Fundamenta Informaticae XX (2007) 1–11
IOS Press

A formalization of transition P systems

Mario J. Pérez-Jiménez

Fernando Sancho-Caparrini

Dpto. Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla, España

{*Mario.Perez,Fernando.Sancho*}*@cs.us.es*

Abstract. In this paper we give a complete formalization of a new computability model of a distributed parallel type which is inspired by some basic features of living cells: transition P systems as they were given in [3], addressed with completely different techniques than in [1] and [2]. For this, we present a formal syntax and semantic of the transition P systems capturing the synchronized work of P systems, and the nondeterministic and maximally parallel manner in which the rules of the system can be applied.

1. Introduction

In [3] a new computability model, of a distributed parallel type, based on the notion of *membrane structure*, is introduced. This model, called *transition P-system*, start from the observation that the processes which take place in the complex structure of a living cell can be considered *computations*. This model verifies interesting properties:

- It is in the framework of *Natural Computing*, a field of research which tries to imitate nature in the way it *computes*. In some way, it simulates the inner membrane structure of certain living organisms, where simultaneous and isolated operations that can be considered as computations, are produced.
- It is a parallel computational model, where parallelism works in two different levels: in a first level, inside every membrane, several operations can be made over different objects simultaneously, and in a second level, all the membranes works simultaneously, with no direct intervention of the produced operations of the other membranes.
- It is a *non deterministic* computational model, because, as we can see later, the rules to execute in each step of the execution are not determined *a priori*.

- It is an universal computational model. In [3] it is proved that P systems are able to compute all Turing computable sets of natural numbers (all recursively enumerable languages, in the case of string-objects).

The current bibliography of membrane computing can be found in [6].

The problem of a complete formalization of the definition of a P system of a given type and, mainly, of its computations and results of computations was explicitly formulated in [5]. A complete formalization of transition P systems is presented in this paper, addressed with completely different techniques than in [1] and [2]. We hope the formalization we present here can be useful for two initial aspects: (a) formal verification of these systems as a mechanical procedure of this new computability model, and (b) a possible implementation of these systems into conventional electronic computers, making an application allowing to define and execute them (with the loss of massive parallelism, not present in conventional computers).

2. A syntax for transition P systems

The *membrane structure* of a P system is a hierarchical arrangement of membranes (understood as vesicles in a space), embedded in a *skin membrane* that separates the system from the environment. When a membrane has not any membrane inside, it is called *elementary*. Each membrane encloses a space between it and the membranes directly included in it (if any). This space (the *region* of the membrane) can contain a multiset (a set where the elements can be repeated) of objects (represented by symbols of a given alphabet) and a set of (evolution) rules for them. Each membrane defines an unique region; that is, each region is delimited (*from the outside*) by an unique membrane.

2.1. Multisets

As a first step in our formalization, we need to define what multisets are, and a partial order and operations over them that will be useful along the paper.

Definition 2.1. A *multiset* over a set, A, is an application $m : A \longrightarrow \mathbf{N}$. The *support* of m is the subset of A: $supp(m) = \{a \in A : \ m(a) > 0\}$. A multiset is said to be empty (resp. finite) when its support is empty (resp. finite), and it will be denoted as $m = \emptyset$, or $m = \vec{0}$.

We will denote by $\mathbf{M}(A)$ the set of multisets over A (usually we denote it briefly by $\mathbf{M}$).

Definition 2.2. Given $m_1, m_2 \in \mathbf{M}$, we define:

- Inclusion: $m_1 \leq m_2 \Leftrightarrow \forall\, j\ (j \in A \rightarrow m_1(j) \leq m_2(j))$.
- Strict inclusion: $m_1 < m_2 \Leftrightarrow m_1 \leq m_2 \ \wedge\ m_1 \neq m_2$.
- Union: $+ : \mathbf{M}^2 \rightarrow \mathbf{M},\ (m_1 + m_2)(j) = m_1(j) + m_2(j)$.
- Difference: $- : \mathbf{M}^2 \rightarrow \mathbf{M},\ (m_1 - m_2)(j) = \max\{m_1(j) - m_2(j), 0\}$.
- Amplification: $\otimes : \mathbf{N} \times \mathbf{M} \rightarrow \mathbf{M},\ (n \otimes m_1)(j) = n \cdot m_1(j)$.

If m is a finite multiset over A, it will be usual to denote it as $m = \{\{a_1, \ldots, a_m\}\}$, where the elements a_i are possibly repeated. The *size* of the multiset m, denoted $|m|$, is defined as $|m| = \sum_{a \in A} m(a)$. That is, the size of a finite multiset is the sum of the multiplicities of the elements appearing in it (note that this sum has only a finite number of non null terms).

2.2. Membrane structures

Following [3], the membrane structures are defined by a language, MS, over the alphabet $\{[,]\}$, whose strings are defined by recursion as follows:

1. $[\,] \in MS$
2. If $\mu_1, \ldots, \mu_n \in MS$ (with $n \geq 1$), then $[\mu_1 \ldots \mu_n] \in MS$

In MS we consider the following relation:

$$x \sim y \leftrightarrow \exists\, \mu_1, \mu_2, \mu_3, \mu_4\ (\mu_2, \mu_3, \mu_1\mu_4 \in MS \wedge x = \mu_1\mu_2\mu_3\mu_4 \wedge y = \mu_1\mu_3\mu_2\mu_4)$$

Let $\sim^*$ be the reflexive and transitive closure of $\sim$, and let us denote $\overline{MS} = MS/\sim^*$. The elements of $\overline{MS}$ are called *membrane structures*. Each matching pair of parenthesis is called a *membrane*.

In a membrane structure with n elements, ordinarily the membranes are identified with a set of labels, usually the set $\{1, \ldots, n\}$.

To carry out our formalization, we need the following concepts about graphs.

An *undirected graph*, G, is an ordered pair, (V, E), where V is a finite set (its elements are called *vertices* or *nodes*) and E is a finite set consisting of unordered pairs of vertices, $\{u, v\}$, such that $u \neq v$. If $\{u, v\} \in E$ we say that vertices u, v are *adjacent*.

A *path of length* k from a vertex u to a vertex v in $G = (V, E)$ is a sequence $(x_0, x_1, \ldots, x_k)$ of vertices such that $u = x_0 \wedge v = x_k \wedge \forall\, j\ (0 \leq j < k \rightarrow \{x_j, x_{j+1}\} \in E)$. The length of the path is the number of edges in the path.

If there is a path from a vertex u to a vertex v, we say that v is *reachable* from u in G, and we denote it as $u \rightsquigarrow_G v$. Also, we say that u and v are *connected* in G.

An undirected graph is *connected* if every pair of different vertices are connected by a path in the graph.

A path is *simple* if all edges and all vertices on the path, except possibly the first and the last vertices, are distinct. A *cycle* is a simple path of length at least 1 which begins and ends at the same vertex. Note that in an undirected graph, a cycle must be of length at least 3. An undirected graph with no cycles is said *acyclic*.

A (*free*) *tree* is a connected, acyclic and undirected graph. A *rooted tree* is a tree with one of its vertices distinguished from the other ones. The distinguished vertex is called the *root* of the tree. Usually, we represent a rooted tree by an ordered pair such that the first component of the pair is the root of the tree and the second component is the adjacency list that consists of n list, one for each vertex i. The list for vertex i contains just those vertices adjacent from i.

Let G be a rooted tree with root r. Given a vertex u, any vertex v ($v \neq u$) on the unique path from r to u is called a *proper ancestor* of u. If v is a proper ancestor of u, then we say that u is a *proper descendant* of v. Any vertex v with no proper descendants is called a *leaf* of the rooted tree.

For every non root vertex, u, we will denote by $f(u)$ (the *father* of u in G) the only proper ancestor of u such that $\{f(u), u\} \in E$. The father of the root is undefined. For any vertex u we will denote $Ch(u)$ (the *children* of u in G) the set of proper descendants of u such that their father is u.

In a rooted tree $G = (V, E)$ with root r we can define a binary relation $E^*(G)$ on V, in a natural way, as follows: $(x, y) \in E^*(G) \Leftrightarrow y \in Ch(x)$. This binary relation establish, in some way, a direction over the edges of the rooted tree.

Definition 2.3. A *membrane structure* is a rooted tree, where the nodes are called *membranes*, the root is called *skin*, and the leaves are called *elementary membranes*.

In this formalization the labels of the membranes (as they appear in [3]) will be the vertices of the graph. In a more general version of P systems, for example, where *creation* is allowed, an independent label for the nodes can be useful.

To give the transition of the P system we will need the concept of a cell. That is, a membrane structure where a multiset (possibly empty) is associated with every membrane. The formalization of this concept is the following one:

Definition 2.4. A *cell* (or *super-cell*) over an alphabet, A, is a pair (μ, M), where μ is a membrane structure, and M is an application, $M : V(\mu) \longrightarrow \mathbf{M}(A)$, such that every node of the tree has an associated multiset over the base alphabet.

If $C = (\mu, M)$ is a cell over A, then we denote as $\mu = (V(\mu), E(\mu))$. That is, $V(\mu)$ and $E(\mu)$ are the set of vertices and edges, respectively, of the labeled rooted tree μ that determine the cell. We define the *degree* of a cell $C = (\mu, M)$, denoted $|C|$, as $|V(\mu)|$; that is, the total number of membranes.

Definition 2.5. Let (μ, M) be a cell over an alphabet, A. Let $x \in V(\mu)$. An (*evolution*) *rule* associated with x is a 3-tuple $r = (\vec{d}_r, \vec{v}_r, \delta_r)$ where

- $\vec{d}_r$ is a multiset over A.
- $\vec{v}_r$ is a function with domain $V(\mu) \cup \{here, out\}$, and range contained in $\mathbf{M}(A)$ where $here, out \notin V(\mu)$ and *here* $\neq$ *out*.
- $\delta_r \in \{\neg\delta, \delta\}$, with $\neg\delta, \delta \notin A$ and $\neg\delta \neq \delta$.

Informally, an evolution rule, r, for a membrane $x \in V(\mu)$, has the form:

$$a_1 \dots a_m \rightarrow (b_1, in_{j_1}) \dots (b_n, in_{j_n})(c_1, out) \dots (c_k, out)(d_1, here) \dots (d_l, here)s$$

with $s = \delta$ or $s = \lambda$ (the empty string).

In this formalization $\vec{d}_r = \{\{a_1, \dots, a_m\}\}$, $\vec{v}_r(y)$ $(y \in V(\mu))$ is the multiset consisting of the elements appearing in the rule under the form (b, in_y), $\vec{v}_r(out) = \{\{c_1, \dots, c_k\}\}$, and $\vec{v}_r(here) = \{\{d_1, \dots, d_l\}\}$. The component δ_r (δ or $\neg\delta$) represents if $s = \delta$ or $s = \lambda$. In the next section the action of the rule is explained in detail.

Finally, to give the syntax of the P system we need to assign to every membrane of the structure a set of rules that will be applied to the objects in the membrane. According to this, we define:

Definition 2.6. Let $C = (\mu, M)$ a cell over an alphabet A. Let $x \in V(\mu)$. A *collection* R of (evolution) *rules* associated with C is a function with domain $V(\mu)$ such that for every membrane $x \in V(\mu)$, $R(x) = \{r_1^x, \ldots, r_{s_x}^x\}$ (denoted R_x) is a finite set (possibly empty) of (evolution) rules associated with x.

Definition 2.7. Let $C = (\mu, M)$ a cell over an alphabet A. Let R be a collection of (evolution) rules associated with C. A *priority relation over* R is a function, ρ, with domain $V(\mu)$ such that for every membrane $x \in V(\mu)$, $\rho(x)$ (denoted ρ_x) is a strict partial order over R_x (possibly empty).

The strict partial order ρ_x over R_x will be interpreted as follows: $(r', r) \in \rho_x$ means that the rule r' has strict higher priority than the rule r.

Definition 2.8. A *transition P system* is a 4-tuple $\Pi = (A, C_0, \mathcal{R}, i_0)$, where:

- A is a non-empty finite set (usually called base alphabet).
- $C_0 = (\mu_0, M_0)$ is a cell over A.
- $\mathcal{R}$ is an ordered pair (R, ρ) where R is a collection of (evolution) rules associated with C_0, and ρ is a priority relation over R.
- i_0 is a node of the rooted tree μ_0, which specifies the output membrane of Π.

3. A semantic for transition P systems

The semantic of a P system, that is the way the P system evolves, is given informally as follows (we study the case of the priority is interpreted in a strong sense: if a rule with a higher priority is used, then no rule of a lower priority can be used, even if the two rules do not compete for objects).

P systems are synchronous, i.e., in each unit time of an assumed global clock a transformation of the system takes place by applying the rules in each membrane, in a nondeterministic and maximally parallel manner. This means that the objects and rules of each membrane are used in a nondeterministic and exhaustive way, such that, after the application of them, no rule can be applied in the same execution step (there are not enough objects for any rule to be applied). In the case we are studying, priority relations among rules of every membranes are given. This means that in each region a rule can be applied if no rule of a higher priority is applicable.

We can define the evolution of a P system formally using the notion of a *configuration*. The idea of a configuration is to represent the status of a computation of a P system. Basically, a P system is a membrane structure with objects in its membranes, with specified evolution rules for objects and with given input–output prescriptions. Such description of a P system only has as variable elements the membrane structure (because of dissolution) and objects contained within each one. So, to describe the status of a P system at some point during a computation, we only must specify the membrane structure of the P system and the contents of the regions associated with each one. Intuitively, a configuration contains a complete description of the current state of the computation: a hierarchical arrangement of membranes with the multisets of objects associated with each one. Configurations play a fundamental role in discussing P system computations.

Definition 3.1. A *configuration*, C, of a P system, $\Pi = (A, C_0, \mathcal{R}, i_0)$ with $C_0 = (\mu_0, M_0)$, is a cell $C = (\mu, M)$ over A, where $V(\mu) \subseteq V(\mu_0)$, and μ has the same root as μ_0. The configuration $C_0 = (\mu_0, M_0)$ will be called the *initial configuration of* Π.

Next we study how to decide if a rule can be applied or not. For this, we remember the informal meaning of an (evolution) rule, $r \in R_x$, in a membrane x such as

$$r : a_1 \dots a_m \to (b_1, in_{j_1}) \dots (b_n, in_{j_n})(c_1, out) \dots (c_k, out)(d_1, here) \dots (d_l, here)s$$

with a_i, b_i, c_i, d_i symbols of A and $s = \delta$ or $s = \lambda$.

- This rule can be applied only when there are enough copies of the objects $a_1, \dots, a_m$ in the membrane x and no rule of R_x with a higher priority can be applied.
- The result of using this rule is determined by its right hand side as follows:
  - The objects $d_1, \dots, d_l$ will remain in the same region x.
  - The objects $c_1, \dots, c_k$ will exit the membrane x and will become elements of the region immediately outside it (they pass to the father of x, and if x is the skin membrane, they are lost in the environment).
  - Every object b_s will be added to the multiset associated with the membrane j_s, provided that j_s is a child of the membrane x. If the membrane j_s is not a child of x, then the application of the rule is not allowed.
  - If the symbol δ appears in the rule ($s = \delta$) then the membrane x is removed (we say *dissolved*) and at the same time the set of the rules R_x (and its associated priority relation) will not be used any longer. The multiset associated with the membrane x is added to the region which is immediately external to it (its father). Of course, not such rule can be applied in the skin membrane.

Definition 3.2. Let $C = (\mu, M)$ a configuration of a P system $\Pi = (A, C_0, \mathcal{R}, i_0)$, with $C_0 = (\mu_0, M_0)$, and $x \in V(\mu_0)$. We say that the (evolution) rule $r \in R_x$ is *semi-applicable* to C if:

- The membrane associated with node x exists in C, that is, $x \in V(\mu)$.
- Dissolution is not allowed in root node, that is, if x is the root node of μ, then $\delta_r = \neg\delta$.
- The membrane associated with x has all the necessary objects to apply the rule, that is, $\vec{d_r} \leq M(x)$.
- Nodes where the rule tries to send objects (by means of in_y) are children of x, that is $\forall\, y \in V(\mu)(\vec{v}_r(y) \neq \vec{0} \to y \in Ch(x))$.

  (Note that rules trying to send objects to x by means of in_x are never semi-applicable).

But in our case, it is not enough to verify the above conditions to be an applicable rule, because nothing about priorities is considered. For that reason, we introduce the concept of applicable rule to a configuration of a P system.

Definition 3.3. Let $C = (\mu, M)$ a configuration of a P system $\Pi = (A, C_0, \mathcal{R}, i_0)$, with $C_0 = (\mu_0, M_0)$, and $x \in V(\mu_0)$. We say that the rule $r \in R_x$ is *applicable* to C, if it is semi-applicable to C and there is no semi-applicable rules in R_x with higher priority. That is:

$$\neg\exists\, r' \ (r' \in R_x \ \wedge\ \rho_x(r', r) \ \wedge\ r' \text{ semi-applicable to } C)$$

We define the *index of applicability of r in C over the node x* as the maximum number of times the rule r can be applied to C in the node x.

$$N_{Ap}(r,C,x) = \begin{cases} 0 & \text{, if } r \text{ is not applicable to } C, \\ \max\{n \ : n \otimes \vec{d}_r \leq M(x)\} & \text{, in other case.} \end{cases}$$

Moreover, an applicable rule may not to be applied, because of the non-determinism of the P system. Next, we define the applicability vector over a node, a numeric vector with as many components as rules in the node, and representing how many times each rule is applied to make the transition in this step.

Definition 3.4. Let $C = (\mu, M)$ a configuration of a P system Π. We will say that $\vec{p} \in \mathbf{N}^{\mathbf{N}}$ is an *applicability vector* over $x \in V(\mu)$ for C, and we will denote it as $\vec{p} \in \mathbf{Ap}(x, C)$, if:

- The node is still alive, that is, $\vec{p} \neq \vec{0} \Rightarrow x \in V(\mu)$.
- It has correct size, that is, $\forall\, j\ (j > s_x \rightarrow \vec{p}(j) = 0)$, (where s_x is the number of rules associated with x).
- Every rule can be applied as many times as the vector $\vec{p}$ indicates, that is,
$$\forall\, j\ (1 \leq j \leq s_x \rightarrow \vec{p}(j) \leq N_{Ap}(r_j^x, C, x))$$
- All the rules can be applied simultaneously, that is, $\displaystyle\sum_{j=1}^{s_x} \vec{p}(j) \otimes \vec{d}_{r_j^x} \leq M(x)$.
- It is maximal, that is, $\neg\exists\, \vec{v} \in \mathbf{N}^{\mathbf{N}}\ (\vec{p} < \vec{v} \wedge \vec{v} \in \mathbf{Ap}(x, C))$ (considering $\vec{p}, \vec{v}$ as infinite multisets over $\mathbf{N}$).

As we said above, the applications of the rules are simultaneous in every membrane of the structure. Because of this, we define applicability matrices, where rows will be applicability vectors over the nodes of P-system in this configuration. That is, an applicability matrix indicates how many times we must apply every rule in every membrane of the P system to get next configuration.

Definition 3.5. Let $C = (\mu, M)$ a configuration of a P system $\Pi = (A, C_0, \mathcal{R}, i_0)$ with $C_0 = (\mu_0, M_0)$. We will say that $P : V(\mu_0) \longrightarrow \mathbf{N}^{\mathbf{N}}$ is an *applicability matrix* over C, denoted $P \in \mathbf{M}_{\mathbf{Ap}}(C)$, if for every $x \in V(\mu_0)$ we have that $P(x) \in \mathbf{Ap}(x, C)$ (usually, we will denote P_x instead of $P(x)$).

If P is an applicability matrix over $C = (\mu, M)$ and $V(\mu) = \{i_1, \ldots, i_k\}$ with $i_1 < \cdots < i_k$, then we denote $P = ((p_1^{i_1}, \ldots, p_{s_{i_1}}^{i_1}), \ldots, (p_1^{i_k}, \ldots, p_{s_{i_k}}^{i_k}))$.

In order to determine how a P system evolves, we must formalize our intuitive understanding of the way that a P system *computes*. To determine a successor configuration of a given configuration, $C = (\mu, M)$, we proceed in two stages. In the first stage all the rules are executed without attending dissolving actions. In the second stage we remove dissolving membranes/nodes and distribute their contents to the region which is immediately external to them, and re-making the connections in the resulting rooted tree.

The first stage (application of rules without attending dissolving actions) produces the configuration $C'' = (\mu, M'')$, where:

$$M''(x) = M(x) - \sum_{j=1}^{s_x} P_x(j) \otimes \vec{d}_{r_j^x} + \sum_{j=1}^{s_x} P_x(j) \otimes \vec{v}_{r_j^x}(here) + \sum_{j=1}^{s_{f(x)}} P_{f(x)}(j) \otimes \vec{v}_{r_j^{f(x)}}(x) + \sum_{y \in Ch(x)} \sum_{j=1}^{s_y} P_y(j) \otimes \vec{v}_{r_j^y}(out)$$

To get the second stage (dissolving actions, and distribution of contents) we need to characterize the nodes to be dissolved by the execution of an applicability matrix.

Definition 3.6. If $C = (\mu, M)$ is a configuration of a P system Π, and $P \in \mathbf{M_{Ap}}(C)$ is an applicability matrix over C, we define $\Delta(P, C) = \{x : x \in V(\mu) \wedge \exists j \, (1 \leq j \leq s_x \wedge P_x(j) \neq 0 \wedge \delta_{r_j^x} = \delta)\}$.

That is, $\Delta(P, C)$ represents the set of nodes in the rooted tree determined by the configuration C that has to be dissolved after the application of the matrix P.

In one step of the transition P system, a set of membranes/nodes can be dissolved. Hence, it will be convenient to determine for every membrane, x, remaining in the next configuration, which is the set of dissolved membranes in the application of an applicability matrix that may give their contents to x (such membranes will be called *donors*) .

Definition 3.7. Let $C = (\mu, M)$ a configuration of a P system Π. Let $P \in \mathbf{M_{Ap}}(C)$ an applicability matrix over C. For each node $x \in V(\mu)$, we define the *donors* of x for C in the application of P as follows:

$$Don(x, P, C) = \begin{cases} \emptyset & \text{, if } x \in \Delta(P, C) \\ \{y \in V(\mu) : y \in \Delta(P, C) \wedge x \rightsquigarrow_\mu y \wedge \\ \wedge \, \forall z \in V(\mu)(x \rightsquigarrow_\mu z \rightsquigarrow_\mu y \rightarrow z \in \Delta(P, C))\} & \text{, if } x \notin \Delta(P, C) \end{cases}$$

That is, if a membrane x is not dissolved, then a membrane y is a donor for x if y is dissolved in the application of P, and every membrane being a proper descendant of x and a proper ancestor of y is also dissolved in the application of P.

Now, we can define the execution of an applicability matrix over a given configuration.

Definition 3.8. Let $C = (\mu, M)$ a configuration of a P system Π. Let $P \in \mathbf{M_{Ap}}(C)$ an applicability matrix over C. We define the *execution* of P over C, denoted $P(C)$, as the configuration of Π, $C' = (\mu', M')$, where:

- μ' is the rooted tree obtained from μ by means of:
  - $V(\mu') = V(\mu) - \Delta(P, C)$
  - If $x, y \in V(\mu')$, then:

$$(x, y) \in E^*(\mu') \Leftrightarrow \exists \, x_0, \ldots, x_n \in V(\mu)(x_1, \ldots, x_{n-1} \in \Delta(P, C) \wedge x_0 = x \wedge x_n = y \wedge \forall \, i \, (0 \leq i < n \rightarrow (x_i, x_{i+1}) \in E^*(\mu)))$$

- $M'(x) = \begin{cases} M''(x) \cup \bigcup_{y \in Don(x,P,C)} M''(y) & , \text{ if } x \notin \Delta(P,C) \\ \emptyset & , \text{ if } x \in \Delta(P,C) \end{cases}$

That is, μ' is the rooted tree obtained from μ erasing all the dissolved nodes by the application of matrix P, and restoring the connections in the right way: if a node y is dissolved, then the node is erased with all the edges adjacent to it, and every edge between the first non dissolved ancestor of y and every first non dissolved descendant of y are added. If a node is dissolved, its content is removed, if not, we must add to it the content of every donor node by the application of P.

Now we formalize the transition from a configuration of a P system to another configuration. That is, we try to precise how they compute, how the P systems evolve.

Definition 3.9. We will say that a configuration C_1 of a P system Π yields a configuration C_2 by a *transition in one step* of Π, denoted $C_1 \Rightarrow_\Pi C_2$, if there exists a non–zero applicability matrix over C_1, P, such that $P(C_1) = C_2$.

Intuitively, $C_1 \Rightarrow_\Pi C_2$ if, in one step of the P system Π, from the configuration C_1 results the configuration C_2.

Once we have defined the relationship of transition in one step among configurations, we can define the relation of *transition* to be its transitive closure. We say that a configuration C yields a configuration C' in k transition steps ($k \geq 0$), if there are configurations $C_1, \ldots, C_{k+1}$ such that

$$C_1 = C \wedge C_{k+1} = C' \wedge \forall i \; (1 \leq i \leq k \Rightarrow C_i \Rightarrow_\Pi C_{i+1})$$

Finally, we say that a configuration C yields a configuration C', denoted $C \stackrel{\star}{\Rightarrow}_\Pi C'$, if there is a $k \geq 0$ such that C yields C' in k transition steps.

With the fixed initial configuration , we can build the computation tree associated with the P system, such that the nodes of this tree are the configurations of the computation, and the edges are the applicability matrices used to get one configuration from another.

Definition 3.10. The *computation tree of a P system* Π, denoted $\mathbf{Comp}(\Pi)$, is a rooted labeled maximal tree defined as follows: the root of the tree is the initial configuration, C_0, of Π. The children of a node are the configurations that follow in one step of transition. Nodes and edges are labeled by configurations and applicability matrices, respectively, in such way that two labeled nodes C, C' are adjacent in $\mathbf{Comp}(\Pi)$, by means an edge labeled with P, if and only if $P \in \mathbf{M_{Ap}}(C) - \{\mathbf{0}\} \wedge C' = P(C)$.

The maximal branches of $\mathbf{Comp}(\Pi)$ will be called *computations* of Π. We will say that a computation of Π *halts* if it is a finite branch. The configurations verifying $\mathbf{M_{Ap}}(C) = \{\mathbf{0}\}$ will be called *halting configurations*, and we will denote the set of halting configurations as $Halt(\Pi) = \{C : \mathbf{M_{Ap}}(C) = \{\mathbf{0}\}\}$.

That is, a computation of a P system $\Pi = (A, C_0, \mathcal{R}, i_0)$ is a sequence (possibly infinite) of configurations $C_0 \Rightarrow_\Pi C_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_n$ (with $n \in \mathbf{N} \cup \{\infty\}$) such that C_0 is the initial configuration of Π and $\forall i \; (i < n \rightarrow C_i \Rightarrow_\Pi C_{i+1})$.

Definition 3.11. We say that a *computation* $C_0 \Rightarrow_\Pi C_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_n$ of a P system $\Pi = (A, C_0, \mathcal{R}, i_0)$ is *successful* if this computation halts, and i_0 is a leaf of the rooted tree μ_n, where $C_n = (\mu_n, M_n)$.

Then, we say that C_n is a *successful configuration* of Π and n is the *length* of the *computation*.

It's clear that we can define P systems where, for a concrete configuration, the applicability matrix for this configuration is not unique. In this moment of the computation, the system has several possibilities to go on, arising the non-determinism we noted in the introduction.

Definition 3.12. If $\mathbf{M_{Ap}}(C)$ is unitary for every configuration C of the P system, we will say that the system is *deterministic*. If not, we will say that the P system is *non deterministic*.

A non deterministic P system allows the possibility of more than one next configuration from a given configuration. At any moment of a computation these P systems may evolve according to several possibilities.

Next we will study the versatility of P systems. For this, we can see that a P system can be seen as a device which *generates* multisets, numbers or relations. It is also possible to interpret a P system as a device *recognizing* a multiset or even as a device *computing* a partial map from natural numbers to sets of natural numbers.

4. P systems generating multisets, numbers or relations

P systems can be seen as devices which generate multisets, numbers or relations. Indeed, let $\Pi = (A, C_0, \mathcal{R}, i_0)$ a P system. Let us consider the set of all successful configurations of Π:

$$S(\Pi) = \{C : C \text{ is the leaf of a successful computation of } \Pi\}$$

Definition 4.1. Let $\Pi = (A, C_0, \mathcal{R}, i_0)$ a P system. The *Parikh set generated* by Π, denoted $Ps(\Pi)$, is the following collection of multisets:

$$P_s(\Pi) = \{M_C(i_0) : C \in S(\Pi) \wedge C = (\mu_C, M_C)\}$$

That is, $P_s(\Pi)$ represents the collection of all multisets over A appearing in the output membrane, i_0, for some successful configuration of the P system Π.

Note: In [4] P systems with output to the *environment* are considered (the multiset of objects sent out of the system during the computation). The above formalization can be easily adapted to this new case. For that, it is enough to change definition 2.8, allowing, for example, output membrane to be $i_0 = env \notin V(\mu_0)$. We also must extend the concept of a cell, where the second component, M, must be an application from $V(\mu) \cup \{env\}$ to $\mathbf{M}$, and $M(env)$ will be interpreted as the content of the environment; also, we must remove the last condition in definition 3.11. Then,

$$P_s(\Pi) = \{M_C(env) : C \in S(\Pi) \wedge C = (\mu_C, M_C)\}$$

Definition 4.2. Let $\mathcal{C} \equiv C_0 \Rightarrow_\Pi C_1 \Rightarrow_\Pi \ldots \Rightarrow_\Pi C_n$ a successful computation of a P system $\Pi = (A, C_0, \mathcal{R}, i_0)$. The *numerical output* of $\mathcal{C}$, denoted by $O(\mathcal{C})$, is $|M_{C_n}(i_0)|$ where n is the length of $\mathcal{C}$ (denoted $|\mathcal{C}|$).

Definition 4.3. Let $\Pi = (A, C_0, \mathcal{R}, i_0)$ a P system. The *set of natural numbers generated* by Π, denoted $\mathbf{N}(\Pi)$, is defined as follows: $\mathbf{N}(\Pi) = \{|M_C(i_0)| : C \in S(\Pi) \wedge C = (\mu_C, M_C)\}$.

That is, $\mathbf{N}(\Pi) = \{O(\mathcal{C}) : \mathcal{C} \text{ is a successful computation of } \Pi\}$ represents the set of all the sizes of all multisets over A appearing in the output membrane, i_0, for some successful configuration of the P system Π.

Definition 4.4. Let $\Pi = (A, C_0, \mathcal{R}, i_0)$ a P system. Let $a_1, \ldots, a_k \in A$ $(k \geq 2)$ pairwise distinct. The k–relation generated by Π associated with $(a_1, \ldots, a_k)$, denoted $R_k(\Pi)$, is defined as follows:
$R_k(\Pi) = \{(n_1, \ldots, n_k) \in \mathbf{N}^k : \exists\, C \in S(\Pi)\ (C = (\mu_C, M_C) \wedge \forall j\ (1 \leq j \leq k \rightarrow$
$\rightarrow (M_C(i_0))(a_j) = n_j))\}$

That is, $R_k(\Pi)$ represents the collection of all k–tuples of natural numbers, $(n_1, \ldots, n_k)$, such that in the output membrane, i_0, of some successful configuration of the P system Π, the elements $(a_1, \ldots, a_k)$, have multiplicities $(n_1, \ldots, n_k)$ respectively.

5. Conclusions

In this paper we have presented a complete formalization for a new computability model which is inspired by some basic features of living cells: transition P systems, as they have been introduced in [3]. For that, a formal syntax and semantic of the P systems capturing the synchronous, in the sense that a global clock is assumed, and the nondeterministic and maximally parallel manner that the rules of the system can be applied, have been given. This formalization can be easily adapted to different variants of P systems that have appeared in recent works (cooperative, priority, rewriting, communication, carriers, porters, and so on).

We think that P systems formalization can represent an important contribution for the treatment of certain questions about this new computability model (programs's verification, validation of the model, etc.) by means of automatic reasoning systems. Also, we think that the formalization can be useful for a possible implementation into conventional electronic computers.

References

[1] A. V. Baranda, J. Castellanos, F. Arroyo, R. Gonzalo, Towards an electronic implementation of membrane computing: A formal description of nondeterministic evolution in transition P systems, *Proc. 7th Intern. Meeting on D NA Based Computers* (N. Jonoska, N.C. Seeman, eds.), Tampa, Florida, USA, 2001, 273–282.

[2] A. Obtulowicz, Membrane computing and one-way functions, *Intern. J. Found. Computer Sci.*, 12, 4 (2001), 551–558.

[3] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).

[4] Gh. Păun, G. Rozenberg, A guide to membrane computing, *Theoretical Computer Science*, to appear.

[5] Gh. Păun, Further research topics about P systems, *Pre-Proceedings of Workshop on Membrane Computing*, Curtea de Argeş, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 243–250.

[6] `http://bioinformatics.bio.disco.unimib.it/psystems`