ copies of b.

At some moment, d will exit membrane 2 by means of $(d, out) \in R_2$; simultaneously, one further copy of b will enter the system by means of $(e, out; eb, in)$, but it will not enter membrane 3. When d is in the skin region, it has to be used by the rule $(de, out; d'a, in) \in R_1$, otherwise the rule $(d, out; f, in) \in R_1$ is used, we get two copies of f in the system and the rule $(ff, out; ff, in) \in R_1$ can be used forever. In this way, we stop bringing further copies of b inside, and we start bringing copies of a.

In the next step, d' is exchanged with cg. From now on, in each step we can use the rules $(a, out; aa, in) \in R_1$ and $(cg, out; cg, in) \in R_1$. The first one doubles successively the number of copies of a present in the system, the second one just keeps busy the object c. After some $m \geq 1$ steps of doubling the number of copies of a (hence, at the end we get 2^m copies of a in the skin membrane), the rule $(c, in)\delta \in R_2$ is used. Because g exits the system at the same time (it has to use the rule $(g, out) \in R_1$), from now on c will still remain in the system.

The use of the rule $(c, in)\delta \in R_2$ dissolves membrane 2, hence, the objects from the skin region become accessible to rules from R_3. In particular, the rule $(b, out; f, in)\delta \in R_3$ can be used – but this should be avoided: if we dissolve membrane 3, the computation will continue forever by means of the rule $(a, out; aa, in) \in R_1$, making use of the copies of a (and there is at least one) from the unique membrane of the system. In order to avoid the use of the rule $(b, out; f, in)\delta \in R_3$ we have to involve all occurrences of b from membrane 3 (there are $n+3$ such copies) in using the rule $(b, out; a, in) \in R_3$. This means that we have enough copies of a in the skin region, that is, $s^m \geq n+3$. Actually, we must have equality: if any copy of a remains outside membrane 3, then it will evolve forever by means of $(a, out; aa, in) \in R_1$.

Consequently, the computation stops in the moment when introducing all copies of a in membrane 3. This means that we have performed $n+3+m = 2^m+m$ steps, for $m \geq 2$: we perform n steps using the rule $(d, out; db, in) \in R_3$, one step for each of $(d, out) \in R_2$ and $(de, out; d'a, in) \in R_1$, m steps of doubling the number of copies of a – in the first one we also use $(d', out; cg, in) \in R_1$, and in the last one $(c, in)\delta \in R_2$, hence we have $m \geq 2$ –, and a final step of introducing all copies of a in membrane 3. Consequently, $lg(\Pi) = \{n + 2^n \mid n \geq 2\}$, which is not a semilinear set. □

A similar result can be obtained for catalytic systems – using membrane dissolution. We below give two proofs of this assertion (the first one is a slight generalization of the set of numbers from the previous theorems).

Theorem 3. *The family $N_{lg}OP_2(cat_1, \delta)$ contains non-semilinear sets of numbers.*

Proof. **First example.**

For some $k \geq 2$, we consider the system

$$\begin{aligned}
\Pi &= (2, \{a, a', a'', b, c\}, \{c\}, [_1\ [_2\ ]_2]_1, c, a'', R_1, R_2),\\
R_1 &= \{ca \to cb\},\\
R_2 &= \{a'' \to a''a'a^{k-2},\\
&\qquad a' \to a'a^{k-1},\\
&\qquad a \to a^k,\\
&\qquad a'' \to a'a^{k-1}\delta\}.
\end{aligned}$$

We start with only one copy of a'' in membrane 2; in each step, the number of copies of a – primed or not – increases k times, always having only one copy of a'' present; in each step, one further copy of a' is introduced. Thus, after $n \geq 0$ steps we have one copy of a'', n copies of a', and $k^n - (n+1)$ copies of a. Eventually we use the rule $a' \to a'a^{k-1}\delta$, thus reaching a multiset with $n+1$ copies of a' and $k^{n+1} - (n+1)$ copies of a. At that time, membrane 2 is dissolved. From now on, we continue by using the rule $ca \to cb$ of R_1, and this is done for each copy of a. Consequently, the computation in total takes k^{n+1} steps, for $n \geq 0$, that is, we have $lg(\Pi) = \{k^n \mid n \geq 1\}$.

Second example.

Let us now consider the system

$$\begin{aligned}
\Pi &= (2, \{a, b, c, d, e\}, \{c\}, [_1\ [_2\ ]_2]_1, c, abd, R_1, R_2),\\
R_1 &= \{c\alpha \to ce \mid \alpha \in \{a, b, d\}\},\\
R_2 &= \{a \to abb,\\
&\qquad d \to abd,\\
&\qquad d \to abdd\delta\}.
\end{aligned}$$

Assume that after $n \geq 0$ steps we have a multiset of the form $a^{n+1}b^{(n+1)^2}d$ in membrane 2 (initially, this is the case, for $n = 0$). If the membrane is not dissolved, then by using the rules of R_2 we get the multiset $a^{n+1}b^{2(n+1)}b^{(n+1)^2}abd = a^{n+2}b^{(n+1)^2+2(n+1)+1}d = a^{n+2}b^{(n+2)^2}d$. The process can be iterated. If the membrane is dissolved (this happens in step $n+1$ for $n \geq 0$), then the obtained multiset is $a^{n+2}b^{(n+2)^2}dd$, which passes to membrane 1. In membrane 1 we perform a step for each copy of a, b, d being present. Thus, the computation lasts in total

$$\begin{aligned}
n+1 &\quad \text{(steps performed in membrane 2)}\\
+(n+2)+(n+2)^2+2 &\quad \text{(steps performed in membrane 1)}\\
= (n+2)^2 + 2(n+2) + 1 &\\
= (n+3)^2 &\quad \text{steps.}
\end{aligned}$$

Consequently, $lg(\Pi) = \{n^2 \mid n \geq 3\}$. □

The use of the dissolving operation can be avoided, at the expense of using cooperative rules.

Theorem 4. *The family $N_{lg}OP_2(coo)$ contains non-semilinear sets of numbers.*

Proof. Let us consider the system

$$\begin{aligned}
\Pi &= (2, \{a, b, c, d, e, f, g, \#\}, [_1\ [_2\ ]_2]_1, c, a, R_1, R_2),\\
R_1 &= \{ca \to c\},\\
R_2 &= \{a \to bbcc,\\
&\qquad c \to d,\\
&\qquad d \to e,\\
&\qquad b \to af,\\
&\qquad b \to a_{out}g,\\
&\qquad ef \to \lambda,\\
&\qquad fg \to \#,\\
&\qquad \# \to \#\}.
\end{aligned}$$

Assume that in membrane 2 we have 2^n copies of a and nothing else; initially, this is the case, for $n = 0$. By using the rule $a \to bbcc$, we introduce the same number of copies of b and c. In the next step, c evolves to d; assume that all copies of b evolve by means of the rule $b \to af$. In the next step, d becomes e and a introduces again b and f. Now, all copies of f and e (they are 2^{n+1} each) are removed by means of $ef \to \lambda$, hence the process can be iterated.

Assume now that at some step we have a number 2^m of copies of b and c in membrane 2, and in the next step we use at least once the rule $b \to a_{out}g$. If at the same time the rule $b \to af$ is used at least once, then the objects f, g have to use the rule $fg \to \#$ and the computation never stops (f cannot be removed by rule $ef \to \lambda$, because e is not available).

Therefore, either all copies of b return to a by the rule $b \to af$ or all of them send a copy of a to membrane 1 by the rule $b \to a_{out}g$. This means that we send 2^n copies of a to membrane 1, for some $n \geq 1$. In membrane 1 we just "count" these objects, removing them one by one by means of the rule $ca \to c$.

Thus, the computation lasts $2n$ steps in membrane 2 and 2^n steps in membrane 1, hence, we obtain $lg(\Pi) = \{2n + 2^n \mid n \geq 1\}$, which is not a semilinear set. □

These results show that the families $N_{lg}OP_m(sym_k, \alpha anti_r, \delta)$, with $\alpha \in \{p, i\}$, and $N_{lg}OP_m(cat_k, \delta)$, $N_{lg}OP_2(coo)$, are non-trivial: they include the semilinear sets of numbers and also contain non-semilinear sets as those from the previous proofs.

On the other hand, these families are not containing all Turing computable sets of numbers, and this is basically due to the fact that during a computation of length n we can only construct a working space which is of the order of k^n, for a constant k depending on the system at hand.

Let us first recall two results from [10].

Given a proper function f, let $SPACE(f)$ denote the deterministic space and $NSPACE(f)$ the non-deterministic space with respect to f.

Theorem 5. *If $f(n)$ is a proper function, then $SPACE(f(n))$ is a proper subset of $SPACE(f(n)logf(n))$.*

Theorem 6. *If $f(n)$ is a proper function with $f(n) \geq log\, n$, then $NSPACE(f(n)$ $\subseteq SPACE(f^2(n))$.*

Theorem 7. *None of the families $N_{lg}OP_*(\alpha sym_*, \beta anti_*, \delta)$, for any $\alpha, \beta \in \{p, i\}$, nor $N_{lg}OP_*(coo, \delta)$ equals NRE.*

Proof. Consider a P system Π with any type of rules (symport/antiport – with or without promoters or inhibitors, cooperative multiset-rewriting, membrane dissolving included) generating as the length of its halting computations the set $lg(\Pi)$.

For each halting computation C of length n of Π, $C = C_1 \Longrightarrow C_2 \Longrightarrow \ldots \Longrightarrow C_n$, define

$$W_\Pi(n, C) = \max\{\text{number of objects present in } C_i \mid 1 \leq i \leq n\}.$$

Now define

$$W_{\Pi}(n) = \max\{W(n, C) \mid C \text{ is a halting computation of length } n \text{ of } \Pi\}.$$

It is clear that $W_{\Pi}(n)$ is bounded by a function of order $O(k^n)$, where k is a constant depending on Π.

We can construct the following non-deterministic Turing machine M accepting $lg(\Pi)$. The machine M uses two tapes. On one tape it writes the input (in binary), and a counter (in binary), separated by a marker, and on the other one the machine simulates the computation of the system Π (in a non-deterministic way, if Π is non-deterministic). Each parallel step in Π is simulated by several successive steps in M – not the duration of the computation in M is relevant here, but the space it uses. For each simulated parallel step of Π, the machine M increases the counter, storing, in this way, the number of parallel steps simulated. When the simulation of Π reaches a halting configuration, the machine M checks whether the counter has reached exactly the number given as input. If this is the case, then it answers `yes`, otherwise it answers `no`.

The machine M uses a space in $O(2^{k^n})$ where n is the size of the input (written in binary; notice that the system Π generates strings in unary numbers, here we have to input them in binary but work with them in unary).

Therefore, $lg(\Pi) \in NSPACE(f(n))$, for $f(n) \in O(2^{k^n})$, hence, by Theorem 6 we get $lg(\Pi) \in SPACE(g(n))$, for $g(n) \in O((2^{k^n})^2)$. In this way, we get the following upper bound,

$$N_{lg}OP_*(coo) \subseteq \bigcup_{j \in \mathbf{N}} SPACE(O((2^{j^n})^2)).$$

According to Theorem 5, given any proper function $h(n)$ such that $h(n) \geq (2^{j^n})^2 log((2^{j^n})^2)$, for any $j \in \mathbf{N}$, there exists a set in $SPACE(h(n))$ that is not in the family $N_{lg}OP_*(coo)$, hence, the theorem follows (note that although the construction of M depends on Π, the space used by M is always bounded in the same way). $\square$

In a more general way, we now give a general view on the length of computations, applicable to any universal computation model (e.g., to register machines).

Let $\varphi(x, y)$ be the result of program number x on input y (φ is usually called a universal function), and $\Phi(x, y)$ be the number of computation steps of program x on input y (typically the computation time). Moreover, for any function f we denote the domain of f by $D(f)$ and the range of f by $R(f)$.

We now can formulate the problem discussed above in a more general way:

Problem 1. Does there exist an x such that $R(\Phi_x)$ is not recursive?

We first observe some simple facts about φ, Φ:

1. $D(\varphi_x) = D(\Phi_x) = \{y \mid (\exists z)\Phi(x, y) = z\}$ for all x.
   Thus, there exists an x such that $D(\Phi_x)$ is undecidable (as the $D(\varphi_x)$ range over all recursively enumerable sets).

2. $\{(x, y, z) \mid \Phi(x, y) = z\}$ is decidable.
   Clearly, even if $\Phi(x, y)$ is undefined (by nontermination) we can check whether termination occurs after z steps.
3. In general, there is no recursive function g such that $\Phi(x, y) \leq g(x, y)$ on $D(\varphi_x)$.

Let $\| (y_1, \dots, y_m) \| = \max\{y_i \mid i = 1, \dots, m\}$ (the usual maximum norm on tuples of natural numbers). Then a strictly monotone function $f \colon \mathbf{N} \to \mathbf{R}$ is called *regular* if the sets $A_k \colon = \{f(l) \mid l \in \mathbf{N}, l \geq k\}$ are infinite for all $k \in \mathbf{N}$. For instance, the identity, log, and $\log\log$, or $c * \log$, etc. are regular monotone functions.

Then, for any Gödel numbering ϕ, the complexity measure (ϕ, Φ) is called *input sensitive* if there exists a regular monotone function $f \colon \mathbf{N} \to \mathbf{R}$ such that

$$\Phi(x, y) > f(\| y \|) \text{ for all } x, y \in \mathbf{N}^m.$$

Theorem 8. *Let (ϕ, Φ) be an input sensitive complexity measure. Then, for all $x \in \mathbf{N}$, $R(\Phi_x)$ is decidable.*

Proof. Let f be the regular strictly monotone function corresponding to (ϕ, Φ) with $\Phi(x, y) > f(\| y \|)$ for all $y \in \mathbf{N}^m$. We define a decision procedure for the set $R(\Phi_x)$ as follows.

Let z be an arbitrary element of $\mathbf{N}$. We define

$$N_z = \min\{n \mid f(n) > z\}.$$

N_z exists by the regularity of f. In particular (by input-sensitivity),

$$\Phi(x, y) > z \text{ for all } y \in \mathbf{N}^m \text{ such that } \| y \| \geq N_z.$$

Therefore $z \in R(\Phi_x)$ if and only if there exists an y with $\| y \| \leq N_z$ such that $\Phi_x(y) = z$. Thus, if we define

$$Y_z = \{y \mid y \in \mathbf{N}^m, \| y \| \leq N_z\},$$

then

$$z \in R(\Phi_x) \text{ if and only if } z \in \Phi_x(Y_z).$$

But the set Y_z is finite and the predicate

$$\Phi(x, y) = z$$

is decidable due to fact 2 observed above. Therefore it is decidable whether $z \in \Phi_x(Y_z)$, and $R(\Phi_x)$ is decidable as well. □

A register machine M has to read its input in order to have an in infinite set $lg(M)$ and therefore is input sensitive, hence, in this case, the set $lg(M)$ is decidable.

Some further light on the characteristic features of the families considered above can be obtained by observing the following connection between the duration of computations in register machines and in P systems:

Theorem 9. *P systems with symport/antiport rules as well as catalytic P systems (with at least two catalysts) can simulate the instructions of a register machine in exactly k steps, where k is a constant only depending on the type of the P system, but not on the particular system itself.*

Proof. For a given register machine $M = (n, H, l_0, l_h, R)$ we consider the alphabet $U = \{a_1, \dots, a_n\}$ (the symbol a_i is associated with register i and the contents of this register is represented by the multiplicity of object a_i in the P system we are going to construct); we now construct the P system with antiport rules

$$\begin{aligned}
\Pi &= (1, O, [_1\ ]_1, l_0, O, R_1),\\
O &= U \cup \{l, l', l'', l''', l^{iv} \mid l \in B\},\\
R_1 &= \{(l_1, out; l_1', in),\\
&\qquad (l_1', out; l_1'', in),\\
&\qquad (l_1'', out; l_2 a_r, in) \mid l_1 : (\mathtt{ADD}(r), l_2) \in R\}\\
&\quad \cup \{(l_1, out; l_1' l_1'', in),\\
&\qquad (l_1' a_r, out; l_1''', in),\\
&\qquad (l_1'', out; l_1^{iv}, in),\\
&\qquad (l_1^{iv} l_1''', out; l_2, in),\\
&\qquad (l_1^{iv} l_1', out; l_3, in) \mid l_1 : (\mathtt{SUB}(r), l_2, l_3) \in R\}.
\end{aligned}$$

We start with l_0 present in the system, and we also introduce a multiset a_1^m, for m being the number to be recognized. The system behaves like M, simulating each of its instructions in exactly three steps (i.e., in this case we have $k = 3$). The construction given above follows the construction elaborated in [5], except that ADD instructions are simulated in three steps here and not in only one step as done in [5].

As is shown in [3], P systems with at least two catalysts can simulate register machines in only one membrane; in the construction given there, an ADD instruction is simulated in one step, whereas the simulation of a SUB instruction of a given register machine takes exactly four steps in the catalytic P system. It is an obvious task to expand the simulation of an ADD instruction to exactly four steps, too, hence, in the case of catalytic P systems the constant k equals four. □

As a corollary, this theorem can again provide examples of non-semilinear sets of numbers as length sets of computations in P systems with symport/antiport systems and catalytic P systems (and without membrane dissolution!) – it just remains to specify register machines working, for instance, for a number of steps which grow exponentially; we leave this easy task as an exercise to the reader. Moreover, from Theorem 8 we know that the length sets of all these P systems are recursive.

From the results elaborated in this section we know that all families of length sets obtained from a specific model of P systems, although not being Turing complete, contain complex enough sets of numbers (which can be generated even by systems of the respective types with reduced parameters – number of membranes, weight of symport/antiport rules, number of catalysts). Still, a series of open problems remains to be investigated; for instance, what about hierarchies on the number of membranes or of catalysts or on the weight of symport/antiport rules?

5 Remarks About Chomsky Grammars

In Section 4 we have used the obvious fact that the length of derivations in a regular grammar is the same as the length of the generated strings, hence, it is a semilinear set. What about other classes of grammars from Chomsky hierarchy?

Given a grammar G of any type, let us denote by $lg(G)$ the set of lengths of terminal derivations in G.

First, let us note that a statement as above for regular grammars also holds for matrix grammars without appearance checking:

Theorem 10. *The length sets of derivations in regular, linear, context-free, and matrix grammars without appearance checking are semilinear.*

Proof. For a matrix grammar $G = (N, T, S, M)$ construct the matrix grammar

$$\begin{aligned} G' &= (N \cup \{S', C\}, T \cup \{c\}, S', M'), \\ M' &= \{(S' \to CS)\} \\ &\quad \cup \{(C \to cC, r_1, \ldots, r_n) \mid (r_1, \ldots, r_n) \in M\} \\ &\quad \cup \{(C \to c, r_1, \ldots, r_n) \mid (r_1, \ldots, r_n) \in M \text{ terminal matrix}\}. \end{aligned}$$

Now consider the morphism which erases all symbols from T and leaves c unchanged. Then we have

$$lg(G) = length(h(L(G'))).$$

Because the languages over a one-letter alphabet generated by matrix grammars are regular, [6], it follows that $lg(G)$ is a semilinear set. □

Thus, there is no difference between the length sets of derivations in regular, linear, context-free, and matrix grammars without appearance checking. This is no longer the case if we pass to monotone grammars. Here is a simple example:

Consider the monotone grammar

$$\begin{aligned} G &= (\{S, B\}, \{a, b, c\}, S, P), \\ P &= \{r_1 : S \to aSBc, \\ &\qquad r_2 : S \to abc, \\ &\qquad r_3 : cB \to Bc, \\ &\qquad r_4 : bB \to bb\}. \end{aligned}$$

We have $L(G) = \{a^n b^n c^n \mid n \geq 1\}$. In the derivation of a string $a^n b^n c^n$ we use $n-1$ times the rule r_1, once the rule r_2, $0 + 1 + 2 + \cdots + (n-2)$ times the rule r_3 (each B has to travel to left until reaching the first occurrence of b, that is, passing over the intermediate occurrences of c), and $n-1$ times the rule r_4. In total, this means $\sum_{i=1}^{n} i = n(n+1)/2$, for some $n \geq 1$, which means that $lg(G)$ is not semilinear. Note that the language $L(G)$ itself is semilinear, in contrast to $lg(G)$.

Actually, in general, $lg(G)$ does not say too much about the language $L(G)$. More precisely, *for each monotone grammar G there is a monotone grammar G' such that $L(G) = L(G')$ and $lg(G') = \{n \mid n \geq 1\}$*. Indeed, assume $G = (N, T, S, P)$ and take a string $w \in L(G)$. We construct $G' = (N \cup \{S'\}, T, S', \{S' \to S', \ S' \to w, \ S' \to S\} \cup P)$. Besides derivations $S' \Longrightarrow^* S' \Longrightarrow w$, of any desired length, all other derivations in G' correspond to derivations in G. Thus, the assertions stated above hold.

Some indications about the “derivation length complexity” of (monotone) languages L probably can be provided by considering all grammars which generate L, thus examining the family of all sets of length of derivations of these grammars, yet we do not persist in this direction here.

6 Final Remarks

Freely chosen events look too easy to handle (too much freedom), on the other hand, taking the length of a computation as the computed number sometimes looks too restrictive. Something in between might be not to count all steps of a computation, but to discard some of them, chosen in a natural manner. This is much in the spirit of using an external observer, as in [2]. If, like in [2], we take as the observer a finite state automaton analyzing multisets, then again we get universality in an easy way, by ignoring all steps before and after given events, like those considered in Section 3. Finding non-trivial observers remains as a further research topic.

References

1. A. Alhazov, M. Margenstern, V. Rogozhin, Y. Rogozhin, S. Verlan: Communicative P systems with minimal cooperation. In [8], 162–178.
2. M. Cavaliere, P. Leupold: Evolution and observation – A new way to look at membrane systems. In [7], 70–87.
3. R. Freund, L. Kari, M. Oswald, P. Sosik: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Sci.*, 330, 2 (2005), 251–266.
4. R. Freund, A. Păun: Membrane systems with symport/antiport rules: Universality results. In [15], 270–287.

5. R. Freund, Gh. Păun, On deterministic P systems. Submitted, 2003.
6. D. Hauschild, M. Jantzen: Petri nets algorithms in the theory of matrix grammars. *Acta Informatica*, 31 (1994), 719–728.
7. C. Martín–Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing. International Workshop, WMC2003, Tarragona, Spain, Revised Papers*, LNCS 2933, Springer-Verlag, Berlin, 2004.
8. G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing. International Workshop WMC5, Milan, Italy, 2004. Revised Papers*,LNCS 3365, Springer-Verlag, Berlin, 2005.
9. M.L. Minsky: *Computation. Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.
10. C.H. Papadimitriou: *Computational Complexity.* Addison-Wesley, 1984.
11. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3 (2002), 295–306.
12. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.
13. Gh. Păun: Further twenty-six open problems in membrane computing. In this volume, 249–262.
14. Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodriguez-Patón: Symport/antiport P systems with three objects are universal. *Fundamenta Informaticae*, 64, 1-4 (2005).
15. Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.: *Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, Revised Papers.* LNCS 2597, Springer-Verlag, Berlin, 2003.
16. A. Salomaa: *Formal Languages.* Academic Press, New York, 1973.