P Systems: from Anti-Matter to Anti-Rules

Artiom Alhazov[1], Rudolf Freund[2], Sergiu Ivanov[3], Mario J. Pérez-Jiménez[4]

[1] Vladimir Andrunachievici Institute of
Mathematics and Computer Science
Academiei 5, Chişinău, MD-2028, Moldova
E-mail: artiom@math.md

[2] TU Wien, Institut für Logic and Computation
Favoritenstraße 9–11, 1040 Wien, Austria
E-mail: rudi@emcc.at

[3] IBISC, Université Évry, Université Paris-Saclay
23, boulevard de France, 91034 Évry, France
E-mail: sergiu.ivanov@univ-evry.fr

[4] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mail: marper@us.es

Summary. The concept of a matter object being annihilated when meeting its corresponding anti-matter object is taken over for rule labels as objects and anti-rule labels as the corresponding annihilation counterpart in P systems. In the presence of a corresponding anti-rule object, annihilation of a rule object happens before the rule that the rule object represents, can be applied. Applying a rule consumes the corresponding rule object, but may also produce new rule objects as well as anti-rule objects, too. Computational completeness in this setting then can be obtained in a one-membrane P system with non-cooperative rules and rule / anti-rule annihilation rules when using one of the standard maximally parallel derivation modes as well as any of the maximally parallel set derivation modes (i.e., non-extendable (multi)sets of rules, (multi)sets with maximal number of rules, (multi)sets of rules affecting the maximal number of objects). When using the sequential derivation mode, at least the computational power of partially blind register machines is obtained.

1 Introduction

The basic model of *P systems* as introduced in [24] can be considered as a distributed multiset rewriting system, where all objects – if possible – evolve in parallel in the membrane regions and may be communicated through the membranes. Overviews on the field of P systems can be found in the monograph [25] and the handbook of membrane systems [26]; for actual news and results we refer to the

P systems webpage [28] as well as to the Bulletin of the International Membrane Computing Society.

Computational completeness (computing any partial recursive relation on non-negative integers) can be obtained with using cooperative rules or with catalytic rules (possibly) together with non-cooperative rules. We recall that non-cooperative rules have the form $a \to w$, where a is a symbol and w is a multiset, catalytic rules have the form $ca \to cw$, where the symbol c is called the catalyst, and cooperative rules have no restrictions on the form of the left-hand side. Without additional control mechanisms, at least two catalysts are needed, see [16]. Using specific control mechanisms, as for example, rule labels or target agreement, only one catalyst is needed, for example, see [14, 19, 20]. In [2, 1], another concept to avoid cooperative rules is investigated: for any object *a (matter)*, its anti-object *(anti-matter)* a^- is considered together with the corresponding *annihilation rule* $aa^- \to \lambda$, which is assumed to exist in all membranes; this annihilation rule is assumed to be a special non-cooperative rule having priority over all other rules in the sense of weak priority (e.g., see [9], i.e., other rules then also may be applied if objects cannot be bound by some annihilation rule any more). For spiking neural P systems, the idea of anti-matter has been introduced in [23] with *anti-spikes* as anti-matter objects. In [12] the power of anti-matter for solving NP-complete problems is exhibited.

Although, as expected (for example, compare with the Geffert normal forms, see [27]), the annihilation rules are rather powerful, it is still surprising that using matter/anti-matter annihilation rules as the only non-cooperative rules, with the annihilation rules having weak priority, computational completeness can already be obtained without using any catalyst, see [2, 1], whereas usually at least one catalyst is needed even when using other control mechanisms, for example, see [2].

In this paper we now consider a rule label as an object itself which cannot evolve as any other object in the system, but only has the task to make the rule applicable; in some sense this can be seen as a variant of rule activation as introduced in P systems with activation and blocking of rules, see [4]; for the concept of activation and blocking of rules also see [3, 5]. Introducing the anti-rule object then can be seen as a variant of blocking the corresponding rule. The main difference between these two concepts – activation and blocking of rules in contrast to rule objects and anti-rule objects – is that with activation and blocking of rules, rules can be activated and blocked for specific time steps in the future, whereas the activation of a rule by its rule object is immediate, and also blocking is immediate, but active until the anti-rule object annihilates with the rule object, which may be an unbounded number of steps in the future. When a rule is applied by consuming its corresponding rule object is also not fixed and may heavily depend not only on the applicability of the rule, but also on the derivation mode. For example, in the sequential mode, several rule objects may compete for the rule each of them represents to be executed; the sequence of applications may be crucial, as in between an anti-rule object may appear and annihilate the rule object. Finally, let us mention that the concepts of activation of rules and of rule objects already

have appeared in some different way in [13] embedded in an even more complex setting.

We also have to emphasize that each copy of a rule object allows for exactly one application of the corresponding rule it represents, i.e., we deal with multisets of rules only applicable if the corresponding multiset of rule objects is present, which restricts the set of applicable sets of multisets of rules in a given derivation mode, especially in the maximally parallel derivation modes. This also means that we have to deal with a subtle technical detail: We either may start with the set of all applicable sets of multisets of rules, no matter which and how many rule objects are present, then take out all the multisets of rules which conform with the given derivation mode, and then only take those for which sufficient resources of rule objects are available. On the other hand, we may also first take only those multisets of rules for which there are sufficient resources of rule objects available, take these as the set of applicable rule multisets and only afterwards apply the condition for the derivation mode. Examples to explain these differences will be given in Section 3.

After explaining some notions and definitions used in this paper in the next section, in Section 3 we will define our new model of P systems with rule and anti-rule objects as well the kind of rules, the derivation modes, and the halting conditions used afterwards; moreover, some examples are given to illustrate the new concept. In Section 4, we then establish our main results. We show that computational completeness can even be obtained with one-membrane P systems using non-cooperative rules and rule / anti-rule annihilation rules as well as one of the standard maximally parallel derivation modes or maximally parallel set derivation modes (i.e., non-extendable (multi)sets of rules, (multi)sets with maximal number of rules, (multi)sets of rules affecting the maximal number of objects). When using the sequential derivation mode, at least the computational power of partially blind register machines is obtained. A summary of the results we obtained as well as an outlook to future research topics conclude the paper.

2 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 $V^* \backslash \{\lambda\}$ is denoted by V^+. For an arbitrary alphabet $V = \{a_1, \ldots, a_n\}$, 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 $|x| = \sum_{a_i \in V} |x|_{a_i}$. The Parikh vector associated with x with respect to $a_1, \ldots, a_n$ is $(|x|_{a_1}, \ldots, |x|_{a_n})$. The Parikh image of an arbitrary language L over $\{a_1, \ldots, a_n\}$ is the set of all Parikh vectors of strings in L, and is denoted by $Ps(L)$. For a family of languages FL, the family of Parikh images of languages

in FL is denoted by $PsFL$, while for families of languages over a one-letter (d-letter) alphabet, the corresponding sets of non-negative integers (d-vectors with non-negative components) are denoted by NFL (N^dFL).

A (finite) multiset over a (finite) alphabet $V = \{a_1, \ldots, a_n\}$, is a mapping $f : V \to \mathbb{N}$ and can be represented by $\langle a_1^{f(a_1)}, \ldots, a_n^{f(a_n)} \rangle$ or by any string x for which $(|x|_{a_1}, \ldots, |x|_{a_n}) = (f(a_1), \ldots, f(a_n))$. In the following we will not distinguish between a vector $(m_1, \ldots, m_n)$, a multiset $\langle a_1^{m_1}, \ldots, a_n^{m_n} \rangle$ or a string x having $(|x|_{a_1}, \ldots, |x|_{a_n}) = (m_1, \ldots, m_n)$. Fixing the sequence of symbols $a_1, \ldots, a_n$ in an alphabet V in advance, the representation of the multiset $\langle a_1^{m_1}, \ldots, a_n^{m_n} \rangle$ by the string $a_1^{m_1} \ldots a_n^{m_n}$ is unique. The set of all finite multisets over an alphabet V is denoted by V°.

The family of regular and recursively enumerable string languages is denoted by REG and RE, respectively. For more details of formal language theory the reader is referred to the monographs and handbooks in this area as [11] and [27].

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:

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \backslash \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Increases the value of register j by one, followed by a non-deterministic jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : (SUB(j), l_2, l_3)$, with $l_1 \in B \backslash \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  If the value of register j is zero then jump to instruction l_3; otherwise, the value of register j 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 of each register and by the value of 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.

For useful results on the computational power of register machines, we refer to [22].

In *partially blind* register machines, the SUB instruction has the form $p : (SUB(r), q)$: if the register r is not empty, it is decremented and the register machine moves to state q, otherwise the machine crashes – the computation stops in a non-halting configuration, yielding no result. Moreover, we require that valid computations of a partially blind register machine have 0 in all non-output registers in halting configurations.

3 P Systems with Rule and Anti-Rule Objects

Formally, a *P system with anti-rules* (a *PARS* for short) is a construct

$$\Pi = \left(V, T, H \cup H^-, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, g_H, f, \Longrightarrow_{\Pi,\delta}\right) \text{ where}$$

- V is the alphabet of *objects*;
- $T \subseteq V$ is the alphabet of *terminal objects*;
- $H \cup H^-$ is the alphabet of *rule objects* (H) and *anti-rule objects* (H^-), respectively; we also define $V_H := V \cup H \cup H^-$;
- μ is the hierarchical membrane structure (a rooted tree of membranes) with the membranes uniquely labeled by the numbers from 1 to m;
- $w_i \in V^*$, $1 \leq i \leq m$, is the *initial multiset* in membrane i;
- R_i, $1 \leq i \leq m$, is a finite set of *rules of type* X assigned to membrane i; we also define $R = \bigcup_{1 \leq i \leq m} \{(i, r) \mid r \in R_i\}$;
- g_H is a function assigning a rule from R to every rule object in H;
- f is the label of the membrane from which the result of a computation has to be taken from (in the generative case) or into which the initial multiset has to be given in addition to w_f (in the accepting case),
- $\Longrightarrow_{\Pi,\delta}$ is the derivation relation under the derivation mode δ.

The symbol X in "rules of type X" may stand for "cooperative", "non-cooperative","purely catalytic", "catalytic", etc., see Subsection 3.1.

A *configuration* is a list of the contents of each membrane region; a sequence of configurations $C_1, \ldots, C_k$ is called a *computation* in the derivation mode δ if $C_i \Longrightarrow_{\Pi,\delta} C_{i+1}$ for $1 \leq i < k$. The derivation relation $\Longrightarrow_{\Pi,\delta}$ is defined by the set of rules in Π and the given derivation mode which determines the multiset of rules to be applied to the multisets contained in each membrane also taking into account the available rule objects, as we will explain in more detail in Subsection 3.4.

3.1 Standard Rule Variants

Non-cooperative rules have the form $a \to w$, where a is a symbol and w is a multiset, catalytic rules have the form $ca \to cw$, where the symbol c is called the *catalyst*, and cooperative rules have no restrictions on the form of the left-hand side. These types of rules will be denoted by *ncoo* (*non-cooperative*), *pcat* (*purely catalytic*), and *coo* (*cooperative*); if both non-cooperative and catalytic rules are allowed, we write *cat* (*catalytic*).

If in general a P system has more than one membrane, each symbol on the right-hand side may have assigned a target where the symbol has to be sent after the application of the rule, where the targets take into account the tree structure of the membranes as follows:

here the symbol stays in the membrane where the rule is applied;
out the symbol is sent to the outer membrane, i.e., the membrane enclosing the membrane where the rule is applied;

in the symbol is sent to an inner membrane, i.e., a membrane enclosed by the membrane where the rule is applied;
in_j the symbol is sent to the inner membrane labeled by j.

3.2 Derivation Modes

In general, the set of all multisets of rules applicable in a P system to a given configuration C is denoted by $Appl(\Pi, C)$ and can be restricted by imposing specific conditions, thus yielding the following basic derivation modes (for example, see [21] for formal definitions):

- asynchronous mode (abbreviated $asyn$): at least one rule is applied;
- sequential mode ($sequ$): only one rule is applied;
- maximally parallel mode (max): a non-extendable multiset of rules is applied;
- maximally parallel mode with maximal number of rules (max_{rules}): a non-extendable multiset of rules of maximal possible cardinality is applied;
- maximally parallel mode with maximal number of objects ($max_{objects}$): a non-extendable multiset of rules affecting as many objects as possible is applied.

In [7], these derivation modes are restricted in such a way that each rule can be applied at most once, thus yielding the set modes $sasyn$, $smax$, $smax_{rules}$, and $smax_{objects}$ (the sequential mode is already a set mode by definition):

- asynchronous set mode (abbreviated $sasyn$): at least one rule is applied, but each rule at most once;
- maximally parallel set mode ($smax$): a non-extendable set of rules is applied;
- maximally parallel set mode with maximal number of rules ($smax_{rules}$): a non-extendable set of rules of maximal possible cardinality is applied;
- maximally parallel set mode with maximal number of objects ($smax_{objects}$): a non-extendable set of rules affecting as many objects as possible is applied.

Let us denote the set of all multisets (possibly only sets) of rules applicable in a (tissue) P system Π to a given configuration C in the derivation mode δ by $Appl(\Pi, C, \delta)$. We immediately observe that $Appl(\Pi, C, asyn) = Appl(\Pi, C)$.
To collect the set and multiset derivation modes, we use the following notations:
$D_S = \{sequ, sasyn, smax, smax_{rules}, smax_{objects}\}$ and
$D_M = \{asyn, max, max_{rules}, max_{objects}\}$.

3.3 Halting Conditions

Besides the standard total halting with no (multi)set of rules being applicable any more to the current configuration, some more variants of halting conditions have been considered in the literature:

total halting (H) the common halting strategy where the computation stops with no (multi)set of rules being applicable any more

unconditional halting (u) the result of a computation can be taken from every configuration derived from the initial one (possibly only taking terminal results)

partial halting (h) the set of rules R is partitioned into disjoint subsets R_1 to R_h, and a computation stops if there is no multiset of rules applicable to the current configuration which contains a rule from every set R_j, $1 \leq j \leq h$

halting with states (s) the configuration with which a derivation may stop must fulfill a recursive condition (which corresponds with a *final state*)

The variant of unconditional halting was introduced in [10]. Partial halting, for example, was investigated in [6, 8, 18], using the membranes for partitioning the rules. Formal definitions for the halting conditions H, h, s can be found in [21].

In the description for P systems, the derivation relation under the derivation mode δ, $\Longrightarrow_{\Pi,\delta}$, is extended by the halting condition, i.e., we then write $\Longrightarrow_{\Pi,\delta,\beta}$ for $\beta \in \{H, h, u, s\}$. By default, β is understood to be the total halting H and then usually omitted.

3.4 Rule and Anti-Rule Objects

In the set of rules R, the rules on the left-hand side may only contain symbols from V, whereas on the right-hand side of a rules any symbol from V_H may appear.

In any computation step of a PARS Π only a multiset R' of rules can be applied such that for each (copy of a) rule r a rule object h with $g(h) = r$ is present in the current configuration. With the application of a rule r, a copy of a rule object h with $g(h) = r$ is consumed, i.e., the number of rule objects in the sense of multisets is important for the applicability of a multiset of rules.

As we allow anti-rule objects to be generated, too, we implicitly assume the rule / anti-rule annihilation rule $hh^- \to \lambda$ to be present in every membrane, for every $h \in H$. Before the next derivation step as described above can be carried out, all such annihilation rules have to be executed, as we assume them to have (weak) priority over all other rules.

As already mentioned in Section 1, each copy of a rule object allows for exactly one application of the corresponding rule it represents, i.e., we deal with multisets of rules only applicable if the corresponding multiset of rule objects is present, which restricts the set of applicable sets of multisets of rules in a given derivation mode, especially in the maximally parallel derivation modes.

Hence, there are two possible ways how to define the applicable multisets of rules applicable to a given configuration; we observe that a configuration may contain elements from V_H, not only V, but as the rule / anti-rule annihilation rules $hh^- \to \lambda$ have weak priority, no pair hh^- can be present any more:

starting with applicable multisets (δ_a) We start with the set of all applicable sets of multisets of rules, no matter which and how many rule objects are present, then take out all the multisets of rules which conform with the given derivation

mode, and then only take those for which sufficient resources of rule objects are available.

starting with available rule objects (δ_r) We first take only those multisets of rules for which there are sufficient resources of rule objects available, take these as the set of applicable rule multisets and only afterwards apply the condition for the derivation mode.

Which variant we choose for a given derivation mode, will be indicated by a subscript a or r to δ, i.e., we write δ_a and δ_r, respectively.

The *language generated by* the PARS Π is the set of all terminal multisets which can be obtained in the output membrane f starting from the initial configuration $C_1 = (w_1, \ldots, w_m)$ using the derivation mode δ_α, $\alpha \in \{a, r\}$, in a halting computation using the halting condition β, i.e.,

$$L_{gen,\delta_\alpha,\beta}(\Pi) = \left\{ C(f) \in T^\circ \mid C_1 \stackrel{*}{\Longrightarrow}_{\Pi,\delta_\alpha,\beta} C \wedge \text{halting}_{\delta_\alpha,\beta}(C) \right\},$$

where $C(f)$ stands for the multiset contained in the output membrane f of the final configuration C and $\text{halting}_{\delta_\alpha,\beta}(C)$ indicates that C is a halting configuration with respect to the halting condition β when using δ_α.

The family of languages of multisets generated by PARSs of type X with at most n membranes in the derivation mode δ_α using the halting condition β is denoted by $Ps_{gen,\delta_\alpha,\beta}OP_n(X)$.

We may also consider PARSs as accepting mechanisms: in membrane f, we add the input multiset w_0 to w_f in the initial configuration $C_1 = (w_1, \ldots, w_m)$ thus obtaining $C_1[w_0] = (w_1, \ldots, w_f w_0, \ldots, w_m)$; the input multiset w_0 is accepted if there exists a halting computation in the derivation mode δ_α starting from $C_1[w_0]$, i.e.,

$$L_{acc,\delta_\alpha,\beta}(\Pi) = \left\{ w_0 \in T^\circ \mid \exists C : \ \left(C_1[w_0] \stackrel{*}{\Longrightarrow}_{\Pi,\delta_\alpha,\beta} C \wedge \text{halting}_{\delta_\alpha,\beta}(C) \right) \right\}.$$

Then the family of languages of multisets accepted by PARSs of type X with at most n membranes in the derivation mode δ_α using the halting condition β is denoted by $Ps_{acc,\delta_\alpha,\beta}OP_n(X)$.

We finally mention that PARSs can also be used to compute functions and relations, with using f both as input and output membrane or even using two different membranes for the input and the output. Yet in this paper we will mainly focus on the generating case.

3.5 Flattening

As many variants of P systems can be *flattened* to only one membrane, see [17], we often may assume the simplest membrane structure of only one membrane which in effect reduces the P system to a multiset processing mechanism, and, observing that $f = 1$, in what follows we then will use the reduced notation

$$\Pi = \left(V, T, H \cup H^-, w, R, g_H, \Longrightarrow_{\Pi,\delta_\alpha,\beta}\right)$$

for a PARS with only one membrane, for which the definitions for the *language generated by* Π and the *language accepted by* Π can be written in an easier way, i.e.,

$$L_{gen,\delta_\alpha,\beta}(\Pi) = \left\{v \in T^\circ \mid w \overset{*}{\Longrightarrow}_{\Pi,\delta_\alpha,\beta} v \wedge \text{halting}_{\delta_\alpha,\beta}(v)\right\} \text{ and}$$
$$L_{acc,\delta_\alpha,\beta}(\Pi) = \left\{w_0 \in T^\circ \mid \exists v : \left(ww_0 \overset{*}{\Longrightarrow}_{\Pi,\delta_\alpha,\beta} v \wedge \text{halting}_{\delta_\alpha,\beta}(v)\right)\right\}.$$

The family of languages of multisets generated by one-membrane PARSs of type X in the derivation mode δ_α using the halting condition β is denoted by $Ps_{gen,\delta_\alpha,\beta}OP(X)$.

The family of languages of multisets accepted by one-membrane PARSs of type X in the derivation mode δ_α using the halting condition β is denoted by $Ps_{acc,\delta_\alpha,\beta}OP(X)$.

The following example illustrates that the two variants δ_a and δ_a need not yield the same results:

Example 1. Take the PARS

$$\Pi = \left(V = \{a\}, T = \{a\}, H \cup H^-, w = ar, R = \{a \to aar\}, g_H, \Longrightarrow_{\Pi,\delta_\alpha,u}\right)$$

with $H = \{r\}$ and $g_H = \{(r, a \to aar)\}$. Then, with every application of the rule $r : a \to aar$ we get one more symbol a, i.e., for $\alpha \in \{a, r\}$, we have

$$\begin{aligned}\{a\}^+ &= L_{gen,sequ_\alpha,u}(\Pi) = L_{gen,smax_\alpha,u}(\Pi)\\ &= L_{gen,max_r,u}(\Pi).\end{aligned}$$

But on the other hand, $L_{gen,max_a,u}(\Pi) = \{a, aa\}$, because after the first application of rule r we obtain the configuration aar, which is a terminal one, as the only applicable multiset of rules which is not extendable would contain two copies of the rule $a \to aar$, yet only one corresponding rule object r is available.

The following example shows that different results are obtained with different derivation modes δ:

Example 2. Take the PARS

$$\Pi = \left(V = \{a\}, T = \{a\}, H \cup H^-, w = ar, R = \{a \to aarr\}, g_H, \Longrightarrow_{\Pi,\delta_\alpha,u}\right)$$

with $H = \{r\}$ and $g_H = \{(r, a \to aarr)\}$. Then, with every application of the rule $r : a \to aarr$ we double the number of symbols a when using a maximally parallel derivation mode, i.e., for $\alpha \in \{a, r\}$, we have

$$L_{gen,\delta_\alpha,u}(\Pi) = \left\{a^{2^n} \mid n \geq 0\right\}$$

for any $\delta \in \{max, max_{rules}, max_{objects}\}$, because now the number of rule objects r grows in the same way as the number of symbols a. But on the other hand, we still get $L_{gen,sequ_\alpha,u}(\Pi) = \{a\}^+$, because in every derivation step the rule $r : a \to aarr$ can only be applied once, although the number of rule objects grows in a similar way as the number of symbols a, i.e., we obtain a sequence of configurations $a^n r^n$, $n \geq 1$, from which we extract the terminal results a^n, $n \geq 1$.

4 Results

As our first result, we will show that with the sequential derivation mode PARSs at least are as powerful as partially blind register machines.

Afterwards, computational completeness will be established for PARSs working in the derivation modes max_r and $smax_r, smax_a$.

4.1 Sequential PARSs

In order to easily comply with the final condition that in halting computations of partially blind register machines all non-output registers should be zero, for the PARSs in the following theorem we use halting with states, where the final states are defined in such a way that the PARS can only halt with yielding a result if only terminal symbols are present any more. Now let $NPBRM$ and $PsPBRM$ denote the families of sets of multisets and vectors of multisets, respectively, generated by partially blind register machines.

Theorem 1. *For any $Y \in \{N, Ps\}$ and $\alpha \in \{a, r\}$,*

$$YPBRM \subseteq Y_{gen,sequ_\alpha,s}OP(ncoo).$$

Proof. Let $M = (m, B, l_0, l_h, P)$ be a partially blind register machine; we assume the output registers to be the first k ones. We now construct a PARS Π which simulates (the computations of) M; the contents of register r is represented by the number of copies of symbols a_r:

$$\Pi = \left(V, T, H \cup H^-, w, R, g_H, \Longrightarrow_{\Pi,sequ_\alpha,s}\right)$$

where

- $V = \{a_j \mid 1 \leq j \leq m\} \cup \{s, \#\}$;
- $T = \{a_j \mid 1 \leq j \leq k\}$;
- $H = B \cup B'_{SUB} \cup \{l_\#\}$, i.e., the *rule objects* are the instruction labels in B, their primed variants, but only for the SUB-instructions, i.e.,

  $$B'_{SUB} = \{l'_1 \in B \mid l_1 : (SUB(r), l_2) \in P\},$$

  and the label $l_\#$ for the trap rule $s \to \#$; we also define $V_H := V \cup H \cup H^-$;

- $w = l_0 s$, i.e., we start with the symbol s and the rule object l_0 representing the initial instruction.

The instructions of M are simulated by the following rules in R; we will write $l : r$ to both indicate the rule r and the label l, having assigned the rule r by the function g_H.

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Simulated by the rules $l_1 : s \to a_r l_2$ and $l_1 : s \to a_r l_3$.

- $l_1 : (SUB(r), l_2)$, with $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, $1 \leq r \leq m$.
  As labeled rule common for the simulations of all SUB-instructions, we have $l_\# : s \to \#$ and the rule / anti-rule annihilation rule $l_\# {l_\#}^- \to \lambda$.
  The application of the rule $l_\# : s \to \#$ traps the whole computation, as $\#$ is no terminal symbol and thus any configuration containing the trap symbol cannot fulfill the final state condition.
  The *decrement case* for instruction l_1 is simulated by the rules
  $l_1 : s \to s l_1' l_\#$ generating the rule objects l_1' and $l_\#$, and
  $l_1' : a_r \to l_2 {l_\#}^-$, which guarantees a correct simulation of a possible decrement instruction in case register r is not empty (and also eliminates the rule object $l_\#$ by generating its anti-rule object ${l_\#}^-$).
  If register r is empty, then the application of the rule labeled with $l_\#$ generating the trap symbol $\#$ is enforced, which can only be avoided if register r is not empty and the rule labeled with l_1' can be applied instead.

- $l_h : HALT$. Simulated by $l_h : s \to \lambda$.

When the computation in M halts, the state symbol s is removed, and no further rules can be applied provided the simulation of a valid computation in M has been carried out correctly, i.e., if no trap symbols $\#$ are present in this situation, which then guarantees that the halting condition is fulfilled. The terminal symbols in the skin membrane represent the result computed by M. □

In the preceding proof we have taken advantage of the halting condition with final states guaranteeing to take only terminal results. Yet we can also take the standard total halting, but paying the price with having rule objects remaining in the halting computations:

Theorem 2. *For any* $Y \in \{N, Ps\}$ *and* $\alpha \in \{a, r\}$,

$$YPBRM \subseteq Y_{gen,sequ_\alpha,H}OP(ncoo).$$

Proof. As in the preceding proof we simulate a partially blind register machine $M = (m, B, l_0, l_h, P)$, where we assume the output registers to be the first k ones.

We then construct a PARS Π' which simulates (the computations of) M, where Π' is obtained from Π as constructed in the proof of Theorem 1 by extending it in the following way:

$$\Pi' = \left(V, T, H' \cup H'^{-}, w, R', g_{H'}, \Longrightarrow_{\Pi, sequ_\alpha, H}\right)$$

where

- $V = \{a_j \mid 1 \leq j \leq m\} \cup \{s, \#\}$;
- $T = \{a_j \mid 1 \leq j \leq k\}$;
- $H' = B \cup B'_{SUB} \cup \{l_\#, l'_\#\} \cup B_r$, i.e., the *rule objects* are the instruction labels in B, their primed variants, but only for the SUB-instructions, i.e.,

  $$B'_{SUB} = \{l'_1 \in B \mid l_1 : (SUB\,(r)\,, l_2) \in P\},$$

  the label $l_\#$ for the trap rule $s \to \#$, as well as, in addition now, the labels in $B_r = \{l_{j,\#} \mid k+1 \leq j \leq m\}$ needed for the final zero check and the label $l'_\#$, which is needed for the additional trap rule $\# \to \#$; we also define $V_{H'} := V \cup H' \cup H'^{-}$;
- $w = l_0 s$, i.e., we start with the symbol s and the rule object l_0 representing the initial instruction.

For simulating the instructions of M we take all rules in R' constructed as follows:

- $l_1 : (ADD\,(j)\,, l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Simulated by the rules $l_1 : s \to a_r l_2$ and $l_1 : s \to a_r l_3$.

- $l_1 : (SUB\,(r)\,, l_2)$, with $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, $1 \leq r \leq m$.
  As labeled rule common for the simulations of all SUB-instructions, we have $l_\# : s \to \#$ and the rule / anti-rule annihilation rule $l_\# {l_\#}^{-} \to \lambda$.
  The application of the rule $l_\# : s \to \#$ traps the whole computation by introducing the trap symbol $\#$.
  The *decrement case* for instruction l_1 is simulated by the rules
  $l_1 : s \to s l'_1 l_\#$ generating the rule objects l'_1 and $l_\#$, and
  $l'_1 : a_r \to l_2 {l_\#}^{-}$, which guarantees a correct simulation of a possible decrement instruction in case register r is not empty (and also eliminates the rule object $l_\#$ by generating its anti-rule object ${l_\#}^{-}$).
  If register r is empty, then the application of the rule labeled with $l_\#$ generating the trap symbol $\#$ is enforced, which can only be avoided if register r is not empty and the rule labeled with l'_1 can be applied instead.

- $l_h : HALT$.
  Instead of $l_h : s \to \lambda$ we now use the final rule $l_h : s \to l'_\# l_{k+1,\#} \ldots l_{m,\#}$, where $l_{j,\#} : a_j \to \#$, $k+1 \leq j \leq m$, introduces the trap symbol $\#$, if any of the non-output registers is not empty when l_h is reached. If at some moment the trap symbol is introduced, an infinite (non-halting) derivation finally is guaranteed by the rule $l'_\# : \# \to \# l'_\#$.

When the computation in M halts, the state symbol s is removed, and no further rules can be applied provided the simulation of a valid computation in M has been carried out correctly, i.e., if no trap symbols # are present in this situation, *AND* the final zero test is successful, i.e., none of the rules $l_{j,\#} : a_j \to \#$, $k+1 \leq j \leq m$ is applicable in the step after the application of the final rule $l_h : s \to l'_{\#} l_{k+1,\#} \ldots l_{m,\#}$.

Only the terminal symbols in the skin membrane represent the result computed by M. □

In the proof of Theorem 1 we needed the anti-rule object ${l_{\#}}^-$ and the rule / anti-rule annihilation rule $l_{\#}{l_{\#}}^- \to \lambda$ to finally obtain a clean result without any rule object. Yet as in the construction of the PARS in Theorem 2 we anyway get rule objects remaining in the final configuration we need not use this anti-rule object ${l_{\#}}^-$ and the corresponding rule / anti-rule annihilation rule $l_{\#}{l_{\#}}^- \to \lambda$ for the PARS Π' constructed in the proof of Theorem 2 any more; the remaining technical details are left to the interested reader.

Denoting a PARS not needing anti-rule objects as well as the corresponding rule / anti-rule annihilation rules by PRS and indicating this by writing α^0 instead of α, we immediately may state the following result:

Corollary 1. *For any* $Y \in \{N, Ps\}$ *and* $\alpha \in \{a, r\}$,

$$YPBRM \subseteq Y_{gen,sequ_{\alpha^0},H}OP(ncoo).$$

We remark that the PARSs in Examples 1 and 2 in fact also are only using rule objects and thus are PRSs only.

It remains a challenging open question if we can go beyond $YPBRM$ when using the sequential derivation mode and non-cooperative rules.

4.2 Computational Completeness

We now show that PARSs characterize the families NRE and $PsRE$, respectively. The main proof idea – as used very often in the area of P systems – is to simulate (the computations of) register machines, as carried out in a similar way in [1] for P systems with anti-matter.

Theorem 3. *For any* $Y \in \{N, Ps\}$ *and* $\gamma \in \{gen, acc\}$,

$$Y_{\gamma,\delta_r,H}OP(ncoo) = YRE$$

for any $\delta \in \{max, max_{rules}, max_{objects}\}$.

Proof. Let $M = (m, B, l_0, l_h, P)$ be a register machine. We now construct a PARS Π which simulates (the computations of) M; the contents of register r is represented by the number of copies of symbols a_r:

$$\Pi' = \left(V, T, H \cup H^-, w, R, g_H, \Longrightarrow_{\Pi, sequ_\alpha, H}\right)$$

where

- $V = \{a_j \mid 1 \leq j \leq m\} \cup \{s\}$;
- $T = \{a_j \mid 1 \leq j \leq k\}$;
- $H = B \cup \tilde{B} \cup B_r$, where
  $\tilde{B} = \{l_1', l_1'' \mid l_1 \in B, l_1 : (SUB\,(r)\,, l_2, l_3) \in P\}$ and
  $B_r = \{l_{j,l_1} \mid l_1 \in B, l_1 : (SUB\,(r)\,, l_2, l_3) \in P, \text{ and } 1 \leq r \leq m\}$;
  we also define $V_H := V \cup H \cup H^-$;
- $w = l_0 s$, i.e., we start with the symbol s and the rule object l_0 representing the initial instruction.

The instructions of M are simulated by the following rules in R (we emphasize that by definition, for all rule objects $l \in H$ also the anti-rule objects l^- are part of the PARS Π as well as all the corresponding rule / anti-rule annihilation rules $l\, l^- \to \lambda$ are in R):

- $l_1 : (ADD\,(j)\,, l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Simulated by the rules $l_1 : s \to a_r l_2$ and $l_1 : s \to a_r l_3$.

- $l_1 : (SUB\,(r)\,, l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq m$.

  We start the simulation by using the rule
  $l_1 : s \to s l_1' l_{r,l_1}$ and then possibly use the two rules
  $l_1' : s \to s l_1''$ and $l_{r,l_1} : a_r \to l_2 {l_1''}^-$.
  If register r is not empty, then $l_{r,l_1} : a_r \to l_2 {l_1''}^-$ can be applied, at the same time generating the label l_2 to continue the computation after the successful simulation of the *decrement case* and eliminating the rule object l_1'' by generating its anti-rule object ${l_1''}^-$.
  In case the register is empty, the rule object l_1'' is still present in the next derivation step and correctly ends the simulation of the *zero test case* with
  $l_1'' : s \to s {l_{r,l_1}}^- l_3$,
  at the same time eliminating the rule object l_{r,l_1} by generating the corresponding anti-rule object ${l_{r,l_1}}^-$.

- $l_h : HALT$. Simulated by $l_h : s \to \lambda$.

When the computation in M halts, the symbol s is removed, and no further rules can be applied; as the simulation has been carried out correctly, the terminal symbols in the skin membrane at the end of a halting computation represent the result computed by M.

We finally observe the important fact that Π simulates the SUB-instructions of M in a deterministic way, i.e., in the accepting case the simulation of a deterministic register machine is deterministic. □

In the maximally parallel derivation modes δ we can only use the variant δ_r, because in case that more than one object a_r is present, we still want only one a_r to be erased in the decrement case, which is guaranteed by having only one rule object l_{r,l_1}, whereas without the restriction for the presence of rule objects, this rule would be used for every copy of a_r. On the other hand, in any of the set maximally derivation modes δ already the definition of the mode guarantees that this rule can only be applied once; hence, using the same construction as in the proof of Theorem 3, we immediately get the corresponding result for all δ_r *and* δ_a in case of the maximally parallel set derivation modes:

Corollary 2. *For any* $Y \in \{N, Ps\}$, $\alpha \in \{a, r\}$, *and* $\gamma \in \{gen, acc\}$,

$$Y_{\gamma,\delta_\alpha,H}OP(ncoo) = YRE$$

for any $\delta \in \{smax, smax_{rules}, smax_{objects}\}$.

5 Conclusion

In this paper we have taken over the idea of matter and anti-matter objects in P systems to P systems with rule objects and anti-rule objects. For each rule to be applied, a rule object must be present, which is consumed by the application of the rule. Whenever a rule object h meets its anti-rule object h^- the rule / anti-rule annihilation rule $hh^- \to \lambda$, independent from the underlying derivation mode, has to be applied before the next derivation step is executed.

The use of anti-rule objects and the corresponding rule / anti-rule annihilation rules allows for the simulation of register machines with only non-cooperative rules and any of the maximally (set) derivation modes. In the sequential mode, non-cooperative rules together with rule objects, but even without anti-rule objects and the corresponding rule / anti-rule annihilation rules at least allow for the simulation of partially blind register machines. Whether we could obtain more, remains as a challenge for future research.

In that sense, anti-rule objects (and the corresponding rule / anti-rule annihilation rules) may also constitute a frontier of tractability as it was shown for anti-matter, for example, see [12]. Investigating these kinds of complexity issues is also a project for future research.

In this paper, we have only investigated some of the possible combinations of derivation modes and halting conditions. Considering other combinations of derivation modes and halting conditions as well as other kinds of rules, for example, insertion, deletion, and substitution, is a promising topic for the future, too.

Another concept that can be added to the model of P systems with anti-rule objects is to use decaying objects as in [15], but only for the rule objects and the

anti-rule objects. A rule object decaying in t time steps means that it can only activate the application of the rule it stands for during the next t derivation steps, whereafter the rule object vanishes even without the application of the rule / anti-rule annihilation rule.

Acknowledgements

The ideas for this paper came up in the inspiring atmosphere of the Brainstorming Week on Membrane Computing in Sevilla this year.

References

1. Alhazov, A., Aman, B., Freund, R.: P systems with anti-matter. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8961, pp. 66–85. Springer (2014). https://doi.org/10.1007/978-3-319-14370-5_5
2. Alhazov, A., Aman, B., Freund, R., Păun, Gh.: Matter and anti-matter in membrane systems. In: Macías-Ramos, L.F., Martínez-del-Amor, M.A., Păun, Gh., Riscos-Núñez, A., Valencia-Cabrera, L. (eds.) Proceedings of the Twelfth Brainstorming Week on Membrane Computing. pp. 1–26 (2014), http://www.gcn.us.es/files/12bwmc/001_bwmc2014AntiMatter.pdf
3. Alhazov, A., Freund, R., Ivanov, S.: Introducing the concept of activation and blocking of rules in the general framework for regulated rewriting in sequential grammars. In: Proceedings of BWMC 2018 (2018)
4. Alhazov, A., Freund, R., Ivanov, S.: P systems with activation and blocking of rules. In: Stepney, S., Verlan, S. (eds.) Unconventional Computation and Natural Computation – 17th International Conference, UCNC 2018, Fontainebleau, France, June 25-29, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10867, pp. 1–15. Springer (2018). https://doi.org/10.1007/978-3-319-92435-9_1
5. Alhazov, A., Freund, R., Ivanov, S.: Sequential grammars with activation and blocking of rules. In: Machines, Computations, and Universality – 8th International Conference, MCU 2018, Fontainebleau, France, June 28-30, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10881, pp. 51–68 (2018). https://doi.org/10.1007/978-3-319-92402-1_3
6. Alhazov, A., Freund, R., Oswald, M., Verlan, S.: Partial halting in P systems using membrane rules with permitting contexts. In: Durand-Lose, J., Margenstern, M. (eds.) Machines, Computations, and Universality. pp. 110–121. Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74593-8_10
7. Alhazov, A., Freund, R., Verlan, S.: P systems working in maximal variants of the set derivation mode. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing – 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10105, pp. 83–102. Springer (2017). https://doi.org/10.1007/978-3-319-54072-6_6
8. Alhazov, A., Oswald, M., Freund, R., Verlan, S.: Partial halting and minimal parallelism based on arbitrary rule partitions. Fundam. Inform. **91**(1), 17–34 (2009). https://doi.org/10.3233/FI-2009-0031

9. Alhazov, A., Sburlan, D.: Static sorting P systems. In: Ciobanu, G., Pérez-Jiménez, M.J., Păun, Gh. (eds.) Applications of Membrane Computing, pp. 215–252. Natural Computing Series, Springer (2006). https://doi.org/10.1007/3-540-29937-8_8
10. Beyreder, M., Freund, R.: Membrane systems using noncooperative rules with unconditional halting. In: Corne, D.W., Frisco, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing. pp. 129–136. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-540-95885-7_10
11. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer (1989), `https://www.springer.com/de/book/9783642749346`
12. Díaz-Pernil, D., Peña-Cantillana, F., Alhazov, A., Freund, R., Gutiérrez-Naranjo, M.A.: Antimatter as a frontier of tractability in membrane computing. Fundam. Inform. **134**(1–2), 83–96 (2014). https://doi.org/10.3233/FI-2014-1092
13. Freund, R.: Generalized P-Systems. In: Ciobanu, G., Păun, Gh. (eds.) Fundamentals of Computation Theory, 12th International Symposium, FCT '99, Iaşi, Romania, August 30 – September 3, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1684, pp. 281–292. Springer (1999)
14. Freund, R.: Purely catalytic P systems: Two catalysts can be sufficient for computational completeness. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu. (eds.) CMC14 Proceedings – The 14th International Conference on Membrane Computing, Chişinău, August 20–23, 2013. pp. 153–166. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova (2013), `http://www.math.md/cmc14/CMC14_Proceedings.pdf`
15. Freund, R.: (Tissue) P systems with decaying objects. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) Membrane Computing – 13th International Conference, CMC 2012, Budapest, Hungary, August 28–31, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7762, pp. 1–25. Springer (2013)
16. Freund, R., Kari, L., Oswald, M., Sosík, P.: Computationally universal P systems without priorities: two catalysts are sufficient. Theoretical Computer Science **330**(2), 251–266 (2005). https://doi.org/10.1016/j.tcs.2004.06.029
17. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 8340, pp. 173–188. Springer (2014). https://doi.org/10.1007/978-3-642-54239-8_13
18. Freund, R., Oswald, M.: Partial halting in P systems. Int. J. Found. Comput. Sci. **18**(6), 1215–1225 (2007). https://doi.org/10.1142/S0129054107005261
19. Freund, R., Oswald, M.: Catalytic and purely catalytic P automata: control mechanisms for obtaining computational completeness. In: Bensch, S., Drewes, F., Freund, R., Otto, F. (eds.) Fifth Workshop on Non-Classical Models for Automata and Applications – NCMA 2013, Umeå, Sweden, August 13 – August 14, 2013, Proceedings. books@ocg.at, vol. 294, pp. 133–150. Österreichische Computer Gesellschaft (2013)
20. Freund, R., Păun, Gh.: How to obtain computational completeness in P systems with one catalyst. In: Neary, T., Cook, M. (eds.) Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9–11, 2013. EPTCS, vol. 128, pp. 47–61 (2013). https://doi.org/10.4204/EPTCS.128.13
21. Freund, R., Verlan, S.: A formal framework for static (tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane

Computing, Lecture Notes in Computer Science, vol. 4860, pp. 271–284. Springer (2007). https://doi.org/10.1007/978-3-540-77312-2_17
22. Minsky, M.L.: Computation. Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ (1967)
23. Pan, L., Păun, Gh.: Spiking neural P systems with anti-matter. International Journal of Computers, Communications & Control **4**(3), 273–282 (2009). https://doi.org/10.15837/ijccc.2009.3.2435, http://univagora.ro/jour/index.php/ijccc/article/download/2435/901
24. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693
25. Păun, Gh.: Membrane Computing: An Introduction. Springer (2002). https://doi.org/10.1007/978-3-642-56196-2
26. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
27. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer (1997). https://doi.org/10.1007/978-3-642-59136-5
28. The P Systems Website. http://ppage.psystems.eu/