Computing partial functions by plasmids

José M. Sempere

Valencian Research Institute for Artificial Intelligence (VRAIN)
and
Valencian Graduate School and Research Network of Artificial Intelligence
(VALGRAI)
Universitat Politècnica de València, Spain
jsempere@dsic.upv.es

Abstract. Computing by plasmids is based on mobile genetic elements such as circular DNA biomolecules that can be transferred from one cell to the other, for example through bacterial conjugation. Recently, they have been proposed as a variant of membrane computing where plasmids are membranes that only contains rules and no object inside. In this work we propose different schemes of P systems to compute recursive functions defined by basic ones, and some functional operatos such as substitution, primitive recursion and unbounded minimization.

Keywords: Membrane Computing · Computing by plasmids · Recursive functions

Introduction

Membrane computing was proposed at the end of the last century by Gh. Păun [4]. It is a computational model inspired by the living eukariotic cell where the information is encoded through objects (inspired by biomolecules) that can be modified by applying rules (inspired by biochemical reactions). Furthermore, the physical structure of the organelles within the living cell marks out differentiated workspaces separated by membranes in which objects can be sent and received, inspired by the actual transport of materials inside and outside the cell. This highly parallel, non-deterministic and distributed model is able to efficiently solve complex problems of very diverse nature. The models to achieve membrane computing are named P systems.

Computing by plasmids is based on mobile genetic elements such as circular DNA biomolecules that can be transferred from one cell to the other, for example through bacterial conjugation [3]. Recently, they have been proposed as a variant of membrane computing where plasmids are membranes that only contain rules and no object is inside [8]. They have also been used in a recent work to simulate general and 2D P colonies [9].

In this work, we propose different schemes of P systems to calculate partial recursive functions defined by some basic functions, and some functional operators such as substitution, primitive recursion and unbounded minimization.

Basic concepts

In the following, we introduce basic concepts of computable functions from [1], and membrane computing and P systems from [7]. We assume that the reader is familiar with the basics of language theory and multisets.

Partial recursive functions

We introduce computable functions defined in the framework of general recursive functions. We refer to [1] for further details. All the functions are defined over natural numbers. Hence, any function f is defined such as $f : \mathbb{N}^k \to \mathbb{N}$.

We define the set of effectively computable functions by introducing the set of partial recursive functions. We define them through an inductive process.

First, we define a set of basic functions:

- The zero function: $z(n) = 0$
- The successor function: $succ(n) = n + 1$
- The projection functions: $f_i^k(n_1, \cdots, n_i, \cdots n_k) = n_i$ for $1 \leq i \leq k$

All the basic functions are primitive recursive functions. Now, we can introduce two operators over functions as follows:

- Substitution
  Let $f : \mathbb{N}^l \to \mathbb{N}$ and $g_i : \mathbb{N}^k \to \mathbb{N}$ for $1 \leq i \leq l$ be primitive recursive functions. Then, the function $h : \mathbb{N}^k \to \mathbb{N}$ defined as

$$h(n_1, \cdots, n_k) = f(g_1(n_1, \cdots, n_k), \cdots, g_l(n_1, \cdots, n_k))$$

  is also a primitive recursive function.
- Primitive recursion
  Let $f : \mathbb{N}^{k+2} \to \mathbb{N}$ and $g : \mathbb{N}^k \to \mathbb{N}$ be primitive recursive functions. Then, the function $h : \mathbb{N}^{k+1} \to \mathbb{N}$ defined as

$$h(0, n_1, \cdots, n_k) = g(n_1, \cdots, n_k)$$

$$h(n+1, n_1, \cdots, n_k) = f(n, h(n, n_1, \cdots, n_k), n_1, \cdots, n_k)$$

  is also a primitive recursive function.

The family of primitive recursive functions is defined inductively as the set of basic functions, and all the functions that can be defined by a finite number of applications of substitution and primitive recursion operators over primitive recursive functions.

The set of computable functions is not covered by the set of primitive recursive functions. For example, Ackermann function, defined as

$$A(m, 0) = m + 1$$

$$A(0, n+1) = A(1, n)$$
$$A(m+1, n+1) = A(A(m, n+1), n)$$

is not primitive recursive.

In order to have a complete definition for effectively computable functions, we must introduce a new operator to define the partial recursive functions. So, we introduce the minimization operator $\hat{\mu}$ defined as follows.

- $\hat{\mu}$ minimization (unbounded minimization)
  Let $g : \mathbb{N}^{k+1} \to \mathbb{N}$ be a function. Then we can define the function $f : \mathbb{N}^k \to \mathbb{N}$ as follows

  $$f(n_1, \cdots, n_k) = \hat{\mu}m[g(m, n_1, \cdots, n_k)]$$

  where $\hat{\mu}m[g(m, n_1, \cdots, n_k)] = m_0 \Leftrightarrow g(m_0, n_1, \cdots, n_k) = 0$ and $\forall m < m_0$ $g(m, n_1, \cdots, n_k) \neq 0$

The family of partial recursive functions is defined inductively as the set of basic functions, together with all the functions that can be defined by a finite number of applications of substitution, primitive recursion and unbounded minimization over partial recursive functions. The set of recursive functions is the set of total functions that are partial recursive.

In the following, we will work with the set of partial recursive functions.

P systems

Definition 1. *A* cell-like *transition P system of degree m is a construct*

$$\Pi = (V, \mu, w_1, \cdots, w_m, (R_1, \rho_1), \cdots, (R_m, \rho_m), i_0),$$

where:

- *V is the alphabet of objects;*
- *μ is a membrane structure consisting of m membranes labeled in a one-to-one manner with the natural numbers $\{1, .., m\}$. The outermost membrane is called the skin membrane;*
- *w_i, $1 \leq i \leq m$, is a string representing a multiset over V associated to the region i of μ;*
- *R_i, $1 \leq i \leq m$, are finite sets of evolution rules over V associated with the regions of μ; the evolution rules are in one of the forms $u \to v$ or $u \to v\delta$, where u is a multiset over V and v is a string from $(V \times \{here, out, in_k : 1 \leq k \leq m\})^*$ that denotes a multiset with target addressings.*
  In the rest of the work, we will omit the addressing $'here'$, so that the symbol a in the right-hand side of any rule will denote a_{here};
  The δ symbol is used to define dissolution rules where the membrane at region i disappears after the application of the rule. In such a case, all the rules of R_i dissapear. Observe that the dissolution rules cannot be applied in the skin membrane.

- ρ_i, $1 \leq i \leq m$, *is a partial order in the rules of* R_i *that denotes the priorities between the rules*
- $i_0 \in \{1, ..., m\} \cup \{\infty\}$ *specifies the output membrane of* Π *(in the case that it equals to* ∞*, the output is read outside the system).*

A configuration of the system consists of the membrane structure and the multisets of objects at every region. The change of a configuration of the system is obtained by applying the rules at every region, for example in a maximally parallel manner [2]. That is, the maximum number of rules that can be applied (with or without repetition) are applied at each computation step. The system halts when no rule can be applied in any of the region. A computation of the system is a finite sequence of configurations that starts from the initial configuration. The result of a computation can be considered as the number of objects in the output region whenever the system halts.

It is a well known result that transition P systems are universal and complete models of computation (given that they are equivalent to Turing machines and register machines). Furthermore, the P systems solve efficiently problems that are catalogued as difficult or intractable problems in the computational complexity theory [7] (they solve NP-complete problems in polynomial time).

Computing by plasmids

In this section, we introduce plasmids as mobile agents that change the behavior of local regions of P systems. In the context of P systems, we consider that plasmids are membranes that only contain rules (there are no objects inside them). They can move throughout the space of regions of the P system by using predefined rules. Whenever a plasmid enters into a region, all its rules are inherited by such a region and its rules compete for the objects at the same level as the rest of the rules that were defined in that region. In addition, there are rules that can move the plasmids by using communication rules as in the classical transition P systems.

In the following, we provide a formal definition of P systems with plasmids.

Definition 2. *A* cell-like *transition P system of degree m with q plasmids and input is a construct*

$$\Pi = (V, H, \mu, w_1, \cdots, w_m, R_o, R_p, \rho, z_1, \cdots, z_m, z_E, p_1, \cdots, p_q, i_i, i_0),$$

where:

- V *is the alphabet of objects;*
- H *is the alphabet of membrane labels;*
- μ *is a membrane structure;*
- w_i, $1 \leq i \leq m$, *is a string representing a multiset over* V *associated to the region* i *of* μ*;*
- R_o *is a finite set of rules for objects and membranes, defined in the same way as the sets* R_i *in Definition 1. Observe that every rule can be referred to a different region by labeling its corresponding membrane;*

- R_p *is a finite set of rules for plasmid mobility, plasmid replication, and plasmid dissolution of the following types*
  1. *in-symport movement*
     $p_i u[\ ]_k \to v[\ p_i\ ]_k$ *where* p_i *is a plasmid with* $1 \leq i \leq q$, *and* $u, v \in V^*$
  2. *out-symport movement*
     $[p_i u]_k \to p_i[v]_k$ *where* p_i *is a plasmid with* $1 \leq i \leq q$, *and* $u, v \in V^*$
  3. *antiport movement*
     $p_i u[p_j v]_k \to p_j w[p_i x]_k$ *where* p_i, p_j *are plasmids with* $1 \leq i, j \leq q$, *and* $u, v, w, x \in V^*$
  4. *replication*
     $[p_i u]_k \to [p_i p_i v]_k$ *where* p_i *is a plasmid with* $1 \leq i \leq q$, *and* $u, v \in V^*$
  5. *dissolution*
     $[p_i u]_k \to [v]_k$ *where* p_i *is a plasmid* $1 \leq i \leq q$, *and* $u, v \in V^*$
- ρ *is a partial order over the rules from* R_o *and* R_p *that denotes the priorities;*
- $z_1, z_2, ..., z_m$ *are the initial multiset of plasmids at every region in* μ*;*
- z_E *is the initial multiset of plasmids in the environment;*
- $p_1, p_2, \cdots, p_q$ *are plasmids, where every plasmid is defined by a pair* (R_{p_i}, ρ_{p_i}) *such that* R_{p_i} *is a finite set of evolution rules of the form* $u \to v$ *where* $u \in V^+$, *and* $v \in (V \times \{here, out, in_k : 1 \leq k \leq m\})^*$, *and* ρ_{p_i} *is a partial order over the rules in* R_{p_i} *that denotes the priorities*
- $i_i \in \{1, ..., m\}$ *is the input membrane of* Π
- $i_0 \in \{1, ..., m\} \cup \{\infty\}$ *is the output membrane of* Π *(in the case that it equals to* ∞*, the output is read outside the system)*

We make some remarks to the previous definition:

1. when a dissolution rule is applied within a region, it causes all plasmids within that region to disappear (they are not moved to the upper region)
2. plasmid rules affect only to objects (they do not affect to other plasmids or membranes).
3. plasmid replication rules are applied in a minimal parallel manner: the rules are applied only to one plasmid regardless of how many copies of that plasmid there are in the region. Therefore, they only produce one copy of every plasmid at every computational step.

The system configuration takes into account not only the objects in each region but also the plasmids that it contains. The change of a configuration is carried out similarly to that in a cell-like transition P system without plasmids. However, when applying rules to objects, the rules of the plasmids compete for objects at the same level as the rest of the rules. The system halts when no rule can be applied within the system, including plasmid mobility rules and the rules contained within each plasmid. The result of the computation can be considered the number of objects found in the output region when the system halts. Note that, in this sense, plasmids are not objects.

In Fig. 1, we show a scheme of a P system with plasmids. Observe that red circles are plasmids and they are membranes that contain only rules and do not contain objects.

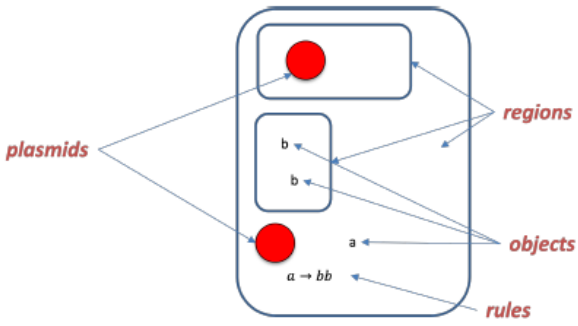

Fig. 1. A scheme of a P system with plasmids

The family of P systems of degree m with q plasmids is denoted by Ppl_m^q. It has been shown in [8] that for every set A in NRE (that denotes the sets of numbers that can be computed) there exist positive integers $m, q > 0$ such that A can be calculated by a P system in Ppl_m^q. Hence, P systems with plasmids are complete models of computation.

Computing partial functions by plasmids

In the following, we work with partial recursive functions, as previously defined. The natural number k will be encoded as the multiset a^k. For any function with p arguments $n_1, n_2, \cdots, n_p$, we will use a multiset support within the alphabet $\{a_1, a_2, \cdots, a_p\}$, and we will encode it as the multiset $a_1^{n_1} a_2^{n_2} \cdots a_p^{n_p}$.

For any function f such that $f(n_1, n_2, \cdots, n_p) = k$, a P system Π computes f if when given the function arguments encoded in the input membrane, the system makes the appropriate transitions so that when it halts, the output membrane has the function value correctly encoded as a^k. If the output membrane is the environment, then the system has sent out a number of predefined objects from the system, corresponding to the function value, that is k.

Observe that the calculation of partial functions has already been addressed for the case of transition P systems [5], and also for virus machines [6]. In our proposal, we will define P systems that only use plasmids and plasmid mobility rules throughout the membrane space. That is, in each membrane there are no other rules than those that only serve to mobilize the plasmids. In addition, we will use two designed symbols # and $ to denote that the plasmid starts and ends its execution, respectively.

P systems for basic functions

– A plasmid for the zero function $z(n) = 0$ is proposed through the following rules:

$r_1 : \#a \to \#$
$r_2 : \# \to \$$
and $\rho_{zero} : r_1 > r_2$.
So, $p_{zero} = (\{r_1, r_2\}, \rho_{zero})$. Then, the P system to compute the zero function is defined as follows:

$$\Pi_{zero} = (\{a, \#, \$\}, \{1\}, [\ ]_1\ , a^n\#, \emptyset, \emptyset, \rho, p_{zero}, \emptyset, p_{zero}, 1, 1)$$

In this case, the relation ρ is empty.

- A plasmid for the successor function $succ(n) = n + 1$ is performed through the rule $r : \#a \to \$aa$. Observe that the 0 value can be encoded by the distinguished symbol @, and it can be incremented by the rule $r' : @ \to \$a$. So, $p_{succ} = (\{r, r'\}, \emptyset)$, and the P system is defined as follows:

$$\Pi_{succ} = (\{a, \#, \$, @\}, \{1\}, [\ ]_1\ , a^n\#, \emptyset, \emptyset, \rho, p_{succ}, \emptyset, p_{succ}, 1, 1)$$

where the relation ρ is empty.

- A plasmid for every projection function $f_k(n_1, \cdots, n_k, \cdots, n_p) = n_k$ can be defined through the following rules:
$r_i : a_i \to \lambda$ for every $1 \leq i \leq p$ and $i \neq k$
$\cdots$
$r_k : \#a_k \to \$a_k$
So, $p_{pr_k} = (\{r_1, r_2 \cdots, r_k, \cdots r_p\}, \emptyset)$, and the P system is defined as follows

$$\Pi_{pr_k} = (\{a_1, a_2, \cdots, a_p, \#, \$\}, \{1\}, [\ ]_1\ , w_1, \emptyset, \emptyset, \rho, p_{pr_k}, \emptyset, p_{succ}, 1, 1)$$

with $w_1 = a_1^{n_1} a_2^{n_2} \cdots a_p^{n_p} \#$. Observe that for any zero value, taken as a function argument, we could provide additional symbols. For example, if $n_j = 0$, then this value could be encoded as $@_j$, and a rule to eliminate it is $@_j \to \lambda$, in addition the rule $\#@_k \to \$@_k$ should be added.
As in the previous cases, the relation ρ is empty.

A P system for substitution

Let us consider the function $f(n_1, \cdots, n_p) = h(f_1(n_1, \cdots, n_p), \cdots, f_k(n_1, \cdots, n_p))$, and the P systems with plasmids $\Pi_h, \Pi_{f_1}, \Pi_{f_2}, \cdots, \Pi_{f_k}$ that compute their corresponding functions.

A P system with plasmids, Π_f for the function f can be defined as follows (we will overview the definition without going into details):

- The membrane structure $\mu_f = [\ [\ ]_{input}\ \mu_h\ \mu_{f_1}\ \cdots \mu_{f_k}\ [\ ]_{aux}]_0$
- In the input membrane the input parameters are encoded as $a_1^{n_1} a_2^{n_2} \cdots a_p^{n_p}$
- There are designated plasmids that activate the execution at every membrane subsystem $\Pi_h, \Pi_{f_1}, \Pi_{f_2}, \cdots, \Pi_{f_k}$
- There are plasmids that transform the output of every subsystem Π_{f_j} as an encoded value a_j^m

- There are mobility rules for plasmids such that the computation of f is made through the following scheme:
  1. In an increasing order
     (a) Move the input values to the subsystem Π_{f_i}
     (b) Calculate f_i by Π_{f_i}
     (c) Encode the output j as a_i^j in the membrane aux
  2. Move the values $a_1^{j_1} a_2^{j_2} \cdots a_k^{j_k}$ from the membrane aux to the subsystem Π_h
  3. Move the output value of Π_h to the output membrane

The computation of the function f is carried out by using a sequential strategy. That is, first the functions f_j are calculated sequentially and, finally, the function h is computed.

A P system for primitive recursion

Let f and g be functions computed by the P systems Π_f and Π_g. Then, the function h defined as

$$h(0, n_1, \cdots, n_k) = g(n_1, \cdots, n_k)$$

$$h(n+1, n_1, \cdots, n_k) = f(n, h(n, n_1, \cdots, n_k), n_1, \cdots, n_k)$$

can be computed by a P system with plasmids Π_h (we will overview the definition without going into details):

- The membrane structure $\mu_h = [\,[\,]_{input}\ \mu_g\ \mu_f\ [\,]_{aux}\ [\,]_{counter}\ [\,]_{comparator}\,]_0$
- In the input membrane, the input parameters are encoded as $a_0^n a_1^{n_1} a_2^{n_2} \cdots a_k^{n_k}$
- In the *counter* membrane, the initial value is zero, encoded by the distinguished symbol @, and the k value is encoded as a^k
- There are designated plasmids that activate the execution at both membrane subsystem Π_g, Π_f
- There are plasmids that transform the output of every subsystem Π_f and Π_g as encoded values a_f^m and a_g^n
- There are mobility rules for plasmids such that the computation of h is made through the following scheme:
  1. If the content of the *counter* membrane is zero then calculate the function g by Π_g, and move the output to membrane aux as an encoded value
  2. Increase the *counter* value by plasmid p_{succ}
  3. Compare the *counter* value with the first parameter of h, n, in membrane *comparator*. If the counter is less than n then
     (a) Move the input parameters $n_1, \cdots, n_k$, the counter value and the value in the membrane aux to the subsystem Π_f
     (b) Calculate function f in Π_f
     (c) Move the output to the aux membrane

     Otherwise go to step 5
  4. Go to step 2

5. Move the output value to the output membrane

The computation of the function h is carried out by using an iteration strategy. First, the base case of recursion is calculated through g. Then, the recursion variable is iteratively increased and the subsequent functions are calculated according to the recursive scheme through f.

A P system for unbounded minimization

Let g be a function, and we define the function f as follows

$$f(n_1, \cdots, n_k) = \hat{\mu}m[g(m, n_1, \cdots, n_k)]$$

where $\hat{\mu}m[g(m, n_1, \cdots, n_k)] = m_0 \Leftrightarrow g(m_0, n_1, \cdots, n_k) = 0$ and $\forall m < m_0$ $g(m, n_1, \cdots, n_k) \neq 0$

Let us assume that the P system Π_g calculates the function g. Then a P system with plasmids Π_f can be defined to compute f as follows (we will overview the definition without going into details):

- The membrane structure $\mu_f = [\ [\ ]_{input}\ \mu_g\ [\ ]_{aux}\ [\ ]_{counter}\ ]_0$
- In the input membrane, the input parameters are encoded as $a_1^{n_1} a_2^{n_2} \cdots a_k^{n_k}$
- In the *counter* membrane, the initial value is zero and the k value is encoded as a^k
- There is a designated plasmid that activates the execution of the membrane subsystem Π_g
- There is a plasmids that transforms the output of the subsystem Π_g as an encoded value a_g^n
- There are movility rules for plasmids such that the computation of f is made through the following scheme:
  1. Set the *counter* membrane to zero encoded as @
  2. Move the input parameters $n_1, \cdots, n_k$, and the counter value to the subsystem Π_g and then calculate the function g by Π_g
  3. Move the output to membrane *aux* as an encoded value
  4. Check if the content of membrane *aux* is zero. If so, move the counter value to the output membrane and finish.
  5. Increase the *counter* value by plasmid p_{succ} and repeat the following steps:
     (a) Move the input parameters $n_1, \cdots, n_k$ and the counter value to the subsystem Π_g
     (b) Calculate function g in Π_g
     (c) Move the output value to the *aux* membrane
  6. go to step 4
  7. Move the output value to the output membrane

As in the case of primitive recursion, the computation of the function f is carried out by using an iteration strategy. First, a counter is set to zero, and it is iteratively increased to find the first value that makes the function equal to zero. Observe that, for the case that f has no value equals to zero, the proposed system never halts. Hence, the function is undefined and partial.

Conclusions

In this work, we have presented several P systems with plasmids to calculate partial recursive functions in the framework of membrane computing, by adjusting the P system schemes to the functional definitions. This is a first approximation that can probably be improved from the computational complexity point of view, by transforming some of the sequential and iterative schemes into purely parallel schemes, maybe by introducing membrane creation and duplication rules. We will explore these aspects in future works.

Acknowledgments. This work has received funding from the Generalitat Valenciana within the Prometeo program in the project *"A disruptive way to ameliorate the diagnosis and treatment of sensorineural diseases."* CIPROM/2023/026.

References

1. Barry Cooper, S.: Computability Theory. Chapman & Hall/CRC Mathematics (2004)
2. Freund, R.: How derivation modes and halting conditions may influence the computational power of P systems. Journal of Membrane Computing **2**, pp 14–25 (2020).
3. Novick, R.P. : Plasmids. Encyclopedia of Life Sciences. John Wiley & Sons (2001)
4. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61(1)**, pp 108–143 (2000)
5. Romero-Jiménez, A., Pérez-Jiménez, M. J.: Computing Partial Recursive Functions by Transition P systems. In Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.), International Workshop WMC 2003, LNCS Vol. 2933, pp 320–340, Springer (2004)
6. Romero-Jiménez, A., Valencia-Cabrera, L., Riscos-Núñez, A., Pérez-Jiménez, M. J.: Computing partial recursive functions by virus machines. In Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron C. (eds.), 16th International Conference CMC, LNCS Vol. 9504, pp 353–368, Springer (2015)
7. Păun, Gh., Rozenberg, G., A. Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, NY (2010)
8. Sempere, J.M.: Computing by plasmids. In 21st Brainstorming Week on Membrane Computing (BWMC) (*submitted*) (2025).
9. Sempere, J.M.: On Some Relationships Between P Colonies and Computing by Plasmids. In: Jiménez López, M.D., Vaszil, G. (eds) Languages of Cooperation and Communication. LNCS vol 15840, pp 252-266, Springer, Cham. (2025)