Variants of P Systems with Toxic Objects

Artiom Alhazov[1], Rudolf Freund[2], and Sergiu Ivanov[3]

[1] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
Email: artiom@math.md

[2] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
Email: rudi@emcc.at

[3] Université Paris Est, France
Email: sergiu.ivanov@u-pec.fr

Summary. Toxic objects have been introduced to avoid trap rules, especially in (purely) catalytic P systems. No toxic object is allowed to stay idle during a valid derivation in a P system with toxic objects. In this paper we consider special variants of toxic P systems where the set of toxic objects is predefined – either by requiring all objects to be toxic or all catalysts to be toxic or all objects except the catalysts to be toxic. With all objects staying inside and being toxic, purely catalytic P systems cannot go beyond the finite sets, neither as generating nor as accepting systems. With allowing the output to be sent to the environment, exactly the regular sets can be generated. With non-cooperative systems with all objects being toxic we can generate exactly the Parikh sets of languages generated by extended Lindenmayer systems. Catalytic P systems with all catalysts being toxic can generate at least $PsMAT$.

1 Definitions

We assume the reader to be familiar with the underlying notions and concepts From formal language theory, e.g., see [16], as well as from the area of P systems, e.g., see [13, 14, 15]; we also refer the reader to [18] for actual news.

1.1 Prerequisites

The set of integers is denoted by $\mathbb{Z}$, and the set of non-negative integers by $\mathbb{N}$. Given an alphabet V, a finite non-empty set of abstract symbols, the free monoid generated by V under the operation of concatenation is denoted by V^*. The elements of V^* are called strings, the empty string is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. For any string $w \in V$, by $alph(w)$ we denote the set of symbols

occurring in w; moreover, the set of all strings which are obtained by permuting the symbols of w is denoted by $Perm(w)$; for a set of stings L, we define $Perm(L) = \{Perm(w) \mid w \in L\}$.

For an arbitrary alphabet $V = \{a_1, \cdots, a_n\}$, the number of occurrences of a symbol a_i in a string x is denoted by $|x|_{a_i}$, while the length of a string x is denoted by $|x| = \sum_{a_i \in V} |x|_{a_i}$. A (finite) multiset over a (finite) alphabet $V = \{a_1, \cdots, a_n\}$ is a mapping $f : V \to \mathbb{N}$ and can be represented by $\left\langle a_1^{f(a_1)}, \cdots, a_n^{f(a_n)} \right\rangle$ or by any string x for which $(|x|_{a_1}, \cdots, |x|_{a_n}) = (f(a_1), \cdots, f(a_n))$. We will denote the vector $(f(a_1), \cdots, f(a_n))$ by $\Psi(f)_V$. The families of regular and recursively enumerable string languages are denoted by REG and RE, respectively.

1.2 Finite Automata

The regular languages in REG are exactly the languages accepted by finite automata. A *finite automaton* is a quintuple $M = (Q, T, \delta, q_0, F)$, where Q is the set of *states*, T is the *input alphabet*, $\delta \subseteq (Q \times T \times Q)$ is the transition function, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of final states. The language over T accepted by M is denoted by $L(M)$. A finite automaton is called *deterministic*, if for every pair (q, a) with $q \in Q$ and $a \in P$ there exists exactly one state $p \in Q$ such that $(q, a, p) \in \delta$.

A *finite automaton with output*, also called *generalized sequential machine* or *gsm* for short, is a construct $M = (Q, T, \Sigma, \delta, q_0, F)$, where Q is the set of *states*, T is the *input alphabet*, Σ is the *output alphabet*, $\delta \subseteq (Q \times T \times Q \times \Sigma^*)$ is the finite transition function, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of final states. M called *deterministic*, if for every pair (q, a) with $q \in Q$ and $a \in P$ there exists exactly one pair $(q, w) \in Q \times \Sigma^*$ such that $(q, a, p, w) \in \delta$. A (deterministic) gsm defines a relation (function) $T^* \to \Sigma^*$, called *(deterministic) gsm mapping*. The sets of all relations (functions) defined by (deterministic) gsm mappings are denoted by $RelREG$ and $FunREG$, respectively.

We also consider a special variant of finite automata which resembles the idea of input-driven push-down automata (for an overview, see [11]), also called visibly push-down automata (for example, see [3]). Hence, we call this variant where the next state only depends on the input symbol *input-driven finite automata*, i.e., for any two triples $(q, a, p), (q', a, p') \in \delta$ with $q, p, q', p' \in Q$ and $a \in T$ we have $p = p'$. In the following, the subclass of regular languages accepted by input-driven finite automata will be denoted by $IDREG$.

A gsm is called input-driven if for any two tuples $(q, a, p, w), (q', a, p', w') \in \delta$ with $q, p, q', p' \in Q$ and $a \in T$ we have $p = p'$ as in the case of finite automata; such a gsm is called deterministic if we even have $(p, w) = (p', w')$. The subclasses of (deterministic) gsm mappings defined by input-driven finite automata with output are denoted by $RelIDREG$ and $FunIDREG$, respectively.

1.3 *ET0L* Systems

An $ET0L$ system is a construct $G = (V, T, P_1, \cdots, P_m, w)$ where $m \geq 1$, V is an alphabet, $T \subseteq V$ is the terminal alphabet, the P_i, $1 \leq i \leq m$, are finite sets (*tables*) of non-cooperative rules over V, and $w \in V^*$ is the *axiom*. In a derivation step in G, all the symbols present in the current sentential form are rewritten using one table. The language generated by G, denoted by $L(G)$, consists of all terminal strings $w \in T^*$ which can be generated by a derivation in G starting from the axiom w. The family of languages generated by $ET0L$ systems and by $ET0L$ systems with at most k tables is denoted by $ET0L$ and ET_k0L, respectively. If only one table is used, we omit the T.

1.4 Register Machines

A *register machine* is a tuple $M = (m, B, l_0, l_h, P)$, where m is the number of registers, B is a set of labels, $l_0 \in B$ is the initial label, $l_h \in B$ is the final label, and P is the set of instructions bijectively labeled by elements of B. The instructions of M can be of the following forms:

- $l_1 : (ADD(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Increases the value of register r by one, followed by a non-deterministic jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : (SUB(r), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  If the value of register r is zero then jump to instruction l_3; otherwise, the value of register r is decreased by one, followed by a jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stops the execution of the register machine.

A *configuration* of a register machine is described by the contents (i.e., by the number stored in the register) of each register and by the current label, which indicates the next instruction to be executed. Computations start by executing the instruction l_0 of P, and terminate with reaching the HALT-instruction l_h.

In order to deal with strings, this basic model of register machines can be extended by instructions for reading from an input tape and writing to an output tape containing strings over an input alphabet T_{in} and an output alphabet T_{out}, respectively:

- $l_1 : (read(a), l_2)$, with $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, $a \in T_{in}$.
  Reads the symbol a from the input tape and jumps to instruction l_2.
- $l_1 : (write(a), l_2)$, with $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, $a \in T_{out}$.
  Writes the symbol a on the output tape and jumps to instruction l_2.

Such a register machine working on strings we call a *register machine with input and output tape*, and we write $M = (m, B, l_0, l_h, P, T_{in}, T_{out})$. If no output is written, we omit T_{out}.

As is well known (e.g., see [10]), for any recursively enumerable set of natural numbers there exists a register machine with (at most) three registers accepting the numbers in this set. Register machines with an input tape, can simulate the computations of Turing machines with two registers and thus characterize RE. All these results are obtained with deterministic register machines, where the ADD-instructions are of the form $l_1 : (ADD\,(r)\,, l_2)$, with $l_1 \in B \backslash \{l_h\}$, $l_2 \in B$, $1 \leq j \leq m$.

Partially blind register machines with d registers use instructions $q_i : (ADD(r), q_j, q_k)$ and $q_i : (SUB(r), q_j)$. Moreover, the result is produced in the first m registers, while in a successful computation registers $m+1, \cdots, d$ are required to be empty in the end (and we assume that the output registers are never decremented).

2 P systems

The ingredients of the basic variants of (cell-like) P systems are the membrane structure, the objects placed in the membrane regions, and the evolution rules. The *membrane structure* is a hierarchical arrangement of membranes. Each membrane defines a *region/compartment*, the space between the membrane and the immediately inner membranes; the outermost membrane is called the *skin membrane*, the region outside is the *environment*, also indicated by (the label) 0. Each membrane can be labeled, and the label (from a set Lab) will identify both the membrane and its region. The membrane structure can be represented by a rooted tree (with the label of a membrane in each node and the skin in the root), but also by an expression of correctly nested labeled parentheses. The *objects* (multisets) are placed in the compartments of the membrane structure and usually represented by strings, with the multiplicity of a symbol corresponding to the number of occurrences of that symbol in the string. The basic *evolution rules* are multiset rewriting rules of the form $u \to v$, where u is a multiset of objects from a given set O and $v = (b_1, tar_1) \cdots (b_k, tar_k)$ with $b_i \in O$ and $tar_i \in \{here, out, in\}$ or $tar_i \in \{here, out\} \cup \{in_j \mid j \in Lab\}$, $1 \leq i \leq k$. Using such a rule means "consuming" the objects of u and "producing" the objects $b_1, \cdots, b_k$ of v; the *target indications* *here*, *out*, and *in* mean that an object with the target *here* remains in the same region where the rule is applied, an object with the target *out* is sent out of the respective membrane (in this way, objects can also be sent to the environment, when the rule is applied in the skin region), while an object with the target *in* is sent to one of the immediately inner membranes, non-deterministically chosen, whereas with in_j this inner membrane can be specified directly. In general, we may omit the target indication *here*.

Yet there are a lot of other variants of rules; for example, if on the right-hand side of a rule we add the symbol δ, the surrounding membrane is dissolved whenever at least one such rule is applied, at the same moment all objects inside this membrane (the objects of this membrane region together with the whole

inner membrane structure) are released to the surrounding membrane, and the rules assigned to the dissolved membrane region get lost.

Another option is to add *promoters* $p_1, \cdots, p_m \in O^+$ and *inhibitors* $q_1, \cdots, q_n \in O^+$ to a rule and write $u \to v|_{p_1,\cdots,p_m,\neg q_1,\cdots,\neg q_n}$, which rule then is only applicable if the current contents of the membrane region includes any of the promoter multisets, but none of the inhibitor multisets; in most cases promoters and inhibitors are rather taken to be singleton objects than multisets.

For all these variants of P systems defined above, the variants of toxic objects defined later in this paper can be defined, too. As this paper is just a starting point of such investigations, in the following we shall restrict ourselves to P systems containing only non-cooperative rules and/or catalytic rules (see definitions given below).

Formally, a (cell-like) *P system* is a construct

$$\Pi = (O, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m, f_I, f_O)$$

where O is the alphabet of *objects*, μ is the *membrane structure* (with m membranes), $w_1, \cdots, w_m$ are multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \cdots, R_m$ are finite sets of *evolution rules*, associated with the membrane regions of μ, and f_O/f_I is the label of the membrane region where the outputs are put in/from where the inputs are taken. ($f_O/f_I = 0$ indicates that the output/input is taken sent to/taken from the environment).

If a rule $u \to v$ has at least two objects in u, then it is called *cooperative*, otherwise it is called *non-cooperative*. In *catalytic P systems* we use non-cooperative as well as *catalytic rules* which are of the form $ca \to cv$, where c is a special object which never evolves and never passes through a membrane (both these restrictions can be relaxed), but it just assists object a to evolve to the multiset v. In a *purely catalytic P system* we only allow catalytic rules. For a catalytic as well as for a purely catalytic P system Π, in the description of Π we replace "O" by "O, C" in order to specify those objects from O which are the catalysts in the set C. As already explained above, cooperative and non-cooperative as well as catalytic rules can be extended by adding promoters and/or inhibitors, thus yielding rules of the form $u \to v|_{p_1,\cdots,p_m,\neg q_1,\cdots,\neg q_n}$.

All the rules defined so far can be used in different derivation modes: in the *sequential* mode (*sequ*), we apply exactly one rule in every derivation step; in the *asynchronous* mode (*asyn*), an arbitrary number of rules is applied in parallel; in the *maximally parallel* (*maxpar*) derivation mode, in any computation step of Π we choose a multiset of rules from the sets $R_1, \cdots, R_m$ in a non-deterministic way such that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the membrane regions $1, \cdots, m$.

The membranes and the objects present in the compartments of a system at a given time form a *configuration*; starting from a given *initial configuration* and using the rules as explained above, we get *transitions* among configurations; a sequence of transitions forms a *computation* (we often also say *derivation*). A computation is *halting* if and only if it reaches a configuration where no rule can

be applied any more. With a halting computation we associate a *result generated* by this computation, in the form of the number of objects present in membrane f_O in the halting configuration. The set of multisets obtained as results of halting computations in Π working in the derivation mode $\delta \in \{sequ, asyn, maxpar\}$ is denoted by $mL_{gen,\delta}(\Pi)$, the set of natural numbers obtained by just counting the number of objects in the multisets of $mL_{gen,\delta}(\Pi)$ by $N_{gen,\delta}(\Pi)$, and the set of (Parikh) vectors obtained from the multisets in $mL_{gen,\delta}(\Pi)$ by $Ps_{gen,\delta}(\Pi)$. If we first project the results in $mL_{gen,\delta}(\Pi)$ to a terminal alphabet O_T, then we add the superscript T to N and Ps.

Yet we may also start with some additional input multiset w_{input} over an *input alphabet* Σ in membrane f_I, i.e., in total we there have $w_{f_I} w_{input}$ in the initial configuration, and *accept* this input w_{input} if and only if there exists a halting computation with this input; the set of multisets accepted by halting computations in

$$\Pi = (O, \Sigma, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m, f_I)$$

working in the derivation mode δ is denoted by $mL_{acc,\delta}(\Pi)$, the corresponding sets of natural numbers and of (Parikh) vectors are denoted by $N_{acc,\delta}(\Pi)$ and $Ps_{acc,\delta}(\Pi)$, respectively.

For the input being taken from the environment, i.e., for $f_I = 0$, we need an additional target indication *come* as, for example, used in a special variant of communication P systems introduced by Petr Sosík (e.g., see [17]) where no objects are generated or deleted, but may only pass through membranes; $(a, come)$ on the right-hand side of a rule applied in the skin membrane means that the object a is taken into the skin membrane from the environment (all objects there are assumed to be available in an unbounded number). The multiset of all objects taken from the environment during a halting computation then is the multiset accepted by this accepting P system, which in this case we shall call a *P automaton*; the idea of *P automata* was first published in [4] and considered at the same time under the notion of *analysing P systems* in [8]. The set of non-negative integers and the set of (Parikh) vectors of non-negative integers accepted by halting computations in Π are denoted by $N_{aut}(\Pi)$ and $Ps_{aut}(\Pi)$, respectively.

The family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\gamma \in \{gen, acc, aut\}$ computed by P systems with at most m membranes working in the derivation mode δ and with rules of type X is denoted by $Y_{\gamma,\delta}OP_m(X)$. If we first project the results in $mL_{gen,\delta}(\Pi)$ to a terminal alphabet O_T, then we add the superscript T to N and Ps.

A P system Π can also be considered as a system computing a partial recursive function (in the deterministic case) or even a partial recursive relation (in the non-deterministic case), with the input being given in a membrane region $f_I \neq 0$ as in the accepting case or being taken from the environment as in the automaton case. The corresponding functions/relations computed by halting computations in Π are denoted by $ZY_\alpha(\Pi)$, $Z \in \{Fun, Rel\}$, $Y \in \{N, Ps\}$, $\alpha \in \{acc, aut\}$.

For example, it is well known (for example, see [12]) that for any $m \geq 1$, for the types of non-cooperative ($ncoo$) and cooperative (coo) rules we have

$$NREG = N_{gen,maxpar}OP_m\,(ncoo) \subset N_{gen,maxpar}OP_m\,(coo) = NRE.$$

For $\gamma \in \{gen, acc, aut\}$ and $\delta \in \{sequ, asyn, maxpar\}$, the family of sets $Y_{\gamma,\delta}\,(\Pi)$, $Y \in \{N, Ps\}$, computed by (purely) catalytic P systems with at most m membranes and at most k catalysts is denoted by $Y_{\gamma,\delta}OP_m\,(cat_k)$ and $Y_{\gamma,\delta}OP_m\,(pcat_k)$, respectively; from [5] we know that, with the results being sent to the environment (which means taking $f_O = 0$), we have

$$Y_{gen,maxpar}OP_1\,(cat_2) = Y_{gen,maxpar}OP_1\,(pcat_3) = YRE.$$

Remark 1. Here we have to add a remark which is important for the rest of this paper. Originally, Gheorghe Păun used an internal elementary membrane to obtain clean results without having to count the catalysts. Hence, sending out the results also uses a second membrane region, thus, from a topological point of view, there in fact is no difference between using the outer region or an inner membrane region without rules to be applied there. In sum, specifying the number of membranes is not sufficient to capture all subtle features of complexity. Hence, in the following, we will write $P_{1,ext}$ to indicate that, besides the single membrane, we also use the environment as a second membrane region. Thus, the result for (purely) catalytic P systems now will be written as

$$Y_{gen,maxpar}OP_{1,ext}\,(cat_2) = Y_{gen,maxpar}P_{1,ext}\,(pcat_3) = YRE.$$

In the general case, we will also use the notation $P_{m,ext}$ for P systems with m membranes and external output, and to contrast this, we will use $P_{m,int}$ for systems with internal output to make a clear difference to the normal notations P_m which might mean both of these cases.

Finally we remark that P systems with internal output still could (mis)use the environment to let objects vanish, yet we will assume that such symbols will be erased instead of being sent out, so for such P systems, without loss of generality, we can assume that there is no communication with the environment at all.

Remark 2. In order to avoid counting the catalysts in the results, we can also make a projection erasing them. Whereas in general we would write $Y_{gen,maxpar}OP_1^T\,(cat_2)$, instead we now would write $Y_{gen,maxpar}OP_{1,int}^{-cat}\,(cat_2)$. In this case, we really use one membrane only, as only one membrane region itself is needed to obtain the results.

Remark 3. Usually, catalytic P systems and many other variants of P systems can be flattened to one membrane, see [6]. Yet in general, flattening means that we have to make a terminal projection to get the results or to use external output for that purpose, i.e., with catalytic P systems flattened to one membrane, clean

results cannot be obtained without using external out or terminal extraction. In fact, all results in the P systems area should be carefully inspected with respect to these subtle details of complexity definitions.

As we shall see in the following sections, the way how to obtain the results in many cases will have a significant influence on the computational power of several variants of P systems with toxic objects.

Remark 4. As in this paper we will only consider P systems using the maximally parallel derivation mode, in the following the subscript *maxpar* will be omitted.

Finally, P systems can also be considered as mechanisms for generating and accepting string languages as well as for computing any partial recursive function $f : \Sigma^* \to \Gamma^*$ on strings. Here the input string consists of the sequence of symbols taken in from the environment during a halting computation and the output string is formed by the sequence of symbols sent out to the environment; hence, the P system works like in the automaton style, but the input and output streams of symbols are interpreted as strings. In general, any number of symbols can be taken in and sent out in one computation step, and any possible sequence of those symbols has to be taken into account as a substring to be concatenated with the strings already computed by the preceding computation steps – thus, not only one input and one output string may result from a successful halting computation.

The string relation computed by halting computations in a P system Π is denoted by $L_{com}(\Pi)$. If we only consider the symbols taken in from the environment, $L_{com}(\Pi)$ can be seen as an automaton accepting the strings computed by the sequences of symbols taken in during halting computations and we also write $L_{aut}(\Pi)$; if no symbols are taken from the environment, $L_{com}(\Pi)$ describes a string language generated by Π and we also write $L_{gen}(\Pi)$. By $L_{\delta}OP_m(X)$, $\delta \in \{gen, aut\}$, as well as by $RelL_{com}OP_m(X)$ and $FunL_{com}OP_m(X)$ we denote the families of string languages generated and accepted as well as the families of string relations and functions computed by P systems with at most m membranes using rules of type X. With $FunRE$ and $RelRE$ denoting the class of partial recursive string functions and relations, respectively, the following results can be derived from the results proved in [5] (for the generating case, also see [15], Theorem 4.17):

Theorem 1. *For any $\delta \in \{gen, aut\}$, $Z \in \{Fun, Rel\}$*

$$RE = L_{\delta}OP_1(cat_2) = L_{\delta}OP_1(pcat_3)$$

as well as

$$ZRE = ZL_{com}OP_1(cat_2) = ZL_{com}OP_1(pcat_3).$$

3 Toxic Objects in P Systems

We specify a specific subset O_{tox} of O as *toxic* objects. Toxic objects must not stay idle as otherwise the computation is abandoned without yielding a result.

In a successful computation, in any computation step continuing a derivation, we always have to apply multisets of rules evolving all toxic objects. On the other hand, if no rule can be applied any more and thus the system halts, toxic objects do no harm and we take out the results in the usual way depending on the specific definition for the systems under consideration.

A P system with toxic objects is only allowed to continue a computation from a configuration $\mathbb{C}$ by using an applicable multiset of rules covering all copies of objects from O_{tox} occurring in $\mathbb{C}$; moreover, if every non-empty multiset of applicable rules is not covering all toxic objects, the whole computation having yielded the configuration $\mathbb{C}$ is abandoned, i.e., no results can be obtained from this computation.

For any variant of P systems, we add the set of *toxic* objects O_{tox} and in the specification of the families of sets of (vectors of) numbers generated/accepted by P systems with toxic objects using rules of type X we add the subscript tox to O, thus obtaining the families $Y_\gamma O_{tox} P_m(X)$, for any $Y \in \{N, Ps, L\}$, $\gamma \in \{gen, acc, aut\}$, and $m \geq 1$.

3.1 Variants of P Systems with Toxic Objects

We may distinguish the following variants:

- all symbols are toxic, i.e., we write $Y_\gamma O_{tox\,all} P_m(X)$;
- in catalytic P systems, exactly the catalysts are toxic, i.e., we write $Y_\gamma O_{tox\,cat} P_m(X)$;
- at least the catalysts are toxic, i.e., we write $Y_\gamma O_{tox \geq cat} P_m(X)$;
- all except the catalysts are toxic, i.e., we write $Y_\gamma O_{tox - cat} P_m(X)$.

In all these notations, we may add the superscript T to indicate terminal extraction or the superscript $-cat$ to indicate that the catalysts are not taken into account for the results; moreover, we replace P_m by $P_{m,ext}$ or $P_{m,int}$ in order to explicitly specify that the system uses external or internal output, respectively.

Remark 5. The results established in the following implicitly may assume the P system to be flattened to one membrane, but in the sense of the previous remarks, we have to be very careful whether we have internal output, so that toxicity of symbols matters, or else we have external output, in which case we assume that the objects sent out do not affect the work of the system any more.

4 Purely Catalytic P Systems with All Objects Being Toxic

We first consider the specific variants of P systems which in any step only allow for a bounded number k of rules to be applied, for example, purely catalytic P systems. Obviously, in this case, as until the end of a computation every symbol has to be affected by a rule, at most k symbols can evolve in any computation

step, which of course bounds the number of possible configurations by a constant number, too.

For the generative case with internal output, we show that we get precisely all finite sets, while for the accepting case (i.e., internal input), the power is also quite limited – everything which is not finite is arbitrary, and, moreover, only specific finite sets are accepted.

Lemma 1. *For all $m \geq 1$ and all $k \geq 1$,*

$$Ps_{gen}O_{toxall}P^{-cat}_{m,int}\left(pcat_k\right) = PsFIN.$$

Proof. For the forward inclusion, we notice that the initial configuration is fixed, and the size of a vector we can generate by halting in any non-initial configuration is bounded by the maximal sum of the right-hand sides of rules over different catalysts.

For the converse inclusion, it is enough to mention that for any finite set F of d-dimensional vectors of non-negative integers, there exists a P system Π of type $O_{toxall}P^{-cat}_{1,int}\left(pcat_1\right)$ such that

$$\Pi = \left(O = \{c_1, a\} \cup T, C = \{c_1\}, \mu = \left[\ \right]_1, w_1 = c_1 a, R_1, 1\right),$$

where $\{c_1, a\} \cap T = \emptyset$, $|T| = d$ with T being written $\langle a_1, \cdots, a_d \rangle$ as an ordered set, and R_1 consists of precisely one rule $c_1 a \to c_1 a_1^{k_1} \cdots a_d^{k_d}$ for every element $(k_1, \cdots, k_d) \in F$, so each element of F is generated in one step; for $F = \emptyset$, we simply take the rule $c_1 a \to c_1 a$, which causes an infinite computation. For both cases, we can define R_1 as follows:

$$R_1 = \left\{ c_1 a \to c_1 a_1^{k_1} \cdots a_d^{k_d} \mid (k_1, \cdots, k_d) \in F \right\} \cup \{c_1 a \to c_1 a\}.$$

□

Lemma 2. *For all $m \geq 1$ and all $k \geq 1$,*

$$\begin{aligned} N_{acc}O_{toxall}P_m\left(pcat_k\right) = &\{\{d\} \mid 0 \leq d \leq k-1\} \\ &\cup \{\{0, k'\} \mid 0 \leq k' \leq k\} \cup \{\emptyset, \mathbb{N}\}. \end{aligned}$$

Proof. We proceed with the forward inclusion. Take an arbitrary P system Π of type $O_{toxall}P_m\left(pcat_k\right)$, where

$$\Pi = (O, C = \{c_1, \cdots, c_{k'}\}, \Sigma \subseteq O \setminus C, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m, i_0),$$

where $k' \leq k$. Before the computation starts, input $w_0 \in \Sigma^*$ is added to w_{i_0}. Only two cases are possible that do *not* lead to a computation which is not abandoned immediately: either the P system halts immediately, or all objects from $w_1, \cdots, w_m$ as well as all the objects from $w_0 \in \Sigma^*$ additionally placed in region i_0 participate in catalytic rules in the first step, hence the number of catalysts must be equal to

the number of non-catalysts. In the second case, it follows that the size $|w_0|$ of the input w_0 must be equal to $k' - \sum_{i=1}^{m} |w_i|$. In the first case, it is easy to see that if Π immediately halts on some *non-empty* input, then it must also immediately halt on the *empty* input. If we have at least one rule for every input symbol in Σ, then immediate halting happens *only* on the empty input. If, however, there exists at least one symbol from Σ that does not appear in the left side of any rule from R_{i_0}, then *any* number of these symbols (let us call them "passive") would be accepted.

We now put it all together. For the same system, having both immediate halting case and later halting case is only possible if besides the input, the initial system has *only* catalysts. This yields exactly $\{0, k'\}$, $k' \leq k$ if there are no passive objects in Σ, or the entire set $\mathbb{N}$ otherwise. Only immediate halting yields $\{0\}$ and $\mathbb{N}$, depending on the presence of passive objects in Σ. Finally, only later halting yields $\{d\}$ for $0 \leq d < k$ (the last inequality is strict since at least one non-catalyst is needed besides the input to reject 0). And of course, we may have a P system with no halting computations, accepting $\emptyset$. The family of all sets mentioned above is $\{\{d\} \mid 0 \leq d \leq k-1\} \cup \{\{0, k'\} \mid 0 \leq k' \leq k\} \cup \{\emptyset, \mathbb{N}\}$, which proves the forward inclusion of the claim of the lemma.

For the converse inclusion, it is enough to exhibit P systems for each of these sets; in every case, the input alphabet is $\Sigma = \{a\}$.

$$\begin{aligned}
&\Pi_{\emptyset} = \left(O = \{c_1, a\}, C = \{c_1\}, \Sigma = \{a\}, \mu = \left[\ \right]_1, w_1 = c_1 a, R_1, i_0 = 1\right), \\
&R_1 = \{c_1 a \to c_1 a\};
\end{aligned}$$

$$\begin{aligned}
&\Pi_{\mathbb{N}} = \left(O = \{c_1, a\}, C = \{c_1\}, \Sigma = \{a\}, \mu = \left[\ \right]_1, w_1 = c_1, R_1, i_0 = 1\right), \\
&R_1 = \emptyset;
\end{aligned}$$

$$\begin{aligned}
&\Pi_{d,0} = \left(O = C \cup \{a\}, C = \{c_1, \cdots, c_d\}, \Sigma = \{a\}, \mu = \left[\ \right]_1, w_1, R_1, i_0 = 1\right), \\
&w_1 = c_1 \cdots c_d, \\
&R_1 = \{c_i a \to c_i \mid 1 \leq i \leq d\},\ 0 \leq d \leq k;
\end{aligned}$$

$$\begin{aligned}
&\Pi_{d} = \left(O = C \cup \{a\}, C = \{c_1, \cdots, c_{d+1}\}, \Sigma = \{a\}, \mu = \left[\ \right]_1, w_1, R_1, i_0 = 1\right), \\
&w_1 = c_1 \cdots c_{d+1} a, \\
&R_1 = \{c_i a \to c_i \mid 1 \leq i \leq d\},\ 0 \leq d \leq k-1.
\end{aligned}$$

Indeed, the only rule of $\Pi_{\emptyset}$ forces an infinite loop on the empty input, while for a non-empty input the computation is blocked because more than one toxic object a cannot be simultaneously taken by c_1. On the other end of the spectrum, $\Pi_{\mathbb{N}}$ accepts any input by immediate halting, because the catalyst always stays idle as there is no rule in the system. P system $\Pi_{d,0}$ either halts immediately with no input, or halts after one step, erasing the input of exactly d objects, $d \leq k$. Finally, the P system Π_d halts after one step, erasing the input of exactly d objects, $d < k$.

These observations conclude the proof. □

Note 1. Of course, characterizing sets of vectors in the accepting case would be more tedious for the following reason. Without passive objects, P system would accept *some* subset of $\Sigma^{\leq k-1}$, as well as a subset of $\Sigma^{\leq k}$ containing *some* vectors of weight $d \leq k$ and a vector of zeros, and the empty set. With passive objects, *any* number of them is allowed for the case of immediate halting, while the projection of the accepted vectors onto the non-passive objects should form a set containing *some* vectors of weight $d \leq k$ and a vector of zeros. The meaning of the word *some* throughout this note can be made more precise by analyzing exactly which multisets of weight d can be consumed by d catalysts, depending on the rules of the system (whereas in the case of accepting numbers, only the total weight of such multisets was taken into consideration).

While Lemma 1 characterized $PsFIN$ by P systems with internal output, in the case of external output their power becomes exactly $PsREG$, as the following theorem shows:

Lemma 3. *For all $m \geq 1$ and all $k \geq 1$,*

$$Ps_{gen}O_{toxall}P_{m,ext}\left(pcat_k\right) = PsREG.$$

Proof. Let $M = (Q, T, \delta, q_0, F)$ be a deterministic finite automaton. Then we construct the P system Π which generates $Ps\left(L\left(M\right)\right)$:

$$\begin{aligned}
\Pi &= \left(O = C \cup Q \cup T, C = \{c_1\}, \mu = \left[\ \right]_1, w_1 = c_1 q_0, R_1, i_0 = 0\right), \\
R_1 &= \{c_1 p \to c_1 q(a, out) \mid (p, a, q) \in \delta, p, q \in Q, a \in T\} \\
&\quad \cup \{c_1 p \to c_1 \mid p \in F\}.
\end{aligned}$$

We conclude that $PsREG \subseteq Ps_{gen}O_{toxall}P_{1,ext}\left(pcat_1\right)$.

The converse inclusion can be argued as follows: In any successful computation step with k catalysts, there must be exactly k non-catalysts, and a computation stops with having yielded a result if all objects inside the system including the catalysts are idle; hence, this finite set of useful configurations is finite and constitutes the set of states of a finite automaton simulating the computations of the P system. Since every rule in such a system involves one catalyst and one non-catalyst, for a configuration C to allow some derivation $C \Rightarrow C'$ it is necessary (although not sufficient) that the number of catalysts equals the number of non-catalysts inside the system. Hence, for a P system

$$\Pi = (O, C, \mu, w_1, \cdots, w_{m'}, R_1, \cdots R_{m'}, i_0)$$

having fixed the set of objects O, the membrane structure μ of $m' \leq m$ membranes, and the set of catalysts C, with the number $k' \leq k$ of catalysts, the set Q of configurations containing a total of exactly k' objects from $O \setminus C$ in m' regions of the P system is bounded. Moreover, the set Q'' of all configurations reachable from Q in one step is also bounded. Finally, we define $Q' = Q'' \cup \{q_0\}$ where q_0 is

the initial configuration, as well as $Q_h \subseteq Q'$ to be the set of halting configurations (in which no rule can be applied any more).

Hence, a P system with external output generating vectors of natural numbers can be modeled by a finite automaton $M = (Q', T, \delta, q_0, Q_h)$ having Q' as the set of states, T contains d symbols for the generation of d-dimensional vectors, δ contains the triple (p, v, q) for any transition from a configuration $p \in Q'$ to a configuration $q \in Q'$ sending out v; the set of the final states is precisely Q_h.

In sum, with all objects being toxic, purely catalytic P systems with external output can exactly generate the regular sets of vectors. □

The statement of Lemma 3 can be generalized to languages, as well as to P automata and P transducers. Indeed, in case of external input (P automaton case) and/or external output, the finite number of different configurations can serve as the finite state set of a finite automaton for the input specified by $(a, come)$ in the rules and/or for the output specified by (a, out) in the rules.

Lemma 4. *For all $m \geq 1$ and all $k \geq 1$,*

$$L_{gen}O_{toxall}P_m\,(pcat_k) = REG.$$

Proof. Using similar arguments as already pointed out in the previous proof, we can easily argue that purely catalytic P systems with all objects being toxic can generate any regular language L; the only difference now is that any sequence of symbols sent out during a successful computation is interpreted as string.

Now we consider the converse, i.e., as in the previous proof, a P system with external output generating strings can be modeled by the finite automaton having Q' as the set of states and Q_h as the set of the final states as constructed there, but now for any transition from a configuration $p \in Q'$ to a configuration $q \in Q'$ sending out v, δ contains the triple (p, v', q) for all $v' \in Perm(v)$. □

Lemma 5. *For all $m \geq 1$ and $k \geq 2$,*

$$L_{aut}O_{toxall}P_m\,(pcat_k) = REG,$$
$$Rel_{aut}O_{toxall}P_m\,(pcat_k) = RelREG.$$

Proof. For a P automaton or a P transducer Π, again take Q' and Q_h as constructed in the proof of Lemma 4.

A P automaton can be modeled by a finite automaton having Q' as the set of states. A transition from configuration $p \in Q'$ to a configuration $q \in Q'$ while having u brought from the environment is simulated by rules (p, u', q) for all $u' \in Perm(u)$; clearly, $|u| \leq k'$. The set of the final states is precisely Q_h.

A P automaton with external output can be modeled by a finite transducer having Q' as the set of states. A transition from configuration $p \in Q'$ to a configuration $q \in Q'$ while having u brought in and v sent out is simulated by rules

$(p, u'/v', q)$, $u' \in Perm(u)$, $v' \in Perm(v)$; clearly, $|u| \leq k'$ and $|v| \leq k'$. The set of the final states is precisely Q_h.

For proving the converse inclusion, take an arbitrary finite automaton $M = (Q, T, \delta, q_0, F)$; without loss of generality, we assume that M has at least one outgoing transition from any non-final state.

A P automaton

$$\Pi = \left(O = \{c_1, c_2, b\} \cup T, C = \{c_1, c_2\}, \Sigma = T, \mu = \left[\ \right]_1, w_1 = c_1 c_2 b, R_1, i_0 = 1\right)$$

$$R_1 = \{c_1 p \to c_1 q(a, come) \mid (p, a, q) \in \delta, p, q \in Q, a \in T\}$$
$$\cup \{c_2 x \to c_2 \mid x \in \{b\} \cup T\} \cup \{c_2 q \to c_2 \mid q \in F\}$$

can simulate M, using two catalysts c_1, c_2. Each transition $(p, a, q) \in \delta$ can be simulated by rule $c_1 p \to c_1 q(a, come)$, but also rule the $c_2 a \to c_2$ is needed to erase the symbol previously brought in. The initial contents of a single membrane is $c_1 c_2 q_0 b$, where besides q_0, one additional non-catalyst $b \notin \{c_1, c_2\} \cup T$ is used to keep c_2 busy in the first step. Halting can be simulated by rule $c_1 q \to c_1$ for each final state q; in the same step c_2 deletes the last symbol brought in.

A P automaton with external output can simulate a finite transducer in the same way as a P automaton without output simulates a finite automaton. The only difference is that now the simulated transitions have the form $(p, a/u, q)$, and the corresponding simulating rules are $c_1 p \to c_1 q(a, come)(u, out)$, the rest of the construction being exactly the same as in the previous paragraph. □

Two catalysts are needed for simulating an arbitrary finite automaton by a P automaton, since both the state symbol and the symbol brought in from the environment have to be processed in parallel. For the case of only one catalyst, the object brought in from the environment itself has to serve as a state. However, in this way, the last object brought inside completely determines the set of possible objects that can be brought inside in the next step, which considerably reduces the generality of finite automata. Having this in mind, we characterize input-driven finite automata:

Lemma 6. *For all $m \geq 1$,*

$$L_{aut} O_{toxall} P_m (pcat_1) = IDREG,$$
$$Rel_{aut} O_{toxall} P_m (pcat_1) = RelIDREG.$$

Proof. The inclusion that at most $IDREG/RelIDREG$ is generated/computed with one catalyst follows from the fact that exactly one non-catalyst may appear in any non-halting non-blocking configuration, and, except the initial configuration, this is precisely the symbol taken from the environment in the previous step. In any successful computation, the only rules applied in any step, possibly except in the last step, are erasing one non-catalyst while bringing in another one instead.

For the inclusion that we can generate/compute the entire families $IDREG/RelIDREG$, we use a construction similar to that of the previous lemma,

except instead of erasing the object a brought in the previous step by c_2, this object is used instead of the state object: each transition (q_a, b, q_b) now is simulated by the rule $c_1 a \to c_1(b, come)$. The initial contents of a single membrane is $c_1 q_0$, where the initial state q_0 is an additional symbol not in the input alphabet. Halting can be simulated by the rules $c_1 a \to c_1$ for all final states q_a.

For the case of a P automaton with external output, the simulation is the same as above, except that now we also have an output: the transitions to be simulated have the form $(q_a, b/u, q_b)$, and the corresponding simulating rules are $c_1 a \to c_1(b, come)(u, out)$; the rest of the construction is exactly the same as in the previous paragraph. □

Finally, the domain of relations computed with internal input (with either output region) corresponds to the sets accepted with internal input, see Lemma 2. Similarly, the range of relations computed with internal output and with internal input corresponds to the sets generated with internal output, see Lemma 1. The nature of these relations always results from a finite-state behavior, but we are not going into further details here; another question to be answered in the future is the exact characterization of P automata with internal output.

5 Non-Cooperative P Systems with All Objects Being Toxic

In this section we consider P systems without catalysts and with only non-cooperative rules, yet with all objects being toxic.

5.1 Connection to L Systems

Example 1. Take the following P system with all objects being toxic.

$$\Pi_{int} = \left(O = \{a, b\}, \mu = \left[\ \right]_1, w_1 = a, R_1 = \{a \to aa,\ a \to b\}, i_0 = 1\right).$$

In n computation steps we obtain a^{2^n} and in a final step b^{2^n}. Only in this last step we may apply the rule $a \to b$ introducing the toxic symbol b for which no rule exists. Hence, the generated set is $N_{gen}(\Pi_{int}) = \{2^n \mid n \geq 0\}$.

Example 2. The same set is accepted by the P automaton Π_{aut} and generated by the P system Π_{ext} with external output:

$$N_{aut}(\Pi_{aut}) = N_{gen}(\Pi_{ext}) = N_{gen}(\Pi_{int}) = \{2^n \mid n \geq 0\} \text{ where}$$
$$\Pi_{aut} = \left(O = \{a, b\}, \mu = \left[\ \right]_1, w_1 = a, R_1 = \{a \to aa,\ a \to (b, come)\}, i_0 = 1\right);$$
$$\Pi_{ext} = \left(O = \{a, b\}, \mu = \left[\ \right]_1, w_1 = a, R_1 = \{a \to aa,\ a \to b(b, out)\}, i_0 = 0\right).$$

Indeed, the behavior of Π_{aut} and Π_{ext} is the same as that of Π_{int}, except producing b inside the membrane is replaced by bringing in b from the environment, or accompanied by sending out b to the environment. Again, if both rules are simultaneously applied, then the toxic objects b will block the computation, but still we will get a result if only symbols b are present in the final configuration.

We now compare non-cooperative P systems with all objects being toxic to L systems.

Lemma 7. *For* $(\gamma, \alpha) \in \{(gen, int), (gen, ext), (aut, -)\}$,

$$PsE0L \subseteq Ps_{\gamma}O_{toxall}P_{1,\alpha}\,(ncoo)\,.$$

Proof. Let $G = (V, T, P, w)$ be an $E0L$ system. We recall that for each symbol $a \in V$, P has to contain some rule with a on the left side. For every $a \in T$, we replace a by N_a throughout all the rules of P. Moreover, we take $N_a \to a$ into R for every $a \in T$. These terminal rules are exactly applied in the last step of a derivation in the P system Π, whereas the other rules in R simulate the rules from P. As the final step with using the rules $N_a \to a$ is an additional derivation step, instead of rules $a \to \lambda$ erasing a symbol a we instead have to use the rules $N_a \to E$ and $E \to \lambda$ where E is a new symbol representing the erased symbol for one step.

Hence, we construct the corresponding non-cooperative P system Π with all symbols being toxic as follows (for the automaton case, we have to insert $\Sigma = T$):

$$\begin{aligned}
\Pi &= (O = V \cup \{N_a \mid a \in T\} \cup \{E\}, \mu = [\ ]_1\ , w_1 = w, R_1, i_0 = 1),\\
R_1 &= \{h(a) \to h(u) \mid a \to u \in P,\ a \in V\} \cup \{N_a \to E \mid a \to \lambda \in P,\ a \in T\}\\
&\quad \cup \{N_a \to a \mid a \in T\} \cup \{E \to \lambda\}\,,
\end{aligned}$$

where $h : V \cup \{E\} \to V \setminus T \cup \{N_a \mid a \in T\} \cup \{E\}$ is the morphism given by $h(a) = a$ for $X \in V \setminus T \cup \{E\}$ and $h(a) = N_a$ for $a \in T$.

By construction, every object from $\{N_a \mid a \in T\} \cup \{V \setminus T\}$ can evolve by rules from R_1, while objects in T cannot. If a computation in Π ends up with a configuration in which the skin contains both objects from T and objects not from T, then the computation is blocked without yielding any result. Therefore, the only derivations of Π which will not be discarded are those in which Π simulates a derivation of G up to some configuration $h(w)$, $w \in (T \cup \{E\})^*$, and then applies the rules $N_a \to a$ and eventually the rule $E \to \lambda$, and only those, to transform $h(w)$ into w.

To show the same result for external output, it suffices to set i_0 to 0 and replace every rule $N_a \to a$ by $N_a \to a(a, out)$. Alternatively, to show the same result for external input, it suffices to set i_0 to 0 and replace every rule $N_a \to a$ by $N_a \to a(a, come)$. □

In the case of P systems with internal output (without terminal filtering) and only one membrane, we can directly show the converse inclusion.

Lemma 8. $Ps_{gen}O_{toxall}P_{1,int}\,(ncoo) \subseteq PsE0L$.

Proof. Let $\Pi = (O, \mu = [\ ]_1\ , w_1, R_1, i_0 = 1)$ be a non-cooperative P system with all objects being toxic. We construct the corresponding $E0L$ system G as follows:

$$G = (V = O \cup \{\#\}, T, P, w = w_1),$$
$$T = \{a \in O \mid \text{there exists no rule } a \to u \in R \text{ with } u \in O^*\},$$
$$P = R_1 \cup \{a \to \# \mid a \in T \cup \{\#\}\}.$$

We immediately observe that, whenever G introduces a terminal symbol, it will be rewritten into a trap symbol in the next step. Thus, the only way for G to produce a terminal string is to move from a string over $V \setminus T$ to a string over T in a single step. But this exactly corresponds to the way in which Π evolves, because rewriting a terminal a into # in G corresponds to discarding the derivation of Π in which a is produced alongside non-terminals. □

Corollary 1. $Ps_{gen}O_{toxall}P_{1,int}(ncoo) = PsE0L$.

Proof. The result follows from Lemma 8 in combination with Lemma 7 for the case of P systems with internal output and only one membrane. □

In case of multiple membranes or terminal filtering or both, however, there is a problem: symbols that represent objects in non-output regions do not contribute to the output. Yet, since $E0L$ is known to be closed under arbitrary morphisms (see, e.g., [16] volume 1 page 34), the result can be strengthened as follows:

Theorem 2. *For all* $m \geq 1$,

$$\begin{aligned} PsE0L &= Ps_{gen}O_{toxall}P^T_{m,int}(ncoo) \\ &= Ps_{gen}O_{toxall}P_{m,int}(ncoo) \\ &= Ps_{gen}O_{toxall}P_{m,ext}(ncoo) \\ &= Ps_{aut}O_{toxall}P_m(ncoo). \end{aligned}$$

Proof. We only have to show that any P system with internal output, eventually even with terminal extraction, or else with external output (terminal extraction need not be considered in this case, as any non-wanted symbol need not be sent out) or external input can be simulated by an $E0L$-system.

First, we can flatten the given P system Π with internal output to only one membrane, yet keeping in mind that then we have to use terminal extraction to obtain the results in a clean form. Hence, in this case, we simply apply the construction from Lemma 8 to the flattened P system Π' thus obtaining an $E0L$-system G generating a set of strings which exactly represent the multisets generated by the P system Π'. In order to obtain the original results, we have to apply a projection h_T erasing all non-terminals only yielding strings/multisets over T. As $E0L$ is closed under arbitrary morphisms (see, e.g., [16] volume 1 page 34), from G we can construct an $E0L$-system G' directly generating the desired results. If the original P system Π used terminal extraction to a terminal alphabet Σ, we can to apply another projection from T to Σ to obtain the desired results (by constructing a corresponding $E0L$-system G'').

If we have a P system Π using external output, we instead first flatten the system to an equivalent P system Π' with only one membrane, but still having external output. Then, from this P system Π' with one membrane and external output we construct an equivalent P system Π'' with only one membrane region having internal output in the skin membrane, but with terminal filtering. Instead of sending a symbol a out by using (a, out) on the right side of a rule in Π, we replace any occurrence of (a, out) by N_a in the rules of the skin membrane in Π''. In that way, the P system Π'' keeps each symbol a sent out by Π in the new inner output membrane $i_0 = 1$ as N_a. In the skin membrane, the rules $N_a \to N_a$ keep these symbols alive until the application of the rules $N_a \to a$ allows the system to halt with yielding the desired result if exactly with the application of these terminal rules no non-terminal afterwards remains in the whole system. As a subtle detail we have to mention that we again, as in the proof of Lemma 7, have to be careful with λ-rules $a \to \lambda$ (again we use the rules $a \to E$ and $E \to \lambda$ instead), but now also with any other "passive" object b which cannot evolve any more - such a symbol has to be treated like a terminal symbol, going to an intermediate symbol N_b before it finally goes to b, and then each of these symbols is projected to λ by using the terminal extraction.

Hence, we conclude that computations in Π'' and in Π yield the same results. According to the construction given above, we can obtain an $E0L$-system G'' such that $Ps\left(L\left(G''\right)\right) = Ps\left(\Pi''\right) = Ps\left(\Pi\right)$.

In the automaton case, we can use similar ideas as in the previous case: instead of $(a, come)$ for terminal symbols from the input alphabet Σ on the right side of rules of a P automaton Π we use $N_a N'_a$ in a flattened P system Π'' with internal output and terminal extraction. Then N'_a is used instead of a in the rules of Π'' used instead of the corresponding rules of Π, and we also add the rules $N_a \to N_a$ and the terminal rules $N_a \to a$. Moreover, any "passive" object b which cannot evolve any more has to be treated as already explained before; the same holds for the dealing with λ-rules $a \to \lambda$. Hence, finally projecting all terminal symbols on themselves and all other symbols on λ with the projection h_Σ, we have got a P system with internal input and terminal extraction Π'' such that $h_\Sigma(Ps(\Pi'')) = Ps(\Pi)$. □

5.2 Internal Input

However, the accepting power of P systems with internal input is much lower, namely subregular.

Lemma 9. $Ps_{acc}O_{tox\,all}P_m(ncoo) \subsetneq PsREG$.

Proof. First, if a P system Π accepts *any* non-empty input over the input alphabet Σ, then also *the empty input* is accepted. Indeed, take an arbitrary P system Π accepting some multiset $w_{in} \in \Sigma^*$, say in m steps. Each of the objects, both initial ones and input ones, initially being in the system, will produce some (possibly

empty) multiset of objects which cannot further evolve by rules of Π. Clearly, replacing w_{in} by λ and following exactly the same evolution of all initial objects, we will get an accepting computation of at most m steps.

Second, for a similar reason, if w_{in} is accepted, than any submultiset of w_{in} is accepted.

Third, if Π accepts some input w_{in} containing *at most* one occurrence of any symbol in Σ, then it also accepts every input $w'_{in} \in (alph\,(w_{in}))^*$, i.e., *any* multiset over $\{a \in \Sigma \mid |w_{in}|_a > 0\}$. Indeed, consider the accepting computation of w_{in}, say of m steps. In this computation, every input object a from w_{in} is either erased in at most m steps, or produces in exactly m steps some non-empty multiset of objects that cannot evolve by rules of Π; let us call such symbols "passive". Replacing each input object by an arbitrary number of its copies, following the same evolution as in the accepting computation before, we again get a computation where every input object is either erased in at most m steps, or produces in exactly m steps some non-empty multiset of passive objects. This computation will either erase everything in at most m steps, or halt in exactly m steps.

Therefore, non-cooperative P systems with internal input with all symbols being toxic can accept *at most* all possible unions of sets from $\{T^* \mid T \subseteq \Sigma\}$. □

It can be shown that last statement from the proof given above actually is an equality.

Theorem 3. *For all $m \geq 1$,*

$$Ps_{acc}O_{toxall}P_m(ncoo) = \left\{ Ps\left(\bigcup_{i=1}^{n} T_i^*\right) \mid \Sigma \text{ alphabet}, T_i \subseteq \Sigma,\ 1 \leq i \leq n,\ n \geq 0 \right\}.$$

Proof. The inclusion $\subseteq$ follows from the proof of the previous theorem, so it suffices to prove that all such sets can indeed be accepted. For $n = 0$, $\emptyset$ can be accepted by the P system

$$\Pi = \left(O = \{a, b\}, \Sigma = \{a\}, \mu = [\ ]_1, w_1 = b, R_1 = \{b \to b\}, i_0 = 1\right).$$

Take arbitrary numbers $n > 0$ and $k > 0$, an input alphabet $\Sigma = \{a_{j,0} \mid 1 \leq j \leq k\}$ and sets $T_i \subseteq \Sigma$, $1 \leq i \leq n$. We construct a P system accepting precisely all inputs from $\bigcup_{i=1}^{n} T_i^*$.

$$\begin{aligned}
\Pi &= \left(O, \Sigma, \mu = [\ ]_1, w_1 = \lambda, R_1, i_0 = 1\right),\\
O &= \{a_{j,i} \mid 0 \leq i \leq n+1,\ 1 \leq j \leq k\} \cup \{b\},\\
R_1 &= \{a_{j,i} \to a_{j,i+1} \mid 0 \leq i \leq n,\ 1 \leq j \leq k\}\\
&\quad \cup \{a_{j,i} \to b \mid 1 \leq i \leq n,\ 1 \leq j \leq k, a_{j,0} \in T_i\}\\
&\quad \cup \{a_{j,n+1} \to a_{j,n+1} \mid 1 \leq j \leq k\}.
\end{aligned}$$

Indeed, every input object $a_{j,0}$ either enters an infinite loop after $n+1$ steps, or evolves into b (that cannot further evolve by the rules of Π) after some number i

of steps if $a_{j,0} \in T_i$. The only way a non-empty input is accepted is if all objects evolve into b in the same number i of steps, which is possible if and only if the input is contained in $\bigcup_{i=1}^{n} T_i^*$. □

5.3 Describing Languages

In the previous subsection we have studied numbers and vectors described by non-cooperative P systems with all objects being toxic. We have shown that they characterize very limited subregular behavior in the case of internal input, while in the cases of external input, internal output or external output their power is strongly related to that of $E0L$ systems. A natural question arises – what can we say about languages? We now illustrate the difficulty of this problem by a few examples, generating non-context-free languages and illustrating the power of synchronization emerging from halting with toxic objects.

Example 3. Our first example shows how a simple non-context-free language can be accepted using external input:

$$\Pi = \left(O = \{S, a, b, c\}, \mu = \left[\ \right]_1, w_1 = S, R_1, i_0 = 0\right),$$
$$R_1 = \{S \to S(a, come),\ S \to a,\ a \to a,\ a \to (bc, come)\}.$$

This P system accepts the language $L_{a(bc)} = \bigcup_{n \geq 1} \{a^n\}\, Perm\,(b^n c^n)$, since the multiplicities of symbols a, b and c brought in are the same, and all objects b and c must be brought inside in the same (i.e., in the last) step of the computation, which must be after all objects a have been brought inside.

Note that, without toxicity, the result would still be some non-context-free language consisting of the same number of symbols a, b and c, but it would also contain strings which are not of the form $\{a\}^* \{b, c\}^*$.

It is no longer surprising that replacing $(x, come)$ by $x(x, out)$ for all $x \in \{a, b, c\}$ in the system above, we get a P system with external output *generating* the same language $L_{a(bc)}$.

Example 4. If, in the previous example, we replace each rule $a \to (bc, come)$ by the two rules $a \to (b, come)$ and $b \to (c, come)$, we get a P automaton accepting language $L_{a(b)(c)} = \{a^n b^n c^n \mid n \geq 1\}$, because to halt without being blocked, all objects c must be brought inside in the same step, and therefore also all objects b must have been brought inside just one step before that, and hence all objects a must have already been brought in by then. Clearly, replacing $(x, come)$ by $x(x, out)$ for all $x \in \{a, b, c\}$ throughout all the rules, we get a P system with external output again *generating* the language $L_{a(b)(c)}$.

In the rest of this section we show that non-cooperative P systems can generate rather complicated languages even without making use of the synchronization power of toxicity, and taking all objects to be toxic does not change the language.

The following example of a difficult language generated by a non-cooperative P system with external output is taken from [2]. This language is considerably more "difficult" than languages in $REG \cdot Perm(REG)$, which informally can be explained as follows: besides permutations of symbols sent out at the same time, it exhibits another kind of non-context-freeness, although this second source of "difficulty" alone, however, could be captured as the intersection of two linear languages.

Example 5. Consider the non-cooperative P system with external output

$$\begin{aligned}\Pi_D &= \left(O = \{D, D', a, b, c, a', b', c'\}, \mu = \left[\ \right]_1, w_1 = DD, R_1, i_0 = 0\right),\\ R_1 &= \big\{D \to (a, out)(b, out)(c, out)D'D',\ D \to (a, out)(b, out)(c, out),\\ &\quad D' \to (a', out)(b', out, (c', out)DD,\ D' \to (a', out)(b', out)(c', out)\big\}.\end{aligned}$$

The contents of region 1 is a population of objects D, initially 2, which are primed if the step is odd. Assume that there are k objects inside the system. In each step, every symbol D is either erased or doubled (and primed or deprimed), so the next step the number of objects inside the system will be any even number between 0 and $2k$. In addition to that, the output during that step is $Perm\left((abc)^k\right)$, primed if the step is odd. Hence, the generated language can be described as

$$\begin{aligned}L(\Pi_D) = \bigcup\nolimits_{k_0=1,\ 0\le k_i\le 2k_{i-1},\ 1\le i\le 2t+1,\ t\ge 0}\ & Perm\left((abc)^{2k_0}\right) Perm\left((a'b'c')^{2k_1}\right)\cdots\\ & Perm\left((abc)^{2k_{2t}}\right) Perm\left((a'b'c')^{2k_{2t+1}}\right).\end{aligned}$$

To give an idea of how complex a language generated by a non-cooperative membrane system can be, imagine that the skin may contain populations of multiple symbols that (like D in the example above) can be erased or multiplied (with different periods), and also be rewritten into each other. The same, of course, happens in usual context-free grammars, but since the terminal symbols in P systems with external output are collected from the derivation tree level by level instead of from left to right as in context-free grammars, the effect is quite different.

We finally again mention that the generated language remains the same even if all objects are toxic. Moreover, by replacing all outputs of the form (x, out) by $(x, come)$ and adding rules $x \to \lambda$ we can convert this P system with external output into a P automaton defining the same language.

6 Catalytic P Systems with Exactly the Catalysts being Toxic Generate at Least $PsMAT$

In this section we investigate catalytic P systems where precisely the catalysts are toxic, i.e., the computation is aborted if any of them is not used in some step

before the computation halts. We prove that at least all Parikh sets of matrix languages can be generated in this setting, using the fact that partially blind register machines generate precisely $PsMAT$.

Theorem 4. *For all $m \geq 1$,*

$$Ps_{gen}O_{toxcat}P_{m,int}(cat_*) \supseteq PsMAT.$$

Proof. Let $M = (d, B, l_0, l_h, P)$ be a partially blind register machine, with the first k registers being its output registers. Let $e = f_{k+1} \cdots f_d$ and let $e(r)$ be e without f_r. Without loss of generality, we assume that the first instruction labeled by l_0 is an ADD-instruction. We then construct the following P system.

$$\begin{aligned}
&\Pi = (O, C, \mu = [\ ]_1, w_1 = l_0 e f_{d+1}, R_1, i_0 = 1), \text{ where}\\
&C = \{c_r \mid k+1 \leq r \leq d\} \cup \{c_p\},\\
&O = C \cup B \cup \{f_r \mid m+1 \leq r \leq d+1\} \cup \{o_r \mid 1 \leq r \leq d\},\\
&R_1 = \{c_p l_i \to c_p l_j o_r e,\ c_p l_i \to c_p l_k o_r e \mid l_i : (\texttt{ADD}(r), l_j, l_k) \in P\}\\
&\quad \cup \{c_p l_i \to c_p l_j e(r) \mid l_i : (\texttt{SUB}(r), l_j) \in P\}\\
&\quad \cup \{c_r o_r \to c_r,\ c_r f_r \to c_r,\ c_r f_{d+1} \to c_r \# \mid k+1 \leq r \leq d\}\\
&\quad \cup \{x \to \# \mid x \in B \cup \{\#\} \cup \{f_r \mid k+1 \leq r \leq d\}\}\\
&\quad \cup \{c_p l_h \to c_p e\} \cup \{c_p f_{d+1} \to c_p\}.
\end{aligned}$$

The catalyst c_r for the working register r, $k+1 \leq r \leq d$, is kept busy by the single copy of the symbol f_r; only in the case of a SUB-instruction on r, in the next step the rule $c_r o_r \to c_r$ should be applied, hence, $e(r)$ is taken instead of e, thus leaving the catalyst c_r free for an object o_r. On the other hand, if such a symbol o_r in that case is not present, due to maximal parallelism the rule $c_r f_{d+1} \to c_r \#$ introducing the trap symbol # has to be applied. The catalyst c_p is used for guiding the computation according to the program given by P. When M reaches the final halting instruction labeled by l_h, then for a last time e is generated, but no instruction label is generated any more, hence, in the last step of a successful computation of Π the rule $c_p f_{d+1} \to c_p$ will be applied together with the rules $c_r f_r \to c_r$ for $k+1 \leq r \leq d$. The computation in Π will then only stop with yielding a result if no rule can be applied any more, i.e., if no trap symbol has been introduced during the computation, and moreover, at the end no symbol o_r for $k+1 \leq r \leq d$ is present, i.e., if all working registers are empty.

The lack of the symbol f_r when a SUB-instruction on register r is carried out guarantees that the computation is trapped if no register symbol o_r, $k+1 \leq r \leq d$, is present by the enforced application of the rule $c_r f_{d+1} \to c_r \#$. If the final rule $c_p f_{d+1} \to c_p$ is applied too early, i.e., as long as some $l \in B$ is present, then the introduction of the trap symbol # by the rule $l \to \#$ is enforced by the maximal parallelism.

We finally observe that only the catalysts are toxic here, so any number of symbols o_r with $1 \leq r \leq d$ can be generated during any successful computation. □

The preceding theorem shows that a partially blind register machine with $m-1$ working registers can be simulated by a P system with internal output in the single membrane with at most m catalysts these being exactly the toxic objects.

7 Conclusion and Future Research

In this paper we have introduced multiple variants of P systems with toxic objects, depending on which objects are toxic. It is important to note that so far in P systems toxic objects and the concept of synchronized halting with toxic objects has not been used in the literature, and thus toxic objects also have not been used to change the computational power of P systems, but rather to decrease the rule complexity of P systems (toxic objects in that case being equivalent to the usual ones, additionally having rules rewriting them into a trap symbol thus forcing the computation to never halt). In this paper, the results are quite different. The most visible one is the non-cooperative case, where toxicity boosts the computational power from $PsREG$ to $PsE0L$. On the other side, requiring certain, or even all, objects to be toxic can also bring limitations to the computational power. The most dramatic limitation is the power of purely catalytic P systems, where complete toxicity lowers the power of internal output from $PsRE$ all the way down to $PsFIN$, and the power of internal input from NRE down to either accepting all numbers, or accepting very restricted finite sets.

Most results we have obtained here describe the computational power of P systems with all objects being toxic or precisely all catalysts being toxic, depending on the kinds of rules used (e.g., non-cooperative, purely catalytic or catalytic), the number of membranes, the output region, in terms of number sets, vector sets or languages, or even computing relations. We repeat the results we obtained in this paper, for easier comparison.

For all $m \geq 1$, we have shown that

$$
\begin{aligned}
Ps_{gen}O_{toxall}P_{m,int}\,(pcat_k) &= PsFIN,\ k \geq 1,\\
N_{acc}O_{toxall}P_m\,(pcat_k) &= \{\{d\} \mid 0 \leq d \leq k-1\}\\
&\quad \cup \{\{0,k'\} \mid 0 \leq k' \leq k\} \cup \{\emptyset, \mathbb{N}\},\ k \geq 1,\\
L_{gen}O_{toxall}P_m\,(pcat_k) &= REG,\ k \geq 1,\\
L_{aut}O_{toxall}P_m\,(pcat_k) &= REG,\ k \geq 2,\\
Rel_{aut}O_{toxall}P_m\,(pcat_k) &= RelREG,\ k \geq 2,\\
L_{aut}O_{toxall}P_m\,(pcat_1) &= IDREG,\\
Rel_{aut}O_{toxall}P_m\,(pcat_1) &= RelIDREG,\\
Ps_{gen}O_{toxall}P^T_{m,int}\,(ncoo) &= PsE0L,\\
Ps_{gen}O_{toxall}P_{m,int}\,(ncoo) &= PsE0L,\\
Ps_{gen}O_{toxall}P_{m,ext}\,(ncoo) &= PsE0L,\\
Ps_{aut}O_{toxall}P_m\,(ncoo) &= PsE0L,\\
Ps_{acc}O_{toxall}P_m(ncoo) &= \{Ps\,(\textstyle\bigcup_{i=1}^{n} T_i^*) \mid \Sigma \text{ alphabet}, T_i \subseteq \Sigma,\ 1 \leq i \leq n,\ n \geq 0\},\\
Ps_{gen}O_{toxcat}P_{m,int}(cat_*) &\supseteq PsMAT.
\end{aligned}
$$

Non-cooperative P systems with all objects being toxic have been shown to be quite versatile, for example, we can easily generate $\{a^n b^n c^n \mid n \geq 1\}$.

Multiple problems remain open. We find particularly interesting the following ones:

- Characterize $Ps_{gen}O_{toxcat}P_{m,int}(cat_*)$ and all other possible variants of restricting the set of toxic objects not yet covered by the results obtained in this paper.
- *There still remains the open problem how to characterize the families of sets of (vectors of) natural numbers generated by [purely] catalytic P systems with only one [two] catalyst[s].*

References

1. A. Alhazov, B. Aman, R. Freund: P systems with anti-matter. In: [9], 66–85.
2. A. Alhazov, C. Ciubotaru, Yu. Rogozhin, S. Ivanov: The family of languages generated by non-cooperative membrane systems. In: Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, A. Salomaa: *Membrane Computing, International Conference*, CMC11, Jena, Lecture Notes in Computer Science **6501**, 2011, 65–79.
3. R. Alur, P. Madhusudan: Visibly pushdown languages. In: L. Babai (Ed.): *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, Chicago, IL, USA, June 13-16, 2004, 202–211, ACM, 2004.
4. E. Csuhaj-Varjú, Gy. Vaszil: P automata or purely communicating accepting P systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 19–23, 2002, Revised Papers*. Lecture Notes in Computer Science **2597**, Springer, 2003, 219–233.
5. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science* **330** (2), 251–266 (2005).
6. R. Freund, A. Leporati, G. Mauri, A. E. Porreca, S. Verlan, C. Zandron: Flattening in (tissue) P systems. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing - 14th International Conference, CMC 2013, Chişinău, Republic of Moldova, August 20-23, 2013, Revised Selected Papers, Lecture Notes in Computer Science **8340**, Springer, 2014, 173–188.
7. R. Freund, I. Pérez-Hurtado, A. Riscos-Núñez , S. Verlan: A formalization of membrane systems with dynamically evolving structures. *International Journal of Computer Mathematics* **90** (4) (2013), 801-815.
8. R. Freund, M. Oswald: A short note on analysing P systems. *Bulletin of the EATCS* **78**, 2002, 231–236.
9. M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron: 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers. Lecture Notes in Computer Science **8961**, Springer, 2014.
10. M. L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
11. A. Okhotin, K. Salomaa: Complexity of input-driven pushdown automata. *SIGACT News* **45** (2), 47–67 (2014).

12. Gh. Păun: Computing with Membranes. *J. Comput. Syst. Sci.* **61**, 108–143 (2000); also see TUCS Report 208, 1998, `www.tucs.fi`.
13. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108–143 (and Turku Center for Computer Science-TUCS Report 208, November 1998, `www.tucs.fi`).
14. Gh. Păun: *Membrane Computing. An Introduction.* Springer, 2002.
15. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
16. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages,* 3 volumes. Springer, 1997.
17. P. Sosík, J. Matyšek: Membrane computing: when communication is enough. In: C. Calude, M. Dinneen, F. Peper (Eds.): *Unconventional Models of Computation 2002*, Lecture Notes in Computer Science **2509**, Springer, 2002, 264–275.
18. The P Systems Website: `http://ppage.psystems.eu`.