Generation of Diophantine Sets by Computing P Systems with External Output

Álvaro ROMERO JIMÉNEZ and Mario J. PÉREZ JIMÉNEZ

Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla, España
E-mail: {Alvaro.Romero,Mario.Perez}@cs.us.es

Abstract. In this paper a variant of P systems with external output designed to compute functions on natural numbers is presented. These P systems are stable under composition and iteration of functions. We prove that every diophantine set can be generated by such P systems; then, the universality of this model can be deduced from the theorem by Matiyasevich, Robinson, Davis and Putnam in which they establish that every recursively enumerable set is a diophantine set.

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 [3]). The framework within which computation are performed in this model is the membrane structure, which recreates 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 [5] 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 kind of P systems have been studied in [6].

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 apparatus, we have been able to define, in a comfortable manner, several operations between computing P systems with external output; more specifically, we have defined composition and iteration, what have allowed

us to prove the universality of these devices through the generation of all the diophantine sets.

2 Multisets. Membrane structures. Evolution rules

A *multiset* over a set, A, is an application $m : A \to \mathbb{N}$. A multiset is said to be empty (resp. finite) if its support, $supp(m) = \{a \in A : m(a) > 0\}$, is empty (resp. finite). If m is a finite multiset over A, we will denote it $m = \{\{a_1, \ldots, a_k\}\}$, where the elements a_i are possibly repeated. We write $M(A)$ for the set of all the multisets over A.

The set of *membrane structures*, MS, is defined by recursion as follows: 1. $[\,] \in MS$; 2. If $\mu_1, \ldots, \mu_n \in MS$, then $[\mu_1 \ldots \mu_n] \in MS$.

A membrane structure, μ, can also be seen as a rooted tree, $\big(V(\mu), E(\mu)\big)$. Then, the nodes of this tree are called *membranes*, the root node the *skin membrane* and the leaves *elementary membranes* of the membrane structure. The *degree* of a membrane structure is the number of membranes in it.

The concepts of *depth* of a membrane structure and *depth* of its membranes are easily defined from those of a tree and its nodes. We will also need the notion of *level* of a membrane within a membrane structure, which is defined as the difference between the depth of the second and the depth of the first.

The *membrane structure with external environment* associated with a membrane structure, μ, is $\mu^E = [_E \mu]_E$. If we consider the latter as a rooted tree, the root node is called the *external environment* of μ.

Given an alphabet, Γ, we associate with every membrane of a membrane structure a finite multiset of elements of Γ, which are called the *objects* of the membrane.

We also associate with every one of these membranes a finite set of *evolution rules*. A evolution rule over Γ 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

$$\Gamma \times \big(\{here, out\} \cup \{in_l : l \in V(\mu)\}\big)$$

The idea behind a rule is that the objects in u "evolve" into the objects in v', moving or not to another membrane and possibly dissolving the original one.

3 Computing P systems with external output

We are now prepared to introduce our new model of computation.

Definition 3.1. *A computing P system with external output of order* (m, n) *and degree* p *is a tuple*

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

where

- Σ *is an ordered alphabet of size* m*, the input alphabet.*
- Λ *is an ordered alphabet of size* n*, the output alphabet.*
- Γ *is an alphabet such that* $\Sigma \cup \Lambda \subset \Gamma$*, the working alphabet.*
- $\#$ *is a distinguished element in* $\Gamma \setminus (\Sigma \cup \Lambda)$*, the halting element.*
- μ_Π *is a membrane structure of degree* p*, whose membranes we suppose labeled from 1 to* p*.*
- *The input membrane of* Π *is labeled by* $\iota \in \{1, \ldots, p\}$*.*
- $\mathcal{M}_i$ *is a multiset over* $\Gamma \setminus \Sigma$ *associated with the membrane labeled by* i*, for every* $i = 1, \ldots, p$*.*
- R_i *is a finite set of evolution rules over* Γ *associated with the membrane labeled by* i*, and* ρ_i *is a strict partial order over it, for every* $i = 1, \ldots, p$*.*

To formalize the semantics of this model we define first what a configuration of a P system is, from what follows the notion of computation.

Definition 3.2. *Let* Π *be a computing P system with external output.*

- *A configuration of* Π *is a pair* (μ^E, M)*, where* μ *is a membrane structure such that* $V(\mu) \subseteq V(\mu_\Pi)$ *and has the same root than* μ_Π*, and* M *is an application from* $V(\mu^E)$ *into* $M(\Gamma)$*. For every node* $nd \in V(\mu^E)$ *we denote* $M_{nd} = M(nd)$*.*
- *Suppose that* Π *is of order* (m, n) *and* $\Sigma = (a_1, \ldots, a_m)$*. Then, any* m*-tuple of natural numbers can be codified by a multiset over* Σ *and given as input to the P system. Thus, the initial configuration of* Π *for a tuple* $(k_1, \ldots, k_m) \in \mathbb{N}^m$ *is the pair* (μ^E, M)*, where* $\mu = \mu_\Pi$*,* $M_E = \emptyset$*,* $M_\iota = \mathcal{M}_\iota \cup \{\{a_1^{k_1} \ldots a_m^{k_m}\}\}$ *and* $M_i = \mathcal{M}_i$*, for every* $i \neq \iota$*.*

We can pass, in a non-deterministic manner, from one configuration of Π to another 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 external 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 not a children membrane of i, 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 dissolve). Moreover, the *priority relation* among the rules forbids the application of a rule if another one of higher priority is applied.

Given two configurations, C and C', of Π, we say that C' is obtained from C in one transition step, and we write $C \Rightarrow C'$, if we can pass from the first to the second by using the evolutions rules appearing in the membrane structure of C in a parallel and maximal way, and for all the membranes at the same time.

Definition 3.3. *Given a computing P system with external output of order* (m, n)*,* Π*, a computation of* Π *with input* $(k_1, \ldots, k_m) \in \mathbb{N}^m$ *is a sequence, possibly infinite, of configurations of* Π*,* $C_0 \Rightarrow C_1 \Rightarrow \ldots \Rightarrow C_q$*,* $q \geq 0$*, such that*

– *C_0 is the initial configuration of Π for $(k_1, \ldots, k_m)$.*
– *Each C_i is obtained from the previous configuration by one transition step.*

We say that a computation, $\mathcal{C}$, is a halting computation of Π, if $q \in \mathbb{N}$ and there is no rule applicable to the objects present in its last configuration.

Then, the output of a halting computation and of a computing P system with external output can be defined in a natural way.

Definition 3.4. *Let Π be a computing P system with external output of order (m, n) and suppose that $\Lambda = (b_1, \ldots, b_n)$. Let $\mathcal{C}$ be a halting computation of Π with input $(k_1, \ldots, k_m) \in \mathbb{N}^m$ and (μ^E, M) its last configuration. Then, the output of that computation is given by*

$$Output(\mathcal{C}) = \big(M_E(b_1), \ldots, M_E(b_n)\big).$$

Definition 3.5. *Let Π be a computing P system with external output of order (m, n). The output of Π with input $(k_1, \ldots, k_m) \in \mathbb{N}^m$ is given by*

$$Output(\Pi; k_1, \ldots, k_m) = \{Output(\mathcal{C}) : \mathcal{C} \text{ is a halting computation of } \Pi \text{ with input } (k_1, \ldots, k_m)\}.$$

The idea behind P systems with external output is that we cannot know what is happening inside the membrane structure, but we can only collect the information thrown from it to the external environment. In accordance with it, it seems natural that the halting computations of these P systems report to the outside when they have reached their final configurations.

Furthermore, the idea behind computing P systems is to use them as computing models of functions between natural numbers. These considerations lead us to the following notions:

Definition 3.6. *A computing P system with external output of order (m, n), Π, is said to be valid when the following is verified:*

– *If $\mathcal{C}$ is a halting computation of Π, then a rule of the form $u \to v(\#, out)$ must have been applied in the skin membrane of μ_Π, and only in the last step of the computation.*
– *If $\mathcal{C}$ is not a halting computation of Π, then no rule of the previous form is applied in the skin membrane in any step of the computation.*
– *For every $(k_1, \ldots, k_m) \in \mathbb{N}^m$ and for every two halting computations, $\mathcal{C}_1$ and $\mathcal{C}_2$, of Π with input $(k_1, \ldots, k_m)$, $Output(\mathcal{C}_1) = Output(\mathcal{C}_2)$.*

Definition 3.7. *A computing P system with external output of order (m, n), Π, computes a partial function, $f : \mathbb{N}^m - \to \mathbb{N}^n$, if*

– *Π is a valid P system.*
– *For every $(k_1, \ldots, k_m) \in \mathbb{N}^m$*
  - *f is defined over $(k_1, \ldots, k_m)$ if and only if there exists a halting computation of Π with input $(k_1, \ldots, k_m)$.*

- *If $\mathcal{C}$ is a halting computation of Π with input $(k_1, \ldots, k_m)$, then $Output(\mathcal{C}) = f(k_1, \ldots, k_m)$.*

We denote $CEP_p^{m,n}(\alpha, \beta, \gamma)$, where $m, n \in \mathbb{N}, p \geq 1$, $\alpha \in \{Pri, nPri\}$, $\beta \in \{Coo, Cat, nCoo\}$ and $\gamma \in \{\delta, n\delta\}$, the family of functions computed by computing P systems with external output of order (m, n), of degree at most p, and with or without priority, with cooperation, only catalysts or without cooperation (see [3]), and with or without dissolution, respectively. The union, for all $p \geq 1$, of the families of one of these types is denoted $CEP^{m,n}(\alpha, \beta, \gamma)$.

4 Composition of computing P systems with external output

We introduce now the operation of composition between computing P systems with external output.

Definition 4.1. *Let $f : \mathbb{N}^m - \rightarrow \mathbb{N}^n$ and $g_1 : \mathbb{N}^r - \rightarrow \mathbb{N}^{s_1}, \ldots, g_t : \mathbb{N}^r - \rightarrow \mathbb{N}^{s_t}$ 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*

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

Theorem 4.2. *Let $f \in CEP^{m,n}(\alpha, \beta, \gamma), g_1 \in CEP^{r,s_1}(\alpha, \beta, \gamma), \ldots, g_t \in CEP^{r,s_t}(\alpha, \beta, \gamma)$, with $\alpha \in \{Pri, nPri\}$, $\beta \in \{Coo, Cat, nCoo\}$ and $\gamma \in \{\delta, n\delta\}$. Then, $C(f; g_1, \ldots, g_t) \in CEP^{r,n}(Pri, Coo, \gamma)$.*

Proof. Let

$$\Pi_f = \left(\Sigma_f, \Lambda_f, \Gamma_f, \#_f, \mu_{\Pi_f}, \iota_f, \mathcal{M}_1^f, \ldots, \mathcal{M}_{p_f}^f, (R_1^f, \rho_1^f), \ldots, (R_{p_f}^f, \rho_{p_f}^f)\right)$$
$$\Pi_{g_1} = \left(\Sigma_{g_1}, \Lambda_{g_1}, \Gamma_{g_1}, \#_{g_1}, \mu_{\Pi_{g_1}}, \iota_{g_1}, \mathcal{M}_1^{g_1}, \ldots, \mathcal{M}_{p_{g_1}}^{g_1}, (R_1^{g_1}, \rho_1^{g_1}), \ldots, (R_{p_{g_1}}^{g_1}, \rho_{p_{g_1}}^{g_1})\right)$$
$$\vdots$$
$$\Pi_{g_t} = \left(\Sigma_{g_t}, \Lambda_{g_t}, \Gamma_{g_t}, \#_{g_t}, \mu_{\Pi_{g_t}}, \iota_{g_t}, \mathcal{M}_1^{g_t}, \ldots, \mathcal{M}_{p_{g_t}}^{g_t}, (R_1^{g_t}, \rho_1^{g_t}), \ldots, (R_{p_{g_t}}^{g_t}, \rho_{p_{g_t}}^{g_t})\right)$$

be computing P systems with external output that compute, respectively, the function f and the functions g_1 to g_t.

By means of a renaming of the elements of the alphabets (and, therefore, also of the rules), we can suppose that

- $\Sigma_{g_1} = \cdots = \Sigma_{g_t} = (a_1, \ldots, a_r)$.
- $\Lambda_{g_1} = (b_1, \ldots, b_{s_1}), \ldots, \Lambda_{g_t} = (b_{s_1 + \cdots + s_{t-1} + 1}, \ldots, b_m)$.
- $\Sigma_f = (c_1, \ldots, c_m)$.
- $\Lambda_f = (d_1, \ldots, d_n)$.
- $\left(\Lambda_{g_1} \cup \cdots \cup \Lambda_{g_t}\right) \cap \Gamma_f = \emptyset$.
- $\#_{g_i} \neq \#_{g_j}$, for every $i \neq j$.

Let us consider the computing P system with external output

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

given by

- $\Sigma = (e_1, \ldots, e_r)$. (We suppose that $\Sigma \cap \bigcup_{i=1}^{t} \Gamma_{g_i} = \emptyset$).
- There exist distinguished elements $\oplus, \ominus, \odot \in \Gamma \setminus (\Gamma_f \cup \bigcup_{i=1}^{t} \Gamma_{g_i})$.
- $\Lambda = (d_1, \ldots, d_n)$.
- $\# \neq \#_{g_i}$, for every $i = 1, \ldots, t$, and $\# \neq \#_f$.
- $\mu_\Pi = [_1 \mu_{\Pi_{g_1}} \ldots \mu_{\Pi_{g_t}} \mu_{\Pi_f}]_1$, where the membranes from $\mu_{\Pi_{g_1}}, \ldots, \mu_{\Pi_{g_t}}, \mu_{\Pi_f}$ have been adequately renamed (and therefore, also the rules of the corresponding P systems have been adapted). We denote $\sigma_{g_1}, \ldots, \sigma_{g_t}, \sigma_f$ the skin membranes of the latter. Also, we consider that $\iota_{g_1}, \ldots, \iota_{g_t}, \iota_f$ reflect the new labeling of the input membranes of $\Pi_{g_1}, \ldots, \Pi_{g_t}, \Pi_f$, respectively.
- $\iota = 1$.
- $p = p_{g_1} + \cdots + p_{g_t} + p_f + 1$.
- $\mathcal{M}_1 = \{\{\#, \ominus\}\}$. The remaining multisets are all empty.
- The evolution rules are the following:
  - Evolution rules for membrane 1:

    $e_i \to (e_i, in_{\sigma_{g_1}}) \ldots (e_i, in_{\sigma_{g_t}}) \quad (i = 1, \ldots, r)$

    $\ominus \to (\ominus, in_{\sigma_{g_1}}) \ldots (\ominus, in_{\sigma_{g_t}})$

    $\#_{g_1} \ldots \#_{g_t} \# \to (\ominus, in_{\sigma_f}) > \# \to \# > b_i \to (b_i, in_{\sigma_f}) \quad (i = 1, \ldots, m)$

    $d_i \to (d_i, out) \quad (i = 1, \ldots, n)$

    $\#_f \to (\#, out)$

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

    $\ominus \to \oplus(\ominus, in_{j_1}) \ldots (\ominus, in_{j_k})$

    $\odot^u \oplus \to \mathcal{M}_j^{fun} > \oplus \to \oplus\odot$

    evolution rules associated with the membrane in Π_{fun}

    where j_1 to j_k are the children membranes of membrane j and u is its level within $\mu_{\Pi_{fun}}$. Moreover, if j is ι_{fun}, then the rule $\oplus \to \oplus\odot$ has higher priority than the original rules of Π_{fun} for this membrane.
  - Let fun be as above and let $j_1, \ldots, j_q$ be the membrane path from σ_{fun} to ι_{fun}. Then, for $k = 1, \ldots, q-1$ the following rules are included in membrane j_k:

    $e_i \to (e_i, in_{j_{k+1}}) \quad (i = 1, \ldots, r) \qquad \text{for } fun = g_1, \ldots, g_t$

    $b_i \to (b_i, in_{j_{k+1}}) \quad (i = 1, \ldots, m) \qquad \text{for } fun = f$

    Also, the following rules are included in membrane $j_q = \iota_{fun}$.

    $e_i \to a_i \quad (i = 1, \ldots, r) \qquad \text{for } fun = g_1, \ldots, g_t$

    $b_i \to c_i \quad (i = 1, \ldots, m) \qquad \text{for } fun = f$

The P system constructed in this way, denoted $C(\Pi_f; \Pi_{g_1}, \ldots, \Pi_{g_t})$, is a valid computing P system with external output which computes the composition of f with g_1 to g_t. Furthermore, it preserves the use or not of dissolution from the P systems which compute the functions.

Indeed, the system works as follows:

- Phase 1: *Computation of the functions g_1 to g_t over the input data*
  To perform this stage, we need to carry out two operations: the first one consists of taking the input arguments from membrane 1, which recall is the input membrane of Π, to all the input membranes of the P systems Π_{g_1} to Π_{g_t}. This is easily done by displacing the objects representing the arguments through all the necessary membranes.
  The second operation is a little bit more complicated: in order for a specific P system Π_{g_j} to compute correctly the value of the function g_j over the input data, we need that all the membranes of this P system start to apply their original rules *at the same time* (that is, we have to synchronize locally the membranes of each Π_{g_j}). We achieve this by using counters for every one of these membranes. First, we use the object $\ominus$ to activate the counters, represented by objects $\oplus$, in all the membranes. These latter objects use objects $\odot$ to count and, when a certain quantity is reached, the corresponding membrane is allowed to use the rules of Π_{g_j}. Because of the way we have implemented this, these quantities turn out to be the levels of the membranes in the structure $\mu_{\Pi_{g_j}}$.
  It is also important that when the P system Π_{g_j} starts to compute the value, the objects representing the input data have reached its input membrane. However, as we perform the two operations above simultaneously, we get it for free.
  Finally, we have to wait until all the values from Π_{g_1} to Π_{g_t} have been computed, before allowing the P system Π_f to be used (that is, there must be a global synchronization in the skin of Π).
  Let us see with greater detail the rules involved in this phase:
  1. At the first step of a computation of Π with input $(k_1, \ldots, k_r)$, we have in membrane 1 the multiset $\{\{e_1^{k_1}, \ldots, e_r^{k_r}, \#, \ominus\}\}$ and the other membranes are empty. Therefore, only the rules which send the objects e_i and the object $\ominus$ into the skins of $\mu_{\Pi_{g_1}}$ to $\mu_{\Pi_{g_t}}$ and the rule $\# \to \#$ in membrane 1 can be applied.
  2. Now, membrane 1 waits for the values of g_1 to g_t over $(k_1, \ldots, k_r)$ by means of the rule $\# \to \#$. With regard to 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 spread to all their membranes, because when it reaches a particular membrane, it is immediately transformed into a counter object $\oplus$ and also sent to the children membranes. Thus, from a step of the computation to the next one, $\ominus$ reaches the membranes one depth greater. Meanwhile, the rule $\oplus \to \oplus\odot$ makes the object $\oplus$ to generate objects $\odot$. A close look to the situation created shows that the activating object $\ominus$ have reached all the membranes exactly when the counter objects $\oplus$ have generated

in each membrane a number of objects $\odot$ equal to their levels in $\mu_{\Pi_{fun}}$ ($fun = g_1, \ldots, g_t$). At that moment, the rule $\odot^u \oplus \rightarrow \mathcal{M}_j^{fun}$ 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 to g_t over $(k_1, \ldots, k_r)$ are computed exactly in the same way than the P systems Π_{g_1} to Π_{g_t} would do it.

3. 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 $e_i \rightarrow (e_i, in_{j_{k+1}})$, and are changed there into the corresponding objects a_i, by means of the rules $e_i \rightarrow a_i$. Note that the objects e_i and the object $\ominus$ reach the input membrane at the same time. So, when Π_{g_j} starts its original functioning, as stated above, the input data is in its place.

– Phase 2: *Computation of the function f*

Phase 1 ends when membrane 1 has collected at least one object of each $\#_{g_1}$ to $\#_{g_t}$. It is then when the values computed have to be sent as input data to the P system Π_f. To synchronize the end of phase 1 with the beginning of phase 2, membrane 1 apply once and again the rule $\# \rightarrow \#$ until the rule $\#_{g_1} \ldots \#_{g_t} \# \rightarrow (\ominus, in_{\sigma_f})$ can be used.

This latter rule sends an object $\ominus$ into the skin of μ_{Π_f}, in order to initiate its membranes' counters so that they start to apply their original rules at the same time (local synchronization within Π_f). This is done just as before. Also, in the next step of the computation the objects b_i, which represent the values obtained in phase 1, are put into the skin of μ_{Π_f} and, subsequently, moved, by means of the rules $b_i \rightarrow (b_i, in_{j_{k+1}})$, through all the membranes from this one to the input membrane of Π_f. Next, the rules $b_i \rightarrow c_i$ change them into the corresponding input objects of Π_f.

It is easy to see that, although there is a gap of one step of computation between when $\ominus$ gets into a membrane and when the b_is do so, this is not at all a problem.

Now, the value of the function f over the arguments represented by the objects c_i is computed, and along this computation objects d_i representing the result are thrown out of μ_{Π_f}. These objects are collected in membrane 1, and immediately expelled from μ_Π. The calculation finishes when some objects $\#_f$ are collected in membrane 1 and they are expelled from μ_Π as objects $\#$.

□

5 Iteration of computing P systems with External Output

We introduce now the operation of iterating a computing P system with external output.

Definition 5.1. *Let $f : \mathbb{N}^m - \rightarrow \mathbb{N}^m$. Then, the iteration function of f, denoted $It(f)$, is a partial function from $\mathbb{N}^{m+1}$ to $\mathbb{N}^m$ defined as follows:*

$$It(f)(\boldsymbol{x}, 0) = \boldsymbol{x}$$
$$It(f)(\boldsymbol{x}, n+1) = It(f)(f(\boldsymbol{x}), n)$$

Theorem 5.2. *Let $f \in CEP^{m,m}(\alpha, \beta, n\delta)$, with $\alpha \in \{Pri, nPri\}$ and $\beta \in \{Coo, Cat, nCoo\}$. Then $It(f) \in CEP^{m+1,m}(Pri, Coo, n\delta)$.*

Proof. Let

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

be a computing P system with external output such that computes f.

By means of a renaming of the elements of the alphabets (and, therefore, also of the rules), we can suppose that

- $\Sigma_f = (a_1, \ldots, a_m)$.
- $\Lambda_f = (b_1, \ldots, b_m)$.

Let us consider the computing P system with external output

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

verifying the following

- $\Sigma = (c_1, \ldots, c_{m+1})$, and is such that $\Sigma \cap \Gamma_f = \emptyset$.
- There exist distinguished elements $\oplus, \ominus, \odot, \otimes, \oslash \in \Gamma \setminus \Gamma_f$.
- $\Lambda = (c_1, \ldots, c_m)$.
- $\# \neq \#_f$.
- $\mu_\Pi = [_1 \mu_{\Pi_f}]_1$, where the membranes from μ_{Π_f} have been adequately renamed (and therefore, also the rules of Π_f have been adapted). We denote σ_f the skin membrane of the latter. Also, we consider that ι_f reflects the new labeling of the input membrane of Π_f.
- $\iota = 1$.
- $p = p_f + 1$.
- $\mathcal{M}_1 = \{\{\#\}\}$. The remaining membranes are all empty.
- The evolution rules are the following:
  - Evolution rules for membrane 1:

$$\begin{aligned} \#c_{m+1} \rightarrow (\ominus, in_{\sigma_f}) > \#c_i \rightarrow \#(c_i, out) \quad (i = 1, \ldots, m) > \\ > \# \rightarrow (\#, out) > \#_f \#_f \rightarrow \#_f > \#_f \rightarrow (\oslash, in_{\sigma_f}) > \\ > \otimes^u b_i \rightarrow \otimes^u c_i \quad (i = 1, \ldots, m) > \otimes^u \rightarrow \# > \\ > c_i \rightarrow (c_i, in_{\sigma_f}) \quad (i = 1, \ldots, m) \end{aligned}$$

    where u is the degree of μ_{Π_f}.

- For every membrane j distinct from membrane 1 the following rules are included:

  $$\ominus \to \oplus(\ominus, in_{j_1}) \dots (\ominus, in_{j_k})$$
  $$\odot^u \oplus \to \mathcal{M}_j^f > \oplus \to \oplus\odot$$
  evolution rules associated with the membrane in Π_f
  $$\oslash \to \otimes(\oslash, in_{j_1}) \dots (\oslash, in_{j_k})$$
  $$ob\otimes \to \otimes \quad (ob \in \Gamma_f) > \otimes \to (\otimes, out)$$

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

  $$c_i \to (c_i, in_{j_{k+1}}) \quad (i = 1, \dots, m)$$

  Also, the following rules are included in membrane j_q:

  $$c_i \to a_i \quad (i = 1, \dots, m)$$

The P system constructed in this way, denoted by $It(\Pi_f)$, is a valid computing P system with external output which computes the iteration of f.

Indeed, the system works as follows:

The number of iterations of f to make is given by the $(m+1)$th argument provided to $It(f)$. What we do then is to reduce this argument by one and, next, carry out a two phases process: first, computing one iteration of f; second "reseting" the P system Π_f to its initial state. We repeat this process until the $(m+1)$th argument gets to zero.

The test to decide if one iteration has to be done is performed in membrane 1 looking how many objects c_{m+1}, which represents the $(m+1)$th argument, there are. If such an object is present, the rule $\#c_{m+1} \to (\ominus, in_{\sigma_f})$ (followed by the rules $c_i \to (c_i, in_{\sigma_f})$) is applied, starting the calculation of a new iteration of f, which is done in two phases.

- Phase 1: *Computing one iteration of f*
  This phase starts when an object $\ominus$ gets into the skin of μ_{Π_f}. This object initiate counters in the membranes of μ_{Π_f}, in the same way than we did for the composition, in order to assure that they start to apply their original rules at the same time (local synchronization within Π_f). Also, with a gap of one step of computation that does not matter, the input data, represented by objects c_i, is taken from the skin of μ_{Π_f} to the input membrane of Π_f. Although through the execution of this phase the result of the iteration is being sent out into membrane 1, they do not activate any rule in it.
- Phase 2: *Reseting the P system Π_f*

Phase 1 ends when some objects $\#_f$ get into membrane 1. Before we could compute another iteration of f, we need to erase all the objects left in the membranes of μ_{Π_f}. This is what we do in this stage, which start by reducing the objects $\#_f$ in membrane 1 to only one. Then the rule $\#_f \to (\oslash, in_{\sigma_f})$ in membrane 1 introduce an object $\oslash$ into the skin of μ_{Π_f}.
This object spreads to all the membranes just as $\ominus$ do in the previous phase, and leave one object $\otimes$ in each of them. This latter objects act as erasers, removing all the objects from the membranes by means of the rules $ob\otimes \to \otimes$. When a membrane has been emptied, the object $\otimes$ is expelled from it.
Therefore, this phase finishes when membrane 1 collects as many objects $\otimes$ as the degree of μ_{Π_f}. Only then the rules $\otimes^u b_i \to \otimes^u c_i$ can be applied, which transform the result of the iteration of f into input data for Π. Finally, the rule $\otimes^u \to \#$ is applied to start the process again.

When no object c_{m+1} is present in membrane 1, no more iteration has to be done. What is left is to send the objects c_1 to c_m of this membrane to the external environment, followed by the object $\#$.

Note that during the evaluation of the test, no rule can be applied in another membrane other than membrane 1, because they are empty. □

6 Diophantine sets

We introduce in this section the notion of diophantine set, which will help us to prove the universality of the model of computing P system with external output.

Definition 6.1. *A set of natural tuples, $A \subseteq \mathbb{N}^m$, is a diophantine set if there exists a polynomial $P(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{Z}[\boldsymbol{a}, \boldsymbol{x}]$ such that*

$$A = \{\boldsymbol{a} \in \mathbb{N}^m : \exists \boldsymbol{x} \in \mathbb{N}^n \big(P(\boldsymbol{a}, \boldsymbol{x}) = 0\big)\}$$

The following property is relatively easy to prove.

Proposition 6.2. *Every diophantine set is a recursively enumerable set.*

The main result about diophantine sets was obtained by Y. Matiyasevich from works by J. Robinson, M. Davis and H. Putnam, providing a negative solution for Hilbert's Tenth Problem.

Theorem 6.3 (MRDP [1]). *Every recursively enumerable set is a diophantine set.*

7 Generation of diophantine sets

First we need to introduce the concept of generation of a set by a P system.

Definition 7.1. *Let Π be a computing P system with external output of order (m, n).*

– *A set $A \subseteq \mathbb{N}^m$ is said to be partially generated by Π if this P system computes its partial characteristic function; that is, the function*

$$\mathcal{C}_A^*(k_1, \ldots, k_m) = \begin{cases} 1, & \text{if } (k_1, \ldots, k_m) \in A \\ \text{undefined}, & \text{otherwise} \end{cases}$$

– *A set $A \subseteq \mathbb{N}^m$ is said to be totally generated by Π if this P system computes its characteristic function.*

The main result of this paper is the following.

Theorem 7.2. *Every diophantine set is partially generated by a computing P system with external output.*

Before beginning with the proof, let us consider the following computing P systems with external output:

– P systems Π_n^{Id}, with $n \geq 1$:

$$\Sigma = \Lambda = (a_1, \ldots, a_n), \quad \mu_{\Pi_n^{Id}} = [\,_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{\#\}\}$$
$$R_1 = \{\# \to (\#, out)\} \cup \{a_i \to (a_i, out) : i = 1, \ldots, n\}, \quad \rho_1 = \emptyset$$

These P systems compute the identity functions, $Id^n : \mathbb{N}^n \to \mathbb{N}^n$, defined as $Id^n(k_1, \ldots, k_n) = (k_1, \ldots, k_n)$.

– P systems $\Pi_{n,j}^{proj}$, with $n \geq 1$ and $1 \leq j \leq n$:

$$\Sigma = (a_1, \ldots, a_n), \quad \Lambda = (a_j), \quad \mu_{\Pi_{n,j}^{proj}} = [\,_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{\#\}\}$$
$$R_1 = \{\# \to (\#, out), a_j \to (a_j, out)\}, \quad \rho_1 = \emptyset$$

These P systems compute the projection functions, $\Pi_j^n : \mathbb{N}^n \to \mathbb{N}$, defined as $\Pi_j^n(k_1, \ldots, k_n) = k_j$.

– P systems $\Pi_{n,c}^{const}$, with $n \geq 1$ and $c \in \mathbb{N}$:

$$\Sigma = (a_1, \ldots, a_n), \quad \Lambda = (b), \quad \mu_{\Pi_{n,c}^{const}} = [\,_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{b^c, \#\}\}$$
$$R_1 = \{\# \to (\#, out), b \to (b, out)\}, \quad \rho_1 = \emptyset$$

These P systems compute the constant functions, $\mathcal{C}_c^n : \mathbb{N}^n \to \mathbb{N}$, defined as $\mathcal{C}_c^n(k_1, \ldots, k_n) = c$.

– P systems $\Pi_2^{sum'}$, $\Pi_2^{prod'}$ and $\Pi_2^{expt'}$:
The first of these P systems is given by

$$\Sigma = \Lambda = (a_1, a_2), \quad \mu_{\Pi_2^{sum'}} = [\,_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{\#\}\}$$
$$R_1 = \{\# \to (\#, out), a_1 \to (a_1, out), a_2 \to (a_1, out)(a_2, out)\}, \quad \rho_1 = \emptyset$$

This P system computes the function $+' : \mathbb{N}^2 \to \mathbb{N}^2$, defined as $+'(k_1, k_2) = (k_1 + k_2, k_2)$.

The iteration of the P system $\Pi_2^{sum'}$ is a P system which computes the function $It(+') : \mathbb{N}^3 \to \mathbb{N}^2$ given by $It(+')(k_1, k_2, k_3) = (k_1 + k_2 k_3, k_2)$. Then, the P system $\Pi_2^{prod'} = C(It(\Pi_2^{sum'}); \Pi_{2,0}^{const}, \Pi_{2,2}^{proj}, \Pi_{2,1}^{proj})$ computes the function $*' : \mathbb{N}^2 \to \mathbb{N}^2$, defined as $*'(k_1, k_2) = (k_1 k_2, k_2)$.
The iteration of the P system $\Pi_2^{prod'}$ is a P system which computes the function $It(*') : \mathbb{N}^3 \to \mathbb{N}^2$ given by $It(*')(k_1, k_2, k_3) = (k_1 k_2^{k_3}, k_2)$. Then, the P system $\Pi_2^{expt'} = C(It(\Pi_2^{prod'}); \Pi_{2,1}^{const}, \Pi_2^{Id})$ computes the function $expt' : \mathbb{N}^2 \to \mathbb{N}^2$, defined as $expt'(k_1, k_2) = (k_1^{k_2}, k_1)$.

– P systems Π_n^{sum}, Π_n^{prod} and Π_2^{expt}:
We defined these P systems by recursion over $n \geq 2$.
$(n = 2)$

$$\Pi_2^{sum} = C(\Pi_{2,1}^{proj}; \Pi_2^{sum'})$$
$$\Pi_2^{prod} = C(\Pi_{2,1}^{proj}; \Pi_2^{prod'})$$
$$\Pi_2^{expt} = C(\Pi_{2,1}^{proj}; \Pi_2^{expt'})$$

$(n > 2)$

$$\Pi_n^{sum} = C(\Pi_{n-1}^{sum}; C(\Pi_2^{sum}; \Pi_{n,1}^{proj}, \Pi_{n,2}^{proj}), \Pi_{n,3}^{proj}, \ldots, \Pi_{n,n}^{proj})$$
$$\Pi_n^{prod} = C(\Pi_{n-1}^{prod}; C(\Pi_2^{prod}; \Pi_{n,1}^{proj}, \Pi_{n,2}^{proj}), \Pi_{n,3}^{proj}, \ldots, \Pi_{n,n}^{proj})$$

They compute, respectively, the n-ary sum function, the n-ary product function and the exponential function.

– P system Π_2^{dif}:

$$\Sigma = (a_1, a_2), \quad \Lambda = (b^+, b^-), \quad \mu_{\Pi_2^{dif}} = [\,]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{\#\}\}$$
$$(R_1, \rho_1) \equiv \{a_1 a_2 \to \lambda > \big(\# \to (\#, out), a_1 \to (b^+, out), a_2 \to (b^-, out)\big)\}$$

This P system computes the function $dif : \mathbb{N}^2 \to \mathbb{N}^2$ defined as $dif(k_1, k_2) = (\max(k_1 - k_2, 0), |\min(k_1 - k_2, 0)|)$.

Proof (of theorem 7.2). Given a polynomial $P(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{Z}[\boldsymbol{a}, \boldsymbol{x}]$, we construct a computing P system with external output which partially generates the diophantine set determined by this polynomial, in several steps:

1. Computing a monomial:
Let us denote $\Pi_{j,k}^{expt_i} = C(\Pi_2^{expt}; \Pi_{j,k}^{proj}, \Pi_{j,i}^{const})$, which computes the function $expt_i^{j,k} : \mathbb{N}^j \to \mathbb{N}$ given by $expt_i^{j,k}(a_1, \ldots, a_j) = a_k^i$.
Let $m(\boldsymbol{a}, \boldsymbol{x}) = c\, a_1^{i_1} \ldots a_m^{i_m} x_1^{j_1} \ldots x_n^{j_n}$, with $c > 0$, be a monomial of $P(\boldsymbol{a}, \boldsymbol{x})$. Then, the following computing P system with external output

$$\Pi_{c,i_1,\ldots,i_m,j_1,\ldots,j_n}^{mon} = C(\Pi_{m+n+1}^{prod}; \Pi_{m+n,c}^{const}, \Pi_{m+n,1}^{expt_{i_1}}, \ldots, \Pi_{m+n,m}^{expt_{i_m}}, \Pi_{m+n,m+1}^{expt_{j_1}}, \ldots, \Pi_{m+n,m+n}^{expt_{j_n}})$$

computes $m(\boldsymbol{a}, \boldsymbol{x})$, considered as a function from $\mathbb{N}^{m+n}$ to $\mathbb{N}$.

2. Computing the polynomial:
   Suppose that

$$P(\boldsymbol{a},\boldsymbol{x}) = \sum_{k=1}^{r_1} c_k a_1^{i_1^k} \cdots a_m^{i_m^k} x_1^{j_1^k} \cdots x_n^{j_n^k} - \sum_{k=r_1+1}^{r_1+r_2} c_k a_1^{i_1^k} \cdots a_m^{i_m^k} x_1^{j_1^k} \cdots x_n^{j_n^k}$$

   with $c_k > 0$ for every $k = 1, \ldots, r_1 + r_2$. Then, the following computing P system with external output

$$\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} = C\big(\Pi^{dif}_2; \Pi^{pol+}_{P(\boldsymbol{a},\boldsymbol{x})}, \Pi^{pol-}_{P(\boldsymbol{a},\boldsymbol{x})}\big)$$

   where

$$\Pi^{pol+}_{P(\boldsymbol{a},\boldsymbol{x})} = C\big(\Pi^{sum}_{r_1}; \Pi^{mon}_{c_1,\boldsymbol{i}^1,\boldsymbol{j}^1}, \ldots, \Pi^{mon}_{c_{r_1},\boldsymbol{i}^{r_1},\boldsymbol{j}^{r_1}}\big)$$
$$\Pi^{pol-}_{P(\boldsymbol{a},\boldsymbol{x})} = C\big(\Pi^{sum}_{r_2}; \Pi^{mon}_{c_{r_1+1},\boldsymbol{i}^{r_1+1},\boldsymbol{j}^{r_1+1}}, \ldots, \Pi^{mon}_{c_{r_1+r_2},\boldsymbol{i}^{r_1+r_2},\boldsymbol{j}^{r_1+r_2}}\big)$$

   computes $P(\boldsymbol{a},\boldsymbol{x})$, considered as a function from $\mathbb{N}^{m+n}$ to $\mathbb{N}$.
3. Computing the diophantine set:
   Considering that $\Sigma^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} = (d_1, \ldots, d_m, e_1, \ldots, e_n)$ and $\Lambda^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} = (b^+, b^-)$, let us define the computing P system with external output

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

   as follows:
   - $\Sigma = (d_1, \ldots, d_m)$.
   - $\Lambda = (b)$.
   - There exist distinct objects $\#', \#_1, \ldots, \#_n \in \Gamma \setminus \{\#, \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}\}$.
   - $\mu_\Pi = [_1[_2]_2 \cdots [_{n+1}]_{n+1} \mu_{\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}}]_1$, where the membranes from $\mu_{\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}}$ have been adequately renamed (and therefore, also the rules of $\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}$ have been adapted). We denote σ_{pol} the skin membrane of the latter.
   - $\iota = 1$.
   - $p = p^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} + n + 1$.
   - $\mathcal{M}_1 = \{\{\#\}\}, \mathcal{M}_2 = \{\{\#_1\}\}, \ldots, \mathcal{M}_{n+1} = \{\{\#_n\}\}$. All the remaining multisets are the empty one.
   - Evolution rules:
     - Rules for membrane 1:

$$\#_1 \ldots \#_n \# \to \#' > \# \to \# > > \begin{cases} d_i \to (d_i, in_{\sigma_{pol}}) & (i = 1, \ldots, m) \\ e_j \to (e_j, in_{\sigma_{pol}}) & (j = 1, \ldots, n) \\ \#' \to (\#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}, in_{\sigma_{pol}})(\ominus, in_{\sigma_{pol}}) \end{cases}$$

$$\left.\begin{matrix} b^+ \to b^+ \\ b^- \to b^- \end{matrix}\right\} > \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} \to \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} > \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} \to (b, out)(\#, out)$$

- Rules for membrane i, $2 \leq i \leq n+1$:

$$\#_{i-1} \rightarrow (\#_{i-1}, out)$$
$$\#_{i-1} \rightarrow \#_{i-1}(e_{i-1}, out)$$

- The rules for the remaining membranes are the same than in $\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}$.

Then Π is a valid computing P system with external output which computes the partial characteristic function of the diophantine set represented by $P(\boldsymbol{a}, \boldsymbol{x})$.
Indeed, the P system works as follows:

(a) First, a tuple $\boldsymbol{x}$ is nondeterministically generated from membranes 2 to $n+1$ into membrane 1.

(b) Second, the input objects d_i and the objects e_i which represent the previous tuple in membrane 1 are sent into the skin membrane of $\mu_{\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}}$ and the computation of P over the input $\boldsymbol{a}$ and the tuple $\boldsymbol{x}$ starts.

(c) Finally, if a non-zero result is obtained, then the computation enters an infinite loop: the rule $b^+ \rightarrow b^+$ or the rule $b^- \rightarrow b^-$ is applied once and again. If a zero result is obtained, then these rules cannot be applied, and an object b and an object # are sent out of the membrane structure.

□

8 Conclusions

We have studied in this paper the computing P systems with external output. This is a variant of the model of computation introduced in [5], which in turn is a variant of the basic model of transition P system introduced by G. Păun in [3]. 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 external environment.

We have defined two operations between computing P systems with external output: composition and iteration. These operations have allowed us to prove the universality of this model, by means of the MRDP theorem about diophantine sets.

References

[1] Y. Matiyasevich. *Hilbert's Tenth Problem.* The MIT Press, 1993.

[2] G. Păun. Computing with membranes. An introduction. *Bull. European Assoc. Theoret. Comput. Sci.*, 67:139–152, 1999.

[3] G. Păun. Computing with membranes. *J. Comput. System Sci.*, 61(1):108–143, 2000.

[4] G. Păun and G. Rozenberg. A guide to membrane computing. *Theoret. Comput. Sci.*, to appear.

[5] G. Păun, G. Rozenberg, and A. Salomaa. Membrane computing with external output. *Fund. Inform.*, 41(3):313–340, 2000.

[6] F. Sancho Caparrini. *Verificación de programas en modelos de computación no convencionales.* PhD thesis, Universidad de Sevilla, Departamento de Ciencias de la Computación e Inteligencia Artificial, 2002.

[7] The P Systems Web Page (`http://bioinformatics.bio.disco.unimib.it/psystems/`).