P Systems with Hybrid Sets

Omar Belingheri, Antonio E. Porreca, and Claudio Zandron

Universitá degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
o.belingheri@campus.unimib.it,
{porreca, zandron}@disco.unimib.it

Abstract. In our investigation of the power of a new type of P system which works with objects that can have negative multiplicities, we prove that it is strictly less powerful than a Turing Machine. We get this result by simulating such device using partially blind register machines.

1 Introduction

P systems are a computational model inspired by the structure of a biological cell. Such a model contains several membranes, which can contain several objects; such objects can be transformed and moved from membrane to membrane according to evolution rules. Evolution rules are applied at each step of the computation, changing the configuration of the system until it eventually halts (when no more rules can be applied). The result of a halting computation is the multiset of objects contained in a specified output membrane (or expelled from the system). If the computation does not halt, then it produces no output.

P systems are particularly interesting for their efficiency: trading space for time, we can deal in a polynomial time with problems normally solvable in an exponential time (for further information see [5]). This is possible thanks to the maximal parallel manner in which they operate. A detailed definition of what a P system and its components are can be found in [6].

Several variants of P systems have been investigated through the years. The main differences among the variants usually include the use of new types of rules, or the addition of special abilities to the membranes. However, our variant stands out a little when compared to the others. In fact, we will deal with an aspect regarding the very nature of the system, rather than just focusing on changing specific aspects like its rules. So far the multiplicities representing the objects within a membrane have only been allowed to be nonnegative. In this paper we will let those multiplicities be negative too, expanding the range of their possible values from the set $\mathbb{N}$ to $\mathbb{Z}$ (another attempt to generalize the multiset has been explored in [3]). In doing so we will have to define a new type of P system, called *hybrid P system*, in some ways similar to a catalytic P system (see [6], page 53). After giving some formal definitions necessary to understand hybrid P systems in Section 2, we will go on to investigate their power in Section 3. It will turn out that such systems are less powerful than the Turing Machine, as we will show.

2 Preliminaries

Definition 1. *A P system is a construct*

$$\Pi = (O, \mu, w_1, ..., w_m, R_1, ..., R_m, i_o),$$

where:

1. *O is an alphabet whose elements are called objects;*
2. *μ is the membrane structure with m membranes, labelled with $1, ..., m$;*
3. *w_i, $1 \leq i \leq m$, are strings of the form $a_1^{M(a_1)} a_2^{M(a_2)} ... a_n^{M(a_n)}$ representing the multisets of objects in the regions $1, ..., m$ they are associated with ($M(a_i)$ is the multiplicity of the i-th symbol);*
4. *R_i, $1 \leq i \leq m$ are finite sets of evolution rules. Each R_i associated with region i of μ. Evolution rules are of the form $u \to v$, where u is a string over O and v is a string over $O_{tar} = O \times TAR$, where $TAR = \{here, out\} \cup \{in_j | 1 \leq j \leq m\}$. The symbols here, out, in_j, are called target commands (or target indications) and specify whether an object has to stay in its current membrane (here), go to the outer membrane (out) or must be sent to an inner membrane labelled with j (in_j). Usually we omit the indication "here" for simplicity;*
5. *$i_o \in \{1, ..., m\}$ is the label of an elementary membrane, called output membrane.*

In *hybrid* P systems we allow only standard rules to be used (like the ones we just presented). In such rules only objects with positive multiplicities appear. However we add one condition: objects do not necessarily have to be present for the rules to be applied. This means we can apply a rule regardless of whether its left-hand objects are present with positive multiplicity in a membrane. When doing so, we must subtract from the membrane the amount of objects that were used to apply the rule. For example, if a membrane contains a^2, the rule $a^3 \to bc$ can still be applied, the result of the application being $a^{-1}bc$.

This immediately raises one problem: if the rules can be applied regardless of the presence of an object, then at every step any rule can be applied an infinite amount of times. One possible solution to prevent this from happening, is to add catalysts to the rules. Catalysts are objects present in a finite number in the system, they are used to apply the rules but are not changed by them, and they are not allowed to have negative multiplicity. An example of this kind of rule could be $ak \to ck$, where k is a catalyst.

Before hybrid P systems were conceived, we have always worked with multisets defined as mappings $M : A \longrightarrow \mathbb{N}$, with A an arbitrary set. In order to consider multisets with negative multiplicities, we need to extend that definition to *hybrid sets*, as defined in [1]:

Definition 2. *Given a universe U, any function $f : U \longrightarrow \mathbb{Z}$ is called a hybrid set. As usual, the value $f(x)$ for an element $x \in U$ is said to be the multiplicity of x.*

From now on we will work with hybrid sets instead of multisets. This means it will be normal, and actually very common, for an object to have a negative multiplicity. What do "negative objects" represent in the real world? How can an object be present a negative number of times? In this paper it is not our purpose to answer that question. Our goal is not to model a real cell, but to define and explore new *theoretical* models inspired by cells. However, notice that some physical quantities, such as electrical charge, may indeed have negative integer values and can possibly be modelled by hybrid sets.

Definition 3. *A hybrid P system is defined as*

$$\Pi = (O, K, \mu, w_1, ..., w_m, R_1, ..., R_m, i_o)$$

where all components are defined as for standard P systems (Definition 1), except that all multisets are replaced by hybrid sets. We also have an alphabet of objects $K \subseteq O$ used as catalysts; we require $M(k) \geq 0$ for all $k \in K$ and all hybrid sets M. Furthermore, the evolution rules are of the form $uk \rightarrow vk_t$, where $u \in O^\star$, $k \in P$ and v is a string over O_{tar}, where $O_{tar} = (O - K) \times TAR$, for $TAR = \{here, out\} \cup \{in_j | 1 \leq j \leq m\}$, and $t \in TAR$.

A *configuration* of a system is the m-tuple of hybrid sets of objects present in the system in its m regions. The *initial configuration* of a system is $(w_1, ..., w_m)$.

As time passes, the configuration of a system changes thanks to the application of the evolution rules. A global clock is assumed to exist. Its function is to mark the time for each membrane within the system, dividing the computation in steps. At each step, the evolution rules are applied in a *non-deterministic* and *maximally parallel* manner. This means that the rules to be applied are chosen in a non-deterministic way within every membrane, and every object that can use a rule to evolve must do so.

A rule $R_i : uk \rightarrow vk_t$ can be applied if k is present in membrane i. If it is, the hybrid set u is removed from the membrane and the objects in v are introduced, according to their multiplicity and target commands, and k is (possibly) moved according to t. If v contains the pair (a, in_j) but j is not a membrane immediately inside i, then that rule cannot be applied. Whenever an object is sent out of the skin into the environment, it cannot come back.

Given two configurations $C_1 = (w'_1, ..., w'_m)$ and $C_2 = (w''_1, ..., w''_m)$ of the same system Π, we say that we have a transition from C_1 to C_2 if we can pass from C_1 to C_2 using the evolution rules $R_1, ..., R_m$ in the regions of the system (i.e. we simultaneously transform the hybrid sets in C_1 to obtain C_2). This can be written as $C_1 \Longrightarrow C_2$.

A *computation* is a sequence of transitions between configurations of a system. It is considered successful if and only if the system halts (i.e. no further rules can be applied). In hybrid P systems, this can only happen when all catalysts have been moved to regions where no rule involves them. Once it halts, the result is given by the multiplicities of objects in the output membrane i_o of the system. A computation that does not halt produces no output. P systems can be seen as generators of numbers: we then denote by $Z(\Pi)$ the set of numbers that can be computed by the system, that is, all the possible vectors of multiplicities of objects in the output membrane when the computation stops.

3 Negative multiplicities weaken the power of a system

We now prove that the use of hybrid sets, where objects are allowed to have negative multiplicities, weakens P systems. Informally, one can expect such a result because of the possibility to transform rules like $uk \to vk_t$ with target $t \in \{here, out, in_j\}$ into the form $k \to u^{-1}vk_t$, that is, we are losing the power of the cooperation by moving u on the other side of the rule with opposite multiplicities.

3.1 Partially blind register machines

We need to formally introduce a particular type of register machine (as in [2]) that is said to be *partially blind*.

We start from a register machine $R = (m, B, l_0, l_h, P)$, where $m \geq 1$ is the number of counters, B is the finite set of instruction labels, l_0 is the initial label, l_h is the halting label, and P is the program, a finite set of instructions from B.

There are three types of instructions:

- $l_1 : (ADD(r), l_2, l_3)$, $1 \leq r \leq m$;
- $l_1 : (SUB(r), l_2, abort)$, $1 \leq r \leq m$ (if r was not empty, then go to l_2, otherwise the machine aborts without producing any result);
- $l_h : HALT$ (which halts the machine).

The main feature of this register machine is that the subtracting instruction does not check if the register is empty. When the RM executes a subtraction from an empty register, it aborts the computation. Technically, even though there does not seem to be a test for zero, this test is implicit: at the end of a successful computation we require some specified registers to be empty; all computations where this does not happen are discarded as aborted computations.

It is known that partially blind register machines are strictly less powerful than Turing machines [4].

3.2 Simulation using partially blind register machines

The language of hybrid sets (resp., the set of vectors of integers) generated by a system Π is denoted by $L(\Pi)$ and it consists of all hybrid sets (resp., vectors of multiplicities of hybrid sets) placed in the output membrane at the end of halting computations. We will denote by NHP and $NPBRG$ the sets generated by a hybrid P system and a partially blind register machine, respectively. We now prove that $NHP \subseteq NPBRG$.

Theorem 1. *Let L be a vector set of numbers generated by a hybrid P system $\Pi = (O, K, \mu, w_1, ..., w_m, R_1, ..., R_m, i_o)$. Then, there exist a PBRM that can generate L.*

Proof. We can build a partially blind register machine

$$R = (2m \cdot |O - K| + 2 \cdot |O - K|,\, B,\, 1,\, 4,\, P)$$

generating L as follows:

1. For each $a \in O - K$ there are pairs of registers labelled a_i^+ and a_i^- such that the value $a_i^+ - a_i^-$ is the multiplicity of object a in membrane i, with $1 \leq i \leq m$ (a total of $2m \cdot |O - K|$ registers). Note that we do not balance out positive and negative occurrences of objects until the computation is over (plus, we do this only in the output membrane); so it will be normal to have the registers associated with the positive and negative multiplicities of an object both nonnegative at the same time. This has no effect on the computation: in fact, the multiplicities of non-catalysts are basically irrelevant until the system stops and they become part of the result;
2. For each $a \in O - K$ we add two output registers labelled output_{a^+} and output_{a^-}. The register output_{a^+} (resp., output_{a^-}) will contain the absolute value of the final multiplicity of object a in the output membrane, if such multiplicity is positive (resp., negative), and zero otherwise.

The simulation works like this: in the first part, the registers from (1) are used to keep track of how many non-catalyst objects are being generated (resp., deleted); this is performed by increasing the registers a_i^+ (resp., increasing the registers a_i^-) without ever decreasing those registers.

The number of catalysts never changes during the computation, but they can only be moved around the membrane structure in a finite number of distinct configurations. Hence, we can keep track of the position of the catalysts as part of the label of the current instruction of the register machine. In particular, some instruction labels correspond to configurations of catalysts where no rule is applicable and thus Π halts its computation.

Hence, each configuration of catalysts enabling one or more rules of Π corresponds to a set of instructions of R updating the counters of (1) according to those rules. In general, several possible maximally parallel choices of rules competing for the catalysts are possible; however, since these are finite in number, one of them can be nondeterministically chosen by jumping to a corresponding instruction label. At the end of this update, the register machine jumps to the first label corresponding to the new configuration of catalysts moved by the rules they enabled (this might be the same as the current one if no catalyst has been moved at this time step).

Then, when the P system cannot apply any more rules, because all catalysts have been moved to a region where they have no applicable rules (this can be detected by the label of the current instruction of R) we enter a second phase of the simulation. The purpose of this phase is that of levelling the values contained in the pairs of registers a_o^+ and a_o^- (the ones associated with the output membrane): that is, we calculate $|a_o^+| - |a_o^-|$ and leave the result in a_o^+ (if the multiplicity is nonnegative) or in a_o^- (otherwise). This process is performed via a nondeterministically guessed number of subtractions, since we cannot perform a zero test (if the number of subtractions leads to decrementing an empty register, we simply obtain an aborted computation); the correct result of the subtraction will be checked by exploiting the halting condition of the register machine.

In the third phase of the simulation we copy the results in $output_{a^+}$ or $output_{a^-}$ while deleting the contents of a^+ or a^- respectively. This is also performed by repeated subtractions without zero testing (i.e., a nondeterministic number of times). Finally, the machine halts.

We require all the registers a_o^+, a_o^- $(\forall a \in O - K)$ to be zero when the machine halts. This way we can make sure that two things have happened: (1) the subtraction $|a_o^+| - |a_o^-|$ has been performed correctly, and (2) the multiplicities of the output objects have been copied correctly to the registers $output_{a^+}$ or $output_{a^-}$.

To generate the instructions in P, we proceed as follows. First of all, we take all the rules of the form $uk \to vk_t$ and we transform them into the form $k \to u^{-1}vk_t$. We can then think of grouping the rules depending on the configuration of catalyst that enable them and depending on simultaneous applicability, and generate all the possible combinations of rules that can be applied in one step. Once we have generated all the combinations, we can translate them into instructions for the machine, adding and subtracting the elements from their respective registers.

From the previous discussion, it is easy to see that the P systems can be simulated by the $PBRM$, generating the same vector set of numbers. $\square$

We show a simple example to clarify the process just described in the proof. The $PBRM$ starts in a configuration that reflects the initial state of the P system, which means that all of its registers have been initialized with the multiplicities of the non-catalyst objects they represent, and that the initial instruction label encodes the initial configuration of the catalysts. At each step of the computation, the instructions from the first phase are used until the machine enters the second phase, which calculates the final multiplicity of each non-catalyst object (a, b, c). Finally, the machine "copies" these values into the output registers using the instructions of the third phase and halts (the copy is attained by means of subtracting and adding 1 to the registers).

Example 1. Let us assume to have only one membrane with label 1 and the following rules (already in context-free notation):

$$k \to a^{-1}b^2k \qquad k \to k_{out} \qquad \ell \to b^{-1}c\ell \qquad \ell \to \ell_{out}$$

and that this membrane only contains the catalysts k and ℓ with multiplicity 1 in the initial configuration. Thus, there exist only 4 possible configurations of catalysts, depending on which have been already sent out. When both have been sent out, the computation of the P system halts. All catalyst configurations (except the halting one) involve one or two conflicting rules: each catalyst can either remain where it is and rewrite some objects, or be sent out.

For instance, the block of instructions corresponding to the catalyst configuration where both catalysts are still inside the membrane and the rules

$k \to a^{-1}b^2k$ and $\ell \to b^{-1}c\ell$ are applied is

$$\begin{aligned}
l_1 &: (ADD(a_1^-), l_{11}, l_{11}) \\
l_{11} &: (ADD(b_1^+), l_{12}, l_{12}) \\
l_{12} &: (ADD(b_1^+), l_{13}, l_{13}) \\
l_{13} &: (ADD(b_1^-), l_{14}, l_{14}) \\
l_{14} &: (ADD(c_1^+), l_0, l_0)
\end{aligned}$$

and the computation proceeds with label l_0, where another maximally parallel choice of rules enabled by the same catalyst configuration is nondeterministically made. If the catalysts had been moved, the computation would instead have jumped to the first label corresponding to the new configuration of catalysts.

Now suppose that the P system reaches a configuration where both catalysts have been sent out (thus no more rule is applicable), and suppose that this catalyst configuration corresponds to label l_4. Now the registers corresponding to the output region (membrane 1 in this case) are levelled by repeatedly applying the following instructions for each object type x:

$$\begin{aligned}
l_x &: (SUB(x_1^+), l_{x2}, abort) \\
l_{x2} &: (SUB(x_1^-), l_{x3}, abort)
\end{aligned}$$

where the instructions at label l_{x3} nondeterministically guess whether one of the two registers x_1^+ and x_1^- have reached zero (and thus the difference has been computed correctly) and, if so, the computation proceeds with the following object type instead of repeating the previous two instructions again.

When all pairs of object registers have been levelled, the register machine guesses, for each object type x, whether the final multiplicity is nonnegative (resp., negative) and copies the value of register x_1^+ (resp., x_1^-) into the output register $output_{x^+}$ (resp., $output_{x^-}$). This is, once again, performed by repeated decrement and increment instructions until nondeterministically guessing that the source register is empty.

This way the machine reaches the halt instruction and succeeds if and only if the P system has stopped and its simulation has been performed correctly; a simulation error is detected either when decrementing a register below zero, or when some registers x_1^+ or x_1^- are nonempty in the final configuration of the register machine.

It is well known that $PBRG \subset RE$, where RE denotes the recursively enumerable languages. Hence, an immediate consequence of Theorem 1 is that hybrid P systems are strictly less powerful than Turing machines.

Corollary 1. *$NHP \subset RE$.*

4 Conclusion

In this paper we introduced a new type of P system, which because of its different nature required new definitions. However, there are other variants that can be similarly conceived.

The first variant is a P system where we allow rules to use objects with negative multiplicities; that is, we allow rules like $u^{-1} \to bc^{-1}$. We do not, however, allow the application of any rules when the left-hand objects are not present (like we do in first-type systems). We conjecture that since their rules cannot be applied without the required left-hand objects, their power is not weakened by the use of hybrid sets. The main problem with systems introduced in this paper is that we can use rules at any time, and that makes us lose control over when to stop using a rule.

The second variant is a combination of the other two: in these systems, rules can deal with objects with negative multiplicities and can also be applied at any time. To limit the use of a certain rule, catalysts need to be introduced (or alternatively, one may think of a different way to prevent the computation from never ending). It is unclear how powerful this device would be, and we leave all possible paths open.

References

1. Carette, J., Sexton, A.P., Sorge, V., Watt, S.M.: Symbolic domain decomposition. In: Autexier, S., Calmet, J., Delahaye, D., Ion, P.D.F., Rideau, L., Rioboo, R., Sexton, A.P. (eds.) Intelligent Computer Mathematics, 10th International Conference, AISC 2010. Lecture Notes in Computer Science, vol. 6167, pp. 172–188. Springer (2010)
2. Freund, R., Ibarra, O.H., Păun, Gh., Yen, H.C.: Matrix languages, register machines, vector addition systems. In: Gutiérrez-Naranjo, M.A., Riscos-Núñez, A., Romero-Campero, F.J., Sburlan, D. (eds.) Third Brainstorming Week on Membrane Computing. Fénix Editora (2005)
3. Freund, R., Ivanov, S., Verlan, S.: P systems with generalized multisets over totally ordered Abelian groups. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) Membrane Computing, 16th International Conference, CMC 2015. Lecture Notes in Computer Science, vol. 9504, pp. 117–136. Springer (2015)
4. Greibach, S.A.: Remarks on blind and partially blind one-way multicounter machines. Theoretical Computer Science 7, 311–324 (1978)
5. Păun, Gh.: P systems with active membranes: Attacking NP-complete problems. Journal of Automata, Languages and Combinatorics 6(1), 75–90 (2001)
6. Păun, Gh.: Membrane Computing: An Introduction. Springer (2002)