Characterizing the Computational Power of Energy-Based P Systems

Artiom Alhazov[1,2], Marco Antoniotti[1], Alberto Leporati[1]

[1] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
{artiom.alhazov,marco.antoniotti,alberto.leporati}@unimib.it

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

Summary. We investigate the computational power of energy-based P systems, a model of membrane systems where a fixed amount of energy is associated with each object and the rules transform single objects by adding or removing energy from them. We answer recently proposed open questions about the power of such systems without priorities associated to the rules, for both sequential and maximally parallel modes. We also conjecture that deterministic energy-based P systems are not computationally complete.

1 Introduction

Membrane systems (also called P systems) have been introduced in [11] as a class of distributed and parallel computing devices, inspired by the structure and functioning of living cells. Since then, many variants of P systems have been defined in the literature. In what follows we assume the reader is familiar with the basic notions and the terminology underlying P systems. A systematic introduction to the area can be found in [12]; a recent overview of the developments is presented in [13], whereas the latest information can be found in [15].

In this paper we consider *energy-based P systems* [7, 8, 6], a model of computation in the framework of Membrane Computing in which a given amount of energy is associated to each object, and the energy manipulated during computations is taken into account by means of conservative rules.

Let us note in passing that there has been other attempts in the literature to incorporate certain conservation laws in membrane computing. One is purely communicative models, of which the most thoroughly studied is P systems with symport/antiport [10]. In these systems the computation is carried out by moving objects between the regions in groups. To reach computational completeness, the workspace is increased by bringing (some types of) objects from the environment, where they can be found in an unbounded supply. Another model is *conformon P systems* [5], where computations are performed by redistributing energy between objects, than can also be renamed and moved. A feature of these systems is that a different amount of energy may be embedded in the same object at different time steps. Yet another approach is to assign energy to membranes, as in P system with Unit Rules and Energy assigned to Membranes (*UREM P systems*, for short) [1]. Here the computations are performed by rules renaming and moving an object across a membrane, possibly modifying the energy assigned to that membrane. It has been proved in [1] that UREM P systems working in the sequential mode characterize $PsMAT$, the family of Parikh sets generated by matrix grammars without appearance checking (and with *erasing rules*), and that their power is increased to $PsRE$ (the family of recursively enumerable Parikh sets) if priorities are assigned to the rules or the mode of applying the rules is changed to maximally parallel.

As stated above, in this paper we consider *energy-based P systems*, in which energy is assigned to objects in a way that each object from the alphabet is assigned a specific value. Instances of a special symbol are used to denote *free energy units* occurring inside the regions of the system. The computations are carried out by rules renaming and possibly moving objects, which may consume or release free energy in the region, respecting the energy conservation law (that is, the total amount of energy associated with the objects that appear in the left hand side of a rule is the same as the energy occurring in the right hand side). The result of a computation may be interpreted in many ways: for example, as the amount of free energy units in a designated output region. Also for this model, to give the possibility to reach computational completeness it is necessary (but not sufficient, as we will see) that there may be an unbounded amount of free energy in (at least one) specified region of the system. In [6] it is proved that energy-based P systems working in the sequential way and using a form of local priorities associated to the rules are computationally complete. Without priorities, their behavior can be simulated by vector addition systems, and hence are not universal. However, in [6] the precise characterization of the computational power of energy-based P systems without priorities is left as an open problem. A related open question was whether energy-based P systems can reach computational completeness by working in the maximally parallel mode, without priorities, as it happens with UREM P systems [1].

In this paper we answer these questions, by showing that the power of energy-based P systems containing an infinite amount of free energy and without priorities is exactly $PsMAT$ when working in the sequential mode, and $PsRE$ when working in the maximally parallel mode. Nonetheless we will end with another open question: what is the power of energy-based P systems under the restriction of determinism? We conjecture non-universality for this case.

The rest of the paper is structured as follows. The next section contains some mathematical preliminaries, to fix the notions, definitions and notations with which we will work. Section 3 contains our results concerning the characterization of the computational power of energy-based P systems working without priorities, either in the sequential or in the maximally parallel mode. Section 4 contains the conclusions and some discussion on the above mentioned open problem, concerning the computational power of deterministic energy-based P systems.

2 Preliminaries

We assume the reader to be familiar with the basics of formal languages; on this subject one may refer to, e.g., [14].

We denote by $\mathbb{N}$ the set of non-negative integers. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* - \{\lambda\}$ is denoted by V^+. By $\mid x \mid$ we denote the length of the word x over V. Let $\{a_1, \ldots, a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $\mid x \mid_{a_i}$; the *Parikh vector* associated with x with respect to $a_1, \ldots, a_n$ is $(\mid x \mid_{a_1}, \ldots, \mid x \mid_{a_n})$. The *Parikh image* of a language L over $\{a_1, \ldots, a_n\}$ is the set of all Parikh vectors of strings in L. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$. A finite multiset $\langle m_1, a_1\rangle \ldots \langle m_n, a_n\rangle$ with $m_i \in \mathbb{N}$, $1 \le i \le n$, is represented as any string x the Parikh vector of which with respect to $a_1, \ldots, a_n$ is $(m_1, \ldots, m_n)$. The family of recursively enumerable languages is denoted by RE, and the family of context-free languages by CF. The family of all recursively enumerable sets of k-dimensional vectors of non-negative integers can thus be denoted by $Ps(k)RE$. Since numbers can be seen as one-dimensional vectors, we can replace $Ps(1)$ by $\mathbb{N}$ in the notation, thus obtaining $\mathbb{N}RE$.

2.1 Matrix Grammars

A context-free *matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$ where N and T are sets of *non-terminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, M is a finite set of *matrices*, $M = \{m_i \mid 1 \le i \le n\}$, where the matrices m_i are sequences of the form $m_i = [m_{i,1}, \ldots, m_{i,n_i}]$, $n_i \ge 1$, $1 \le i \le n$, and the $m_{i,j}$, $1 \le j \le n_i$, $1 \le i \le n$, are context-free productions over (N, T). For $m_i = [m_{i,1}, \ldots, m_{i,n_i}]$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \le j \le n_i$, w_j is the result of the application of $m_{i,j}$ to w_{j-1}. The language generated by G is

$$\begin{aligned} L(G) = \{w \in T^* \mid\ & S \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w, \\ & w_j \in (N \cup T)^*,\ m_{i_j} \in M \ \text{ for } 1 \le j \le k, k \ge 1\}. \end{aligned}$$

According to the definitions given in [2], the last matrix can already finish with a terminal word without having applied the whole sequence of productions. The family of languages generated by matrix grammars without appearance checking is denoted by MAT. It is known that $PsCF \subset PsMAT \subset PsRE$. Further details about matrix grammars can be found in [2] and in [14].

2.2 Energy-based P systems

Let us now recall the definition of energy-based P systems as given in [8].

An *energy-based P system* of degree $m \geq 1$ is a tuple

$$\Pi = (A, \varepsilon, \mu, e, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{in}, i_{out})$$

where:

- A is a finite set of objects called the alphabet;
- $\varepsilon : A \to \mathbb{N}$ is a mapping that associates to each object $a \in A$ the value $\varepsilon(a)$ (also denoted by ε_a), which can be viewed as the "energy value of a". If $\varepsilon(a) = \ell$, we also say that object a *embeds* ℓ units of energy;
- μ is a description of a tree structure consisting of m membranes, injectively labeled with elements from the set $\{1, \ldots, m\}$;
- $e \notin A$ is a special symbol denoting one free energy unit;
- w_i, for $1 \leq i \leq m$, specifies multisets (over $A \cup \{e\}$) of objects initially present in region i. We will sometimes assume that the number of e's (but not of objects from A) in some regions of the system is unbounded;
- R_i, for $1 \leq i \leq m$, is a finite set of multiset rewriting rules over $A \cup \{e\}$ associated with region i. Rules can be of the following types:

$$ae^k \to (b, p) \qquad \text{and} \qquad b \to (a, p)e^k \tag{1}$$

  where $a, b \in A$, $p \in \{here, in(j), out \mid 1 \leq j \leq m\}$, and k is a non-negative integer. Rules satisfy the conservativeness condition $\varepsilon(a) + k = \varepsilon(b)$;
- $i_{in} \in \{1, 2, \ldots, m\}$ specifies the input region of Π;
- $i_{out} \in \{0, 1, \ldots, m\}$ specifies the output region of Π ($i_{out} = 0$ corresponds to the environment).

Remark 1. In the above definition we excluded rules of types $e \to (e, p)$, originally included in [8]. It is easy to see that this does not influence the computational power of energy-based P systems.

When a rule of the type $ae^k \to (b, p)$ is applied, the object a, in presence of k free energy units, is allowed to be transformed into object b (note that $\varepsilon_a + k = \varepsilon_b$, for the conservativeness condition). If $p =$ here, then the new object b remains in the same region; if $p =$ out, then b exits from the current membrane. Finally, if $p =$ in($name$), then b enters into the membrane labelled with $name$, which must be directly contained inside the current membrane in μ. The meaning of rule $b \to (a, p)e^k$, where k is a positive integer number, is similar: the object b is allowed to be transformed into object a by releasing k units of free energy (also here, $\varepsilon_b = \varepsilon_a + k$). As above, the new object a may optionally move one level up or down into the membrane structure. The k free energy units might then be used by another rule to produce "more energetic" objects from "less energetic" ones. When $k = 0$ the rule $ae^k \to (b, p)$, also written as $a \to (b, p)$, transforms the object a into the object b (note that in this case $\varepsilon_b = \varepsilon_a$) and moves it (if $p \neq$ here) upward or downward into the membrane hierarchy, without acquiring or releasing any free energy unit. A similar observation applies to rules $b \to (a, p)e^k$ when $k = 0$.

Rules can be applied either in the sequential or in the maximally parallel mode. In either cases, we assume that the execution of rules does not consume energy. When working in the *sequential mode*, at each computation step (a global clock is assumed) exactly one enabled rule is nondeterministically chosen and applied in the whole system. When working in the *maximally parallel mode*, instead, at each computation step in each region of the system a maximal multiset of applicable rules is selected, and then all those rules are applied in parallel. Here *maximal* means that no further rule is applicable to objects that are "idle", that is, not already used by some other rule. If two or more maximal sets of applicable rules exist, then one of them is nondeterministically chosen.

A configuration of Π is the tuple $(M_1, \ldots, M_m)$ of multisets (over $A \cup \{e\}$) of objects contained in each region of the system; $(w_1, \ldots, w_m)$ is the *initial* configuration. A configuration where no rule can be further applied on is said to be *final*. A computation is a sequence of transitions between configurations of Π, starting from the initial one. A computation is *successful* if and only if it reaches a final configuration or, in other words, it *halts*. The multiset $w_{i_{\text{in}}}$ of objects occurring inside the input membrane is the *input* for the computation, whereas the multiset of objects occurring inside the output membrane (or ejected from the skin, if $i_{\text{out}} = 0$) in the final configuration is the *output* of the computation. A non-halting computation produces no output. As an alternative, we can consider the Parikh vectors associated with the multisets, and see energy-based P systems as computing devices that transform (input) Parikh vectors to (output) Parikh vectors. We may also assume that energy-based P systems have $\alpha \geq 1$ input membranes and $\beta \geq 1$ output membranes, instead of one. This modification does not increase the computational power of energy-based P systems, since for any fixed value of $\alpha \geq 1$ (resp., $\beta \geq 1$), the set $\mathbb{N}^\alpha$ (resp., $\mathbb{N}^\beta$) is isomorphic to $\mathbb{N}$, as it is easily shown by using the well known Cantor mapping.

In what follows sometimes we will use energy-based P systems as generating devices: we will disregard the input membrane, and will consider the multisets (or Parikh vectors) produced in the output membrane at the end of the (halting) nondeterministic computations of the system. In particular, in the output multisets we will only count the number of free energy units contained in the β output regions in the final configuration. We will denote the family of β-dimensional vectors generated in this way by energy-based P systems with at most m membranes and unbounded energy by $Ps(\beta)OP_m(energy_*)$. The union of all these classes for β ranging through the set of all non-negative integers is denoted by $PsOP_m(energy_*)$. When $\beta = 1$, the class $Ps(\beta)OP_m(energy_*)$ will be written as

$\mathbb{N}OP_m(energy_*)$. In all cases we will replace the subscript m by $*$ if no bound is placed on the number of membranes. If instead of maximal parallelism we assume that the P system evolves sequentially, we will add the superscript *seq* to P in the notation.

Our results will thus be proved for energy-based P systems working as generating devices; however, the extension to the cases in which P systems are computing functions or are accepting multisets of objects (or Parikh vectors) will be straightforward.

2.3 Register machines

In what follows we will need to simulate register machines; here we briefly recall their definition and some of their computational properties. A *register machine* is a tuple $M = (m, B, l_0, l_h, P)$, where m is the number of registers, P is the set of instructions bijectively labeled by elements of B, $l_0 \in B$ is the initial label, and $l_h \in B$ is the final label. The instructions of M can be of the following forms:

- $l_1 : ADD(j, l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  Increase the value of register j by one, and nondeterministically jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : SUB(j, l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  If the value of register j is zero then jump to instruction l_3, otherwise decrease the value of register j by one and jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stop the execution of the register machine. Note that, without loss of generality, we may assume that this instruction always appears exactly once in P, with label l_h.

A register machine is *deterministic* if $l_2 = l_3$ in all its ADD instructions. A *configuration* of a register machine is described by the contents of each register and by the value of the program counter, that indicates the next instruction to be executed. Computations start by executing the first instruction of P (labelled with l_0), and terminate if and when they reach instruction l_h.

Register machines provide a simple universal computational model [9]. In particular, the results proved in [4] immediately lead to the following proposition.

Proposition 1. *For any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ there exists a deterministic $(\max\{\alpha, \beta\}+2)$-register machine M computing f in such a way that, when starting with $(n_1, \ldots, n_\alpha) \in \mathbb{N}^\alpha$ in registers 1 to α, and all other registers empty, M has computed $f(n_1, \ldots, n_\alpha) = (r_1, \ldots, r_\beta)$ if it halts in the final label l_h with registers 1 to β containing r_1 to r_β, and all other registers being empty. If the final label cannot be reached, then $f(n_1, \ldots, n_\alpha)$ remains undefined.*

Register machines can also be used as *accepting* or as *generating* devices. In accepting register machines, a vector of non-negative integers is accepted if and only if the register machine halts. The following Proposition is a direct consequence of Proposition 1.

Proposition 2. *For any recursively enumerable set $L \subseteq Ps\,(\alpha)\,RE$ of vectors of non-negative integers there exists a deterministic register machine M with $(\alpha+2)$ registers accepting L in such a way that, when starting with n_1 to n_α in registers 1 to α, M has accepted $(n_1, \ldots, n_\alpha) \in L$ if and only if it halts in the final label l_h with all registers being empty.*

To generate vectors of non-negative integers we have to use nondeterministic register machines. The following Proposition is also a direct consequence of Proposition 1.

Proposition 3. *For any recursively enumerable set $L \subseteq Ps\,(\beta)\,RE$ of vectors of non-negative integers there exists a nondeterministic register machine M with $(\beta+2)$ registers generating L in such a way that, when starting with all registers being empty, M has generated $(r_1, \ldots, r_\beta) \in L$ if it halts in the final label l_h with registers 1 to β containing r_1 to r_β, and all other registers being empty.*

A direct consequence of the results exposed in [9] is that in Propositions 1 and 3 we may assume without loss of generality that only ADD instructions are applied to the output registers. This fact will be used to decrease the number of membranes of energy-based P systems simulating register machines; in particular, for each output register one membrane will suffice, whereas to simulate the behaviour of the other registers we will need two membranes.

A register machine is called *partially blind* if performing a zero-test blocks the computation, thus leading to no result. We can reflect this situation by omitting l_3 from all SUB instructions. However, unless all non-output registers have value zero at halting, the result of a computation is discarded; note that this is an implicit final zero-test, imposed by the definition and not affecting the power of register machines in the general case. It is known [3] that partially blind register machines characterize $PsMAT$.

3 Characterizing the Power of Energy-based P Systems

In this section we characterize the computational power of energy-based P systems without priorities associated to the rules and with an unbounded amount of free energy units. As stated in the previous section, we use energy-based P systems as generating devices; the extension to the computing and accepting cases (concerning computational completeness) is easy to obtain.

We start with systems working in the sequential mode; with the next two theorems we prove that they characterize $PsMAT$. We first prove the inclusion $PsMAT \subseteq PsOP_*^{seq}(energy_*)$, obtained by simulating partially blind register machines.

Theorem 1. $PsOP_*^{seq}(energy_*) \supseteq PsMAT$.

Proof. Consider a partially blind register machine $M = (m, B, l_0, l_h, P)$, generating β-dimensional vectors, and assume that registers $\beta+1, \dots, m$ are empty in the final configuration of successful computations, corresponding to an implicit final zero-test.

We construct a corresponding energy-based P system Π_M, containing an infinite amount of free energy units in the skin, as follows:

$$\begin{aligned}
\Pi &= (A, \varepsilon, \mu, e, w_s, \dots, w_f, R_s, \dots, R_f), \text{ where} \\
A &= \{l, l', l'' \mid l \in B\} \cup \{t_j, T_j \mid \beta+1 \le j \le m\} \cup \{t, T, H, H'\}, \\
\varepsilon(l) &= 1,\ \varepsilon(l') = 2,\ \varepsilon(l'') = 0,\ l \in B, \\
\varepsilon(t_j) &= 0,\ \varepsilon(T_j) = 2,\ \beta+1 \le j \le m, \\
\varepsilon(t) &= 0,\ \varepsilon(T) = 1,\ \varepsilon(H) = 2m+1,\ \varepsilon(H') = 2\beta+2, \\
\mu &= [\ [\ ]_1 \cdots [\ ]_m\ [\ ]_f\ ]_s\ , \\
w_s &= l_0, \\
w_j &= \lambda,\ 1 \le j \le m, \\
w_f &= t_{\beta+1} \cdots t_m T, \\
R_s &= \{l_1 e \to (l_1', in(j)) \mid l_1 : ADD(j, l_2, l_3) \in P\} \\
&\quad \cup \{l_1 \to (l_1'', in(j))\, e \mid l_1 : SUB(j, l_2) \in P\} \\
&\quad \cup \{T \to (T, in(f)),\ l_h e^{2m} \to (H, in(f))\} \\
&\quad \cup \{T_j \to (t, in(j))\, e^2 \mid \beta+1 \le j \le m\}. \\
R_j &= \{l_1' \to (l_2, out)\, e,\ l_1' \to (l_3, out)\, e \mid l_1' : ADD(j, l_2, l_3) \in P\} \\
&\quad \cup \{l_1'' e \to (l_2, out) \mid l_1 : SUB(j, l_2) \in P\} \cup R_j',\ 1 \le j \le m, \\
R_j' &= \emptyset,\ 1 \le j \le \beta, \\
R_j' &= \{T \to te,\ te \to T\},\ \beta+1 \le j \le m, \\
R_f &= \{T \to te,\ te \to T,\ H \to H' e^{2(m-\beta)-1}\} \\
&\quad \cup \{t_j e^2 \to (T_j, out) \mid \beta+1 \le j \le m\}.
\end{aligned}$$

Note that if $\beta = m$, then we replace $H \to H'e^{-1}$ in R_f by $He \to H'$. The sets of rules R_j and R_j', $1 \le j \le m$, are both intended to be associated with region j, and hence should be joined. As explained below, the rules from R_j' are used to produce infinite $T \leftrightarrow te$ loops in the regions corresponding to non-output registers of M if such registers are nonempty when the computation halts.

The simulation consists of a few parts. Every object associated with an instruction label l_1 embeds 1 unit of energy. To simulate an increment instruction $l_1 : ADD(j, l_2, l_3)$, the corresponding object l_1 consumes 1 more unit of energy, enters the region associated with register j as l_1', releases e there, and returns to the skin either as the object l_2 or as l_3 (the choice being made in a nondeterministic way), indicating the next instruction of M to be simulated. To simulate a decrement instruction $l_1 : SUB(j, l_2)$, the corresponding object l_1 releases e in the skin and enters as l_1'' the region associated with register j. There it consumes 1 unit of energy, and returns to the skin as the object l_2 associated with the next instruction of M to be simulated. If the register was empty then this process is blocked.

Meanwhile, another process takes place in region f; the order of execution of the two processes is nondeterministic, but both must finish in order for the system to terminate the computation and produce a result. The aim of this latter process is indeed to make Π_M "compute" forever if the above simulation of the SUB instruction gets blocked when trying to decrement an empty register, so that no result is produced in this case. The process consists of object T cyclically releasing e and capturing it, generating a possibly infinite loop. The only way to stop the loop is to alter the free energy occurring in region m. This is done when the simulation of M is finished, leading to object H releasing $e^{2(m-\beta)-1}$. If $\beta = m$ this consumes 1 unit of energy, thus stopping the $T \leftrightarrow te$ loop. Otherwise it releases enough energy for objects t_j, $\beta + 1 \leq j \leq m$ to leave the region, and the last one of them stops the $T \leftrightarrow te$ loop. The objects t_j are used to ensure that registers $\beta + 1 \leq j \leq m$ are empty, otherwise causing a $T \leftrightarrow te$ loop in the corresponding region. □

The opposite inclusion, $PsOP_*^{seq}(energy_*) \subseteq PsMAT$, is proved by simulating energy-based P systems by matrix grammars.

Theorem 2. *$PsOP_*^{seq}(energy_*) \subseteq PsMAT$.*

Proof. Let Π be an energy-based P system containing an infinite amount of free energy units and applying the rules in the sequential mode. Each rule of Π can be simulated by a corresponding rewriting rule on multisets of object-region pairs, ignoring those pairs involving energy objects in the regions containing infinite free energy. Such a multiset rewriting rule can be written as a matrix, yielding a matrix grammar. Clearly no matrix can be applied if and only if no rule can be applied. Since matrix languages are closed under morphisms, when this happens we can apply a morphism that erases all object-region pairs except those involving free energy objects in the output regions. The resulting language belongs to MAT, and its Parikh mapping yields exactly $Ps(\Pi)$. □

By joining the two inclusions proved in Theorems 1 and 2 we obtain our characterization of $PsMAT$ by energy-based P systems working in the sequential mode with an unbounded amount of free energy:

Corollary 1. *$PsOP_*^{seq}(energy_*) = PsMAT$.*

Running energy-based P systems in the maximally parallel mode allows them to reach computational completeness without using priorities, as shown in the next theorem.

Theorem 3. *$Ps(\beta)RE = Ps(\beta)OP_{\beta+6}(energy_*)$ for all integers $\beta \geq 1$.*

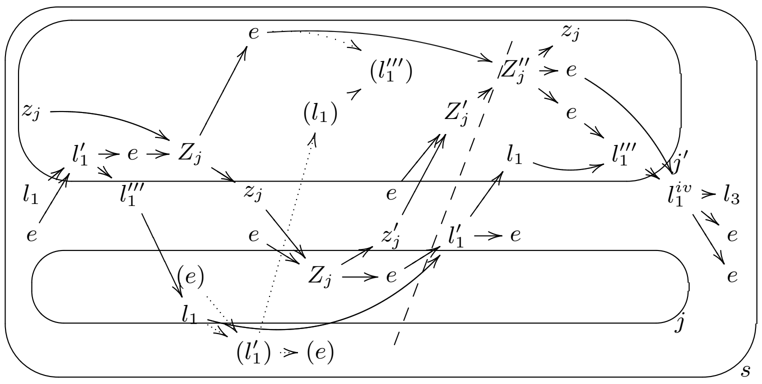

Fig. 1. Simulation of the zero-test of $l_1 : SUB(j, l_2, l_3)$. The case when the register is not zero is shown by dotted lines and symbols in parentheses, and the computation stops before the dashed line.

Proof. We start by noticing that the construction from Theorem 1 produces the same result when the P system works in the maximally parallel mode. Thus, it suffices to only add a simulation of zero-test instructions without disrupting the existing machinery. The new system nondeterministically chooses between decrement and zero-test, and blocks the simulation process if the zero-test fails.

We thus add membranes $[\ ]_{1'}, [\ ]_{2'}, \dots, [\ ]_{m'}$ and the following sets of rules to the energy-based P system Π_M mentioned in the proof of Theorem 1:

$$\begin{aligned}
R_s^0 = \{&1 : l_1 e \to (l_1', in(j')),\ 3a : l_1''' \to (l_1, in(j)),\\
&5a : l_1' \to (l_1, in(j'))\, e,\ 5b : z_j e \to (Z_j, in(j))\, e,\\
&7b : z_j' e \to (Z_j', in(j')),\ 12a : l_1^{(iv)} \to l_3 e^2\\
&\mid l_1 : SUB(j, l_2, l_3) \in P\},\\
R_j^0 = \{&4a : l_1 e \to (l_1', out),\ 6b : Z_j \to (z_j', out)\, e \in R_j''\\
&\mid l_1 : SUB(j, l_2, l_3) \in P\},\\
R_{j'}^0 = \{&2 : l_1' \to (l_1''', out)\, e,\ 3b : z_j e \to Z_j,\ 4b : Z_j \to (z_j, out),\\
&8b : Z_j' e \to Z_j'',\ 9b : Z_j'' \to z_j e^2,\ 10a : l_1 e \to l_1''',\\
&11a : l_1''' e \to (l_1^{(iv)}, out) \mid l_1 : SUB(j, l_2, l_3) \in P\}.
\end{aligned}$$

The case of correct simulation of a zero-test is illustrated in Figure 1. Indeed, if region j does not contain any object e, then the following sequence of multisets of rules is applied: $1, 2, (3a, 3b), (4a, 4b), (5a, 5b), 6b, 7b, 8b, 9b, 10a, 11a, 12a$. In this way, l_1 is transformed to l_3 while the other objects used (energy and z_j) are reproduced. On the other hand, if region j contains some object e, corresponding to a non-zero value of the corresponding register, then the sequence of multisets of rules applied is $1, 2, (3a, 3b), (4a, 4b), (5a, 5b), (6b, 10a), 7b$, and the simulation process is blocked. We recall that blocking the simulation process leads to an infinite computation due to the $T \leftrightarrow te$ loop in region f. In words, because of free energy in region j, object l_1 left that region three steps earlier. As a result, instead of object Z'_j consuming 1 unit of energy and then releasing 2 units needed for l_1, the existing unit of energy has been consumed by l_1, leaving the computation unfinished.

The full P system, defined using components of the construction from Theorem 1, is given below:

$$\begin{aligned}
\Pi'' &= (A'', \varepsilon'', \mu'', e, w''_s, \ldots, w''_f, R''_s, \ldots, R''_f), \text{ where}\\
A'' &= A \cup \{l''' \mid l \in B\} \cup \{z_j, z'_j, Z_j, Z'_j, Z''_j \mid 1 \le j \le m\},\\
\varepsilon''(x) &= \varepsilon(x),\ \forall\, x \in A,\ \varepsilon(l''') = 1,\ \varepsilon(l^{(iv)}) = 2,\ \forall\, l \in B,\\
\varepsilon''(z_j) &= \varepsilon''(z'_j) = 0,\ \varepsilon''(Z_j) = \varepsilon''(Z'_j) = 1,\ \varepsilon''(Z''_j) = 2,\ 1 \le j \le m,\\
\mu &= [\ [\ ]_1 \cdots [\ ]_m\ [\ ]_{1'} \cdots [\ ]_{m'}\ [\ ]_f\ ]_s\ ,\\
w''_s &= w_s,\ w''_j = w_j,\ 1 \le j \le m,\ w''_f = w_f,\\
w''_{j'} &= z_j,\ 1 \le j \le m,\\
R''_s &= R_s \cup R^0_s,\ R''_j = R_j \cup R^0_j,\ 1 \le j \le m,\\
R''_{j'} &= R^0_{j'},\ 1 \le j \le m,\ R''_f = R_f.
\end{aligned}$$

As we can see, system Π'' uses the skin membrane, one membrane to control the halting, and two membranes for each of the m registers, for a total of $2m + 2$ membranes.

Since it is known (see Proposition 3) that $m = \beta+2$ registers suffice to generate any recursively enumerable set $L \subseteq Ps(\beta)RE$ of vectors of non-negative integers by nondeterministic register machines, we would obtain $2\beta + 6$ membranes. However, as recalled above, when using register machines as generating devices we can assume without loss of generality that only ADD instructions are applied to the output registers. So the number of membranes needed to simulate M reduces to $\beta + 2(m - \beta) + 2 = \beta + 6$. □

By putting $\beta = 1$ in the above theorem we obtain a characterization of $\mathbb{N}RE$:

Corollary 2. *$\mathbb{N}OP_7(energy_*) = \mathbb{N}RE$.*

whereas if we make the union of all classes $Ps(\beta)OP_{\beta+6}(energy_*)$ for β ranging through the set of non-negative integers we obtain a characterization of $PsRE$:

Corollary 3. *$PsOP_*(energy_*) = PsRE$.*

As stated above, these results can be easily generalized to the cases in which energy-based P systems are used as accepting devices or as devices computing partial recursive functions. First of all note that the energy-based P systems built in the proofs of Theorems 1 and 3 can be easily modified to simulate deterministic register machines. Considering the computing case, we know from Proposition 1 that $m = \max\{\alpha, \beta\} + 2$ registers suffice to compute any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$. To simulate such a register machine we would obtain $2\max\{\alpha, \beta\}+6$

membranes for the system Π'' built in the proof of Theorem 3. However, this number can be reduced to $\alpha + \max\{\alpha, \beta\} + 6$ by considering that:

- as stated above, we can assume that only ADD instructions are applied to the output registers. This means that only one membrane (instead of two) is needed to simulate the behaviour of each output register;
- in general some input registers may also be used as output registers. However, any "primed" membrane j' associated with an input register, $1 \leq j' \leq \alpha$, cannot be used also as a membrane associated to an output register, due to the object z_j residing in the membrane. Hence, with α inputs and β outputs we need α primed membranes plus $\max\{\alpha, \beta\}$ non-primed membranes. By adding two membranes for each of the 2 additional registers of M, plus membranes f and s, we obtain $\alpha + \max\{\alpha, \beta\} + 6$ membranes.

As particular cases, we need $2\alpha + 6$ membranes for the accepting case and $\beta + 6$ membranes for the generating case.

4 Conclusions and Future Work

In this paper we have considered *energy-based P systems*, a model of membrane systems with energy assigned to objects. We have answered two questions about their computational power, and we have thus proved that it matches Parikh mapping of matrix languages when the rules of the P systems are applied in the sequential mode, whereas there is computational completeness in the maximally parallel case.

As a direction for future research, we propose the following problem: What is the computational power of *deterministic* energy-based P systems? We conjecture that they are not universal. The question originates from the fact that in [7, 8] energy-based P systems are used to simulate Fredkin gates and Fredkin circuits, respectively; however, the simulation is performed in a nondeterministic way, relying on the fact that sooner or later the simulation will choose the correct sequence of rules. Note that if the wrong rules are chosen the simulation is not aborted; the state of the system is "rolled back" so that a new nondeterministic choice can be made, hopefully the correct one. Clearly this situation could produce infinite loops; this is why one would like instead to have a *deterministic* simulation.

Here we can only give an *informal* justification for our conjecture. Notice that objects only interact indirectly, via releasing free energy units in a region or consuming them. Consider a dependency graph whose nodes are identified by object-region pairs. Two nodes are connected if the corresponding objects are present in the associated regions in some rule. A system is deterministic if no branching can be effectively used in its computations, so removing unusable rules would lead to a dependency graph where each node has out-degree at most one. Hence, any object occurring in the initial configuration of the system has some predetermined evolution path, and one of the following cases must happen:

- the path is finite, and the object evolves until there are no associated rules,

- the path leads to a cycle, and the object evolves forever (the computation yields no result),
- the evolution is "frozen" because there is not enough energy for the associated rule.

In energy-based P systems, the only way one object can influence the behavior of another object is by manipulating energy, leading to freezing or unfreezing the computational path of another object. There is no deterministic way to set an object to two different paths. If a "frozen" object receives enough energy to continue its evolution, then its computational path is the same as if it was never frozen.

So the information that can be passed from an object to another one is quite limited: giving the latter energy, as opposed to letting it freeze forever. However, every time this happens, some object must stop evolving forever. Since the initial number of objects is fixed and cannot increase, the communication complexity is bounded and this should imply non-universality.

However, even if deterministic energy-based P systems were not universal, they could nonetheless be able to simulate Fredkin gates. This should be doable if leaving some "garbage" into the system at the end of the computation is allowed. Indeed, the active objects could unfreeze the desired ones, producing the needed result. More difficult would be designing an energy-based P system that can be reused to simulate a Fredkin gate as many times as desired. We expect the reusable construction to be impossible, for the same reasons as exposed above.

Acknowledgements

The ideas exposed in this paper emerged during and after the *Tenth Brainstorming Week on Membrane Computing (BWMC 10)*, held in Seville in conjunction with the *International Conference on Developments in Membrane Computing*, from January 30th to February 3rd, 2012.

The first author gratefully acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618). The work of the other two authors was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca (FAR) 2011.

References

1. A. Alhazov, R. Freund, A. Leporati, M. Oswald, C. Zandron: (Tissue) P Systems with Unit Rules and Energy Assigned to Membranes. *Fundamenta Informaticae* **74**(4), 2006, 391–408.
2. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
3. R. Freund, O.H. Ibarra, Gh. Păun, H.-C. Yen: Matrix Languages, Register Machines, Vector Addition Systems. In: Proceedings of the *Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, 155–168. Available at: `http://www.gcn.us.es/3BWMC/bravolpdf/bravol155.pdf`

4. R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae* **49**(1-3), 2002, 81–102.
5. P. Frisco: The Conformon-P system: A Molecular and Cell Biology-Inspired Computability Model. *Theoretical Computer Science* **312**(2-3), 2004, 295–319.
6. A. Leporati, D. Besozzi, P. Cazzaniga, D. Pescini, C. Ferretti: Computing with Energy and Chemical Reactions. *Natural Computing* **9**, 2010, 493–512.
7. A. Leporati, C. Zandron, G. Mauri: Simulating the Fredkin Gate with Energy-Based P Systems. *J Univers Comput Sci* **10**(5), 2004, 600-619.
8. A. Leporati, C. Zandron, G. Mauri: Reversible P Systems to Simulate Fredkin Circuits. *Fundamenta Informaticae* **74**, 2006, 529–548.
9. M.L. Minsky: *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.
10. A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing* **20**(3), 2002, 295–306.
11. Gh. Păun: Computing with Membranes. *J Comput Syst Sci* **1**(61), 2000, 108-143. See also Turku Centre for Computer Science, *TUCS Report* No. **208**, 1998.
12. Gh. Păun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
13. Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.
14. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*, 3 vol., Springer, 1997.
15. P Systems Webpage. `http://www.ppage.psystems.eu/`