P Systems with Anti-Matter

Rudolf Freund[1], Gheorghe Păun[2]

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

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700, Bucharest, Romania
gpaun@us.es, curteadelaarges@gmail.com

Summary. After a short introduction to the area of membrane computing (a branch of natural computing), we introduce the concept of anti-matter in membrane computing. First we consider spiking neural P systems with anti-spikes, and then we show the power of anti-matter in cell-like P systems. As expected, the use of anti-matter objects and especially of matter/anti-matter annihilation rules, turns out to be rather powerful: computational completeness of P systems with anti-matter is obtained immediately, even without using catalysts. Finally, some open problems are formulated, too.

1 Introduction

First we give a brief introduction to membrane computing, a branch of natural computing having widely developed during the more than fifteen years since its initiation, see [19]. In some details we present a specific class of membrane systems (usually called *P systems*) with motivation coming from the way neurons interact, the spiking neural (in short, SN) P systems. In particular, we discuss SN P systems with *anti-spikes*, and then we generalize this idea, considering P systems of any type with *anti-objects*: for an object a, we say that $\bar{a}$ is an anti-object if an *annihilation rule* $a\bar{a} \to \lambda$ is assumed to exist in all membranes, which may either be an explicit rule or else act in an implicit way by removing a pair $a, \bar{a}$ in zero time. These annihilation rules turn out to be rather powerful, as somehow expected if, for example, we look at the λ-rules as the only non-context-free rules in the Geffert normal forms, e.g., see [22].

2 Elements of Membrane Computing

Membrane computing is a branch of natural computing, aiming to abstract computing models from the structure and the functioning of living cells. The models obtained in that way are called *P systems*. Single cells (leading to *cell-like P systems*) as well as communities of cells, like tissues or organs (leading to t*issue-like P systems*), or neural cells (the associated models are called *spiking neural P systems*; these are one of the main topics in the present survey paper) have been considered in the literature. Basically, a P system consists of an arrangement of *membranes* (arranged in a hierarchical manner in the cell-like case and placed in the nodes of an arbitrary graph in the tissue-like case), which determine *compartments* where *multisets of objects* are placed, together with *evolution rules* inspired from biochemistry. Using these rules, the objects evolve, and these evolutions of objects are considered as *computations*. A result is associated with certain computations, hence, a computing device is obtained (working in the generative, the accepting, or the computing mode).

This very general framework lead to a large number of specific classes of P systems. Details can be found, for example, in [20] and [21]; recent information is available at the membrane computing website [27].

As objects in a P system we may use multisets of symbols from a given (finite) alphabet, strings, or more complex structures, such as graphs or d-dimensional arrays. In the case of spiking neural P systems, only multisets over a single object – the *spike*, an electrical impulse used by neurons to communicate with each other – are used. The rules used in a P system are of various types: multiset rewriting rules (similar to chemical reactions), string processing rules, specific rules for handling spikes, or rules directly inspired from biology, such as symport/antiport rules (for moving coupled symbol objects through membranes, corresponding to the functioning of selective protein channels in biology), or rules for handling membranes (dividing or creating membranes, exocytosis, endocytosis, etc.). The rules can be used sequentially or in parallel; the basic strategy in membrane computing is to use the rules in the maximally parallel way (in each step, a maximal multiset of rules is used in each compartment, in the multiset inclusion sense: no rule can be added to a chosen multiset of rules such that the resulting multiset of rules would still be applicable). Most of the investigations carried out in the literature concern synchronized P systems, but also non-synchronized systems were considered. In what concerns the ways to associate a result to a computation, there also are several possibilities: usually, only halting computations are considered to be successful (those computations which reach a configuration of the system where no rule can be applied any more). When dealing with multisets (which is also the case when dealing with SN P systems), the natural result of a computation is a number, but also strings can be associated in various ways.

The computing power of these devices is rather large: Turing computability can be obtained by many classes of P systems. In the cases when an exponential working space can be created during the computation in polynomial time, then, by a time-space trade-off, polynomial, often even linear, solutions to computationally hard problems (typically, **NP**-complete problems, but sometimes even **PSPACE**-problems) can be obtained.

Power and efficiency are computer science issues. Membrane computing proved to be rather attractive as a modeling framework, too. The reader can consult [3] and [4] in this respect. The most numerous and advanced applications are those in biology and biomedicine, but there are also well-investigated applications in approximate optimization, computer graphics, robot control, etc.

In this paper, we formally introduce only the basic model of membrane computing, the cell-like P systems with symbol objects (which will also be considered in Section 5 below). Hierarchical membrane structures (which can be described by a tree) are represented by strings of labeled matching parentheses, and the multisets over an alphabet V are represented by strings over V; a string and all its permutations represent the same multiset. For an alphabet V, by V^* we denote the set of all strings over V, including the empty string. The length of $x \in V^*$ is denoted by $|x|$; the empty string, of length zero, is denoted by λ. Basic knowledge in formal language theory as well as some familiarity with basic elements of membrane computing is assumed in what follows.

A *cell-like P system*, of degree m, with catalysts, is a construct

$$\Pi = (O, C, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_{in}, i_{out})$$

where O is the alphabet of objects, $C \subset O$ is the set of catalysts, μ is the *membrane structure* (with m membranes), $w_1, \dots, w_m$ are strings over O representing multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \dots, R_m$ are finite sets of evolution rules associated with the regions of μ, and i_{in} and i_{out} are the labels of the input and output regions, respectively; if the input or output is taken from the environment, this is indicated by taking the label 0 for i_{in} or i_{out}, respectively.

The *evolution rules* are multiset rewriting rules of the form $u \to v$, where u is a non-empty multiset over O and $v = (b_1, tar_1) \dots (b_k, tar_k)$ with $b_i \in O$ and $tar_i \in \{here, out, in\}$, i.e., the objects b_i in v have associated a *target indication* tar_i. Using such a rule means "consuming" the objects of u and "producing" the objects from $b_1, \dots, b_k$ of v, where the target indication *here* means that the objects remain in the same region where the rule is applied, *out* means that they are 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), and *in* means that they are sent to one of the immediately inner membranes, chosen in a non-deterministic way; in general, the target indication *here* is omitted.

A rule $u \to v$ with $|u| = 1$ is said to be *non-cooperative.* A rule of the form $ca \to cv$, where $c \in C$, $a \in V \setminus C$, and the objects in v are from $V \setminus C$, too, is called *catalytic*; C is the set of catalysts, objects which are not changed by evolution rules. Arbitrary rules are called *cooperative.*

If the system is used in the generative mode, then i_{in} is omitted, and if the system is used in the accepting mode, then i_{out} is omitted. The number m of membranes in μ is called the *degree* of Π.

In the generative case, the set of numbers computed by Π (in the maximally parallel non-deterministic mode) is denoted by $N(\Pi)$. The family of all sets $N(\Pi)$

computed by systems Π of degree at most $m \geq 1$ and using rules of form α is denoted by $NOP_m(\alpha)$; if there is no bound on the degree of the systems, then the subscript m is replaced by $*$. According to the previous classification, $\alpha \in \{ncoo, cat, coo\}$, with the obvious meaning.

It is known that $NOP_1(coo) = NOP_1(cat_2) = NRE$, where cat_2 indicates the fact that only two catalysts are used with catalytic rules together with non-cooperative rules, and NRE is the family of recursively enumerable (Turing computable) sets of natural numbers. In turn, $NOP_*(ncoo) = NREG$, where $NREG$ is the family of length sets of regular languages (i.e., the family of semilinear sets of natural numbers).

3 Spiking Neural P Systems

Spiking neural P systems, see [8], have a completely different architecture and functioning, as they are not based on the standard eukaryotic cell, but on brain biology. Here we only consider the neurons cooperating by means of spikes, electrical impulses of identical forms, moving along axons. Spiking neurons are also investigated in the current neural computing area (e.g., see [11]). We do not define SN P systems in a formal way, instead we only describe such a system and then also introduce anti-spikes, and we will give a simple example.

In short, an SN P system consists of a set of *neurons* (represented by membranes) placed in the nodes of a directed graph (the arcs are called *synapses*) and containing *spikes*, denoted by copies of the symbol a. Thus, the architecture is that of a tissue-like P system, with only one kind of objects present in the cells. The objects evolve by means of *spiking rules*, which are of the form $(E/a^c \to a; d)$, where E is a regular expression over a and c, d are natural numbers, $c \geq 1$, $d \geq 0$. The meaning is that a neuron containing k spikes such that $a^k \in L(E)$, $k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*, of the form $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are removed, provided that the neuron contains exactly s spikes. The system works in a synchronized manner, i.e., in each time unit, every neuron which can use a rule should do that, but the work of the system is sequential in each neuron: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output* one, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system; it might be infinite if the computation does not stop.

The result of a computation is encoded in the distance between the first two spikes sent to the environment by the (output neuron of the) system. Other ways to associate a result with a computation were considered, for instance, the total number of spikes emitted by the output neuron during a halting computation, or else the number of spikes contained in the output neuron at the end of a halting computation; the spike train itself can be taken as the result of the computation, and in this way the system generates a binary sequence (a finite string, if the computation halts).

There are several classes of SN P systems, using various combinations of ingredients – rules of restricted forms, for example, without a delay, without forgetting rules, or extended rules, e.g., producing more than one spike, as well as asynchronous SN P systems (no clock is considered, any neuron may use a rule or not), with exhaustive use of rules (when enabled, a rule is used as many times as made possible by the spikes present in a neuron), with certain further conditions imposed on the halting configuration, with the same sets of rules in each neuron (the system then is called *homogeneous*), containing further biological ingredients, such as *astrocytes*, with inhibitory synapses, etc. For most SN P systems with unbounded neurons (arbitrarily many spikes can be found in each of them), characterizations of Turing computable sets of natural numbers are obtained. When the neurons are bounded, usually characterizations of the family $NREG$ are obtained. SN P systems can also be used in the accepting and in the computing modes.

4 SN P Systems with Anti-Spikes

A natural feature added to an SN P system is that of *anti-spikes*, proposed in [17] and then investigated in a series of papers. For the reader's convenience, the bibliography below contains many titles of papers dealing with this subject, yet not all of them are explicitly referred to in the present suvey paper.

The main point of the new notion is to interpret the "anti-spikes" as "anti-matter", hence to assume that when a piece of matter meets the corresponding piece of anti-matter, they will annihilate each other. This corresponds to the existence of rules of the form $a\bar{a} \to \lambda$, which are used immediately when a and $\bar{a}$ are present in the same neuron.

Thus, in an SN P system with anti-spikes, the spiking rules and the forgetting rules are of the forms $E/b^c \to b'^c$ and $b^c \to \lambda$ where E is a regular expression over a or over $\bar{a}$, while $b, b' \in \{a, \bar{a}\}$ and $c \geq 1$. If $L(E) = b^c$, then we write the first rule as $b^c \to b'$. As usual, a delay can be added to the spiking rules, too.

Note that we have four categories of rules, identified by $(b, b') \in \{(a, a), (a, \bar{a}), (\bar{a}, a), (\bar{a}, \bar{a})\}$. Of course, it is of interest to restrict the type of rules, and this is the case in most papers found in the literature.

The rules are used as usual in SN P systems, with the additional fact that a and $\bar{a}$ "cannot stay together", they instantaneously annihilate each other: if in a neuron there are either objects a or objects $\bar{a}$, and further objects of either type (maybe both) arrive from other neurons, such that we end with a^r and $\bar{a}^s$ inside, then immediately the rule of the form $a\bar{a} \to \lambda$ is applied in the maximal manner, so that either the multiset of spikes a^{r-s} – if $r \geq s$ – or of anti-spikes $\bar{a}^{s-r}$ – if $s \geq r$ – remains.

In the definition from [17], the mutual annihilation of spikes and anti-spikes takes no time, so that the neurons always contain either only spikes or only anti-spikes. That is why, for instance, the regular expressions of the spiking rules are defined either over a or over $\bar{a}$, but not over both symbols. Moreover, annihilation has priority over spiking and forgetting rules. Later, also the case when the annihilation takes one time unit was considered, with explicitely using the rule $a\bar{a} \to \lambda$, eventually even without priority over other rules.

The computations and the results of computations are defined in the same way as for usual SN P systems. In most investigations, the restriction was considered that the output neuron produces only spikes, not also anti-spikes. The anti-spikes are sometimes used to encode, in a natural way, negative numbers.

By $N_2(\Pi)$ we denote the family of numbers generated by an SN P system (with anti-spikes) as the distance between the first two spikes sent to the environment by the output neuron, and by $N_2S_aNP_m$ the families of all sets $N_2(\Pi)$, computed by SN P systems with anti-spikes and at most $m \geq 1$ neurons. When the number of neurons is not bounded, we replace the subscript m by $*$.

We illustrate the previous definition by an example recalled from [17]; it is, in fact, part of the proof showing computational completeness of SN P systems with anti-spikes (i.e., $N_2S_aNP_* = NRE$), namely, the module which simulates a SUB-instruction of a register machine. We present the module in the graphical form, a usual way of presentation in membrane computing: neurons are given as ovals containing spiking and forgetting rules, and in addition indicating the initial spikes and anti-spikes; the synapses are represented by arrows linking the neurons.

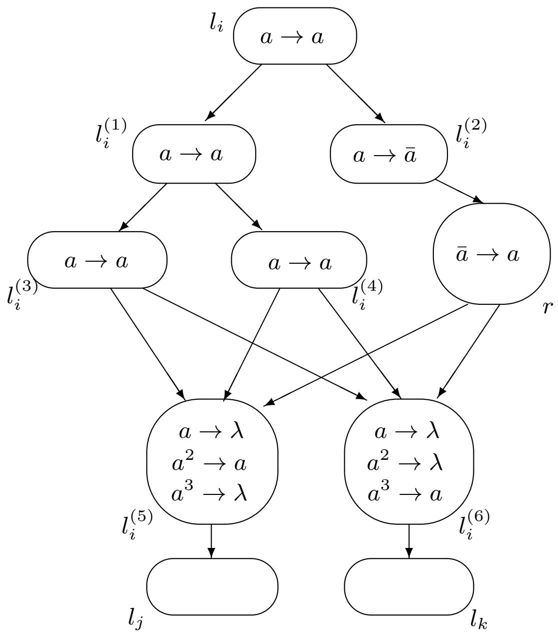

Fig. 1. Module SUB, simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

Figure 1 shows the module associated with an instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$. The module is activated when neuron σ_{l_i} receives a spike. Initially, no neuron contains any spike, except for the neuron σ_{l_0} associated with l_0, the initial label of the register machine; each label has such an associated neuron, and also each register r has associated a neuron σ_r. Neuron σ_{l_i} sends a spike to neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$. In the next step, neuron $\sigma_{l_i^{(2)}}$ sends an anti-spike to neuron σ_r, which corresponds to register r; at the same time, $\sigma_{l_i^{(1)}}$ sends a spike to the neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$. If register r is non-empty, that is, neuron σ_r contains at least one a, then $\bar{a}$ removes one occurrence of a, which corresponds to subtracting one from register r, and no rule is applied in σ_r. This means that $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive only two spikes, from $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$, hence, σ_{l_j} is activated, whereas σ_{l_k} is not activated. If register r is empty, then the rule $\bar{a} \rightarrow a$ is used in σ_r, hence, $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive three spikes, and this leads to the activation of σ_{l_k}, which is the correct continuation in this case.

The reader is referred to [17] for further details concerning the functioning of this module, and in general, for the proof of the universality of SN P systems with anti-spikes.

We cannot present all the developments concerning SN P systems with anti-spikes; most of the titles of the related articles listed at the end of the paper are self-explanatory. We only mention an important research direction in membrane computing in general and in the SN P systems area in particular, reminding the "old times" of investigations in formal language theory (see a survey in [7]) concerning the *descriptional complexity* of grammars and languages: considering size parameters for P systems. Because most of the classes of P systems are universal, for those classes the basic question is to find the smallest number of membranes in order to get the equivalence with Turing machines. For subuniversal classes, an important question of interest is whether or not the number of membranes induces an infinite hierarchy.

These questions are of interest for SN P systems, too, with or without anti-spikes. Further questions appear, resembling those mentioned in [7]: How many rules per neuron are needed? How many different types of neurons are needed? Can rules of a specific type be avoided?

Another question of interest is to find universal systems for a given class of devices with a small descriptional complexity; like in the case of universal Turing machines, we search for fixed P systems which can simulate any P system from a given class, as soon as the code of a particular system is introduced as an input to the universal one. For SN P systems, the "race" was started in [18], with several subsequent papers succeeding to decrease the complexity of the universal systems constructed there.

According to our knowledge, for SN P systems with anti-spikes the best results currently available are those from [24]: a universal system is constructed, for the case of computing functions, having 75 neurons and 125 rules, with 6 types of neurons and 8 types of rules. A related result is reported in [12], where a similar system is described, containing 91 neurons, each of them containing only one rule, of one of the simple forms $a \to a$ and $a \to \bar{a}$. This once again proves the power of annihilation rules.

5 P Systems with Anti-Matter

The idea of considering "anti-matter" objects and their corresponding matter/anti-matter annihilation rules can be extended to all types of P systems. We briefly discuss it here for cell-like P systems.

Formally, a cell-like P system (of degree m, with catalysts) with anti-matter is a construct

$$\Pi = (O, A_O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{in}, i_{out})$$

where all the components are as in a usual P system and A_O is a set of symbols $\bar{a}$, for $a \in O \setminus C$ (obviously, we do not allow the catalysts to have anti-objects). In each compartment of μ we assume the matter/anti-matter annihilation rules $a\bar{a} \to \lambda$ to be present, for all $\bar{a} \in A_O$. As in SN P systems we might assume that these rules are used "automatically", in zero time, as soon as they can be applied. Yet in the following we assume the annihilation rules to be used as other rules, yet eventually with *weak priority* (e.g., see [2]) over all other rules, i.e., other rules then also may be applied if objects cannot be bound by some annihilation rule any more. In both cases, the rules in the sets R_i, $1 \leq i \leq m$, of the form $u \to v$ have to obey to the condition that neither u nor v may contain both the symbol a and its anti-matter object $\bar{a}$ for any $\bar{a} \in A_O$.

The functioning of such a system is as usual in membrane computing, keeping in mind that the annihilation rules have to be added to all sets of rules R_i, $1 \leq i \leq m$. By $NO_aP_m(ncoo, pri)$ we denote the family of sets of numbers generated by P systems with at most m membranes, using anti-objects, with non-cooperative rules. The parameter pri indicates the use of annihilation rules with priority over the other rules; it is omitted if we do not use this implicit priority. If in addition to non-cooperative rules we also allow catalytic rules and at most k catalysts, $ncoo$ is replaced by cat_k in these notations.

Although the annihilation rules are expected to add a lot of computational power, it is still surprising that together with giving the annihilation rules priority over all other rules, non-cooperative rules are already sufficient to obtain computational completeness, whereas without this priority condition, in addition we need catalytic rules with one catalyst; in both cases rather simple proofs can be obtained, whereas without these matter/anti-matter annihilation rules, non-cooperative rules together with catalytic rules with two catalysts are needed, see the rather complex proof given in [5].

Theorem 1. $NO_aP_1(ncoo, pri) = NRE$.

Proof. Let $M = (3, H, l_0, l_h, I)$ (number of registers, labels of instructions, initial label, halt label, set if instructions) be a register machine with three registers; register 1 is the output register containing the result at the end of a successful computation, it is never decremented; registers 2 and 3 are empty at the begin and at the end of a successful computation. We now construct the (generating, hence, we omit i_{in}) P system with anti-matter

$$\Pi = (O, A_O, [\ \ ]_1, l_0, R_1, 1)$$

with only one membrane and the following components:

$$\begin{aligned} O &= \{l, l' \mid l \in H\} \cup \{a_r \mid r \in \{1,2,3\}\} \cup \{\#\}, \\ A_O &= \{\bar{a}_2, \bar{a}_3, \bar{\#}\}; \end{aligned}$$

the non-cooperative rules in R_1 are described below.

The contents of register r is represented by the number of copies of the object a_r, $r \in \{1,2,3\}$, in the system. The P system starts with the object l_0 representing the initial label of M.

For each instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ in I, $r \in \{1,2,3\}$, we take the rules

$$\begin{aligned} &l_i \to l_j a_r \text{ and} \\ &l_i \to l_k a_r, \end{aligned}$$

which obviously simulate the given ADD-instruction.

For each instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in I, $r \in \{2,3\}$, we consider the three rules

$$\begin{aligned} &l_i \to l_j \bar{a}_r, \\ &l_i \to l_i' \bar{a}_r, \\ &l_i' \to \# l_k. \end{aligned}$$

As rules common for all SUB-instructions, we also add the rules $\bar{a}_r \to \bar{\#}$, $r \in \{2,3\}$, the matter/antimatter annihilation rules $a_r \bar{a}_r \to \lambda$ and $\#\bar{\#} \to \lambda$ as well as the trap rules $\# \to \#\#$ and $\bar{\#} \to \#\#$.

When simulating a SUB-instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$, we have to make a non-deterministic choice between the decrement case and the zero-test. The decrement case of the SUB-instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated by the rule $l_i \to l_j \bar{a}_r$ and the subsequent application of the annihilation rule $a_r \bar{a}_r \to \lambda$. If this rule is not applicable, i.e., if register r is empty, the rule $\bar{a}_r \to \bar{\#}$ will be applied instead, which in absence of its counterpart $\#$ immediately evolves to $\#\#$ and thus leads to an infinite computation.

The zero-test is initiated with the rule $l_i \to l_i' \bar{a}_r$. If register r is empty, then $\bar{a}_r$ cannot be annihilated and therefore evolves to $\bar{\#}$, which then annihilates the symbol $\#$ generated by the rule $l_i' \to \# l_k$; if register r is not empty, $\bar{a}_r$ is annihilated by some copy of a_r, hence, the trap symbol $\#$ generated by the rule $l_i' \to \# l_k$ does not find its anti-matter $\bar{\#}$ and therefore evolves to $\#\#$, thus leading to an infinite computation. Here we find the crucial situation where we need the constraint that annihilation rules have priority over all other rules, i.e., $\bar{a}_r \rightarrow \bar{\#}$ cannot be applied if the annihilation rule $a_r\bar{a}_r \rightarrow \lambda$ can be applied.

The rule $l_h \rightarrow \lambda$ is applied at the end of a successful simulation of the instructions of the register machine M, and the computation halts if no trap symbol $\#$ is present; the number of symbols a_1 in the skin membrane then represents the result of this halting computation. In sum, we obtain $N(M) = N(\Pi)$. □

Returning to descriptional complexity issues, it is worth noting that the P system constructed in the preceding proof has only one membrane and only three matter/anti-matter annihilation rules.

If we look for small universal systems, we may start with the universal register machine U_{32} from [9], with 8 registers which are decremented during the computations, and apply the construction given in the preceding proof, thus needing $8+1$ matter/anti-matter annihilation rules. An optimized P system with matter/anti-matter annihilation rules having priority over all other rules can be found in [1].

Without this priority of the annihilation rules, the construction is not working, hence, a characterization of the class $NO_aP_1(ncoo)$ remains as an open problem. Yet in addition using catalytic rules with one catalyst again allows us to obtain computational completeness:

Theorem 2. $NO_aP_1(cat_1) = NRE$.

Proof. We again consider a register machine $M = (3, H, l_0, l_h, I)$ as in the previous proof, and construct the (generating) catalytic P system

$$\Pi = (O, A_O, [\ \ ]_1, \{c\}, cl_0, R_1, 0)$$

with only one membrane (containing the single catalyst c) and the following components:

$$\begin{aligned} O &= \{l, l', l'' \mid l \in H\} \cup \{a_r \mid r \in \{1,2,3\}\} \cup \{\#, d\}, \\ A_O &= \{\bar{a}_2, \bar{a}_3, \bar{\#}\}; \end{aligned}$$

the non-cooperative rules in R_1 are described below. The output symbols a_1 now are sent to the environment, in order not to have to count the catalyst in the skin membrane; for that purpose, we simply use the rule $a_1 \rightarrow (a_1, out)$.

For each instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ in I, $r \in \{1,2,3\}$, we again take the rules

$$\begin{aligned} &l_i \rightarrow l_j a_r \text{ and} \\ &l_i \rightarrow l_k a_r. \end{aligned}$$

For each instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in I, $r \in \{2,3\}$, we now consider the following four rules:

$$\begin{aligned} &l_i \rightarrow l_j \bar{a}_r, \\ &l_i \rightarrow l_i'' d \bar{a}_r, \\ &l_i'' \rightarrow l_i', \\ &l_i' \rightarrow \# l_k. \end{aligned}$$

As rules common for all SUB-instructions, we again add the matter/antimatter annihilation rules $a_r\bar{a}_r \to \lambda$ and $\#\bar{\#} \to \lambda$ as well as the trap rules $\# \to \#\#$ and $\bar{\#} \to \#\#$, but in addition, also $d \to \#\#$ as well as the catalytic rules $cd \to c$ and $c\bar{a}_r \to c\bar{\#}$, $r \in \{2,3\}$.

The decrement case of the SUB-instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is simulated as in the previous proof, by using the rule $l_i \to l_j\bar{a}_r$ and then applying the annihilation rule $a_r\bar{a}_r \to \lambda$. If this rule is not applicable, i.e., if register r is empty, the rule $\bar{a}_r \to \bar{\#}$ will be applied instead, which in absence of its counterpart $\#$ immediately evolves to $\#\#$ and thus leads to an infinite computation.

The zero-test now is initiated with the rule $l_i \to l_i''d\bar{a}_r$ thus introducing the (dummy) symbol d which keeps the catalyst busy for one step, where the catalytic rule $cd \to c$ has to be applied in order to avoid the application of the trap rule $d \to \#\#$. If register r is empty, then $\bar{a}_r$ cannot be annihilated and therefore evolves to $\bar{\#}$ in the third step by the application of the catalytic rule $c\bar{a}_r \to c\bar{\#}$, which symbol $\bar{\#}$ then annihilates the symbol $\#$ generated by the rule $l_i' \to \#l_k$ in the same step; if register r is not empty, $\bar{a}_r$ is annihilated by some copy of a_r already in the first step, hence, the trap symbol $\#$ generated by the rule $l_i' \to \#l_k$ does not find its anti-matter $\bar{\#}$ and therefore evolves to $\#\#$, thus leading to an infinite computation. Altough the annihilation rule $a_r\bar{a}_r \to \lambda$ now does not have priority over the catalytic rule $c\bar{a}_r \to c\bar{\#}$, maximal parallelism enforces $a_r\bar{a}_r \to \lambda$ to be applied, if possible, already in the first step instead of $c\bar{a}_r \to c\bar{\#}$, as in a successful derivation the catalyst c first has to eliminate the dummy symbol d.

The rule $l_h \to \lambda$ is applied at the end of a successful simulation of the instructions of the register machine M, and the computation halts if no trap symbol $\#$ is present; the number of symbols a_1 sent out to the environment during the computation represents the result of this halting computation. In sum, we obtain $N(M) = N(\Pi)$. $\square$

6 Concluding Remarks

In this survey paper we have briefly recalled some basic ideas of membrane computing, and especially have given some information about spiking neural P systems, including spiking neural P systems with anti-spikes. We have also extended this idea of anti-objects ("anti-matter") to cell-like P systems with symbol objects, which can be proved to be computationally complete when the annihilation rules are applied with having priority over the remaining non-cooperative rules; without this priority, in addition catalytic rules with a single catalyst are needed to obtain computational completeness.

Several problems are still open in this area of P systems with anti-matter. Some of them have been formulated in this paper; the interested reader can find many more in the literature, for instance, in [6].

References

1. A. Alhazov, B. Aman, R. Freund, Gh. Păun: Matter and anti-matter in membrane systems. *Brainstorming Week in Membrane Computing*, Sevilla, February 2014.
2. A. Alhazov, D. Sburlan: Static Sorting P Systems. In: G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.): Applications of Membrane Computing. *Natural Computing Series*, Springer, 2005, 215–252.
3. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing.* Springer, Berlin, 2006.
4. P. Frisco, M. Gheorghe, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing in Systems and Synthetic Biology.* Springer, Berlin, 2014.
5. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330 (2005), 251–266.
6. M. Gheorghe, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Frontiers of membrane computing: Open problems and research topics, *Intern. J. Found. Computer Sci.*, 24, 5 (2013), 547–623.
7. J. Gruska: Descriptional complexity of context-free languages. *Proc. Symp. Math. Found. Computer Sci.*, High Tatras, 1973, 71–83.
8. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
9. I. Korec: Small universal Turing machines. *Theoretical Comp. Sci.*, 168 (1996), 267–301.
10. K. Krithivasan, V.P. Metta, D. Garg: On string languages generated by spiking neural P systems with anti-spikes. *Intern. J. Found. Computer Sci.*, 22, 1 (2011).
11. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks.* MIT Press, Cambridge, 1999.
12. V.P. Metta, A. Kelemenova: More on universality of spiking neural P systems with anti-spikes. Manuscript, 2013.
13. V.P. Metta, K. Krithivasan, D. Garg: Some characteristics of spiking neural P systems with anti-spikes. *Proc. 11th Intern. Conf. on Membrane Computing*, Jena, Germany, August 2010, 291–303.
14. V.P. Metta, K. Krithivasan, D. Garg: Modelling and analysis of spiking neural P systems with anti-spikes using Pnet lab. *Nano Comm. Networks*, 1, 2 (2011), 141–149.
15. V.P. Metta, K. Krithivasan, D. Garg: Computability of spiking neural P systems with anti-spikes. *New Math. and Natural Comput.*, 8, 3 (2012), 283–295.
16. V.P. Metta, K. Krithivasan, D. Garg: Spiking neural P systems with anti-spikes as transducers. *Romanian J. Info. Sci. and Tehnology*, 14, 1 (2011), 20–30.
17. L. Pan, Gh. Păun: Spiking neural P systems with anti-spikes. *Int. J. Comoputers, Comm. and Control*, 4, 3 (2009), 273–282.
18. A. Păun, Gh. Păun: Small universal spiking neural P systems. *BioSystems*, 90 (2007), 48–60.
19. 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`).
20. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
21. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing.* Oxford University Press, 2010.

22. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 vols., Springer, Berlin, 1997.
23. T. Song, L. Pan, J. Wang, I. Venkat, K.G. Subramanian, R. Abdullah: Normal forms for spiking neural P systems with anti-spikes. *IEEE Trans. Nanobioscience*, 22, 4 (2012), 352–359.
24. T. Song, Y. Jiang, X. Shi, X. Zeng: Small universal spiking neural P systems with anti-spikes. *J. Comput. and Th. Nanoscience*, 10, 4 (2013), 999–1006.
25. T. Song, X. Wang, Z. Zhang, Z. Chen: Homogeneous spiking neural P systems with anti-spikes. *Neural Comput. and Applic.*, DOI 10.1007/s00521-0123-1397-8 (June 2013).
26. G. Tan, T. Song, Z. Chen, X. Zeng: Spiking neural P systems with anti-spikes and without annihilating priority working in a ”flip-flop” way. *Intern. J. Computing Sci. and Math*, 4, 2 (2013), 152–162.
27. The P Systems Website: `www.ppage.psystems.eu`.