Computing Partial Recursive Functions by Transition P Systems

Alvaro Romero-Jiménez and Mario J. Pérez-Jiménez

Abstract. In this paper a variant of transition P systems with external output designed to compute partial functions on natural numbers is presented. These P systems are stable under composition, iteration and unbounded minimization (μ–recursion) of functions. We prove that every partial recursive function can be computed by such P systems, from which the computational completeness of this model can be deduced.

1 Introduction

In 1998 G. Păun initiated a new branch of the field of *Natural Computing* by introducing a new model of molecular computation, based on the structure and functioning of the living cell: *transition P systems* (see [2]). The framework within which computations are performed in this model is the membrane struc-ture, which resembles the cell-like one. Multisets of symbol-objects are processed along the computations, making them to evolve and distributing them among the membranes. The result of a halting computation is the number of objects collected in a specified output membrane.

Since the introduction of this model of computation many variants of it have been proposed. One of them, presented in [4] by G. Păun, G. Rozenberg and A. Salomaa, is the model of *transition P systems with external output*. In this model, the result of a halting computation is not collected in a fixed membrane of the membrane structure, but in the external environment associated with it. In this way, the output of a computation can be thought as a set of strings, instead of as a natural number, as occurred in the basic model.

P systems are usually considered as devices which generate numbers. Nevertheless, besides generating devices, they can also be thought as recognizing devices and as computing devices. These kinds of P systems have been studied in [6] and [8].

In this paper we work with computing P systems, but instead of the basic transition ones we consider those with external output. Thanks to the special functioning of these devices, we have been able to define, in a suitable manner, several operations between computing P systems with external output. More specifically, we have defined the following operations: composition, iteration and unbounded minimization (or μ–recursion). This has allowed us to prove the

computational completeness of these devices through the capability of computing any partial recursive function.

2 Transition P Systems with External Output

A *multiset* over a set A is an application $m : A \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. A subset $B \subseteq A$ can be identified with the multiset over A given by the application $m(a) = 1$, if $a \in B$, and $m(a) = 0$, if $a \notin B$. We denote by $\mathbf{M}(A)$ the set of all the multisets over A. Note that if A is a non-empty finite set, then $\mathbf{M}(A)$ is a countable set.

The *support* of $m \in \mathbf{M}(A)$ is the set $supp(m) = \{a \in A \mid m(a) > 0\}$. A multiset is said to be finite if its support is finite. Analogously, a multiset is said to be empty, and it is denoted by $m = \emptyset$, if its support is the empty set.

2.1 Syntax

The framework within which computations of a cellular computing system with membranes take place is a membrane structure. The latter can be thought as a hierarchically arranged collection of vesicles.

Definition 1. *A* membrane structure *is a rooted tree in which the nodes are called* membranes*, the root is called* skin*, and the leaves are called* elementary membranes.

The degree *of a membrane structure is the number of membranes it contains (that is, the number of nodes of the tree).*

The skin membrane of a membrane structure, to which we will generically refer using the meta–label *skin*, isolates the structure from what is known as the environment of the structure, to which we will refer with the meta–label *env*. In the variant of P systems that we are going to consider, it is in the environment where the output of the computations will be collected. This is why we must associate it in some way with the membrane structure.

Definition 2. *Let $\mu = (V(\mu), E(\mu))$ be a membrane structure. The* membrane structure with environment *associated with μ is the rooted tree $Ext(\mu)$ where: (a) $V(Ext(\mu)) = V(\mu) \cup \{env\}$; (b) $E(Ext(\mu)) = E(\mu) \cup \{\{env, skin\}\}$; and (c) the root of the tree is the node env.*

The new node is called the environment *of the structure μ.*

Observe that what we do is only adding a new node that represents the environment and that, therefore, is only adjacent to the skin, whereas the original membrane structure remains unchanged.

Next, we define what we understand by a transition P system with external output, describing the syntax and semantics of this computing model in an informal manner. Nevertheless, a formalization for transition P systems can be found in [5] and an improved one for those with external output can be found in [6].

Definition 3. *A* transition P system with external output *(and without input) is a tuple*

$$\Pi = \big(\Gamma, \Lambda, \#, \mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p)\big)$$

where:

- Γ *is the working alphabet of the system.*
- Λ *is the output alphabet, and it is contained in* Γ*.*
- $\#$ *is a distinguished element in* $\Gamma \setminus \Lambda$*.*
- μ_Π *is a membrane structure of degree* p*. We suppose that the membranes are labelled, in a one-to-one manner, from 1 to* p*.*
- $\mathcal{M}_i$ *is a multiset over* Γ *associated with membrane* i*, for each* $i = 1, \ldots, p$*.*
- R_i *is a finite set of (transition) evolution rules associated with membrane* i*, for each* $i = 1, \ldots, p$*.*
  An evolution rule over is a pair (u, v)*, usually written* $u \to v$*, where* u *is a string over* Γ *and* $v = v'$ *or* $v = v'\delta$*, where* v' *is a string over*

$$\Gamma \times \big(\{here, out\} \cup \{in_{mb} \mid mb \text{ is a membrane in } \mu_\Pi\}\big).$$

- ρ_i *is a strict partial order over* R_i*, for each* $i = 1, \ldots, p$*. Given* $(r_1, r_2) \in \rho_i$*, usually written* $r_1 > r_2$*, we will say that* r_1 *has higher priority than* r_2*.*

For such system with input *we also consider an input alphabet,* Σ *(such that* $\Sigma \subseteq \Gamma$ *and* $\# \in \Gamma \setminus (\Sigma \cup \Lambda)$*), and an input membrane, im.*

2.2 Semantics

Now we show in which way a transition P system with external output evolves according to the multisets of objects contained in each of the compartments of its membrane structure, as well as to the evolution rules associated with the membranes.

Definition 4. *Let* Π *be a transition P system with external output. A configuration of* Π *is a pair* $C = \big(Ext(\mu), M\big)$ *such that it verifies the following conditions:*

- $\mu = (V(\mu), E(\mu))$ *is a membrane structure.*
- $Ext(\mu)$ *is the membrane structure with environment associated with the structure* μ*.*
- *The set* $V(\mu)$ *of nodes of* μ *is a subset of* $V(\mu_\Pi)$*, and contains the root of* μ_Π*.*
- *The roots of both membrane structures coincide.*
- M *is a function with domain* $V(Ext(\mu))$ *and range contained in* $\mathbf{M}(\Gamma)$*.*

Notation. *We will denote by* $C = (\mu, M_{env}, M_{i_1}, \ldots, M_{i_q})$ *a configuration of* Π*, where* $V(\mu) = \{i_1, \ldots, i_q\}$*,* $M_{env} = M(env)$ *is the multiset associated with the environment of* μ *and* $M_{i_j} = M(i_j)$ *is the multiset associated with the membrane* i_j *of* μ*, for each* $j = 1, \ldots, q$*.*

When defining the configurations that specify the initial state of a P system (that is, its initial configurations) we must take into account whether the system has an input membrane or not.

Definition 5. *Let* $\Pi = \big(\Gamma, \Lambda, \#, \mu_{\Pi}, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p)\big)$ *be a transition P system with external output.*

- *If* Π *has no input membrane, then there exists an unique initial configuration of the system, namely*

$$C^0 = (\mu_{\Pi}, \emptyset, \mathcal{M}_1, \ldots, \mathcal{M}_p).$$

- *If* Π *has an input membrane, then there exists an initial configuration for each multiset* $m \in \mathbf{M}(\Sigma)$ *that can be introduced in the input membrane, namely,*

$$C^0(m) = (\mu_{\Pi}, \emptyset, \mathcal{M}_1, \ldots, \mathcal{M}_{im} + m, \ldots, \mathcal{M}_p).$$

We can pass, in a non-deterministic manner, from one configuration of Π to another configuration by applying to its multisets the evolution rules associated with their corresponding membranes. This is done as follows: given a rule $u \to v$ of a membrane i, the objects in u are removed from M_i; then, for every $(ob, out) \in v$ an object ob is put into the multiset associated with the parent membrane (or the environment if i is the skin membrane); for every $(ob, here) \in v$ an object ob is added to M_i; for every $(ob, in_j) \in v$ an object ob is added to M_j (if j is a child membrane of i; otherwise, the rule cannot be applied). Finally, if $\delta \in v$, then the membrane i is dissolved, that is, it is removed from the membrane structure (the objects associated with this membranes are collected by the parent membrane, and the rules are lost. The skin membrane cannot be dissolved). Moreover, the *priority relation* among the rules forbids the application of a rule if another one of higher priority can be applied.

Given two configurations, C and C', of Π, we say that C' is obtained from C in one transition step, and we write $C \Rightarrow_{\Pi} C'$, if we can pass from the first one to the second by using the evolution rules associated with the membranes appearing in the membrane structure of C in a parallel and maximal way, and for all the membranes at the same time. If no configuration can be derived from C by applying those evolution rules, then we say that it is a *halting configuration.*

Definition 6. *A* computation, $\mathcal{C}$, *of a P system is a (finite or infinite) sequence of configurations,* $\{C^i\}_{i<r}$, *where:*

- C^0 *is an initial configuration of the system.*
- $C^i \Rightarrow_{\Pi} C^{i+1}$, *for every* $i < r$.
- *Either* $r \in \mathbb{N}^+$ *and* C^{r-1} *is a halting configuration (*$\mathcal{C}$ *is then a* halting *computation performing* $r-1$ *steps), or* $r = \infty$ *(*$\mathcal{C}$ *is then* not halting*).*

Notation. *Let* Π *be a transition P system with external output and let* $\mathcal{C} = \{C^i\}_{i<r}$ *be a computation of* Π. *Then we denote* $C^i = \big(Ext(\mu^i), M^i\big)$.

The idea of a transition P system with external output consists of not taking into account what happens inside the system, but only focusing on what it expels to the external environment. It emerges then the question of determining when a computation halts, and it is here where the distinguished object # comes into play.

Definition 7. *We say that* $r = u_r \rightarrow v_r\delta_r$ *is a* halting indicator rule *if* $(\#, out) \in v_r$.

Definition 8. *We say that a transition P system with external output* Π *is* valid *if given a computation* $\mathcal{C} = \{C^i\}_{i<t}$ *of the system it is verified the following:*

- *If* $\mathcal{C}$ *is halting, then a halting indicator rule must be applied in the skin membrane, and only in the last step of the computation.*
- *If* $\mathcal{C}$ *is not halting, then no halting indicator rule is applied in the skin membrane in any step of the computation.*

In this way, the fact that a computation has halted or not is determined by the presence of an object # in the environment of the system.

Note that we have not defined what the output of a computation is. This is because we can consider different modes for transition P systems by only fixing a definition for an *Output* function over the set of the computations of the system (see [6]).

3 (Function) Computing P Systems

Given an order between the symbols of an alphabet, we can represent tuples of natural numbers by means of multisets over this alphabet with only focusing on the multiplicities of the symbols in the multiset. Thus, in the cellular computing systems with membranes defined below we will impose that both the input and the output alphabets are ordered. In this way, it makes sense to consider that the multisets received as input and obtained as output represent tuples of natural numbers.

Definition 9. *A* computing P system*,* Π*, of the order* (m, n) *is a cellular computing system with membranes that verifies the following properties:*

- Π *is a transition P system with external output and with input membrane.*
- *The input alphabet,* Σ*, of* Π *is an ordered alphabet with* m *elements. We denote it by* $\Sigma = \{a_1, \ldots, a_m\}$.
- *The output alphabet,* Λ*, of* Π *is an ordered alphabet with* n *elements. We denote it by* $\Lambda = \{b_1, \ldots, b_n\}$.
- *The output of a computation* $\mathcal{C} = \{C^i\}_{i<r}$ *is given by the following function:*

$$Output(\mathcal{C}) = \begin{cases} undefined, & \text{if } \mathcal{C} \text{ is not halting,} \\ \left(M_{env}^{r-1}(b_1), \ldots, M_{env}^{r-1}(b_n)\right), & \text{if } \mathcal{C} \text{ is a halting computation performing } r-1 \text{ steps.} \end{cases}$$

In this way, the output of a halting computation of Π is a tuple of n natural numbers.

According to the previous definition, in a computing P system every halting computation returns a tuple of natural numbers. However, for the same input data there can exist computations that are halting and others that are not halting. Furthermore, the output of two halting computations over the same input data do not have to be the same tuple. This does not happen for functions: given a tuple of natural numbers, either the function is undefined over that tuple, or it is defined and returns a single value. Therefore, we must impose that the systems we are going to work with capture these properties.

Definition 10. *A computing P system, Π, of order (m, n) is said to be* valid *if it verifies the following properties:*

- *Π is a valid transition P system with external output.*
- *Given an initial configuration, C, of Π, either no computation with initial configuration C is halting, or every computation with initial configuration C is halting.*
- *If $\mathcal{C}_1$ and $\mathcal{C}_2$ are two halting computations of Π with the same initial configuration, then $Output(\mathcal{C}_1) = Output(\mathcal{C}_2)$.*

Notation. *We will denote by $\mathcal{FC}^{m,n}$ the class of valid computing P systems of order (m, n). The class $\mathcal{FC}$ is the union of all the previous collections.*

The cellular computing systems with membranes belonging to the class $\mathcal{FC}$ allow us to compute partial functions between natural numbers, according to the following criterion.

Definition 11. *We say that a system $\Pi \in \mathcal{FC}^{m,n}$ computes the partial function $f : \mathbb{N}^m - \to \mathbb{N}^n$ if the following conditions are verified for each $(k_1, \ldots, k_m) \in \mathbb{N}^m$:*

- *f is defined over $(k_1, \ldots, k_m)$ if and only if there exists a halting computation of Π with the multiset $a_1^{k_1} \ldots a_m^{k_m}$ as input.*
- *If $\mathcal{C}$ is a halting computation of Π with the multiset $a_1^{k_1} \ldots a_m^{k_m}$ as output, then $Output(\mathcal{C}) = f(k_1, \ldots, k_m)$.*

From Definition 10, in the previous definition the expression "a computation" can be substituted by the expression "any computation".

4 Computational Completeness through Partial Recursive Functions

The purpose of this section is to point out that using valid computing P systems we are able to reproduce the behaviour of any partial recursive function. Indeed, we are going to design systems such that:

1. Compute the *basic or initial functions*: constant zero function, successor function and projection functions.
2. Compute the *composition* of functions, from systems computing the functions to compose.
3. Compute the *iteration* of functions, from a system computing the function to iterate.
4. Compute the *unbounded minimization* of functions, from a system computing the function to minimize.

Taking into account that the class of partial recursive functions coincides with the least class that contains the basic functions and is closed under composition, iteration and unbounded minimization (see [1]), it is then guaranteed that it is possible to construct cellular computing systems with membranes that compute any partial recursive function.

4.1 Basic or Initial Functions

We begin by describing computing P systems that allow us to compute the basic functions.

– The *constant zero function*, $\mathcal{O} : \mathbb{N} \to \mathbb{N}$, defined by $\mathcal{O}(k) = 0$, for every $k \in \mathbb{N}$, can be computed by the system

$$\Pi^{zero} = \big(\Sigma, \Gamma, \Lambda, \#, \mu_{\Pi^{zero}}, \mathcal{M}_1, (R_1, \rho_1), im\big),$$

where:

$$\begin{aligned} \Sigma = \{a\}, \qquad & \Gamma = \{a, b, \#\}, \qquad \Lambda = \{b\}, \\ \mu_{\Pi^{zero}} = [\,_1]_1, \qquad & \mathcal{M}_1 = \#, \qquad im = 1, \\ R_1 = \{\# \to (\#, out)\}, \qquad & \rho_1 = \emptyset. \end{aligned}$$

– The *successor function*, $\mathcal{S} : \mathbb{N} \to \mathbb{N}$, defined by $\mathcal{S}(k) = k+1$, for every $k \in \mathbb{N}$, can be computed by the system

$$\Pi^{suc} = \big(\Sigma, \Gamma, \Lambda, \#, \mu_{\Pi^{suc}}, \mathcal{M}_1, (R_1, \rho_1), im\big),$$

where:

$$\begin{aligned} \Sigma = \{a\}, \qquad & \Gamma = \{a, b, \#\}, \qquad \Lambda = \{b\}, \\ \mu_{\Pi^{suc}} = [\,_1]_1, \qquad & \mathcal{M}_1 = \#, \qquad im = 1, \\ R_1 = \{a \to (b, out),\ \# \to (b, out)(\#, out)\}, \qquad & \rho_1 = \emptyset. \end{aligned}$$

– The *projection functions*, $\Pi^n_j : \mathbb{N}^n \to \mathbb{N}$, with $n \geq 1$ and $1 \leq j \leq n$, defined by $\Pi^n_j(k_1, \ldots, k_n) = k_j$, for every $(k_1, \ldots, k_n) \in \mathbb{N}^n$, can be computed by the systems

$$\Pi^{proj}_{n,j} = \big(\Sigma, \Gamma, \Lambda, \#, \mu_{\Pi^{proj}_{n,j}}, \mathcal{M}_1, (R_1, \rho_1), im\big),$$

where:

$$\Sigma = \{a_1, \ldots, a_n\}, \qquad \Gamma = \{a_1, \ldots, a_n, b, \#\}, \qquad \Lambda = \{b\},$$
$$\mu_{\Pi^{proj}_{n,j}} = [\,_1]_1, \qquad \mathcal{M}_1 = \#, \qquad im = 1,$$
$$R_1 = \{a_j \to (b, out),\ \# \to (\#, out)\}, \qquad \rho_1 = \emptyset.$$

4.2 Composition of Functions

We introduce now the operation of composition between computing P systems. For that we start by defining the corresponding operation for functions.

Definition 12. *Let $f : \mathbb{N}^m - \to \mathbb{N}^n$ and $g_1 : \mathbb{N}^r - \to \mathbb{N}^{s_1}, \ldots, g_t : \mathbb{N}^r - \to \mathbb{N}^{s_t}$ such that $s_1 + \cdots + s_t = m$. Then, the composition of f with g_1 to g_t, denoted $C(f; g_1, \ldots, g_t)$, is a partial function from $\mathbb{N}^r$ to $\mathbb{N}^n$ defined as follows*

$$C(f; g_1, \ldots, g_t)(k_1, \ldots, k_r) = f(g_1(k_1, \ldots, k_r), \ldots, g_t(k_1, \ldots, k_r))$$

Next, we are going to design a P system that computes the composition of functions, from systems that compute the given functions. Let $\Pi_f, \Pi_{g_1}, \ldots, \Pi_{g_t} \in \mathcal{FC}$ be systems computing, respectively, the function $f : \mathbb{N}^m - \to \mathbb{N}^n$ and the functions $g_1 : \mathbb{N}^r - \to \mathbb{N}^{s_1}, \ldots, g_t : \mathbb{N}^r - \to \mathbb{N}^{s_t}$, with $s_1 + \cdots + s_t = m$.

We can suppose that

$$\Pi_f = \big(\Sigma_f, \Gamma_f, \Lambda_f, \#_f, \mu_{\Pi_f}, \mathcal{M}^f_1, \ldots, \mathcal{M}^f_{p_f}, (R^f_1, \rho^f_1), \ldots, (R^f_{p_f}, \rho^f_{p_f}), im_f\big),$$
$$\Pi_{g_1} = \big(\Sigma_{g_1}, \Gamma_{g_1}, \Lambda_{g_1}, \#_{g_1}, \mu_{\Pi_{g_1}}, \mathcal{M}^{g_1}_1, \ldots, \mathcal{M}^{g_1}_{p_{g_1}}, (R^{g_1}_1, \rho^{g_1}_1), \ldots, (R^{g_1}_{p_{g_1}}, \rho^{g_1}_{p_{g_1}}), im_{g_1}\big),$$
$$\ldots$$
$$\Pi_{g_t} = \big(\Sigma_{g_t}, \Gamma_{g_t}, \Lambda_{g_t}, \#_{g_t}, \mu_{\Pi_{g_t}}, \mathcal{M}^{g_t}_1, \ldots, \mathcal{M}^{g_t}_{p_{g_t}}, (R^{g_t}_1, \rho^{g_t}_1), \ldots, (R^{g_t}_{p_{g_t}}, \rho^{g_t}_{p_{g_t}}), im_{g_t}\big).$$

Renaming adequately the elements of the alphabets (and, consequently, also of the rules) we can suppose, besides, that

- $\Sigma_{g_1} = \cdots = \Sigma_{g_t} = \{a_1, \ldots, a_r\}$.
- $\Lambda_{g_1} = \{b_1, \ldots, b_{s_1}\}, \ldots, \Lambda_{g_t} = \{b_{s_1+\cdots+s_{t-1}+1}, \ldots, b_m\}$.
- $\Sigma_f = \{c_1, \ldots, c_m\}$ and $\Lambda_f = \{d_1, \ldots, d_n\}$.
- $\Big(\Lambda_{g_1} \cup \cdots \cup \Lambda_{g_t}\Big) \cap \Gamma_f = \emptyset$.
- The object $\#_{g_i}$ is distinct from the object $\#_f$, for each $i = 1, \ldots, t$.
- The object $\#_{g_i}$ is distinct from the object $\#_{g_j}$, for each $i \neq j$.

Let us consider the computing P system

$$\Pi = \big(\Sigma, \Gamma, \Lambda, \#, \mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p), im, env\big),$$

given by:

- $\Sigma = \{e_1, \ldots, e_r\}$. We suppose, besides, that Σ is disjoint from $\bigcup_{i=1}^t \Gamma_{g_i}$.

– There exist the distinguished elements $\oplus, \ominus, \odot \in \Gamma \setminus (\Gamma_f \cup \bigcup_{i=1}^{t} \Gamma_{g_i})$.
– $\Lambda = \{d_1, \ldots, d_n\}$.
– The object $\#$ is distinct from the objects $\#_f$ and $\#_{g_i}$, for each $i = 1, \ldots, t$.
– $\mu_\Pi = [_1 \mu_{\Pi_{g_1}} \ldots \mu_{\Pi_{g_t}} \mu_{\Pi_f}]_1$, where the membranes of $\mu_{\Pi_{g_1}}, \ldots, \mu_{\Pi_{g_t}}, \mu_{\Pi_f}$ have been adequately renamed (and, consequently, also the rules of the corresponding systems have been adapted). We will denote by $\sigma_{g_1}, \ldots, \sigma_{g_t}, \sigma_f$ the skin membranes of these systems. Besides, we consider that $im_{g_1}, \ldots, im_{g_t}$, im_f reflect the new labeling of the input membranes of $\Pi_{g_1}, \ldots, \Pi_{g_t}, \Pi_f$, respectively.
– $p = p_{g_1} + \cdots + p_{g_t} + p_f + 1$.
– $\mathcal{M}_1 = \#\ominus$. The remaining multisets are all empty.
– $im = 1$.
– The evolution rules and their priorities are the following:
  • Evolution rules for membrane 1:

$$
\begin{array}{l}
e_i \to (e_i, in_{\sigma_{g_1}}) \ldots (e_i, in_{\sigma_{g_t}}) \quad \text{for } i = 1, \ldots, r, \\
\ominus \to (\ominus, in_{\sigma_{g_1}}) \ldots (\ominus, in_{\sigma_{g_t}}), \\
\#_{g_1} \ldots \#_{g_t} \# \to (\ominus, in_{\sigma_f}) > \# \to \# > b_i \to (b_i, in_{\sigma_f}) \quad \text{for } i = 1, \ldots, m, \\
d_i \to (d_i, out) \quad \text{for } i = 1, \ldots, n, \\
\#_f \to (\#, out).
\end{array}
$$

  • For every function $fun \in \{g_1, \ldots, g_t, f\}$ and for every membrane j of $\mu_{\Pi_{fun}}$, the following rules are included:

$$
\begin{array}{l}
\ominus \to \oplus(\ominus, in_{j_1}) \ldots (\ominus, in_{j_k}) \\
\odot^u \oplus \to \mathcal{M}_j^{fun} > \oplus \to \oplus \odot
\end{array}
$$

The rules and priorities associated with membrane j in Π_{fun}

Here, $j_1, \ldots j_k$ are the children membranes of membrane j and u is its depth level within the tree $\mu_{\Pi_{fun}}$. Moreover, if $j = im_{fun}$, then the rule $\oplus \to \oplus\odot$ has higher priority than the original rules of Π_{fun} for this membrane.

  • Let $fun \in \{g_1, \ldots, g_t, f\}$ and let $j_1, \ldots, j_q$ be the membrane path in $\mu_{\Pi_{fun}}$ from σ_{fun} to the input membrane, im_{fun}. Then, for each $k = 1, \ldots, q-1$ the following rules are included in membrane j_k:

$$
\begin{array}{ll}
e_i \to (e_i, in_{j_{k+1}}), & \text{for } i = 1, \ldots, r, \text{ and } fun = g_1, \ldots, g_t, \\
b_i \to (b_i, in_{j_{k+1}}), & \text{for } i = 1, \ldots, m, \text{ and } fun = f.
\end{array}
$$

The following rules are also included in membrane $j_q = im_{fun}$:

$$
\begin{array}{ll}
e_i \to a_i, & \text{for } i = 1, \ldots, r, \text{ and } fun = g_1, \ldots, g_t, \\
b_i \to c_i, & \text{for } i = 1, \ldots, m, \text{ and } fun = f.
\end{array}
$$

Thus, the initial membrane structure of this system can be pictorially represented as in figure 1. Furthermore, its functioning can be considered arranged in two stages:

Stage 1: *Calculation of the functions g_1 to g_t over the input data*:

- Sending of the input data to the input membranes of Π_{g_1} to Π_{g_t}.
- Local synchronization of the membranes in each Π_{g_i}.
- Global synchronization in the skin of Π of the computed values.

Stage 2: *Calculation of the function f*:

- Sending of the computed values in the previous phase from the skin to the input membrane of Π_f.
- Local synchronization of the membranes in Π_f.
- Sending the result to the environment.

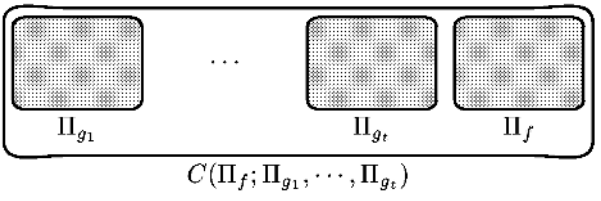

Fig. 1. Composition of computing P systems

Notation. *We say that the system Π designed above, which we denote by $C(\Pi_f; \Pi_{g_1}, \ldots, \Pi_{g_t})$, is the system obtained from the composition of Π_f with $\Pi_{g_1} \ldots, \Pi_{g_t}$.*

Next we are going to justify, in an informal manner, that $C(\Pi_f; \Pi_{g_1}, \ldots, \Pi_{g_t})$ is a computing P system that is valid and that, besides, computes the composition of f with $g_1, \ldots, g_t$. This system also preserves the use or not of membrane dissolution from the P systems that compute the functions.

Stage 1: *Computing the functions $g_1, \ldots, g_t$ over the input data.*

To perform this stage, it is necessary to carry out two operations: the first one consists of *transporting* the input arguments from membrane 1, which recall is the input membrane of Π, to each of the input membranes of the systems $\Pi_{g_1}, \ldots, \Pi_{g_t}$. This is easily done by displacing the objects that represent the arguments through all the necessary membranes.

The second operation is a little bit more difficult: for a specific system, Π_{g_j}, to correctly compute the value of the function g_j over the input data, we need that all the membranes of this system start to apply their *original* rules at the same time (that is, we have to achieve a *local synchronization* of all

the membranes of each Π_{g_j}). This can be done by using counters for each of these membranes. First, we use the object $\ominus$ to activate the counters, represented by objects $\oplus$, in all the membranes. These last objects use, in turn, objects $\odot$ to count and, when a certain quantity has been reached, the corresponding membrane is allowed to apply the rules of Π_{g_j}. From the way it has been implemented, these quantities coincide with the depth levels of each one of the membranes in the structure $\mu_{\Pi_{g_j}}$.

It is also important to observe that when the system Π_{g_j} begins to compute the value, the objects that represent the input data must have reached their corresponding input membrane. However, as we perform the two previous operations simultaneously, this is obtained automatically.

Finally, before permitting that the system Π_f activates itself, it is necessary to make sure that all the values of $\Pi_{g_1}, \ldots, \Pi_{g_t}$ have been computed (that is, there must be a *global synchronization* in the skin of Π).

Let us see in detail the rules involved in this stage:

a) In the first step of a computation of Π in which the value of the composition function over the tuple $(k_1, \ldots, k_r)$ is computed, in membrane 1 we have the multiset $e_1^{k_1} \ldots e_r^{k_r} \# \ominus$ and the remaining membranes are empty. Therefore, the only rules that can be applied are those that send the objects e_i and the object $\ominus$ into the corresponding skin membranes of $\mu_{\Pi_{g_1}}, \ldots, \mu_{\Pi_{g_t}}$, and the rule $\# \to \#$ in membrane 1.

b) Now membrane 1 waits for the values of the functions $g_1, \ldots, g_t$ over the tuple $(k_1, \ldots, k_r)$ by means of the rule $\# \to \#$. With respect to the membrane structures $\mu_{\Pi_{g_1}}$ to $\mu_{\Pi_{g_t}}$, the rule $\ominus \to \oplus(\ominus, in_{j_1}) \ldots (\ominus, in_{j_k})$ makes the object $\ominus$ to propagate to all of their membranes, since when it reaches a specific membrane, it immediately transforms itself into a counter object $\oplus$ and it is also sent to the children membranes. Thus, from a computation step to the next one, the object $\ominus$ reaches the membranes with one level below. Meanwhile, the rule $\oplus \to \oplus\odot$ makes the object $\oplus$ to generate objects $\odot$. A close look to the situation created shows us that the activating object $\ominus$ has reached all the membranes exactly when the counter object $\oplus$ has generated in each membrane a number of objects $\odot$ equal to their levels in the tree $\mu_{\Pi_{fun}}$ (for $fun \in \{g_1, \ldots, g_t\}$). At that moment, the rule $\odot^u \oplus \to \mathcal{M}_j^{fun}$ introduces in membrane j the objects associated with it in Π_{fun}, and this is done for all the membranes of each Π_{fun} at the same time. From now on, the values of $g_1, \ldots, g_t$ over $(k_1, \ldots, k_r)$ are computed exactly in the same way than the systems $\Pi_{g_1}, \ldots, \Pi_{g_t}$ would do it.

An example of how the process of local synchronization works is shown in figure 2, for a P system with ten membranes.

c) Simultaneously, the objects e_i cover the path from the skin membrane of each $\mu_{\Pi_{g_j}}$ to the input membrane of Π_{g_j}, by means of the rules $e_i \to (e_i, in_{j_{k+1}})$, and they evolve there into the corresponding objects a_i, by means of the rules $e_i \to a_i$. Take into account that the objects e_i and the object $\ominus$ reach the input membrane of Π_{g_j} at the same time. In this

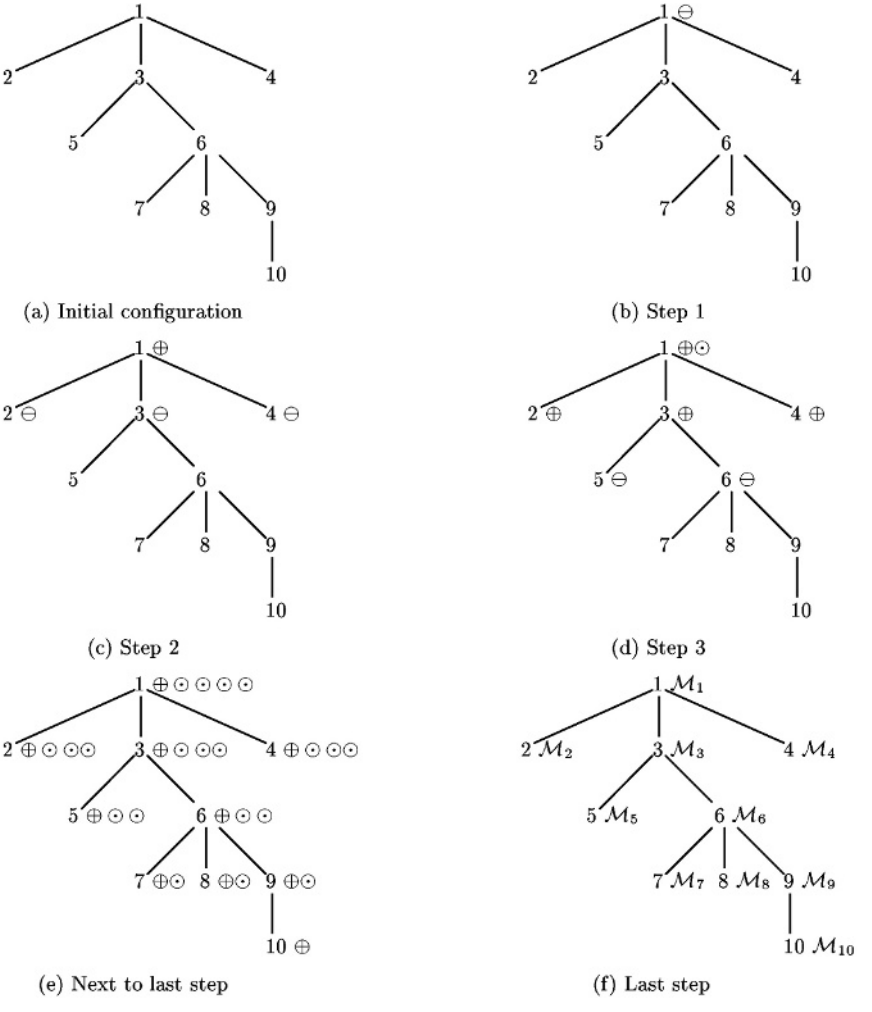

Fig. 2. Local synchronization of a P system

way, when Π_{g_j} begins to perform its original work, the input data is in the suitable place.

Stage 2: *Computing the function f over* $(g_1(k_1, \ldots, k_r), \ldots, g_t(k_1, \ldots, k_r))$

The first stage ends when membrane 1 has collected at least one copy of each object $\#_{g_1}, \ldots, \#_{g_t}$. In that moment the computed values have to be sent as input data to the system Π_f. To synchronize the end of the first stage with the beginning of the second one, in membrane 1 the rule $\# \to \#$ is repeatedly applied until the rule $\#_{g_1} \ldots \#_{g_t} \# \to (\ominus, in_{\sigma_f})$ can be used.

This rule sends an object $\ominus$ to the skin of μ_{Π_f}, with the goal of initiating the counters of its membranes in such a way that these membranes can start

to apply their original rules at the same time (*local synchronization* within Π_f). This process is performed in a similar way as the previous one of stage 1. Also in the next step of the computation the objects b_i, that represent the values obtained in the first stage, are put inside the skin of μ_{Π_f} and, subsequently, are moved by means of the rules $b_i \to (b_i, in_{j_{k+1}})$, through all the membranes of μ_{Π_f}, from the skin to the corresponding input membrane of Π_f.

It is easy to check that, although there exists a gap of one computation step between the moment when the object $\ominus$ arrives into a membrane and the moment when the objects b_i arrive, this entails no problem.

Next, the value of the function f over the arguments represented by the objects c_i is computed and, along this computation, objects d_i that represent the result are expelled from μ_{Π_f}. These objects are collected in membrane 1 and are immediately expelled from μ_{Π}. The computing process ends when some object $\#_f$ is collected in membrane 1, and all these objects are sent to the environment of μ_{Π} as objects $\#$.

4.3 Iteration of a Function

We introduce now the operation of iterating a computing P system. For that, we begin by defining the corresponding operation for functions.

Definition 13. *Let $f : \mathbb{N}^m - \to \mathbb{N}^m$. Then, the iteration of f, denoted $It(f)$, is a partial function from $\mathbb{N}^{m+1}$ to $\mathbb{N}^m$ defined as follows:*

$$It(f)(x_1, \ldots, x_m, 0) = (x_1, \ldots, x_m),$$
$$It(f)(x_1, \ldots, x_m, n+1) = It(f)(f(x_1, \ldots, x_m), n).$$

Next, let us see how we design, from a system $\Pi_f \in \mathcal{FC}$ that computes the function $f : \mathbb{N}^m - \to \mathbb{N}^m$ *without using the dissolution of membranes*, a computing P system that computes the iteration of f.

Let us suppose that

$$\Pi_f = \big(\Sigma_f, \Gamma_f, \Lambda_f, \#_f, \mu_{\Pi_f}, \mathcal{M}_1^f, \ldots, \mathcal{M}_{p_f}^f, (R_1^f, \rho_1^f), \ldots, (R_{p_f}^f, \rho_{p_f}^f), im_f\big).$$

Renaming adequately the elements of the alphabets (and, therefore, also of the rules) we suppose, besides, that $\Sigma_f = \{a_1, \ldots, a_m\}$ and $\Lambda_f = \{b_1, \ldots, b_m\}$.

Let us consider the computing P system

$$\Pi = \big(\Sigma, \Gamma, \Lambda, \#, \mu_{\Pi}, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p), im, env\big)$$

that verifies the following:

- $\Sigma = \{c_1, \ldots, c_{m+1}\}$. We can suppose, besides, that the condition $\Sigma \cap \Gamma_f = \emptyset$ is satisfied.
- There exist the distinguished elements $\oplus, \ominus, \odot, \otimes, \oslash \in \Gamma \setminus \Gamma_f$.
- $\Lambda = \{c_1, \ldots, c_m\}$.

– The object # is distinct from the object $\#_f$.
– $\mu_{\Pi} = [_1 \mu_{\Pi_f}]_1$, where the membranes of μ_{Π_f} have been adequately renamed (and, therefore, the rules of Π_f have also been adapted). We denote by σ_f the skin membrane of this system. Also, we consider that im_f is the new label of the input membrane of Π_f.
– $p = p_f + 1$.
– $\mathcal{M}_1 = \#$. The remaining multisets are all empty.
– $im = 1$.
– The evolution rules and their priorities are the following:
  • Evolutions rules for membrane 1:

$$\begin{aligned}&\#c_{m+1} \to (\ominus, in_{\sigma_f}) > \#c_i \to \#(c_i, out) > \\ &> \# \to (\#, out) > \#_f\#_f \to \#_f > \#_f \to (\oslash, in_{\sigma_f}) > \\ &> \otimes^u b_i \to \otimes^u c_i > \otimes^u \to \# > \\ &> c_i \to (c_i, in_{\sigma_f});\ \text{in all cases } i = 1, \ldots, m;\end{aligned}$$

  u is the number of membranes of the structure μ_{Π_f}.
  • For each membrane j distinct from membrane 1 the following rules are included:

$$\begin{aligned}&\ominus \to \oplus(\ominus, in_{j_1}) \ldots (\ominus, in_{j_k}),\\ &\odot^v \oplus \to \mathcal{M}_j^f > \oplus \to \oplus\odot,\\ &\oslash \to \otimes(\oslash, in_{j_1}) \ldots (\oslash, in_{j_k}),\\ &ob\otimes \to \otimes > \otimes \to (\otimes, out), \quad \text{for all } ob \in \Gamma_f.\end{aligned}$$

  The rules and priorities associated with membrane j in Π_f

  Here, $j_1, \ldots, j_k$ are the children membranes of membrane j and v is its depth level within μ_{Π_f}. Moreover, if $j = im_f$, then the rule $\oplus \to \oplus\odot$ has higher priority than the original rules of this membrane in Π_f.
  • Let $j_1, \ldots, j_q$ be the membrane path in μ_{Π_f} from σ_f to the input membrane, im_f,. Then, for each $k = 1, \ldots, q-1$, the following rules are included in membrane j_k:

$$c_i \to (c_i, in_{j_{k+1}}), \quad \text{for } i = 1, \ldots, m.$$

  The following rules are also included in membrane $j_q = im_f$:

$$c_i \to a_i, \quad \text{for } i = 1, \ldots, m.$$

Thus, the initial membrane structure of this system can be represented as in figure 3. Furthermore, its functioning can be considered arranged in two stages:

Stage 1: *Computation of one iteration of f*:

– Sending of the input data to the input membrane of Π_f.
– Local synchronization of the membranes in Π_f.

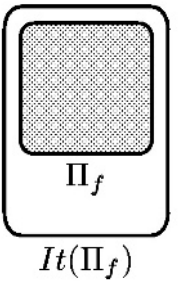

Fig. 3. Iteration of computing P systems

Stage 2: *Restarting of the system* Π_f:

- Erasing of the objects remaining in Π_f.
- Beginning of a new iteration of f.

Notation. *We say that the system* Π *designed above, which we denote by* $It(\Pi_f)$*, is the system obtained by the iteration of the system* Π_f*.*

Next, we are going to justify, in an informal manner, that the system $It(\Pi_f)$ is a computing P system that is valid and computes the iteration of f.

The number of iterations of f to perform is given by the $(m+1)$-th argument supplied to $It(f)$. What we do then is to reduce this argument by one and, next, we perform a process consisting of two stages: the first one consists of computing one iteration of f; the second one consists of "reseting" the system Π_f to its initial state. We iterate this process until the $(m+1)$-th argument makes zero.

The condition to decide if a iteration has to be performed or not is checked in membrane 1 examining how many objects c_{m+1}, that represent the $(m+1)$-th argument are present. If any of those objects is present, then the rule $\#c_{m+1} \rightarrow (\ominus, in_{\sigma_f})$ is applied (followed by the rules $c_i \rightarrow (c_i, in_{\sigma_f})$), starting the calculation of a new iteration of the function f.

Stage 1: *Computation of one iteration of* f:

This stage begins when an object $\ominus$ is introduced in the skin of μ_{Π_f}. This object initiates counters in the membranes of μ_{Π_f}, in an analogous manner as it was done for composition, in order to make sure that they will begin to apply their *original* rules at the same time (*local synchronization* within Π_f). Also, with a gap of a computation step that is not relevant, the input data, represented by the objects c_i, is transported from the skin of μ_{Π_f} to the input membrane of Π_f. Although along the execution of this stage the result of a iteration is sent out of μ_{Π_f}, being collected in membrane 1 of Π, it is necessary to observe that in this membrane no rule is activated.

Stage 2: *Restarting of the system* Π_f

The first stage ends when some object $\#_f$ is collected in membrane 1 of Π. Before we can begin the simulation of another iteration of f, it is necessary to erase all the objects that remain in the membranes of μ_{Π_f}. This is done in this stage, that begins by reducing the number of objects $\#_f$ present in membrane 1 to only one. Then the rule $\#_f \to (\ominus, in_{\sigma_f})$ in this membrane introduces an object $\oslash$ in the skin of μ_{Π_f}.

This object spreads to all the membranes in the same way as $\ominus$ does in the previous stage, and put an object $\otimes$ in each of them. These last objects act as erasers, eliminating all the objects in the membranes by means of the rule $ob\otimes \to \otimes$. When a membrane has been emptied (that is, when only an object $\otimes$ remains in it), then the object $\otimes$ is expelled.

Therefore, this stage finishes when membrane 1 collects as many objects $\otimes$ as the degree of μ_{Π_f} indicates. It is only then when the rules $\otimes^u b_i \to \otimes^u c_i$ can be applied, transforming the result of one iteration of f into input data of Π. Finally, the rule $\otimes^u \to \#$ is applied to start the process again.

An example of how the process of restarting a P system works is shown in figure 4, for a P system with ten membranes.

At the moment when no object c_{m+1} is present in membrane 1, it is necessary to finish the simulation of iterations. Then it is necessary to send the objects $c_1, \ldots, c_m$ of this membrane to the environment, followed by an object $\#$.

Note that along the evaluation of the halting condition no rule can be applied in any membrane distinct from the skin membrane, because they are empty.

4.4 Unbounded Minimization of a Function

We introduce now the operation of unbounded minization of a computing P system. For that, we begin by defining the corresponding operation for functions.

Definition 14. *The operation of* unbounded minimization *or* μ–recursion *applied to the partial function* $f : \mathbb{N}^{n+1} - \to \mathbb{N}$ *produces the function* $Min(f) : \mathbb{N}^n - \to \mathbb{N}$ *given by*

$$Min(f)(x_1, \ldots, x_n) = \begin{cases} y_{x_1,\ldots,x_n}, & \text{if } y_{x_1,\ldots,x_n} \text{ exists,} \\ \text{undefined}, & \text{otherwise,} \end{cases}$$

for every $(x_1, \ldots, x_n) \in \mathbb{N}^n$*, where*

$$y_{x_1,\ldots,x_n} = \min\{y \in \mathbb{N} \mid \forall z < y\big(f \text{ is defined over } (x_1, \ldots, x_n, z)\big) \wedge f(x_1, \ldots, x_n, y) = 0\}.$$

Finally, we are going to describe a computing P system that, from a system $\Pi_f \in \mathcal{FC}$ computing the function $f : \mathbb{N}^{m+1} - \to \mathbb{N}$ *without using the dissolution of membranes*, computes the function obtained by the unbounded minimization from f.

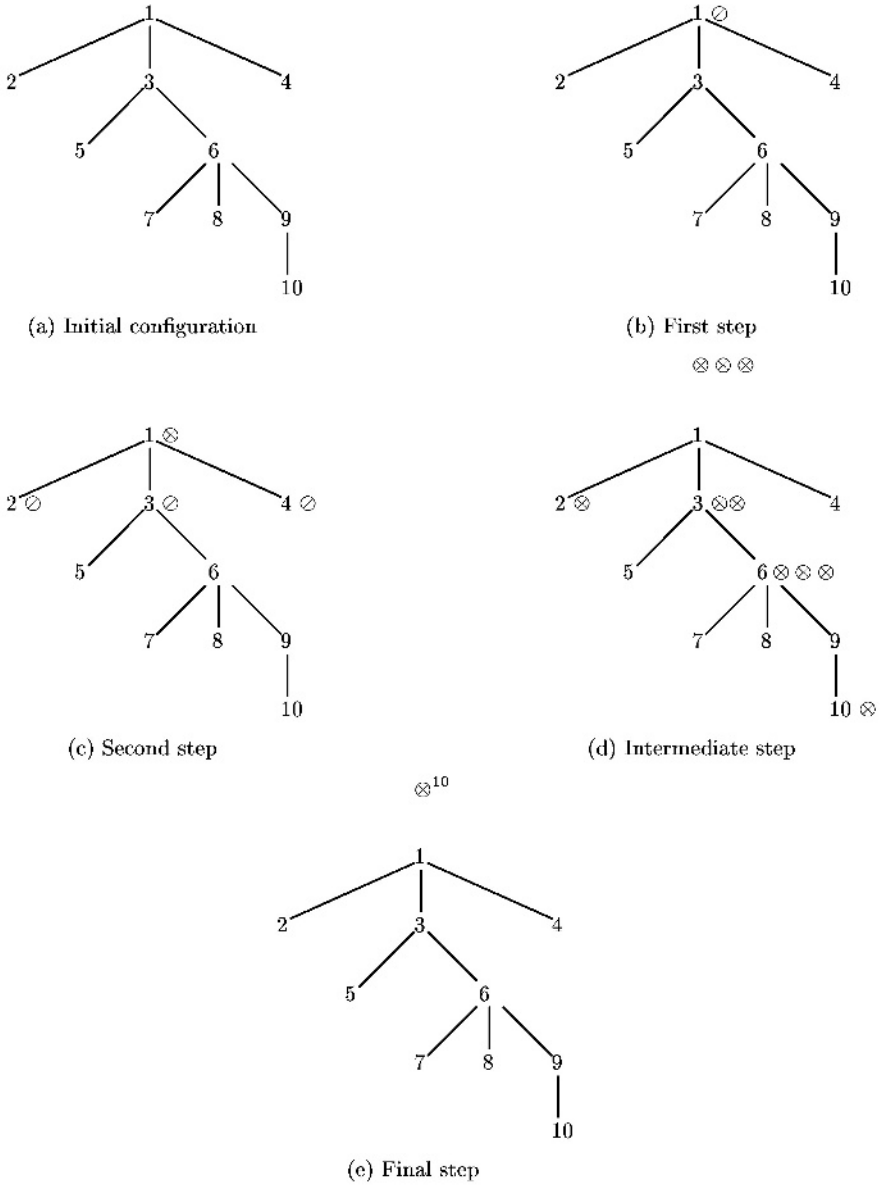

Fig. 4. Restarting of a P system

Let us suppose that

$$\Pi_f = \left(\Sigma_f, \Gamma_f, \Lambda_f, \#_f, \mu_{\Pi_f}, \mathcal{M}_1^f, \ldots, \mathcal{M}_{p_f}^f, (R_1^f, \rho_1^f), \ldots, (R_{p_f}^f, \rho_{p_f}^f), im_f\right)$$

Renaming adequately the elements of the alphabets (and, therefore, also of the rules) we can also suppose that $\Sigma_f = \{a_1, \ldots, a_{m+1}\}$ and $\Lambda_f = \{b\}$.

Let us consider the computing P system

$$\Pi = \big(\Sigma, \Gamma, \Lambda, \#, \mu_{\Pi}, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p), im, env\big)$$

verifying the following conditions:

- $\Sigma = \{c_1, \ldots, c_m\}$. We can also suppose that the condition $\Sigma \cap \Gamma_f = \emptyset$ is satisfied.
- There exist distinguished elements $\oplus, \ominus, \odot, \otimes, \oslash \in \Gamma \setminus \Gamma_f$.
- $\Lambda = \{c_{m+1}\}$.
- The object $\#$ is distinct from the object $\#_f$.
- $\mu_{\Pi} = [_1 \mu_{\Pi_f}]_1$, where the membranes of μ_{Π_f} have been adequately renamed (and, therefore, the rules of Π_f have also been adapted). We denote by σ_f the skin membrane of this system. Moreover, we consider that im_f is the new label of the input membrane of Π_f.
- $p = p_f + 1$.
- $\mathcal{M}_1 = \#$. The remaining multisets are all empty.
- $im = 1$.
- The evolution rules and priorities are the following:
  - Evolution rules for membrane 1:

    $$\begin{aligned}
    &\#_f \#_f \to \#_f > \#_f b \to b c_{m+1}(\oslash, in_{\sigma_f}) > \otimes^u \to \# > \#b \to \# > \\
    &\#c_i \to \#d_i > \# \to (\ominus, in_{\sigma_f}) > \\
    &d_i \to c_i(d_i, in_{\sigma_f}) > \#_f c_{m+1} \to \#_f(c_{m+1}, out) > \\
    &\#_f \to (\#, out), \qquad \text{for } i = 1, 2, \ldots, m+1;
    \end{aligned}$$

    u is the number of membranes of the structure μ_{Π_f}.
  - For each membrane j distinct from membrane 1 the following rules are included:

    $$\begin{aligned}
    &\ominus \to \oplus(\ominus, in_{j_1}) \ldots (\ominus, in_{j_k}) \\
    &\odot^v \oplus \to \mathcal{M}_j^f > \oplus \to \oplus \odot \\
    &\oslash \to \otimes(\oslash, in_{j_1}) \ldots (\oslash, in_{j_k}) \\
    &ob\otimes \to \otimes > \otimes \to (\otimes, out), \qquad \text{for } ob \in \Gamma_f.
    \end{aligned}$$

    The rules and priorities associated with membrane j in Π_f .

    Here, $j_1, \ldots, j_k$ are the children membranes of membrane j and v is its depth within μ_{Π_f}. Moreover, if $j = im_f$, then the rule $\oplus \to \oplus\odot$ has higher priority than the original rules for this membrane in Π_f.
  - Let $j_1, \ldots, j_q$ be the membrane path in μ_{Π_f} from σ_f to the input membrane, im_f. Then, for each $k = 1, \ldots, q-1$, the following rules are included to membrane j_k:

    $$d_i \to (d_i, in_{j_{k+1}}), \qquad \text{for } i = 1, \ldots, m+1.$$

The following rules are also included in membrane $j_q = im_f$:

$$d_i \to a_i, \quad \text{for } i = 1, \ldots, m+1.$$

Thus, the initial membrane structure of this system can be represented as shown in figure 5. Furthermore, its functioning can be considered arranged in two stages:

Stage 1: *Calculation of* $f(\boldsymbol{x}, y)$ *(starting with* $y = 0$*)*:

- Erasing of the results obtained previously.
- Sending of the input data to the input membrane of Π_f.
- Local synchronization of the membranes in Π_f.

Stage 2: *Checking the result of* $f(\boldsymbol{x}, y)$:

- If the result is zero, sending y to the environment and halting.
- If the result is not zero, reseting the system Π_f and increasing y by 1. Then go back to stage 1.

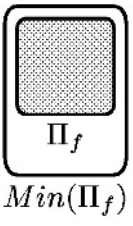

Fig. 5. Unbounded minimization of computing P systems

Notation. *We say that the system* Π *defined above, which we denote by* $Min(\Pi_f)$*, is the system obtained by the unbounded minimization of the system* Π_f*.*

Next, we are going to justify, in an informal manner, that the system $Min(\Pi_f)$ is a computing P system that is valid and computes the unbounded minimization of f.

Given an input data $(x_1, \ldots, x_m) \in \mathbb{N}^m$ we have to compute the values $f(x_1, \ldots, x_m, y)$ for $y = 0, 1, 2$ and so on, until finding the first one that is zero, in which case we return the corresponding value of y. The data $(x_1, \ldots, x_m)$ is represented by the objects c_i, with $i = 1, \ldots, m$, and the number y will be given by the number of objects c_{m+1} present in the system.

To perform this, the system Π repeats a process arranged in two stages: the first one consists of computing the value of f applied to the input data $(x_1, \ldots, x_m)$ and to a specific number y; in the second stage the obtained result

is checked. If it is zero, then we have finished and it suffices to expel the objects c_{m+1} to the environment. If it is not zero, then we add a new object c_{m+1}, in such a way that these objects represent the number $y+1$, and return the system Π_f to its initial configuration, starting again with the first stage.

Stage 1: *Calculation of* $f(x_1, \ldots, x_m, y)$

This stage is activated with the presence of an object # in membrane 1 of Π. What is done first is the erasing, by means of the rule $\#b \to \#$, the result of $f(x_1, \ldots, x_m, y-1)$ that we would have obtained previously. Next, we change the objects c_i into objects d_i, with the goal of being able to send them to the system Π_f and, at the same time, keep them in the input membrane of Π. Once done this, we send an object $\ominus$ to the skin of μ_{Π_f} in order to perform, in an analogous way as we have seen for composition and iteration, a *local synchronization* of its membranes. Also, with a gap of one computation step that is not relevant, the objects d_i, that represent the arguments to which we are going to apply the function f, is transported from the skin of μ_{Π_f} to the input membrane of Π_f. Furthermore, we keep a copy in membrane 1 using objects c_i.

From now on no rule can be applied in membrane 1 until Π_f does not finish computing the value of the function f applied to the tuple $(x_1, \ldots, x_m, y)$.

Stage 2: *Checking of the result*

In this stage what is first done is reducing to only one the number of objects $\#_f$ collected in membrane 1 of Π. Then, if the result of $f(x_1, \ldots, x_m, y)$ has been zero, the only rules applicable are the rule $\#_f c_{m+1} \to \#_f(c_{m+1}, out)$, that sends the objects c_{m+1} to the external environment, followed by the rule $\#_f \to (\#, out)$, that finishes the computation.

If the result of $f(x_1, \ldots, x_m, y)$ has been different from zero, then in membrane 1 of Π some object b has been collected and, therefore, the rule $\#_f b \to b c_{m+1}(\oslash, in_{\sigma_f})$ will be applicable. This rule adds a new object c_{m+1}, for its multiplicity to represent the number $y+1$. Furthermore, that rule sends and object $\oslash$ to the skin membrane of μ_{Π_f} to restart the system Π_f, exactly in the same way as we did with iteration. Then no rule in membrane 1 of Π can be applied until as many objects $\otimes$ as membranes in μ_{Π_f} do not appear. At this moment, the rule $\otimes^u \to \#$ introduces an object #, so that stage 1 starts again.

From these constructions and discussions we infer the following result.

Theorem 1. *Let $f \in \mathcal{P}$ be a partial recursive function. Then there exists a system $\Pi_f \in \mathcal{FC}$, which uses priority and cooperation, but not dissolution, computing the function f.*

Proof. It suffices to take into account that if f is a recursive function then there exist functions $g_1, \ldots, g_n$ such that $g_n = f$ and for each $j = 1, \ldots, n$ either g_j is a basic function, or g_j is obtained from some of the functions $g_1, \ldots, g_{j-1}$ by means of the operations of composition, iteration or unbounded minimization (see [1]).

5 Conclusions

We have studied in this paper *computing P systems*. This is a variant of the model of computation introduced in [4], which in turn is a variant of the basic model of transition P system introduced by G. Păun in [2]. The idea behind this new model is to be able to compute functions without worrying about the content of the membrane structure used to do it, but only considering the objects collected in its environment.

We have defined three operations for computing P systems with external output: composition, iteration and minimization. These operations have allowed us to prove, in a constructive manner, the computational completeness of this model, since using these operations any partial recursive function can be computed by such a system.

Acknowledgement. The authors gratefully acknowledge the support of the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds.

References

1. D.E. Cohen. *Computability and Logic*. Ellis Horwood, 1987.
2. G. Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
3. G. Păun. *Membrane Computing. An Introduction*. Springer–Verlag, 2002.
4. G. Păun, G. Rozenberg, and A. Salomaa. Membrane Computing with External Output. *Fundamenta Informaticae*, 41(3):259–266, 2000.
5. M.J. Pérez-Jiménez and F. Sancho-Caparrini. A Formalization of Transition P Systems. *Fundamenta Informaticae*, 49(1–3):261–272, 2002.
6. A. Romero-Jiménez. *Complejidad y Universalidad en Modelos de Computación Celular*. PhD thesis, Universidad de Sevilla, 2003.
7. A. Romero-Jiménez and M.J. Pérez-Jiménez. Generation of Diophantine Sets by Computing P Systems with External Output. In C.S. Calude, M.J. Dinneen, and F. Peper, editors, *Unconventional Models of Computation*, volume 2509 of *Lecture Notes in Computer Science*, pages 176–190. Springer–Verlag, 2002.
8. F. Sancho-Caparrini. *Verificación de Programas en Modelos de Computación no Convencionales*. PhD thesis, Universidad de Sevilla, 2002.
9. The P Systems Web Page. (URL `http://psystems.disco.unimib.it/`).
10. Web Page of the Research Group on Natural Computing of University of Sevilla. (URL `http://www.cs.us.es/gcn/`).