Journal of Membrane Computing (2020) 2:95–107
https://doi.org/10.1007/s41965-020-00037-z

REGULAR PAPER

Tissue P systems with promoter simulation with MeCoSim and P-Lingua framework

Luis Valencia-Cabrera[1] · **Bosheng Song**[2]

Received: 7 December 2019 / Accepted: 13 March 2020 / Published online: 6 April 2020

Abstract
Tissue P systems constitute a well-known class of computing models within membrane computing. Inspired by the information exchange among cells and with the environment, many interesting variants emerged along the years, attracting significant attention. One of such variants, tissue P systems with promoters, was proved to be Turing-universal (even when restricting to a very limited number of elements) and able to solve NP-complete problems. On the other hand, P-Lingua framework provides useful tools to model, debug and simulate different types of P systems with a special relevance of the P-Lingua language, pLinguaCore library and MeCoSim environment. This work presents new features introduced in the framework to cover functionalities associated with tissue P systems with promoters, including extensions of the language, variants of tissue models and their simulators within P-Lingua version of MeCoSim. The new elements are described in detail, and the use of the tools is described through basic examples. Besides, a solution for SAT is experimentally validated using the developed software.

Keywords Membrane computing · tissue P system · NP-complete problem · P-Lingua · MeCoSim

1 Introduction

Membrane computing [14, 16] has been providing new computing models for 2 decades, inspired from features observed in the structure and functioning of living cells, tissues or neurons. Many of the computational devices defined, generically called P systems, have proved their computational completeness (Turing-universal) [28, 29] Besides, a number of them have shown their ability to solve computationally hard problems as SAT [11, 13, 15, 24], HAM-CYCLE [20] or 3-COL [1], among others [12, 22– 24].

One of the groups of P systems attracting more attention over the years has been tissue P systems, inspired by the way cells organize and communicate in tissues [9]. Many variants of these systems exist, with ability to solve challenging problems [26]. In particular, in 2016, tissue P systems with promoters were proved to be both Turing-universal and able to solve SAT [24, 25], probably the best-known computationally hard problem.

In any kind of P system, when defining a new computing model, it can possibly include interesting specific syntactic features, semantic restrictions, dynamic aspects or subtleties to consider. This implies that it might be worth studying different properties the new variants will offer. Thus, computability and complexity properties are usually explored for different variants of the models. Besides, for those providing solutions to hard problems, the designs proposed can be too complex and tedious to follow the evolution of the systems *without some external help*.

Therefore, this seems the perfect context to make use of helpful tools making our lives easier when debugging new models and simulating them under certain inputs or scenarios of interest, both for theoretical and practical applications [30]. However, it is a time-consuming task to develop such kind of tools for every model we can conceive [6, 27]. Besides, for every variant proposed new elements can be present in many aspects, so despite having rich frameworks covering many types of P systems, the compliance for future models cannot be ensured.

✉ Bosheng Song
boshengsong@hnu.edu.cn

Luis Valencia-Cabrera
lvalencia@us.es

[1] Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Sevilla, Spain

[2] College of Information Science and Engineering, Hunan University, Changsha, China

To overcome the highlighted limitation, significant efforts have been made in the membrane computing community, trying to cover as many features as possible in a generic way, with different approaches. Thus, P-Lingua framework [4] includes many general features as a common language and some shared features for all the models, but cannot cover by default (in the last official version 4.0, nor in the last version included in MeCosim) new variants unless some further development is made.

On the other hand, UPSimulator [5] provides a set of features of P systems that can be generically combined in a flexible way, thus allowing the use of new models not studied so far, by the combination of syntactic and semantic ingredients. This interesting software provides a significant contribution for the syntactic parsing and the simulation of P systems. However, it must pay a price for not restricting to the highly specific semantic peculiarities of each P system variant. This goal (i.e., taking into account fine-grained subtle aspects of each variant) was present from the beginning in the former P-Lingua, to control that the proper specification of each variant is absolutely respected by the developer and the P system designer providing a solution for a given problem.

To provide general functionalities for computing models within the field of Membrane Computing, P-Lingua framework [4, 7, 8, 34] including MeCoSim environment [17, 33] serve the double purpose of (1) a rich set of features for modeling, debugging, simulation, and final apps development, among others; (2) the strict control, for the type and variant of P system used, of all the syntactic, semantic and dynamic aspects defined within the computing model. We consider at the same level the flexibility of the tools and the fidelity to the rules of the game defined with any new computing model. Thus, the framework can control the use that any P system designer can make of the elements present in a solution given within a specific framework.

The present work extends the functionality provided by P-Lingua framework (specifically, the P-Lingua version—language and library—running inside MeCoSim environment) with all the features required to work with tissue P systems with promoters. These features must include, at least (1) the specification in the P-Lingua language of the ingredients trying to mimic the syntax of the promoters appearing in papers studying their computational completeness and complexity [24]; (2) the semantic restrictions imposed on different variants in terms of the types of rules allowed within the model; (3) the specific simulator following the dynamics described in the definition of the system [24]; (4) the connection with all the pre-existing generic tools within the framework provided by P-Lingua and MeCoSim.

The paper is organized as follows: in Sect. 2, tissue P systems with promoters are defined; following, in Sect. 3, the main elements of the framework given by P-Lingua and MeCoSim are recalled; then the new syntactic ingredients introduced in the P-Lingua language to define tissue P systems with promoters is presented in Sect. 4, along with a brief description of the simulator adapted to work with promoters; finally, the work with tissue P systems in our platform is illustrated in Sect. 5, with simple examples plus the translation of the solution to SAT in P-Lingua, taken from [24].

2 Tissue-like P systems with promoters

This section introduces the computing model of tissue P systems with promoters, subject of the present paper and the simulator developed. More specifically, the type of devices covered is tissue P systems with promoters and cell division, thus incorporating the division rules to the basic tissue P systems with promoters.

2.1 Recognizer tissue P systems with promoters and cell division

In what follows, the definition of our devices is recalled, as appeared in [24].

Definition 1 A recognizer tissue P system with promoters and cell division, of degree $q \geq 1$, is a tuple

$$\Pi = (\Gamma, \Sigma, \mathcal{E}, w_1, \ldots, w_q, \mathcal{R}, i_{in}, i_{out}),$$

where

- Γ, Σ are alphabets of objects, with Σ strictly contained in Γ;
- $\mathcal{E} \subseteq \Gamma$ is an alphabet;
- $w_i, 1 \leq i \leq q$, are finite multisets over $\Gamma \setminus \Sigma$;
- $\mathcal{R}$ is a finite set of rules of the following forms:
  - Communication rules:
    - Symport rules: $(pro \mid i, u/\lambda, j)$ or $(pro \mid i, \lambda/u, j)$, where $0 \leq i \neq j \leq q$, with pro, u being finite multisets over Γ and $u \neq \lambda$;
    - Antiport rules: $(pro \mid i, u/v, j)$, where $0 \leq i \neq j \leq q$, with pro, u, v being finite multisets over Γ and $u, v \neq \lambda$;
  - Division rules:
    - $[a]_i \rightarrow [b]_i[c]_i$, where $i \in \{1, \ldots, q\}$, $i \neq i_{out}$, $a, b, c \in \Gamma$;
- $i_{in} \in \{1, \ldots, q\}$;
- $i_{out} = 0$;
- all computations halt;
- if $\mathcal{C}$ is a computation of Π, then either object `yes` or object `no` (but not both) must be released into the envi-

ronment (labeled by 0), precisely at the last step of the computation.

As also detailed in [24], a tissue P system with promoters and cell division of degree $q \geq 1$ can be viewed as a set of q cells labeled by $1, \ldots, q$, providing the nodes of a directed graph; $w_1, \ldots, w_q$ represent the finite multisets of objects initially placed in these q cells; the important alphabet $\mathcal{E}$ contains the set of objects initially located in the environment of the system, which will be available in an arbitrary number of copies; it is contained in the working alphabet Γ, as it also does the input alphabet Σ. $\mathcal{R}$ is the finite set of rules of the system, including symport/antiport rules operating on *pro*, u, v (finite multisets over the working alphabet), plus division rules (defined as usual); i_{out} is a distinguished region that will hold the output of the system.

A *configuration* of this kind of system is described by the multisets of objects over Γ associated with all cells in the system, plus the multiset of objects over $\Gamma \backslash \mathcal{E}$ placed in the environment at that moment (as the objects from $\mathcal{E}$ are present in the environment in an arbitrarily large number of copies, the specific number of these objects is not relevant along the computation, considering they will be always enough to trigger the proper communication rules with the cells interacting with it). For each multiset m over the input alphabet Σ, the initial configuration of the system Π with input m is the following: $(\mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} + m, \ldots, \mathcal{M}_q)$. Any computation of the system Π with input m starts from such configuration, is a halting computation, and either object `yes` or object `no` (but not both) must appear in the environment at the last step.

Despite the fact that the applicability and effect produced by the different types of rules is properly described in [24], as it is crucial for the understanding of the semantics of the system, we will include such information in this preliminary section to avoid the reader the need of such additional access to the corresponding sources, thus making the reading of this document mostly self-contained. Therefore, in what follows, we keep describing the semantic aspects of the system:

- A symport rule $(pro \mid i, u/\lambda, j) \in \mathcal{R}$ is applicable to a configuration if region i contains multiset u and the objects of the promoter *pro* are present in such i. Then if such a rule is applied, u is sent from region i to j. Please note that other variant of the rule, $(pro \mid i, \lambda/u, j)$, would behave similarly.
- An antiport rule $(pro \mid i, u/v, j) \in \mathcal{R}_i$ is applicable to a configuration if region i contains multiset u, region j contains multiset v, and the objects of the promoter *pro* are present in i. Thus, when it is applied, the objects of u are sent from i to region j, in exchange of objects of v, which are sent from j to i.
- A division rule $[a]_i \rightarrow [b]_i[c]_i$ is applicable to a configuration if the cell i contains object a, and i is not the output cell. When such a rule is applied, i is divided into two cells (*with the same label*): in the first one, a is replaced by object b, and in the second one by object c; the rest of objects present in the original cell are all kept (replicated) in the two new cells.

Let us recall that, in a tissue P system with promoters and cell division, the presence of the promoter objects enables the use of the associated rule *as many times as possible*, without any restriction; consequently, a promoter can be used simultaneously by any number of rules associated with this promoter, which is not consumed nor blocked by the rule. Let us note that, if the rule does not require any promoter, the initial part of the rule is omitted, writing the rule as $(i, u/\lambda, j)$ (resp., $(i, u/v, j)$) instead of $(\emptyset \mid i, u/\lambda, j)$ (resp., $(\emptyset \mid i, u/v, j)$).

Regarding the dynamics of the system (*i.e.*, the execution strategy), the rules of these systems are applied in a maximally parallel manner, in the following sense: at any given instant, the multiset of rules selected to be applied in the whole system at that step cannot be extended to include any further rule unless some applicability condition on the rule is violated. It is worth recalling that, to check for the applicability of a rule, the corresponding objects should be available to use by each rule, the promoter conditions should be present, and the constraints imposed by each rule type must be respected. This last condition implies that while many symport/antiport rules can simultaneously affect the same cell (assuming the previous conditions apply), division rules are more restrictive in their applicability. Thus, if a division rule is being applied at a given instant, this rule will be the only one to be applied affecting such cell at that step.

The concepts of computation following the sequence of configurations, and the halting configuration are the usual ones in membrane systems. Concerning the result of the halting computations, it will be encoded by the number of objects present in the output region in the halting configuration.

3 P-Lingua framework for P systems

Along the last 2 decades, many types and variants of P systems have been defined, providing computing theoretical devices showing interesting properties. However, no physical implementation of such machines exists so far, and the solutions based on such systems are difficult to work with at certain level in a manual way.

In this context, having at disposal automatic tools to support the design, debug, analysis and simulation of these novel solutions may result crucial. In this sense, P-Lingua

[4, 18] provides a uniform framework for the specification, debugging and simulation of this kind of computing models. Additionally, on top of P-Lingua, MeCoSim [17, 27] provides a user-friendly environment using the previous core to parse and simulate models (for technical users), along with a higher level interface to handle models and deliver end-user applications based on this framework (for end users, not requiring knowledge about P systems).

This section outlines the main elements included in the framework provided by P-Lingua and MeCoSim.

3.1 P-Lingua framework

The main components of *P-Lingua* framework [3] were initially a specification language to define P systems, and a Java [31] library called *pLinguaCore*, including parsers and simulators for different models within membrane computing. Later on, additional types of P systems have been covered, along with additional features of the language. Along with these main elements, different command-line tools were provided from the beginning.

P-Lingua is intended to be a *standard* language for the definition of P systems through simple *text files*, easily processed by pLinguaCore directly or using some command-line tool or visual client (as MeCoSim, described in Sect. 3.2). These files can specify P systems of families of them, depending on certain parameters included in the files (so that different values for the parameters instantiate different members of the family). The common syntax shared by many kinds of P systems including membrane structures, initial multisets, some types of rules, etc. decreases the learning curve for P system designers learning different computing models within membrane computing.

Figure 1 shows a small specification in the P-Lingua language, for the definition of a simple P system. In this case, we are showing a tissue P system with promoters, using some new features introduced in this work. However, many elements (as the specification of the model, the definition of functions, the membrane structure or the initial multisets) take advantage of the general features provided by the language. This implies that someone familiar with P-Lingua and transition P systems, for instance, can understand most of the code, without prior experience with tissue P systems with promoters and cell division.

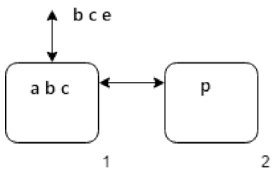
Apart from the language, as mentioned above, P-Lingua framework includes the library `pLinguaCore`, providing *parsers* and *simulators* for the variants of P systems supported by the language, plus other useful tools. A detailed explanation can be found in [3, 4], and further information on a deeper level is given in [18] (this last one in Spanish).

3.2 MeCoSim (membrane computing simulator)

MeCoSim provides visual tools to manage membrane computing models using P-Lingua. P system designers may explore the models as white boxes to deepen in the study of the P systems themselves, while end users (possibly unrelated to membrane computing) can use them as black boxes to focus on the problems, abstracting from the internal details of the P systems actually solving such problems.

On the one hand, model designers find in MeCoSim a graphic tool to design, simulate, analyse and verify their models. On the other hand, end users receive MeCoSim-based applications (customized by the designers) adapted to their problems, where they simply enter the input data and check the results. As previously introduced, MeCoSim is built on top of P-Lingua: models are specified in the P-Lingua language, processed by their parsers, and then the simulations are performed by executing simulation algorithms provided by pLinguaCore library, or through external simulators connected to MeCoSim.

In this visual environment, there exists a general application where any specific *single P system* written in P-Lingua can be processed, from the load and parsing to its simulation. However, if we want to have custom applications where the end user can introduce data to instantiate some member of a family of P systems and/or providing the external input to the system, the designer can configure a custom application



$$r_1 : (1, a/p, 2)$$
$$r_2 : (p \mid 1, b/be, 0)$$
$$r_3 : (p \mid 1, c/ce, 0)$$
$$r_4 : (p \mid 1, bc/\lambda, 0)$$

```
@model<tpdc>
def main()
{
@mu = [[]'1 []'2]'0;
@ms(0) = b,c,e;
@ms(1) = a,b,c;
@ms(2) = p;
[a]'1 <--> [p]'2;
[p | b]'1 <--> [b,e]'0;
[p | c]'1 <--> [c,e]'0;
[p | b,c]'1 <--> []'0;
}
```

Fig. 1 A simple P system specification in P-Lingua

through a spreadsheet file with `.xls` extension, adapting the inputs and outputs to the desired P system family, and indicating the way the input data by the end user will populate the parameters for the final P system to generate with each run of the system.

The overall process to customize apps and to use them, enriching the core parsing and simulation functionalities with further debugging and visualization tools, plus other add-ons using MeCoSim extension mechanisms, can be reviewed in detail in [27, 33].

4 Extensions of the P-Lingua language

The previous section has outlined the main elements of our general framework provided by P-Lingua and MeCoSim. In the present work, the general tools of this framework are extended to handle tissue P systems with promoters and cell division. This includes genuine features added to the specific language for this kind of systems, including in their syntax some lexicon and grammar elements. Additionally, the existence of new ingredients (promoters) in the rules of these systems implies the need of new simulators capturing the dynamic behavior of these systems, following the description in [24].

In summary, we will take as a reference the existing P-Lingua syntax for P systems introduced in [18, 19], and will introduce the syntactic elements for tissue P systems with promoters and cell division along the following subsections. Then a brief description of the simulator developed for this variant of P systems is given.

4.1 Model specification

Any P-Lingua file defining a tissue P system with cell division can be specified, in general, using `TPDC` as its model (similar to `TPDC` [24]):

```
@model<tpdc>
```

This use of `tpdc` will imply the allowance in (`t`)issue P systems of the use of (`p`)romoters (not present nor allowed in pre-existing models as `tissue_psystems` or `TSCS`), and cell (`d`)ivision, as well as the usual (`c`)ommunication rules.

Similarly, two additional restricted variants have been defined for this new type of P systems, both available to use instead of `tpdc`, depending on the type of constraints imposed:

```
@model<tpds>
@model<tpda>
```

As one might expect, the suffix `s` means that communication rules are restricted to (`s`)ymport rules only, while the `a` constraints the systems to allow (`a`)ntiport rules only instead.

The rest of the file will then define the main elements describing the P system, including the membrane structure containing the different cells, the initial multisets present in each one of them at the beginning of the computation and the set of rules. Let us note that some elements, as Γ or the underlying graph connecting the cells, are inferred from the initial multisets and the rules. On the other hand, $\mathcal{E}$ is also inferred from the initial multiset associated with the environment in the P-Lingua file.

The membrane structure, initial multisets, sets of rules and output region will be set, as explained in the following subsections.

4.2 Membrane structure and initial multisets

Tissue P systems with promoters are based on a graph structure as any other tissue–like P systems. As commonly used in P-Lingua models, to specify the initial membrane structure the reserve word `@mu` is used, defining the corresponding μ, as defined in Sect. 2. More specifically, if we have q cells, it would be defined as:

```
@mu = [ []'1 []'2 ... []'q ]'0;
```

This can be seen for a specific example in Fig. 1:

```
@mu = [ []'1 []'2 ]'0;
```

Regarding the objects in the initial configuration of the system, they can be defined by the reserved word `@ms`, assigning a certain multiset to the desired membrane. Thus, for the example in Fig. 1, the initial multisets are defined in P-Lingua as:

```
@ms(1) = a,b,c;
@ms(2) = p;
```

Similarly, the alphabet of the environment, establishing which objects will be present in the environment in an arbitrarily large number of copies along the computation, is defined with the same syntax, but assigning a set to label 0.

```
@ms(0) = b,c,e;
```

Let us recall that in the rest of regions (in the cells) we may have more than one copy of certain objects, and that P-Lingua expresses this using operator `*`, followed by the multiplicity of the object, as in the following example:

```
@ms(1) = a,b*3,c*2;
```

Alternatively, the definition of these initial objects can be included directly in the definition of μ presented above, so that any membrane may include both child membranes and objects inside, as in the following code:

```
@mu = [ a*2 [ b ]'2 [ ]'3 [ c*5 ]'4 ]'1;
```

Besides the previous separate definitions of cell structure and initial multisets (plus definition of the alphabet of the environment), these elements might be combined to express in a more succinct way the same ideas. Thus, the corresponding lines into the example in Fig. 1 could be rewritten as:

```
@mu = [ b, c, e [ a, b, c ]'1 [ p ]'2 ]'0;
```

4.3 Definition of rules

As we recalled in Sect. 2, tissue P systems with promoters and cell division introduce a new feature with respect to *classical* tissue P systems with cell division, in such a way that any symport/antiport rule can be conditioned by the presence of promoters. These elements were not present in P-Lingua so far, so they have been included in the language, and applied to the rules of the new models (`tpdc`, `tpds` and `tpda`). The example shown in Fig. 1 illustrates the use of the new feature:

```
[p | b]'1 <--> [b,e]'0;
[p | c]'1 <--> [c,e]'0;
[p | b,c]'1 <--> []'0;
```

As we can see in the example, the syntax `prom |` inside the left-hand side of any of the rules is used to express that the promoter object (`p` in the example) must be present to make the rule applicable.

4.4 Simulation of the new model with P-Lingua and MeCoSim

A new simulator has been developed within P-Lingua framework, making use of the general simulator available for tissue P systems, but including the semantic and dynamic aspects derived from the use of promoters in the rules of the new model.

Thus, once a P system in the P-Lingua language is available in its text file, the simulator will perform a possible computation (let us recall that our systems are non-deterministic) from the initial configuration given by the structure and multisets specified, producing the corresponding transitions until a halting configuration is reached.

Thus, the simulation algorithm follows the general schema present in most of the simulators included in the platform:

1. Initialization
2. For each computation step, while there are applicable rules:
   (a) Selection of rules
   (b) Execution of rules

The *initialization stage* sets initial structures used by the algorithm (technical details are not relevant here). Then the main loop runs until a halting configuration is reached (i.e., until no applicable rule is present).

The *selection phase* checks for the applicability of the rules for each cell. The applicability of a rule is determined by the presence of at least the number of objects of each type present in each interacting region cell (let us recall that, if the environment is one of such regions, the objects of the alphabet of the environment will be present in an arbitrarily large number of copies). Additionally, if promoters are included in the rule, then their presence will also be required to make the rule applicable.

Then, given all the applicable rules, this phase will select a maximal set of rules. According to the theoretical model, this set should be selected non-deterministically; however, in the particular recognizer tissue P systems considered here, to properly solve the problems under study (e.g., `SAT`) we impose the solution the condition of being *confluent*, i.e., given a certain input m, the output of all the computations of the system with such input m must be equal. Therefore, to simulate the solution of the system it is enough to consider a single computation. Thus, the simulator developed always performs the same simulation given the same input. Instead of choosing non-deterministically the rules, it starts taking each applicable communication rule while they have objects available to perform their computations, and they reserve (*block*) those objects, to be consumed during the execution phase. Then after checking all the communication rules, if there are division rules still applicable to the remaining (i.e., not blocked) objects, then one division rule is selected for each cell having such applicable division rules. For more details about the algorithmic schema of this simulator, please see [10], given that the same approach has been followed (in this case extended to consider the influence of promoters).

Finally, as a result of this selection phase, a set of rules have been selected. In this case, let us also recall that, while the multiplicity of each object restricts the choice of rules for our maximal set, in the case of promoters is different: with only one promoter object affecting many rules, all of them would be enabled in with respect to the promoter. In

this sense, we say that rules do not compete for the promoters, they imply more a context condition that an object to consume (it is, in fact, not removed).

Then the *execution phase* applies the change in the configuration, passing from C_t to C_{t+1}, removing the objects consumed by the rules selected, and adding the objects produced, with the semantics specified in Sect. 2.

4.5 Availability of the software tools developed

The tools described in the previous sections, concerning the language, parsing and simulation have been made publicly available in the current version of MeCoSim that can be downloaded from [33]. Once downloaded, whenever the software runs, if an Internet connection is active, it checks the presence of new versions of any of the files involved, thus guarantees it always provides the user with the last version of MeCoSim (that includes in its installation pLinguaCore).

4.6 Further technical considerations

The tools developed along this work aim to complement P-Lingua framework, to provide both P system experts/designers and end users of the P-system-based applications with an environment where they can debug their designs, verify solutions to hard problems and run virtual experiments for a variant of P system not previously supported.

The main challenges faced during the development are derived from the definition of a sub-language for tissue P systems with promoters integrated in P-Lingua framework, plus the corresponding simulator handling this variant of P system.

On the one hand, the extension of the language includes new syntactic and grammatical elements for the parser, based on Java [32], and the proper integration with the rest of the P-Lingua language. On the other hand, the definition of the corresponding semantic constraints and the logic of the specific simulor, taking into account the semantic and dynamic aspects defined by tissue P systems with promoters and cell division [24], was developed in Java [31], and following the structure defined for other simulators and semantic decorations.

Let us recall that the aim of P-Lingua framework and MeCoSim environment is the precise definition of each variant of P system covered from a functional point of view. Other aspects as the development of the fastest possible solutions for these systems (e.g., with parallel simulators using high-performance computing—HPC) would be out of the scope of this work, and could improve the performance of the simulator developed. Those potential alternatives could still make use of the parsing tools developed with P-Lingua, and also be connected with MeCoSim environment acting as a client providing the interface and pre-/post-processing tool for a server HPC-based simulator.

5 Case studies

In the previous section, the software tools developed for the design and simulation of tissue P systems with promoters and cell division have been described. In this last section, we will illustrate the use of the software, showing the corresponding P-Lingua files specifying the solutions in the input format for the computer tools, and some runs for simple and complex problems.

5.1 Basic example

To check the proper behavior of the new features included in our framework, we started with a very simple example, as included at the beginning of he paper:

```
@model<tpdc>
def main()
{
@mu = [[]'1 []'2]'0;
@ms(0) = b,c,e;
@ms(1) = a,b,c;
@ms(2) = p;
[a]'1 <--> [p]'2;
[p | b]'1 <--> [b,e]'0;
[p | c]'1 <--> [c,e]'0;
[p | b,c]'1 <--> []'0;
}
```

This simple example included both symport and antiport rules, so we made use of `tpdc` model, and started debugging in MeCoSim, as shown in Fig. 2.

If we had used a different model, as `tpda` or `tpds`, our environment would inform us about the error, thus disabling the possibility to run it until we correct this aspect, as we can see in Fig. 3.

5.2 Solving `SAT` by tissue P systems with promoters and cell division

In this case study, we will recall the solution for SAT given in [24], illustrating the behavior of these systems, and we will show the corresponding P-Lingua files specifying the solutions in the input format for the computer tools.

The `SAT` problem is a well-known **NP**-complete problem [2], which is defined as follows: *given a Boolean formula in conjunctive normal form (CNF), determine whether or not there exists an assignment to its variables such that the formula is evaluated to be true.*

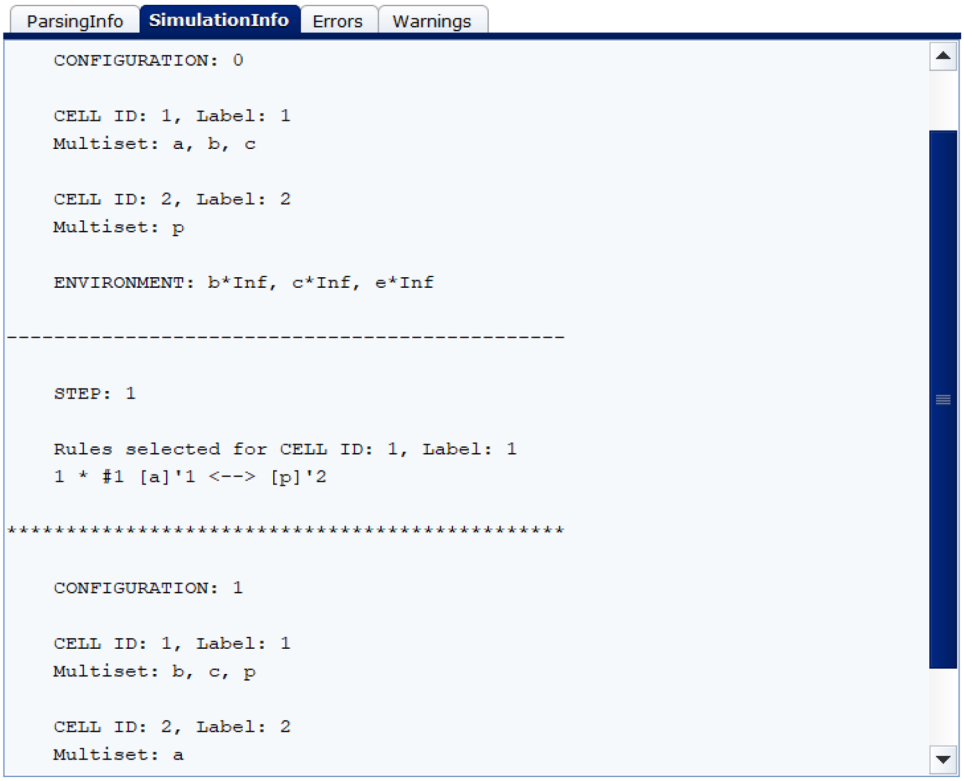


Fig. 2 Debugging a simple tissue P system with promoters in MeCoSim

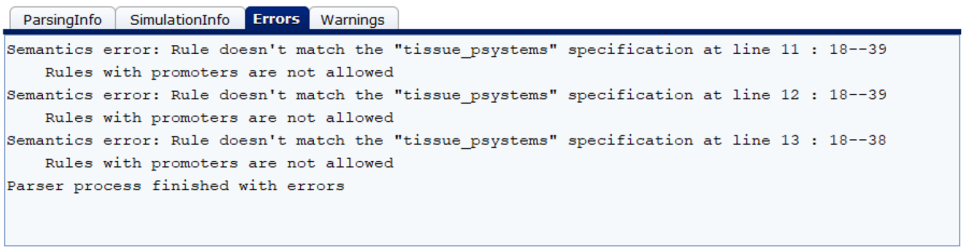


Fig. 3 Informing about semantic errors

In what follows, we provide here the linear time solution to the SAT problem by a family of recognizer tissue P systems with promoters and cell division directly taken from [24]: $\Pi = \{\Pi(t) \mid t \in \mathbb{N}\}$. As expressed there, each system $\Pi(t)$ processes all Boolean formulas φ in conjunctive normal form with n variables and m clauses, where $t = \langle n, m\rangle = ((n+m)(n+m+1)/2) + n$, assuming the corresponding input multiset $cod(\varphi)$ is supplied to the system.

For each $n, m \in \mathbb{N}$, we consider the recognizer tissue P system with promoters and cell division

$$\Pi(\langle n, m\rangle) = (\Gamma, \Sigma, \mathcal{E}, w_1, w_2, \mathcal{R}, i_{in}, i_{out}),$$

where

- $\Gamma = \Sigma \cup \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{b_j, c_j \mid 1 \leq j \leq m\}$ $\cup \{z_i \mid 0 \leq i \leq 2n+m+3\} \cup \{b_{m+1}, e, p, q, q', \mathtt{yes}, \mathtt{no}\}$,
- $\Sigma = \{x_{i,j}, \bar{x}_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$,
- $\mathcal{E} = \{a_i \mid 1 \leq i \leq n+1\} \cup \{b_j, c_j \mid 1 \leq j \leq m\}$ $\cup \{z_i \mid 0 \leq i \leq 2n+m+3\} \cup \{b_{m+1}, e, q'\}$,
- $w_1 = \{a_1\}$, $w_2 = \{p, q, z_0, \mathtt{yes}, \mathtt{no}\}$,
- $i_{in} = 1$ is the input cell,
- $i_{out} = 0$ is the output region,
- The set $\mathcal{R}$ consists of the following rules:

$r_{1,i} \equiv [a_i]_1 \rightarrow [t_i]_1[f_i]_1, 1 \leq i \leq n,$
$r_{2,i,j} \equiv (t_i \mid 1, x_{i,j}/c_j, 0), 1 \leq i \leq n, 1 \leq j \leq m,$
$r_{3,i,j} \equiv (f_i \mid 1, \bar{x}_{i,j}/c_j, 0), 1 \leq i \leq n, 1 \leq j \leq m,$
$r_{4,i} \equiv (1, t_i/a_{i+1}, 0), 1 \leq i \leq n,$
$r_{5,i} \equiv (1, f_i/a_{i+1}, 0), 1 \leq i \leq n,$
$r_6 \equiv (1, a_{n+1}/b_1, 0),$
$r_{7,j} \equiv (b_j \mid 1, c_j/b_{j+1}, 0), 1 \leq j \leq m,$
$r_8 \equiv (1, b_{m+1}/p, 2),$
$r_9 \equiv (b_{m+1} \mid 2, \mathtt{yes}/e, 0),$
$r_{10} \equiv (b_{m+1} \mid 2, q/e, 0),$
$r_{11,i} \equiv (2, z_i/z_{i+1}, 0), 0 \leq i \leq 2n + m + 2,$
$r_{12} \equiv (z_{2n+m+3} \mid 2, q/q', 0),$
$r_{13} \equiv (q' \mid 2, \mathtt{no}/e, 0).$

All the details about the behavior and verification of the model can be read in [2]. For our purposes, it is interesting to see the translation of this system to the P-Lingua language, using the new model and elements introduced:

```
@model<tpda>

def main()
{
call module_init_conf(n,m);
call module_rules(n,m);
call module_input();
}

def module_init_conf(n,m)
{
@mu = [[]'1 []'2]'0;
@ms(0) = b{m+1},e,qp;
@ms(0) += a{i} : 1 <= i <= n+1;
@ms(0) += b{j},c{j} : 1 <= j <= m;
@ms(0) += z{i} : 0 <= i <= 2*n+m+3;
@ms(1) = a{1};
@ms(2) = p,q,z{0},yes,no;
}

def module_rules(n,m)
{
/* r1{i} */    [a{i}]'1 --> [t{i}]'1 [f{i}]'1 : 1<=i<=n;
/* r2{i,j} */  [t{i} | x{i,j}]'1 <--> [c{j}]'0: 1<=j<=m, 1<=i<=n;
/* r3{i,j} */  [f{i} | nx{i,j}]'1 <--> [c{j}]'0: 1<=j<=m, 1<=i<=n;
/* r4{i} */    [t{i}]'1 <--> [a{i+1}]'0 : 1<=i<=n;
/* r5{i} */    [f{i}]'1 <--> [a{i+1}]'0 : 1<=i<=n;
/* r6 */       [a{n+1}]'1 <--> [b{1}]'0;
/* r7{j} */    [b{j} | c{j}]'1 <--> [b{j+1}]'0 : 1<=j<=m;
/* r8 */       [b{m+1}]'1 <--> [p]'2;
/* r9 */       [b{m+1} | yes]'2 <--> [e]'0;
/* r10 */      [b{m+1} | q]'2 <--> [e]'0;
/* r11{i} */   [z{i}]'2 <--> [z{i+1}]'0 : 0<=i<=2*n+m+2;
/* r12 */      [z{2*n+m+3} | q]'2 <--> [qp]'0;
/* r13 */      [qp | no]'2 <--> [e]'0;
}

def module_input()
{
/* We define here the input for the P system */
/* Let be m: number of clauses, and n: number of variables */

@ms(1) += nx{variable{i},clause{i}}*valn{i}, x{variable{i},clause{i
     }}*val{i} : 1<=i<=nvals;
}
```

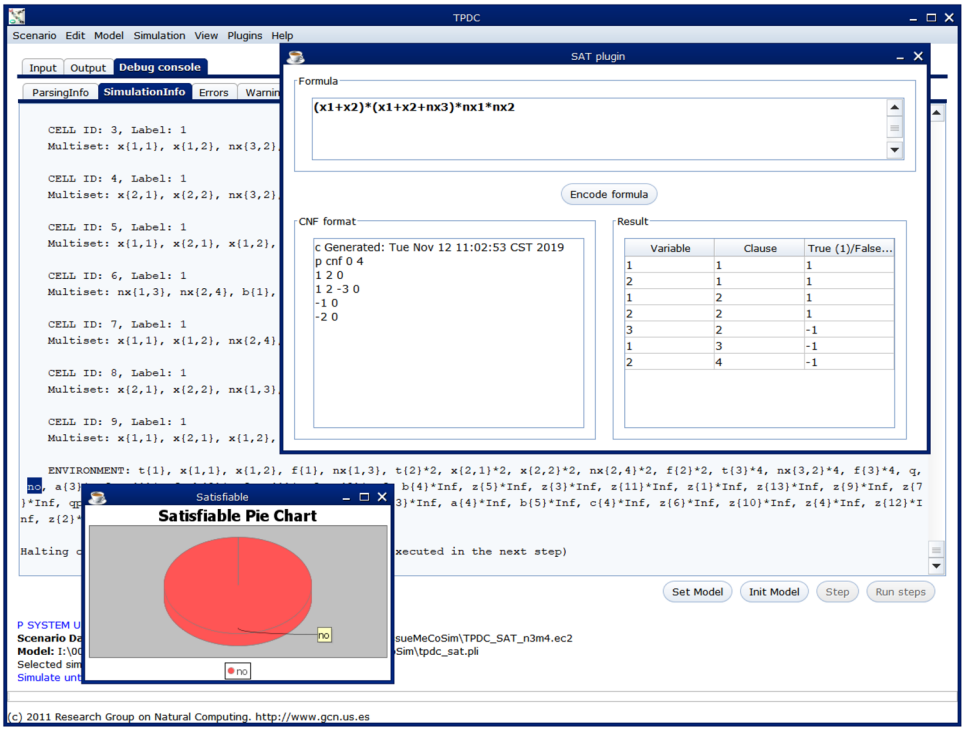


Fig. 4 SAT simulation in MeCoSim

Figure 4 shows this last example loaded in MeCoSim, running the simulation for the corresponding input formula and returning the result, while in background the previous debugging screen is visualized.

Let us note that previous to getting this result a debugging process was conducted through the platform generic tools with the new features incorporated. Additionally, the formal verification of this solution given in [24] could be checked with the step-by-step run, thus checking that the statements made in the original paper are indeed satisfied for specific examples, what enriches the previous result with experimental validation.

6 Results and conclusions

The main purpose of the simulator developed along this work was to provide a useful tool for P systems designers to experiment with their models based on tissue P systems with promoters. Bearing in mind this idea, the approach followed was to take advantage of the well-known framework available with P-Lingua and MeCoSim, including the language, the simulation engine and the visual tools to make it easier to design and verify solutions based on membrane systems. As a case study, a strong result given by the solution of SAT analyzed in previous sections was chosen, to actually check that the behavior of the extended language and simulation tools were performing as expected.

As a pleasant unexpected surprise, a minor but illustrative fact happened: the discovery of a subtle mistake in the original solution to SAT while debugging the model in Sect. 5.2. Specifically, a_{n+1} had to be incorporated in $\Leftarrow E$), to make it possible to apply rules $r_{4,i}$ and $r_{5,i}$. While not significant, this fact shows the relevance of the simulation tools in designing and verifying solutions to hard problems.

In addition to the step-by-step traces with a small example formula, others were tested to experimentally validate the results of the simulator. Table 1 shows the results obtained, including the input formula, number of variables and clauses, output and running time. However, this last field is only relevant to see how it increases exponentially, as expected, when n and m start increasing. In absolute terms, to compare with other tools would not make too much sense, given the overhead of information the simulation is producing to activate certain visualization tools and plugins in MeCoSim environment. This is due to the nature of these tools not focusing on getting the best performance but getting the best user experience, saving time in designing, debugging and verifying solutions, and visually introducing new scenarios to experiment without technical background in P systems.

As a future work, the development of high-performance simulators based on different platforms could help people being interested in narrowing the gap between the real power of P systems and the clear limitations of the solutions based on software simulation.

Table 1 Simulation results for SAT instances

Ex.	n	m	Out	t (ms)	Formula
1	3	2	T	7	$x_1 \wedge (\bar{x}_2 \vee x_3)$
2	2	3	F	6	$(\bar{x}_1 \vee \bar{x}_2) \wedge x_1 \wedge x_2$
3	2	3	T	5	$(\bar{x}_1 \vee \bar{x}_2) \wedge x_2 \wedge (\bar{x}_1 \vee x_2)$
4	3	4	F	7	$(x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \bar{x}_3) \wedge \bar{x}_1 \wedge \bar{x}_2$
5	3	4	T	6	$(\bar{x}_1 \vee x_2) \wedge \bar{x}_1 \wedge x_3 \wedge (\bar{x}_1 \vee x_3)$
6	4	5	F	12	$(x_1 \vee x_4) \wedge (x_1 \vee \bar{x}_4) \wedge x_3 \wedge (x_2 \vee \bar{x}_3 \vee x_4) \wedge \bar{x}_1$
7	4	5	T	11	$(x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee x_3)$
8	5	6	F	28	$(x_1 \vee \bar{x}_2 \vee x_3 \vee x_5) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4) \wedge x_4 \wedge x_2 \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4)$
9	5	6	T	24	$(x_3 \vee x_4) \wedge (x_4 \vee \bar{x}_5) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_2 \vee x_4) \wedge (x_1 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee x_5)$
10	6	7	F	66	$(x_3 \vee x_5 \vee x_6) \wedge (x_3 \vee \bar{x}_4 \vee x_5 \vee \bar{x}_6) \wedge \bar{x}_3 \wedge \bar{x}_6 \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_5 \vee x_6) \wedge (x_1 \vee x_4 \vee x_5) \wedge (\bar{x}_5 \vee x_6)$
11	6	7	T	65	$(\bar{x}_1 \vee \bar{x}_2 \vee x_5) \wedge (x_2 \vee x_3) \wedge (x_3 \vee \bar{x}_5 \vee \bar{x}_6) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4 \vee x_5 \vee x_6) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3 \vee x_6) \wedge (x_1 \vee \bar{x}_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6)$
12	7	8	F	184	$(\bar{x}_5 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_3 \vee \bar{x}_4 \vee x_7) \wedge (\bar{x}_1 \vee x_3 \vee x_5 \vee x_6 \vee \bar{x}_7) \wedge (x_1 \vee x_3 \vee \bar{x}_5 \vee x_6 \vee x_7) \wedge (x_2 \vee x_6) \wedge (x_2 \vee \bar{x}_6) \wedge \bar{x}_2 \wedge (x_2 \vee x_3 \vee x_4 \vee \bar{x}_5 \vee x_7)$
13	7	8	T	184	$(\bar{x}_2 \vee x_5 \vee x_6 \vee x_7) \wedge (x_2 \vee \bar{x}_4 \vee \bar{x}_5 \vee \bar{x}_7) \wedge (x_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_6 \vee x_7) \wedge (x_1 \vee x_2 \vee x_3 \vee \bar{x}_5 \vee x_6 \vee \bar{x}_7) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee x_6 \vee \bar{x}_7) \wedge (x_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_7) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4 \vee \bar{x}_6) \wedge (x_3 \vee x_5 \vee x_6 \vee \bar{x}_7)$
14	8	9	F	430	$(x_3 \vee x_4 \vee \bar{x}_6 \vee \bar{x}_8) \wedge (x_6 \vee \bar{x}_7) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4 \vee x_5 \vee x_8) \wedge x_7 \wedge (x_1 \vee \bar{x}_2 \vee x_5 \vee \bar{x}_7 \vee \bar{x}_8) \wedge (x_2 \vee x_7 \vee x_8) \wedge (\bar{x}_6 \vee \bar{x}_7) \wedge (x_1 \vee x_5 \vee \bar{x}_8) \wedge (x_1 \vee \bar{x}_4 \vee x_5 \vee \bar{x}_6 \vee x_7)$
15	8	9	T	460	$(x_1 \vee \bar{x}_5 \vee \bar{x}_6 \vee \bar{x}_7 \vee \bar{x}_8) \wedge (x_2 \vee x_3 \vee x_4 \vee \bar{x}_6 \vee \bar{x}_7 \vee x_8) \wedge (x_3 \vee x_4 \vee \bar{x}_5 \vee \bar{x}_6 \vee \bar{x}_7 \vee \bar{x}_8) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5 \vee x_6 \vee \bar{x}_7 \vee x_8) \wedge (\bar{x}_3 \vee \bar{x}_7) \wedge (x_4 \vee x_5 \vee \bar{x}_7) \wedge (x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_4 \vee \bar{x}_5 \vee \bar{x}_6 \vee x_7 \vee \bar{x}_8)$
16	9	10	F	1135	$(\bar{x}_2 \vee \bar{x}_3 \vee x_5 \vee x_7) \wedge (x_2 \vee x_5 \vee x_6 \vee x_7 \vee x_9) \wedge (\bar{x}_3 \vee x_5 \vee x_7 \vee x_8) \wedge (x_1 \vee \bar{x}_4 \vee \bar{x}_5 \vee x_6 \vee x_8) \wedge (\bar{x}_2 \vee x_3 \vee x_5 \vee x_7 \vee x_8 \vee \bar{x}_9) \wedge (\bar{x}_2 \vee \bar{x}_4 \vee x_7 \vee x_9) \wedge (\bar{x}_2 \vee x_4 \vee x_6 \vee x_9) \wedge x_1 \wedge x_5 \wedge (\bar{x}_1 \vee \bar{x}_5)$
17	9	10	T	1155	$(x_3 \vee x_8) \wedge (x_1 \vee \bar{x}_2 \vee x_5 \vee \bar{x}_6 \vee x_9) \wedge (x_3 \vee x_6 \vee x_9) \wedge (x_3 \vee x_5 \vee \bar{x}_6 \vee \bar{x}_8) \wedge (x_1 \vee x_2 \vee \bar{x}_5 \vee x_7 \vee \bar{x}_8 \vee \bar{x}_9) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4 \vee x_5 \vee \bar{x}_6 \vee \bar{x}_7 \vee x_9) \wedge (x_1 \vee x_2 \vee x_4 \vee \bar{x}_6 \vee x_8 \vee \bar{x}_9) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_7 \vee \bar{x}_8) \wedge (\bar{x}_1 \vee x_2 \vee x_3 \vee x_5 \vee \bar{x}_6 \vee x_8 \vee \bar{x}_9) \wedge (x_2 \vee \bar{x}_3 \vee x_4 \vee \bar{x}_6 \vee \bar{x}_7 \vee \bar{x}_9)$

Acknowledgements The work of Luis Valencia-Cabrera was partially supported in part by the research project TIN2017-89842-P, cofinanced by Ministerio de Economía, Industria y Competitividad (MINECO) of Spain, through the Agencia Estatal de Investigación (AEI), and by Fondo Europeo de Desarrollo Regional (FEDER) of the European Union. The work of Bosheng Song was supported in part by National Natural Science Foundation of China (61972138, 61602192), and in part by the Fundamental Research Funds for the Central Universities (531118010355).

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Díaz-Pernil, D., Gutiérrez-Naranjo, M. A., & Pérez-Jiménez, M. J. (2006). Solving 3-COL with Tissue P Systems *Proceeding of Fourth Brainstorming Week on Membrane Computing*, Sevilla, 17–30.
2. Garey, M. R., & Johnson, D. J. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco: W.H. Freeman.
3. Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M. J., & Riscos-Núñez, A. (2009). A P-Lingua programming environment for membrane computing. *Lecture Notes in Computer Science*, *5391*, 187–203.
4. García-Quismondo, M., Gutiérrez-Escudero, R., Pérez-Hurtado, I., Pérez-Jiménez, M. J., & Riscos-Núñez, A. (2010). An overview of P-Lingua 2.0. *Lecture Notes in Computer Science*, *5957*, 264–288.
5. Guo, P., Quan, C., & Ye, L. (2019). UPSimulator: A general P system simulator. *Knowledge-Based Systems*, *170*, 20–25.
6. Lefticaru, R., Ipate, F., Valencia-Cabrera, L., Turcanu, A., Tudose, C., Gheorghe, M., et al. (2012). Towards an integrated approach for model simulation, property extraction and verification of P systems. *Proceedings of Tenth Brainstorming Week on Membrane Computing, Sevilla*, *I*, 291–318.
7. Macías-Ramos, L. F. (2016). Developing efficient simulators for cell machines. PhD thesis. University of Seville.
8. Macías-Ramos, L. F., Pérez-Hurtado, I., García-Quismondo, M., Valencia-Cabrera, L., Pérez-Jiménez, M. J., & Riscos-Núñez, A. (2012). A P-Lingua based simulator for spiking neural P systems. *Lecture Notes in Computer Science*, *7184*, 257–281.
9. Martín-Vide, C., Păun, Gh, Pazos, J., & Rodríguez-Patón, A. (2003). Tissue P systems. *Theoretical Computer Science*, *296*(2), 295–326.
10. Martínez-del-Amor, M. A., Pérez-Hurtado, I., Pérez-Jiménez, M. J., & Riscos-Núñez, A. (2010). A P-Lingua based simulator for tissue P systems. *Journal of Logic and Algebraic Programming*, *79*, 374–382.
11. Pan, L., & Alhazov, A. (2006). Solving HPP and SAT by P systems with active membranes and separation rules. *Acta Informatica*, *43*(2), 131–145.
12. Pan, L., & Song, B. (2020). P systems with rule production and removal. *Fundamenta Informaticae*, *2020*(171), 313–329.
13. Pan, L., & Pérez-Jiménez, M.J. (2010). Computational complexity of tissue-like P systems. *Journal of Complexity*, *26*(3), 293–315.
14. Păun, Gh. (2000). Computing with membranes. *Journal of Computer and System Sciences*, *61*(1), 108–143.
15. Păun, Gh. (2001). P systems with active membranes: Attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics*, *6*(1), 75–90.
16. Păun, Gh, Rozenberg, G., & Salomaa, A. (Eds.). (2010). *The Oxford Handbook of Membrane Computing*. New York: Oxford University Press.
17. Pérez-Hurtado, I., Valencia-Cabrera, L., Pérez-Jiménez, M. J., Colomer, M. A., & Riscos-Núñez, A. (2010). Mecosim: A general purpose software tool for simulating biological phenomena by means of p systems. *IEEE Fifth International Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010)*, *I*, 637–643.
18. Pérez-Hurtado, I. (2010). Desarrollo y aplicaciones de un entorno de programación para Computación Celular: P-Lingua. PhD thesis. University of Seville.
19. Pérez-Hurtado, I., Valencia-Cabrera, L., Chacón, J. M., Riscos-Núñez, A., & Pérez-Jiménez, M. J. (2014). A P-Lingua based simulator for tissue P systems with cell separation. *Romanian Journal of Information Science and Technology*, *17*, 89–102.
20. Porreca, A. E., Murphy, N., & Pérez-Jiménez, M. J. (2012). An optimal frontier of the efficiency of tissue P systems with cell division. In Proceedings of 10th Brainstorming Week on Membrane Computing, Sevilla, II, 141–166.
21. Song, B., & Kong, Y. (2019). Solution to PSPACE-complete problem using P systems with active membranes with time-freeness. *Mathematical Problems in Engineering*, 5793234.
22. Song, B., Li, K., Orellana-Mart, D., Valencia-Cabrera, L., Perez-jemenz, M.J. (2020). Cell-like P systems with evolutional symport/antiport rules and membrane creation. *Information and Computation*. https://doi.org/10.1016/j.ic.2020.104542.
23. Song, B., Perez-jemenz, M.J., Pan, L. (2017). An efficient time-free solution to QSAT problem using P systems with proteins on membranes. *Information and Computation*. https://doi.org/10.1016/j.ic.2017.06.005.
24. Song, B., & Pan, L. (2016). The computational power of tissue-like P systems with promoters. *Theoretical Computer Science*, *641*, 43–52.
25. Song, B., Zhang, C., & Pan, L. (2017) Tissue-like P systems with evolutional symport/antiport rules. *Information Sciences*, *378*, 177–193.
26. Song, T., Pan, L., Wu, T., Zheng, P., Dennis, W.M.L., Rodriguez-Paton, A. (2019). Spiking neural P systems with learning functions. *IEEE Transactions on NanoBioscience*, *18*(2), 176–190.
27. Valencia Cabrera, L. (2015). An environment for virtual experimentation with computational models based on P systems. PhD thesis. University of Seville.
28. Wu, T., Păun, A., Zhang, Z., Pan, L. (2018). Spiking neural P systems with polarizations. *IEEE Transactions on Neural Networks and Learning Systems*, *8*, 3349–3360.
29. Zhang, X., Pan, L., Păun, A., (2015) On the universality of axon P systems. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(11), 2816–2829.
30. Zhang, G., Pérez-Jiménez, M. J., & Gheorghe, M. (2017). *Real-life applications with membrane computing*. New York: Springer.
31. Java tutorial Website, https://docs.oracle.com/javase/tutorial.
32. Javacc, https://javacc.org.
33. MeCoSim Website. http://www.p-lingua.org/mecosim/.
34. P-Lingua Website. http://www.p-lingua.org/.

Luis Valencia-Cabrera finished his BSc degree in Computer Engineering in 2005, Advanced studies diploma in 2011, Masters MSc in Logics, Computing and Artificial Intelligence in 2013, and got his PhD in 2015, in Universidad de Sevilla (Spain). He worked in IT Consulting from 2005 to 2010, obtaining with a Professional Postgraduate Masters in 2007 in Development of Enterprise Applications JEE, later also getting official certifications by the IFPUG (cert. CFPS, in 2006) and Oracle (SCJP, SCWCD, SCBCD, in 2008). From 2017 to early 2018 he collaborated with a startup in the development of projects related with Natural Language Processing, getting in 2018 the first prize in the category "General Corpus" en el II Hackathon of Natural Language Technologies (Ministry of Economy and Enterprises of Spain). He is currently an Assistant Professor (certified by ANECA to the next level of Associate Professor) in the Department of Computer Science and Artificial Intelligence of the University of Sevilla (Spain). He belongs to the department since 2011, and to the Research Group on Natural Computing from 2010. He got the PhD Extraordinary Award for his PhD. Has supervised many final BSc and MSc final projects, along with two PhD (currently advising the third one). His main research interests include the modelling and simulation of complex systems, natural computing (especially, membrane computing),theoretical computer science, and software engineering and development. His publications include 28 papers in ISI-JCR indexed journal, along with book chapters, journals indexed in other indices, invited talks and conference communications, for a total of 98 scientific publications. He has been involved in more than 10 research projects at a regional, national and international level. He has participated in the organization of a dozen international conferences, and in the edition of the volumes of such conferences.

Bosheng Song received the Ph.D. degree in control science and engineering from Huazhong University of Science and Technology, Wuhan, China, in 2015. He spent eighteen months working in the Research Group on Natural Computing, University of Seville, Seville, Spain, from November, 2013 to May, 2015. He was worked as a post-doctoral researcher with the School of Automation, Huazhong University of Science and Technology, Wuhan, China, from March, 2016 to February, 2019. He is currently an Associate Professor with the College of Information Science and Engineering, Hunan University, Changsha, China. His current research interests include membrane computing and formal language theory.