Computing with Genetic Gates, Proteins and Membranes

Nadia Busi[1] and Claudio Zandron[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A. Zamboni 7, I-40127 Bologna, Italy.
busi@cs.unibo.it
[2] Dipartimento di Informatica, Sistemistica e Comunicazione,
Università di Milano-Bicocca,
via Bicocca degli Arcimboldi 8, I-20126, Milano, Italy.
zandron@disco.unimib.it

Abstract. We introduce Genetic P systems, a class of P systems with evolution rules inspired by the functioning of the genes.
The creation of new objects – representing proteins – is driven by genetic gates: a new object is produced when all the activator objects are present, and no inhibitor object is available. Activator objects are not consumed by the application of such an evolution rule. Objects disappear because of degradation: each object is equipped with a lifetime; when such a lifetime expires, the object decays.
Then, we extend the basic model with *bind and release* rules and *repressor* rules, that simulate the action of protein channels and the action of substances which connect to other objects to block their use. We provide a universality result for such a class of systems.

1 Introduction

Membrane computing is a branch of natural computing, initiated by Gheorghe Păun with the definition of P systems in [3–5]. The aim is to provide a formal modeling of the structure and the functioning of the cell, making use especially of automata, languages and complexity theoretic tools.

Membrane systems (also called P systems) are based upon the notion of *membrane structure,* which is a structure composed by several cell-membranes, hierarchically embedded in a main membrane called the *skin membrane*. A plane representation of a membrane structure can be given by means of a Venn diagram, without intersected sets and with a unique superset. The membranes delimit *regions* and we associate with each region a set of *objects*, described by some symbols over an alphabet, and a set of *evolution rules*.

In the basic variant, the objects evolve according to the evolution rules, which can modify the objects to obtain new objects and send them outside the membrane or to an inner membrane. The evolution rules are applied in a maximally parallel manner: at each step, all the objects which can evolve should evolve.

A computation device is obtained: we start from an initial configuration, with a certain number of objects in certain membranes, and we let the system evolve. If a computation *halts,* that is no further evolution rule can be applied, the result of the computation is defined to be the number of objects in a specified membrane (or expelled through the skin membrane). If a computation never halts (i.e. one or more object can be rewritten forever), then it provides no output.

An up-to-date bibliography of the area and other useful resources can be found at [9].

The goal of this paper is to introduce systems which mimic the functioning of the genes. The relevance of such a subject has been recently pointed out in [6]. Genetic gates work in the following way: the production of a substance is the result of the activation of a gene, when certain substances (activators) are present while other substances (inhibitors) are absent. It is important to stress the fact that the production of the object does not require that one or more objects are consumed in order to do this. Nonetheless, objects can disappear due to a decay process. For this reason, objects are marked with a lifetime, which is decreased by one at each computation step. When this value becomes equal to zero, the object disappears.

We also consider rules to simulate the action of protein on membranes to communicate objects through protein channels, by defining *bind and release* rules, and the action of certain substances which act as repressor by connecting to other objects so to block their action, by defining *repressor* rules. We show that systems with all these types of rules are universal, and we point out various questions and investigation topics for further research.

The rest of the paper is organized as follows. In section 2 we give some basic definitions which will be used throughout the paper. In section 3 we define Genetic P systems and in section 4 we extend the the basic class with*Bind and Release* rules and repressor rules. In section 5 we provide an universality result for such an extended class of systems. Section 6 gives some conclusive remarks and presents various research topics.

2 Basic definitions

In this section we provide some basic definitions that will be used throughout the paper. We start with the definition of multisets and multiset operations.

Definition 1. *Given a set S, a* finite multiset *over S is a function $m : S \to \mathbb{N}$ such that the set $dom(m) = \{s \in S \mid m(s) \neq 0\}$ is finite. The* multiplicity *of an element s in m is given by the natural number $m(s)$. The set of all finite multisets over S, denoted by $\mathcal{M}_{fin}(S)$, is ranged over by m. A multiset m such that $dom(m) = \emptyset$ is called* empty*. The empty multiset is denoted by $\emptyset$.*

Given the multiset m and m', we write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$ while $\oplus$ denotes their multiset union*: $m \oplus m'(s) = m(s) + m'(s)$. The operator $\setminus$ denotes* multiset difference*: $(m \setminus m')(s) =$ if $m(s) \geq m'(s)$ then $m(s) - m'(s)$ else 0. The* scalar product*, $j \cdot m$, of a number j with m is $(j \cdot m)(s) = j \cdot (m(s))$.*

The cardinality of a multiset is the number of occurrences of elements contained in the multiset: $|m| = \sum_{s \in S} m(s)$.

The set of parts of a set S is defined as $\mathcal{P}(S) = \{X \mid X \subseteq S\}$.

Given a set $X \subseteq S$, with abuse of notation we use X to denote also the multiset

$$m_X(s) = \begin{cases} 1 & \text{if } s \in X \\ 0 & \text{otherwise} \end{cases}$$

The restriction to a subset of a multiset is defined as follows:

Definition 2. *Let m be a finite multiset over S and $X \subseteq S$. The multiset $m|_X$ is defined as follows: for all $s \in S$,*

$$m|_X(s) = \begin{cases} m(s) & \text{if } s \in X \\ 0 & \text{otherwise} \end{cases}$$

We provide some basic definitions on strings, cartesian products and relations.

Definition 3. *A string over S is a finite (possibly empty) sequence of elements in S. Given a string $u = x_1 \dots x_n$, the length of u is the number of occurrences of elements contained in u and is defined as follows: $|u| = n$. The empty string is denoted by λ.*

With S^* we denote the set of strings over S, and $u, v, w, \dots$ range over S. Given $n \geq 0$, with S^n we denote the set of strings of length n over S.

Given a string $u = x_1 \dots x_n$ and i such that $1 \leq i \leq n$, with $(u)_i$ we denote the i-th element of u, namely, $(u)_i = x_i$.

Given a string $u = x_1 \dots x_n$, the multiset corresponding to u is defined as follows: for all $s \in S$, $m_u(s) = |\{i \mid x_i = s \wedge 1 \leq i \leq n\}|$. With abuse of notation, we use u to denote also m_u.

Definition 4. *With $S \times T$ we denote the cartesian product of sets S and T, with $\times_n S$, $n \geq 1$, we denote the cartesian product of n copies of set S and with $\times_{i=1}^n S_i$ we denote the cartesian product of sets $S_1, \dots, S_n$, i.e., $S_1 \times \dots \times S_n$. The ith projection of $(x_1, \dots, x_n) \in \times_{i=1}^n S_i$ is defined as $\pi_i(x) = x_i$, and lifted to subsets $X \subseteq \times_{i=1}^n S_i$ as follows: $\pi_i(X) = \{\pi_i(x) \mid x \in X\}$.*

Given a binary relation R over a set S, with R^n we denote the composition of n instances or R, with R^+ we denote the transitive closure of R, and with R^* we denote the reflexive and transitive closure of R.

3 Genetic P systems

In this section, we present the definition of Genetic P systems and the definitions which we need to describe their functioning. To this aim, we start with the definition of *membrane structure*:

Definition 5. *Given the alphabet $V = \{[,]\}$, the set MS is the least set inductively defined by the following rules:*

- $[\ ] \in MS$
- *if $\mu_1, \mu_2, \ldots, \mu_n \in MS$, $n \geq 1$, then $[\mu_1 \ldots \mu_n] \in MS$*

We define the following relation over MS: $x \sim y$ if and only if the two strings can be written in the following form: $x = [_1 \ldots [_2 \ldots]_2 \ldots [_3 \ldots]_3 \ldots]_1$ and $y = [_1 \ldots [_3 \ldots]_3 \ldots [_2 \ldots]_2 \ldots]_1$ (i.e., if two pairs of parentheses that are neighbors can be swapped together with their contents).

The set $\overline{MS}$ of membrane structures *is defined as the set of equivalence classes w.r.t. the relation $\sim^*$.*

We say that i is the father *of j (and j is a* child *of i) if the membrane j is contained in i, and no membrane exists that contains j and is contained in i.*

The partial function $father : \{1, \ldots, d\} \to \{1, \ldots, d\}$ returns the father of a membrane i, or is undefined if i is the external membrane.

The function $children : \{1, \ldots, d\} \to \mathcal{P}(\{1, \ldots, d\})$ returns the set of children of a membrane.

We call a *membrane* each matching pair of parentheses appearing in the membrane structure. A membrane structure μ can be represented as a Venn diagram, in which any closed space (delimited by a membrane and by the membranes immediately inside) is called a *region* of μ.

We can give now the definition of Genetic P systems (or GP systems for short).

To this aim, given a set X, we define $\mathcal{R}_X = \mathcal{P}(X) \times \mathcal{P}(X) \times X$.

Definition 6. *A* Genetic P system with timed degradation *(of degree d, with $d \geq 1$) is a construct*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$$

where

1. *V is a finite alphabet whose elements are called* objects*;*
2. *μ is a* membrane structure *consisting of d membranes (usually labelled with i and represented by corresponding brackets $[_i$ and $]_i$, with $1 \leq i \leq d$);*
3. *w_i^0, $1 \leq i \leq d$, are strings over $V \times (I\!N \cup \infty)$ associated with the regions $1, 2, \ldots, d$ of μ; they represent multisets of objects of the form (a, t) present in the regions of μ, where a is a symbol of the alphabet V and $t > 0$ represents the decay time of that object. The multiplicity of a pair in a region is given by the number of occurrences of this pair in the string corresponding to that region;*
4. *R_i, $1 \leq i \leq d$, are finite multisets[3] of* genetic gates *over V associated with the regions $1, 2, \ldots, d$ of μ; these gates are of the forms $u_{act}, \neg u_{inh} :\rightarrow (b, t)$*

[3] Here we use multisets of rules, instead of sets, because each rule can be used at most once in each computational step.

where $u_{act} \cap u_{inh} = \emptyset$. $u_{act} \subseteq V$ *is the* positive regulation (activation)[4], $u_{inh} \subseteq V$ *is the* negative regulation (inhibition), $b \in V$ *is the transcription of the gate*[5] *and* $t \in I\!N \cup \infty$ *is the* duration *of object* b;

5. i_0 *is a number between* 1 *and* d *and it specifies the* output *membrane of* Π.

We say that a gate is *unary* if $|u_{act} \oplus u_{inh}| = 1$.

The membrane structure and the multisets represented by w_i, $1 \leq i \leq d$, in Π constitute the *initial state*[6] of the system. A transition between states is governed by an application of the transcriptions specified by the genetic gates which is done in parallel; all objects, from all membranes, which *can be* the subject of local evolution (that is, that can be used to apply the rule of a gate which is not used in the same step by other objects) *have to* evolve simultaneously.

The gate $u_{act}, \neg u_{inh} :\rightarrow (b, t)$ can be activated if the region it belongs to contains enough free activators and no free inhibitors. If the gate is activated, the regulation objects (activators) in the set u_{act} are bound to such a gate, and they cannot be used for activating any other gate in the same maximal parallelism evolution step. On the contrary, if one or more free inhibitor objects are present in the region where the gate is placed, then one of these objects (non-deterministically chosen) is bound to the gate, which cannot then be activated.

In other words, the gate $u_{act}, \neg u_{inh} :\rightarrow (b, t)$ in a region containing a multiset of (not yet bound) objects m can be activated if u_{act} is contained in m and no object in u_{inh} appears in m; if the gate performs the transcription, then a new object (b, t) is produced. Note that the objects in u_{act} and u_{inh} are not consumed by the transcription operation, but will be released at the end of the operation and (if they do not disappear because of the decay process) they can be used in the next maximal parallelism evolution step. Each object starts with a decay number, which specify the number of steps after which this object disappears. The decay number is decreased after each parallel step; when it reaches the value zero, the object disappears. If the decay number of an object is equal to ∞, then the object is persistent and it never disappears.

Note that the decay number associated to an object depends on the gate that produced the object (if the object is not present in the initial system), and not on the type of the object. Hence,a system may contain two gates, say, e.g. $a :\rightarrow (b, 5)$ and $a, \neg c :\rightarrow (b, \infty)$: the first gate produces one copy of object b that decays after 5 time units, whereas the second gate produces a persistent copy of object b.[7]

[4] We consider sets of activators, meaning that a genetic gate is never activated by more than one instance of the same protein.

[5] Usually the expression of a genetic gate consists of a single protein.

[6] Here we use the term *state* instead of the classical term *configuration* because we will define a (essentially equivalent but syntactically) different notion of configuration in section 5.

[7] We could also consider a variant of GP systems where the decaying time is a function of the type of the object, i.e., all the objects b that are produced in the system will have the same decaying time. We plan to deserve future investigation to this variant.

We adopt the following notation for gates. The activation and inhibition sets are denoted by one of the corresponding strings, i,e, $a, b, \neg c :\rightarrow (c, 5)$ denotes the gate $\{a, b\}, \neg\{c\} :\rightarrow (c, 5)$. If either the activation or the inhibition is empty then we omit the corresponding set, i.e., $a :\rightarrow (b, 3)$ is a shorthand for the gate $\{a\}, \neg\emptyset :\rightarrow (b, 3)$. The *nullary gate* $\emptyset, \neg\emptyset :\rightarrow (b, 2)$ is written as $:\rightarrow (b, 2)$.

3.1 Partial configurations, reaction relation and maximal parallelism step

Once defined genetic P systems, we are ready to describe their functioning. Hence, we give now the definitions for partial configuration, configuration, reaction relation, and heating and decaying function.

Definition 7. *Let $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$ be a genetic P system.*

A partial configuration *of Π is a tuple $(w_1, R_1, \bar{w}_1, \bar{R}_1), \ldots, (w_d, R_d, \bar{w}_d, \bar{R}_d) \in \times_d((V \times I\!N) \times \mathcal{R}_V \times (V \times I\!N) \times \mathcal{R}_V)$.*

We use $\times_{i=1}^{d}(w_i, R_i \bar{w}_i, \bar{R}_i)$ to denote the partial configuration above.

The set of partial configurations of Π is denoted by $Conf_\Pi$. We use γ, γ', γ_1, ... to range over $Conf_\Pi$.

$w_1, \ldots, w_d$ represent the active multisets, whereas $\bar{w}_1, \ldots, \bar{w}_d$ represent the frozen (already used) multisets, $R_1, \ldots, R_d$ represent the active gates, while $\bar{R}_1, \ldots, \bar{R}_d$ represent the frozen (already used) gates.

A *configuration* is a partial configuration containing no frozen objects; configurations represent the states reached after the execution of a maximal parallelism computation step.

Definition 8. *Let $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$ be a GP system.*

A configuration *of Π is a partial configuration $\times_{i=1}^{d}(w_i, R_i, \bar{w}_i, \bar{R}_i)$ satisfying the following: $\bar{w}_i = \emptyset$ and $\bar{R}_i = \emptyset$ for $i = 1, \ldots, d$.*

The initial configuration *of Π is the configuration $\times_{i=1}^{d}(w_i^0, R_i, \emptyset, \emptyset)$.*

The activation of a genetic gate is formalized by the notion of reaction relation. In order to give a formal definition we need the function $obj : (V \times I\!N)^* \rightarrow V^*$, defined as follows. Assume that $(a, t) \in (V \times (I\!N \cup \infty))$ and $w \subseteq (V \times (I\!N \cup \infty))^*$. Then, $obj(\lambda) = \lambda$ and $obj((a, t)w) = a\ obj(w)$.

We also need to define a function DecrTime which is used to decrement the decay time of objects, destroying the objects which reached their time limit.

Definition 9. *The function $DecrTime : (V \times I\!N)^* \rightarrow (V \times I\!N)^*$ is defined as follows:*

$DecrTime(\lambda) = \lambda$

and

$$DecrTime((a, t)w) = \begin{cases} (a, t-1)DecrTime(w) & \text{if } t > 1 \\ DecrTime(w) & \text{if } t = 1 \end{cases}$$

We are now ready to give the notion of *reaction relation.*

Definition 10. *Let* $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$ *be a GP system.*

The reaction relation $\mapsto$ *over* $Conf_\Pi \times Conf_\Pi$ *is defined as follows:*

$\times_{i=1}^d (w_i, R_i, \bar{w}_i, \bar{R}_i) \mapsto \times_{i=1}^d (w_i', R_i', \bar{w}_i', \bar{R}_i')$ *iff there exist* k, *with* $1 \leq k \leq d$ *and* $u_{act}, \neg u_{inh} :\rightarrow (b,t) \in R_k$ *such that*

- $R_k' = R_k \setminus (u_{act}, \neg u_{inh} :\rightarrow (b,t))$
- $\bar{R}_k' = \bar{R}_k \oplus (u_{act}, \neg u_{inh} :\rightarrow (b,t))$
- $\forall i : 1 \leq i \leq d$ *and* $i \neq k$ *implies* $w_i' = w_i$, $\bar{w}_i' = \bar{w}_i$, $R_i' = R_i$ *and* $\bar{R}_i' = \bar{R}_i$
- *if* $u_{inh} \cap dom(obj(w_k)) = \emptyset$ *and* $\exists w_{act} \subseteq w_k$ *such that*[8] $obj(w_{act}) = u_{act}$ *then*
  - $w_k' = w_k \setminus w_{act}$
  - $\bar{w}_k' = \bar{w}_k \oplus \{(b,t)\} \oplus DecrTime(w_{act})$
- *if* $\exists (s,t) \in dom(w_k)$ *such that* $s \in u_{inh}$ *then*
  - $w_k' = (w_k) \setminus (s,t)$
  - $\bar{w}_k' = \bar{w}_k \oplus DecrTime((s,t))$

Definition 11. *The function* $heat\&decay : Conf_\Pi \rightarrow \mathcal{P}(Conf_\Pi)$ *is then defined as follows:*

$heat\&decay(\times_{i=1}^d (w_i, R_i, \bar{w}_i, \bar{R}_i)) = \times_{i=1}^d ((DecrTime(w_i)) \oplus \bar{w}_i), R_i \oplus \bar{R}_i, \emptyset, \emptyset)$

Now we are ready to define the maximal parallelism computational step $\Rightarrow$:

Definition 12. *Let* $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, i_0)$ *be a GP system.*

The maximal parallelism computational step $\Rightarrow$ *over (nonpartial) configurations of* Π *is defined as follows:* $\gamma_1 \Rightarrow \gamma_2$ *iff there exists a partial configuration* γ' *such that* $\gamma_1 \mapsto^+ \gamma'$, $\gamma' \not\mapsto$ *and* $\gamma_2 = heat\&decay(\gamma')$.

4 Genetic P Systems with Bind&Release and Repressor Rules

The use of genetic gates alone is quite restrictive. For instance, no communication of objects is possible through the membranes, a feature which is fundamental in the basic variant of P systems. In fact, without communication the system would not act as a whole unit, but instead as a collection of separate systems or processes of various type, without interaction.

In order to enrich the model described so far, we consider also two other types of rules which mimic two different important cellular reactions.

The first type of rules we consider (Bind&Release), mimics the communication of objects through a protein channel. Two (multisets of) substances are *bound* to both side of a membrane. Then, by means of a channel in the membrane, they pass in opposite directions through the membrane itself, exchanging in this way their position. Finally, they can be *released* in their (new) region.

The second type of rules we consider (Repressor) mimics the action of certain substances that act to disactivate other substances present in the cell. When such a substance (a repressor) get in contact with another object, it creates a bond which cannot be destroyed. The object (and the repressor substance) cannot be

[8] The symbol = should be intended here as working on multisets.

used anymore for any other reaction. We notice that the repressor can create such a bond in any region of the cell. For this reason, the set of repressor rules will be valid for the whole system (i.e., we will not define different set of repressor rules for each region).

We provide the definition of Genetic P systems extended with Bind&Release and Repressor rules:

Given a set X, we define $\mathcal{R}_X = \mathcal{P}(X) \times \mathcal{P}(X) \times X$ and $\mathcal{BR}_X = \mathcal{P}(X) \times \mathcal{P}(X) \times \mathcal{P}(X) \times \mathcal{P}(X)$

Definition 13. *A* Genetic P system with timed degradation, Bind and Release actions and repressor rules *(of degree d, with $d \geq 1$), or* G^+P system *for short, is a construct*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$$

where

1. *V, μ, i_0, and w_i^0, R_i, for $1 \leq i \leq d$, are defined as in Definition 6.*
2. *BR_i, $1 \leq i \leq d$, are finite multisets of* Bind and Release rules *over V associated with the regions $1, 2, \ldots, d$ of μ; these rules are of the forms $u[v] \rightarrow v[u]$ where $u, v \in V^*$, and $|uv| > 0$. The* weight *of a Bind and Release rule $u[v] \rightarrow v[u]$ is $|u| + |v|$.*
3. *R_s is a finite multiset of* repressor rules*; these rules are associated with the system (and not to each region), and they are of the form $a, b \rightarrow a\&b$ where $a, b \in V$;*

Besides evolution driven by the application of transcriptions specified by genetic gates and object degradation, evolution steps in G^+P systems are also concerned with object migration through membranes and proteins repression.

Objects can be moved through membranes using bind and release operations. If outside a region i is present a multiset u of objects in $(V \times I\!N)$ and inside i a multiset v of objects in $(V \times I\!N)$, then a rule $u[v] \rightarrow v[u]$ in BR_i can be activated, moving the multisets u and v outside and inside region i, respectively.

Finally some objects can act as repressor objects, by means of repressor rules R_s. Such a rules of the form $a, b \rightarrow a\&b$ is activated when a repressor object b is present, thus binding to an object a and creating a new object which cannot be used anymore with any other rule.

4.1 Partial configurations, reaction relation and maximal parallelism step

As we did for the basic case, we give now the definitions for partial configuration, configuration, reaction relation, and heating and decaying function for G^+P systems.

Definition 14. *Let*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$$

be a G^+P system.

A partial configuration *of Π is a tuple*
$(w_1, R_1, BR_1, \bar{w}_1, \bar{R}_1, \overline{BR}_1), \ldots, (w_d, R_d, BR_d, \bar{w}_d, \bar{R}_d, \overline{BR}_d)$
$\in \times_d((V \times I\!N) \times \mathcal{R}_V \times \mathcal{B}R_V \times (V \times I\!N) \times \mathcal{R}_V \times \mathcal{B}R_V)$.

We use $\times_{i=1}^{d}(w_i, R_i, BR_i, \bar{w}_i, \bar{R}_i, \overline{BR}_i)$ to denote the partial configuration above. The set of partial configurations of Π is denoted by $Conf_\Pi$. We use $\gamma, \gamma', \gamma_1, \ldots$ to range over $Conf_\Pi$.

$w_1, \ldots, w_d$ represent the active multisets, $\bar{w}_1, \ldots, \bar{w}_d$ represent the frozen (already used) multisets, $R_1, \ldots, R_d$ represent the active gate rules, $\bar{R}_1, \ldots, \bar{R}_d$ represent the frozen (already used) gate rules, $BR_1, \ldots, BR_d$ represent the active Bind&Release rules, $\bar{R}_1, \ldots, \bar{R}_d$ represent the frozen (already used) Bind&Release rules.

A *configuration* is a partial configuration containing no frozen objects; configurations represent the states reached after the execution of a maximal parallelism computation step.

Definition 15. *Let*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$$

be a G^+P system.

A configuration *of Π is a partial configuration $\times_{i=1}^{d}(w_i, R_i, BR_i, \bar{w}_i, \bar{R}_i, \overline{BR}_i)$ satisfying the following: $\bar{w}_i = \emptyset$, $\bar{R}_i = \emptyset$ and $\overline{BR}_i = \emptyset$ for $i = 1, \ldots, d$.*

The initial configuration *of Π is the configuration $\times_{i=1}^{d}(w_i^0, R_i, BR_i, \emptyset, \emptyset, \emptyset)$.*

The activation of a genetic gate is formalized by the notion of reaction relation. In order to give a formal definition we need the function $obj : (V \times I\!N)^* \rightarrow V^*$, defined as follows. Assume that $(a, t) \in (V \times (I\!N \cup \infty))$ and $w \subseteq (V \times (I\!N \cup \infty))^*$. Then, $obj(\lambda) = \lambda$ and $obj((a, t)w) = a\ obj(w)$.

We also need to define a function DecrTime which is used to decrement the time index used objects, destroying the objects which reached their time limits. The function $DecrTime : (V \times I\!N)^* \rightarrow (V \times I\!N)^*$ is defined as follows:

Definition 16. $DecrTime(\lambda) = \lambda$
and

$$DecrTime((a, t)w) = \begin{cases} (a, t-1)DecrTime(w) & \text{if } t > 1 \\ DecrTime(w) & \text{if } t = 1 \end{cases}$$

We are now ready to give the notion of *reaction relation.*

Definition 17. *Let*

$$\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$$

be a G^+P system.

The reaction relation $\mapsto$ *over $Conf_\Pi \times Conf_\Pi$ is defined as follows:*

$\times_{i=1}^{d}(w_i, R_i, BR_i, \bar{w}_i, \bar{R}_i, \overline{BR}_i) \mapsto \times_{i=1}^{d}(w_i', R_i', BR_i', \bar{w}_i', \bar{R}_i', \overline{BR}_i')$ iff there exist k, with $1 \leq k \leq d$ and a rule R in $R_k \cup BR_k \cup R_s$ such that

CASE 1: IF $R : u_{act}, \neg u_{inh} :\rightarrow (b, t) \in R_k$, THEN

- $R'_k = R_k \setminus (u_{act}, \neg u_{inh} :\rightarrow (b,t))$
- $\bar{R}'_k = \bar{R}_k \oplus (u_{act}, \neg u_{inh} :\rightarrow (b,t))$
- $\forall i : 1 \leq i \leq d$ *and* $i \neq k$ *implies* $w'_i = w_i$, $\bar{w}'_i = \bar{w}_i$, $R'_i = R_i$ *and* $\bar{R}'_i = \bar{R}_i$
- *if* $u_{inh} \cap dom(obj(w_k)) = \emptyset$ *and* $\exists w_{act} \subseteq w_k$ *such that* $obj(w_{act}) = u_{act}$ *then*
  - $w'_k = w_k \setminus w_{act}$
  - $\bar{w}'_k = \bar{w}_k \oplus \{(b,t)\} \oplus DecrTime(w_{act})$
- *if* $\exists (s,t) \in dom(w_k)$ *such that* $s \in u_{inh}$ *then*
- $w'_k = (w_k) \setminus (s,t)$
- $\bar{w}'_k = \bar{w}_k \oplus DecrTime((s,t))$

CASE 2: IF $R : u[v] \rightarrow v[u] \in BR_k$ *THEN*

- $\exists U_{br} \subseteq w_{father(k)}$ *and* $\exists V_{br} \subseteq w_k$ *such that* $u = obj(U_{br})$ *and* $v = obj(V_{br})$
- $\forall i : 1 \leq i \leq d$, $i \neq k$ *and* $i \neq father(k)$ *implies* $w'_i = w_i$, $\bar{w}'_i = \bar{w}_i$, $BR'_i = BR_i$ *and* $\overline{BR}'_i = \overline{BR}_i$
- $BR'_{father(k)} = BR_{father(k)}$
- $\overline{BR}'_{father(k)} = \overline{BR}_{father(k)}$
- $w'_{father(k)} = w_{father(k)} \setminus U_{br}$
- $\bar{w}'_{father(k)} = \bar{w}_{father(k)} \oplus DecrTime(V_{br})$
- $BR'_k = BR_k \setminus (u[v] \rightarrow v[u])$
- $\overline{BR}'_k = \overline{BR}_k \oplus (u[v] \rightarrow v[u])$
- $w'_k = (w_k) \setminus V_{br}$
- $\bar{w}'_k = \bar{w}_k \oplus DecrTime(U_{br})$

CASE 3: $R : a, b \rightarrow a\&b \in R_s$

- $\exists (a,t_1), (b,t_2) \in (V \times I\!N)$ *and* $(a,t_1), (b,t_2) \in w_k$
- $\forall i : 1 \leq i \leq d, i \neq k$ $w'_i = w_i$ *and* $\bar{w}'_i = \bar{w}_i$
- $\forall i : 1 \leq i \leq d$, $R'_i = R_i$ *and* $\bar{R}'_i = \bar{R}'_i$
- $\forall i : 1 \leq i \leq d$, $BR'_i = BR_i$ *and* $\overline{BR}'_i = \overline{BR}'_i$
- $w'_k = w_k \setminus (a,t_1) \setminus (b,t_2)$
- $\bar{w}'_k = \bar{w}_k \oplus (a\&b, min(t_1,t_2))$

Definition 18. *The function* $heat\&decay : Conf_\Pi \rightarrow \mathcal{P}(Conf_\Pi)$ *is defined as follows:*

$heat\&decay(\times_{i=1}^{d}(w_i, R_i, BR_i, \bar{w}_i, \bar{R}_i, \overline{BR}_i)) =$
$\times_{i=1}^{d}(DecrTime(w_i) \oplus \bar{w}_i), R_i \oplus \bar{R}_i, BR_i \oplus \overline{BR}_i, \emptyset, \emptyset, \emptyset)$

Now we are ready to define the maximal parallelism computational step $\Rightarrow$:

Definition 19. *Let* $\Pi = (V, \mu, w_1^0, \ldots, w_d^0, R_1, \ldots, R_d, BR_1, \ldots, BR_d, R_s, i_0)$ *be a* G^+P *system.*

The maximal parallelism computational step $\Rightarrow$ *over (nonpartial) configurations of* Π *is defined as follows:* $\gamma_1 \Rightarrow \gamma_2$ *iff there exists a partial configuration* γ' *such that* $\gamma_1 \mapsto^+ \gamma'$, $\gamma' \not\mapsto$ *and* $\gamma_2 = heat\&decay(\gamma')$.

5 Turing equivalence of G^+P systems

In this section we show that G^+P systems with Bind and Release rules of weight one are Turing powerful. The result is proved by showing how to model Random Access Machines (RAMs) [8], a well known Turing powerful formalism.

We start recalling the definition of RAMs.

5.1 Random Access Machines

RAMs are a computational model based on finite programs acting on a finite set of registers. More precisely, a RAM R is composed of the registers $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a sequence of indexed instructions $(1 : I_1), \ldots, (m : I_m)$. In [2] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Succ(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;
- $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the first instruction and it continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached.

A state of a RAM is modelled by $(i, c_1, \ldots, c_n)$, where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current contents of the registers $r_1, \ldots, r_n$, respectively. We use the notation $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ to denote that the state of the RAM R changes from $(i, c_1, \ldots, c_n)$ to $(i', c'_1, \ldots, c'_n)$, as a consequence of the execution of the i-th instruction.

A state $(i, c_1, \ldots, c_n)$ is *terminated* if the program counter i is strictly greater than the number of instructions m. We say that a RAM R *terminates* if its computation reaches a terminated state. The *output* of the RAM is the contents of register r_1 in the terminated state of the RAM (if such a state exists).

5.2 Encoding RAMS in G^+P systems

In this section we show how to model RAMs in G^+P systems. Given a RAM with n registers, the system is composed by an external membrane, containing n children membranes, each one representing one register: $[_0[_1\ ]_1 \ldots [_n\ ]_n]_0$ (to simplify the notation, we label the external membrane with 0 instead of 1). The fact that register r_i contains value c_i is represented by the presence of c_i copies of object (r_i, ∞) in the membrane i. The instructions are encoded by genetic gates. The presence of object p_i in some part of the system represents the fact that the program counter contains the value i (i.e., the next instruction to be

executed is the ith). At the beginning of the computation, an object $(p_i, 2)$ is in the external membrane. All the objects representing the program counter will be produced with duration 2.

As the output of the RAM is the contents of register r_1 in the terminated state of the RAM, the output of the RAM encoding is the number of occurrences of object (r_1, ∞) in membrane 1.

Usually, when providing a RAM encoding of a P system, the output of the RAM encoding is taken in (one of the) halting configurations of the encoding. When considering GP systems, we note that it is not trivial to define what is a halting configuration. Take, e.g., the system with a negative gate $\neg a :\rightarrow (b, t)$, reaching a configuration containing only a persistent object (a, ∞): according to the reaction relation, this system never terminates. Actually, no real computation is performed, but what happens is that the inhibitor protein (a, ∞) is attacked to the negative regulation part of the gene.

Hence, here we adopt a different "termination" condition for GP systems, quite similar to the acceptance condition of automata with final states. Namely, we consider a computation to be successfully terminated if a configuration is reached which contains a distinguished, persistent object (end, ∞) in the external membrane. The definition of other suitable notions of termination for GP systems is left for future investigation.

We provide a RAM encoding which satisfies the following condition: the RAM terminates with output k if and only if the encoding of the RAM reaches a configuration containing the object (end, ∞) in the membrane 0, and containing exactly k occurrences of object (r_1, ∞) in membrane 1.

We consider RAMs that satisfy the following constraints:

1. If the RAM has m instructions, then all the jumps to addresses higher than m are jumps to the address $m + 1$.
2. The "self-loops" on DecJump instructions – i.e., instruction of the kind $(i : DecJump(r_j, i))$ – are forbidden.
3. The instruction following a DecJump (either if the decrement or if the jump is performed) is an increment.

Such constraints are not restrictive, as for any RAM not satisfying the constraints it is possible to construct an equivalent RAM (i.e., a RAM computing the same function) which satisfies the constraints above.

Consider a RAM with m instructions and n registers.

The first constraint can be easily satisfied by replacing each jump to an address higher than m to a jump to the address $m + 1$.

The second constraint can be satisfied by adding to the RAM a new register r_{n+1} that always contains the value zero, and by replacing each instruction $(i : DecJump(r_j, i))$ with a pair of instructions $(i : DecJump(r_j, i + 1))$ and $(i + 1 : DecJump(r_{n+1}, i)$. This means that the instructions following the ith instruction are shifted of one position. More in detail, for all $h : i + 1 \leq h \leq m$ we replace h with $h + 1$ in all the labels of the program, as well as in all the labels occurring in the jump instructions of the program.

The third constraint can be satisfied by adding a new register r_{n+2} – that will ever be incremented and never tested – and by replacing each instruction that can be reached after performing a $DecrJump$ instruction with the instruction $Succ(r_{n+2})$, and by shifting accordingly the other instructions.

If the RAM has m instructions, then the following gate belongs to membrane 0:

$p_{m+1} :\rightarrow (end, \infty)$

This rule permits to the system to signal termination when the instruction p_{m+1} is reached. (Actually, as we will see in the following, two instances of (end, ∞) are produced, but this is not a problem.)

If the ith instruction is $(i : Succ(r_j))$, then the following sequence of rules of membrane 0 is executed:

step 1: $p_i, \neg r_j :\rightarrow (r_j, \infty)$

step 2: $p_i, r_j, \neg p_{i+1} :\rightarrow (p_{i+1}, 2)$

step 3:$r_j[\ ]_j \rightarrow [r_j]_j$

If object r_j enters membrane j before the object p_{i+1} is created, no new program counter $i + 1$ will be created and the system will either stop in a failed computation or diverge without reaching a configuration with object end. As the program counters have duration equal to 2, at step 3 the object p_i decays.

If the i-th instruction is $(i : DecJump(r_j, s))$ then the following sequence of rules is executed:

step1: $p_i[\ ]_{l_j} :\rightarrow [p_i]_{l_j}$ (in membrane j)

If the contents of register r_j is zero (no occurrences of r_j in membrane j):

step 2: $p_i, \neg r_j :\rightarrow (p_s, 3)$ (in membrane j)

step 3: $[p_s]_j \rightarrow p_s\ [\ ]_j$ (in membrane j)

After step 2 the object p_i decays. If object p_i erroneously exits the membrane, then p_i decays just after exiting, and the system reaches a failed computation (or will diverge).

If the contents of register r_j is greater than zero:

step 2: $p_i, r_j, \neg dec_{i,j} :\rightarrow decr_{i,j}$ (in membrane j)

step 3: $r_j, decr_{i,j} \rightarrow (p_{i+1}, 3)$ (in membrane j)

step 4(1): $r_j, decr_{i,j} \rightarrow r_j \& decr_{i,j}$ (in membrane j)

step 4(2): $[p_{i+1}]_{l_j} \rightarrow p_{i+1}\ [\ ]_{l_j}$ (in membrane j)

After step 2 the object p_i decays. Steps 4(1) and 4(2) are executed in the same maximal parallelism step. If the rule at step 4(1) takes place before step 3 (i.e., the repressor bounds to r_j before that p_{i+1} is created), then no new program counter is created and the system reaches a failed configuration (or will diverge).

The formal definition of the encoding of a RAM R with m instructions and n registers, whose registers $r_1, \ldots, r_n$ contain values $c_1, \ldots, c_n$ is reported in Table 1.

If some of the registers contain a value greater than zero when the RAM terminates, then the system reaches a configuration containing the end object, but because of gates $p_i, \neg r_j :\rightarrow (p_s, 3)$ the system will never terminate. To obtain an encoding that guarantees that the configurations containing the end object

$\Pi(R) = (V, \mu, w_0^0, \ldots, w_d^0, R_0, \ldots, R_d, BR_0, \ldots, BR_d, R_s, i_0)$

$V = \{p_i \mid 1 \leq i \leq m+1\} \cup \{r_i \mid 1 \leq i \leq n\} \cup$
$\{decr_{i,j} \mid 1 \leq i \leq m+1 \wedge 1 \leq j \leq n\} \cup \{end\}$

$\mu = [_0[_1\ ]_1 \ldots [_n\ ]_n]_0$

$w_0^0 = (p_1, 2)$

$|w_j^0| = c_j$ and $(w_j^0)_i = (r_j, \infty)$ $j = 1, \ldots, n$ and $i = 1, \ldots, c_j$

$R_0 =$
$\{p_{m+1} :\rightarrow (end, \infty)\} \cup$
$\{p_i, r_j, \neg p_{i+1} :\rightarrow (p_{i+1}, 2) \mid$ the ith instr. is $(i : Succ(r_j)),\ i = 1, \ldots, m\} \cup$
$\{p_i, r_j, \neg p_{i+1} :\rightarrow (p_{i+1}, 2) \mid$ the ith instr. is $(i : Succ(r_j)),\ i = 1, \ldots, m\}$

$R_j =$
$\{p_i, \neg r_j :\rightarrow (p_s, 3) \mid$ the ith instr. is $(i : DecrJump(r_j, s))\} \cup$
$\{p_i, r_j, \neg dec_{i,j} :\rightarrow decr_{i,j} \mid$ the ith instr. is $(i : DecrJump(r_j, s))\} \cup$
$\{r_j, decr_{i,j} \rightarrow (p_{i+1}, 3) \mid$ the ith instr. is $(i : DecrJump(r_j, s))\}$

$BR_0 = \{r_j[\ ] \rightarrow [r_j] \mid 1 \leq j \leq n\}$

$BR_j =$
$\{r_j[\ ] \rightarrow [r_j]\} \cup$
$\{p_i[\ ] \rightarrow [p_i] \mid$ the ith instr. is $(i : DecrJump(r_j, s)),\ i = 1, \ldots, m\} \cup$
$\{[p_i] \rightarrow p_i[\ ] \mid$ the ith instr. is $(i : Succ(r_j)),\ i = 1, \ldots, m\}$

$R_s = \{r_j, decr_{i,j} \rightarrow r_j \& decr_{i,j} \mid 1 \leq j \leq n \wedge 1 \leq i \leq m+1\}$

$i_0 = 1$

Table 1. The G^+P system encoding a RAM R.

can perform no further computation, we could add to the RAM a further register r_{n+1}, that will never be decreased, and consider only RAMs that terminate with all registers empty but r_{n+1}, and the result is contained in register r_{n+1}. If we provide a slight variation of the encoding, where membrane $n+1$ contains no gates (as register r_{n+1} can only be increased), then the above requirement is fulfilled.

Another feature of the encoding is the fact that, if an erroneous action is performed, then the system can reach a failed configuration (i.e., a deadlocked configuration that does not contain the *end* object). It is possible to produce an encoding that diverges when an erroneous action is performed, by adding to the membrane 0 the gate $\neg end :\rightarrow loop$. However, in such a case, the configuration containing the *end* object is no longer terminated. A possible solution could be to signal termination by emitting the *end* object outside the external membrane.

In this section, we only use a restricted version of the Bind and Release rules, namely, rules with weight 1. We claim that, by using cooperative symport or antiport rules in combination with very simple genetic gates permitting to generate as many copies as you want of any object, Turing equivalence can be obtained as an easy consequence of the results recalled in [7].We stress the fact that we use Bind and Release rules of weight 1 to get universality, as symport rules of weight 2 (or alternatively antiport rules with one object entering the membrane and one object exiting the membrane) are already universal, without taking into account genetic gates.

We proved Turing equivalence of G^+P systems with Bind and Release rules of weight one and suppressor rules. We started some investigation on the expressiveness of more restricted versions of G^+P systems.

We conjecture that in G^+P systems with only positive gates and with Bind and Release rules of weight 1 (and without repressor rules) it is possible to decide if a system can reach a configuration containing a *end* object. This result could be proved by using the set saturation methods for well-structured transition systems defined in [1]. A consequence of this conjecture is the fact that such a class of systems is not Turing equivalent, according to the encoding rules defined above.

If we consider systems with both positive and negative gates and with persistent objects (i.e., objects with an infinite duration) only (and without bind and release rules and without repressor rules), we conjecture that the set of configurations of the system with the maximal parallelism rule is a finite state machine, hence most of the behavioural properties can be decided.

6 Conclusions

We have presented Genetic P systems, a new class of P systems where objects can be produced by means of evolution rules which are inspired from the functioning of the genes: a gene is activated (producing a new object), when certain substances (activators) are present while other substances (inhibitors) are absent.

We have also considered rules that mimic the action of proteins on membranes to communicate objects through protein channels, and rules simulating the action of repressor substances. We showed that systems with all these types of rules are universal.

Many investigations and research directions can be explored.

For instance, we can consider different kind of genetic gates, where more objects can be created at the same time by a single activation of the gate, or where the inhibition requires the presence of all inhibiting substances.

Also genetic gates where both inhibitors and activators can be attached to the gate at the same time can be considered.

For what concern the decaying process of the objects, we could also consider a non–deterministic decay process: at each parallel evolution step some objects are non–deterministically chosen to be eliminated from the set of objects in the system.

Various questions already investigated for "classic" P systems, could be investigated also for the systems defined in this paper, such as, for example, decidability, computational power, comparison with other formalism.

We also think that such a model would be very useful to be used in the systems biology area, to simulate various biological cell processes.

References

1. A. Finkel and Ph. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256:63–92, Elsevier, 2001.
2. M.L. Minsky. Computation: finite and infinite machines. Prentice-Hall, Englewood Cliffs, 1967.
3. G. Păun. Computing with membranes: an Introduction. *Bull. EATCS 67*, 1999.
4. G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
5. G. Păun. *Membrane Computing. An Introduction.* Springer, 2002.
6. G. Păun. 2006 Research Topics in Membrane Computing. *Proc. Fourth Brainstorming Week on Membrane Computing*, Felix Editora, Sevilla, 2006.
7. Y. Rogozhin, A. Alhazov, R. Freund, Computational Power of Symport/Antiport: History, Advances and Open Problems. *Proc 6th International Workshop on Membrane Computing (WMC6)*, LNCS 3850, Springer, 2006.
8. J.C. Shepherdson and J.E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.
9. P Systems webpage. http://psystems.disco.unimib.it.