P–Lingua 2.0: A software framework for cell–like P systems

Manuel García-Quismondo, Rosa Gutiérrez-Escudero, Miguel A Martínez-del-Amor Enrique Orejuela-Pinedo, I. Pérez-Hurtado

Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mangarfer2@alum.us.es, {rgutierrez, mdelamor, orejuela, perezh}@us.es

Abstract: P-Lingua is a programming language for membrane computing. It was first presented in Edinburgh, during the Ninth Workshop on Membrane Computing (WMC9). In this paper, the models, simulators and formats included in P-Lingua in version 2.0 are explained. We focus on the stochastic model, associated simulators and updated features. Finally, we present one of the first applications based on P-Lingua: a tool for describing and simulating ecosystems.
Keywords: Programming languages, software development, P systems, Membrane Computing, P-Lingua

1 Introduction

Membrane computing (or cellular computing) is a branch of Natural Computing that was introduced by Gh. Păun [10]. The main idea is to consider biochemical processes taking place inside living cells from a computational point of view. The initial definition of this computing paradigm is very flexible and many different models have been defined.

Each model displays characteristic semantic constraints that determine the way in which rules are applied. Hence, the need for software simulators capable of taking into account different scenarios when simulating P system computations comes to the fore. An initial approach could be defining inputs for each simulator specifically. Nevertheless, this approach involves defining new input formats for each simulator, so designing simulators would take a great effort. A second approach could be standardizing the simulator input, so all simulators need to process inputs specified in the same format. These two approaches raise up a trade-off: On the one hand, specific simulator inputs could be defined in a more straightforward way, as the used format is closer to the P system features to simulate. On the other hand, although the latter approach involves analyzing different P systems and models to develop a standard format, there is no need to develop a new simulator every time a new P system should be simulated, as it is possible to specify it in the standard input format. Moreover, researches would not have to devise a new input format every time they specify a P system and would not need to change the way to specify P systems which need to be simulated every time they move on to another model, as they would keep on using the standard input format.

This second approach is the one considered in P-Lingua project, a programming language whose first version, presented in [3], is able to define P systems within the active membrane P system model with division rules. The authors also provide software tools for compilation, simulation and debug tasks. From now on, we will call P-Lingua 1.0 this version of the language and its associated tools.

As P-Lingua is intended to become a standard for P systems definition, it should also consider other models. In this paper, we present P-Lingua 2.0 as a framework to define cell-like P system models, including several algorithms to simulate P system computations for the supported models (from now on,

simulators), as well as different formats to represent P systems with associated parsers to translate from each other.

This paper is structured as follows. In Section 2 the supported models at this stage are enumerated. The next section introduces some algorithms used to simulate P systems, focusing on the stochastic and the probabilistic P system models. In Section 4 different file formats to representate cell-like P systems are presented, for example, P-Lingua 2.0 programming language. Model definitions, simulators and parsers have been encoded in a JAVA library, pLinguaCore ©, presented in Section 6, this library is free software and it can be easily expanded. Command-line tools to compile files and simulate P systems have been slightly modified - in Section 5 these changes are presented. The next section introduces one of the first applications of P-Lingua, a software tool to describe and simulate ecosystems. Finally, some conclusions and future work are enumerated in Section 8.

2 Supported P system models

The supported models developed so far are enumerated below, but a standard mechanism for defining new cell-like models has been included on the P-Lingua 2.0 framework. Each model displays characteristic semantic constraints entailing the rules applied, such as the number of objects specified on the left-hand side, membrane creation, polarization, and so on. It is possible to define additional models by including the corresponding semantic constraints within the plinguaCore JAVA library. This mechanism has been used on all the existent models.

The supported P system models in P-Lingua 2.0 are Transition P Systems, Symport/Antiport P Systems, P Systems with active membranes, with membrane division and membrane creation rules, Probabilistic P Systems and Stochastic P Systems. More details on those models can be found in [13], except for Stochastic P Systems, which are described in [10].

3 Simulators

In P-Lingua 1.0, only one simulator was supported, since there was only one P system model definition. However, as new models have been included, new simulators have been developed, providing at least one simulator for each supported model.

All simulators in P-Lingua 2.0 can step backwards (as well as the simulator in P-Lingua 1.0), but this option should be set before the simulation starts.

P-Lingua 2.0 also takes into account the existence of different simulation algorithms for the same model and provides a means for selecting a simulator among the ones which are suitable to simulate the P system, by checking its model. So far, only the stochastic P system model provides several simulation algorithms to choose, but the plugin-oriented architecture of the pLinguaCore JAVA library allows easily to encode new simulators.

3.1 Simulators for Stochastic P Systems

In the original definitions P systems evolve in a non-deterministic and maximally parallel manner (that is, all the objects in every membrane that can evolve by a rule must do it [10]). When trying to simulate biological phenomena, like living cells, the classical non-deterministic and maximally parallel approach is not valid anymore. First, biochemical reactions, which are modelled by rules, occur at a specific rate (determined by the propensity of the rule), therefore they cannot be selected in an arbitrary and non-deterministic way. Second, in the classical approach all time steps are equal and this does not represent the time evolution of a real cell system.

The strategies to replace the original approach are based on Gillespie's Theory of Stochastic Kinetics

[6]. A constant c is associated to each rule, which provides P systems with a stochastic extension. The constant c depends on the physical properties of the molecules involved in the reaction modeled by the rule and other physical parameters of the system. Besides, it represents the probability per time unit at which the reaction takes place. Also, it is used to calculate the propensity of each rule which determines the probability and time needed to apply the rule.

Two different algorithms based on the principles stated above have been implemented and integrated in pLinguaCore.

Multicompartimental Gillespie Algorithm

The Gillespie [6] algorithm or SSA (Stochastic Simulation Algorithm) was developed for a single, well-mixed and fixed volume/compartment. P systems generally contain several compartments or membranes. For that reason, an adaptation of this algorithm was presented in [10] and it can be applied in the different regions defined by the compartmentalised structure of a P system model. The next rule to be applied in each compartment and the waiting time for this application is computed using a *local* Gillespie algorithm. The Multicompartimental Gillespie Algorithm can be broadly summarized as follows:

Repeat until a prefixed simulation time is reached:

1. Calculate for each membrane $i, 1 \leq i \leq m$, and for each rule $r_j \in R_{l_i}$ the propensity, a_j, by multiplying the stochastic constant $c_j^{l_i}$ associated to r_j by the number of distinct possible combinations of the objects and substrings present of the left-side of the rule with respect to the current contents of membranes involved in the rule.

2. Compute the sum of all propensities

$$a_0 = \sum_{i=1}^{m} \sum_{r_j \in R_{l_i}} a_j$$

3. Generate two random numbers r_1 and r_2 from the uniform distribution in the unit interval and select τ_i and j_i according to

$$\tau_i = \frac{1}{a_0} \ln(\frac{1}{r_1})$$

$$j_i = \text{the smallest integer satisfying } \sum_{j=1}^{j_i} a_j > r_2 a_0$$

   In this way, we choose τ_i according to an exponential distribution with parameter a_0.

4. The next rule to be applied is r_{j_i} and the waiting time for this rule is τ_i. As a result of the application of this rule, the state of one or two compartments may be changed and has to be updated.

The Multicompartimental Next Reaction Method

The Gillespie Algorithm is an exact numerical simulation method appropriate for systems with a small number of reactions, since it takes a time proportional to the number of reactions (i.e., the number of rules). An exact algorithm which is also efficient is presented in [5], the Next Reaction Method. It uses only a single random number per simulation event (instead of two) and takes a time proportional to the logarithm of the number of reactions. We have adapted this algorithm to make it compartimental.

The idea of this method is to be extremely sensitive in recalculating a_j and t_i, trying to recalculate them only if they change. In order to do that, a data structure called *dependency graph* [5] is introduced.

Let $r : u[v]_l \xrightarrow{c} u'[v']_l$ be a given rule with propensity a_r and let the parent membrane of l be labelled with l'. We define the following sets:

- DependsOn$(a_r) = \{(b,t) \mid b$ is an object or string whose quantity affect the value.

  a_r, $t = l$ if $b \in v$ and $t = l'$ if $b \in u\}$.

  Generally, DependsOn$(a_r) = \{(b,l) \mid b \in v\} \cup \{(b,l') \mid b \in u\}$

- Affects$(r) = \{(b,t) \mid b$ is an object or string whose quantity is changed when the rule.

  r is excuted, $t = l$ if $b \in v \vee b \in v'$ and $t = l'$ if $b \in u \vee b \in u'\}$.

  Generally, Affects$(r) = \{(b,l) \mid b \in v \vee b \in v'\} \cup \{(b,l') : b \in u \vee b \in u'\}$.

Definition 1. Given a set of rules $R = R_{l_1} \cup \cdots \cup R_{l_m}$, the dependency graph is the directed graph $G = (V,E)$, with vertex set $V = R$ and edge set $E = \{(v_i, v_j) \mid \text{Affects}(v_i) \cap \text{DependsOn}(a_{v_j}) \neq \emptyset\}$

In this way, if there exists an edge $(v_i, v_j) \in E$ and v_i is executed, as some objects affected by this execution are involved in the calculation of a_{v_j}, this propensity would have to be recalculated. The dependency graph depends only on the rules of the system and is static, so it is built only once.

The times τ_i, that represent the waiting time for each rule to be applied, are stored in an *indexed priority queue*. This data structure, discussed in detail in [5], has nice properties: finding the minimum element takes constant time, the number of nodes is the number of rules $|R|$, because of the indexing scheme it is possible to find any arbitrary reaction in constant time and finally, the operation of updating a node (only when τ_i is changed, which we can detect using to the dependency graph) takes $\log|R|$ operations.

The Multicompartimental Next Reaction Method can be broadly summarized as follows:

1. Build the dependency graph, calculate the propensity a_r for every rule $r \in R$ and generate τ_i for every rule according to an exponential distribution with parameter a_r. All the values τ_r are stored in a priority queue. Set $t \leftarrow 0$ (this is the global time of the system).

2. Get the minimum τ_μ from the priority queue, $t \leftarrow t + \tau_\mu$. Execute the rule r_μ (this is the next rule scheduled to be executed, because its waiting time is least).

3. For each edge (μ, α) in the dependency graph recalculate and update the propensity a_α and

   - if $\alpha \neq \mu$, set
   $$\tau_\alpha \leftarrow \frac{a_{\alpha,old}(\tau_\alpha - \tau_\mu)}{a_{\alpha,new}} + \tau_\mu$$
   - if $\alpha = \mu$, generate a random number r_1, according to an exponential distribution with parameter a_μ and set $\tau_\mu \leftarrow \tau_\mu + r_1$

   Update the node in the indexed priority queue that holds τ_α.

4. Go to 2 and repeat until a prefixed simulation time is reached.

Both Multicompartimental Gillespie Algorithm and Multicompartimental Next Reaction Method are the core of the Direct Stochastic Simulator and Efficient Stochatic Simulator, respectively. One of them, which can be chosen in runtime, will be executed when compiling and simulating a P-Lingua file that starts with `@model<stochastic>`.

3.2 A Simulator for Probabilistic P Systems

Next, we describe how the simulator for probabilistic P systems implements the applicability of the rules to a given configuration.

(a) Rules are classified into sets so that all the rules belonging to the same set have the same left–hand side.

(b) Let $\{r_1, \ldots, r_t\}$ be one of the sets of rules. Let us suppose that the common left-hand side is $u\ [v]_i^{\alpha}$ and their respective probabilistic constants are $c_{r_1}, \ldots, c_{r_t}$. In order to determine how these rules are applied to a given configuration, we proceed as follows:

- One computes the greatest number N so that u^N appears in the parent membrane of i and v^N appears in membrane i.
- N random numbers x such that $0 \leq x < 1$ are generated.
- For each k $(1 \leq k \leq t)$ let n_k be the amount of numbers generated belonging to interval $[\sum_{j=0}^{k-1} c_{r_j}\ ,\ \sum_{j=0}^{k} c_{r_j})$ (assuming that $c_{r_0} = 0$).
- For each k $(1 \leq k \leq t)$, rule r_k is applied n_k times.

4 File formats to define P systems

Together with models and simulators, new formats have been included in P-Lingua 2.0. P-Lingua 1.0 provided a programming language to define P systems and an XML file format [3]. Both have been upgraded to allow representations of P systems which have a cell-like structure. It also supports backwards compatibility, so any file which defines a P system by using P-Lingua 1.0 is also recognized by P-Lingua 2.0 tools. A detailed description of the syntax of P-Lingua programming language, including the new extensions added in order to support the new models, can be found in [4].

A new format has been included as well, the binary format, whose purpose is to use less disk space than the XML format.

At this point, the concepts *input format* and *output format* should be introduced. An input format is a file format which, if a P system is specified in a file by following that format, the P system specified can be processed by the pLinguaCore JAVA library. An output format is a file format which, if a P system is specified in a file by following that format, that file can be generated by the library. These concepts are similar to the source code and object code concepts [3].

For P-Lingua 2.0 framework, P-Lingua programming language is an input format, the binary format is an output format and, eventually, XML is both an input and an output format. This means that P-Lingua programs can be processed by pLinguaCore, binary files can be generated by pLinguaCore and XML files can be both processed and generated by the library.

5 Command-line tools

P-Lingua 1.0 provided command-line tools for simulating P systems and compiling files which specify P systems [3]. In P-Lingua 2.0, the command-line tool general syntax has changed but, as it provides backwards compatibility, all valid actions in P-Lingua 1.0 are still valid in P-Lingua 2.0, as well.

5.1 The compilation command-line tool

The command-line tool general syntax for compiling input files is defined as follows:

```
plingua [-input_format] input_file [-output_format]
output_file [-v verbosity_level] [-h]
```

The command header `plingua` requests the system to compile the P system specified on a file to a file specified on another, whereas the file `input_file` contains the programme that we want to be compiled, and `output_file` is the name of the file that is generated [3]. Optional arguments are in square brackets:

- The option `-input_format` defines the format followed by `input_file`, which should be an input format.
- At this stage, valid input formats are `P-Lingua` and `XML`.
- If no input format is set, the P-Lingua format is assumed.
- The option `-output_format` defines the format followed by `output_file`, which should be an output format.
- At this stage, valid output formats are `XML` and `bin`.
- If no input format is set, the XML format is assumed by default.
- The option `-v` verbosity level is a number between 0 and 5 indicating the level of detail of the messages shown during the compilation process [3].
- The option `-h` displays some help information [3].

5.2 The simulation command-line tool

The simulations are launched from the command line as follows:

```
plingua_sim input_file -o output_file [-v verbosity level] [-h] [-to timeout]
[-st steps] [-mode simulatorID] [-a] [-b]
```

The command header `plingua_sim` requests the system to simulate the P system specified on a file, whereas `input_xml` is an XML document where a P system is formatted on, and output file is the name of the file where the report about the simulated computation will be saved [3]. Optional arguments are in brackets:

- The option `-v` verbosity level is a number between 0 and 5 indicating the level of detail of the messages shown during the compilation process [3]. If no value is specified, it is 3 by default.
- The option `-h` displays some help information [3].
- The option `-to` sets a timeout for the simulation defined in timeout (in milliseconds), so when the time out has elapsed the simulation is halted. If the simulation has reached a halting configuration before the time out has elapsed this option has no effect.
- The option `-st` sets a maximum number of steps the simulation can take (defined in steps), so when the time out has elapsed the simulation comes to a halt. If the simulation has reached a halting configuration or the time out has elapsed (in case the option `-to` is set) before the specified number of steps have been taken this option has no effect.

- The option `-mode` sets the specific simulator to simulate the P system (defined in simulatorID). This option reports an error in case the simulator defined by simulatorID is not a valid simulator for the P system model.

- The option `-a` defines if the simulation can take alternative steps. This option reports an error if the simulator does not support alternative steps.

- The option `-b` defines if the simulation can step backwards. As every simulator supports stepping backwards, this option does not report errors.

6 The pLinguaCore JAVA library

pLinguaCore © is a JAVA library which performs all functions supported by P-Lingua 2.0, that is, models definition, simulators and formats. This library reports the rules and membrane structure read from a file where a P system is defined, detects errors in the file, reports them. If the P system is defined in P-Lingua programming language, it locates the error in the file. This library performs simulations by using the simulators implemented as well as taking into account all options defined. It reports the simulation process, by displaying the current configuration as text and reporting the elapsed time. Eventually, this library translates files that define a P system between formats, for instance, from P-Lingua language format to binary format. This library is free software published under LGPL license [12], so everyone who is interested can upgrade, change and distribute it respecting the license restrictions.

7 A tool for simulating ecosystems based on P-Lingua

The Bearded Vulture (Gypaetus barbatus) is an endangered species in Europe that feeds almost exclusively on bone remains of wild and domestic ungulates. In [1], it is presented a first model of an ecosystem related to the Bearded Vulture in the Pyrenees (NE Spain), by using probabilistic P systems where the inherent stochasticity and uncertainty in ecosystems are captured by using probabilistic strategies. In order to validate experimentally the designed P system, the authors have developed a simulator that allows them to analyze the evolution of the ecosystem under different initial conditions. That software application is focused on a particular P system, specifically, the initial model of the ecosystem presented in [1]. With the aim of improving the model, the authors are adding ingredients to it, as new species and more complex behaviour for the animals. The improved model, together with results of virtual experiments made with this software application, is exhaustively described in [2].

A new GPL [11] licensed JAVA application with a friendly user-interface sitting on the pLinguaCore JAVA library has been developed. This application provides a flexible way to check, validate and improve computational models of ecosystem based on P systems instead of designing new software tools each time new ingredients are added to the models. Furthermore, it is possible to change the initial parameters of the modelled ecosystem in order to make the virtual experiments suggested by experts. These experiments will provide results that can be interpreted in terms of hypotheses. Finally, some of these hyphoteses will be selected by the experts in order to be checked in real experiments.

8 Conclusions and future work

Creating a programming language to specify P systems is an important task in order to facilitate the development of software applications for membrane computing.

In [3] P-Lingua was presented as a programming language to define active membrane P systems with division rules. The present paper extends that language to other models: transition P systems,

symport/antiport P systems, active membranes P systems with division or creation rules, probabilistic P systems and stochastic P systems.

We have developed a JAVA library which recognizes the models, implements several simulators for each model and defines different formats to codify P systems, like the P-Lingua one or a new binary format. This library can be expanded to define new models, simulators and formats.

It is possible to select different algorithms to simulate a P system, for example, there are two different algorithms for stochastic P systems. The library can be used inside other software applications, in this sense, we present a tool for virtual experimentation of ecosystems.

An internet website [14], still under construction, will be available to download the applications, libraries, source-code and technical reports, as well as provide information about the progress of the P-Lingua project. In addition, this site aims to be a meeting point for users and developers through the use of web-tools as forums.

The syntax of the P-Lingua programming language is sufficiently standard for specifying different models of cell-like P systems. However, a new version of the language is necessary in order to specify tissue-like P systems but this will be the aim of a future work.

Although P-Lingua 2.0 provides a way to simulate and compile P-systems, command-line tools are usually not user-friendly. It means it is not easy and intuitive for people to use them. For this purpose, P-Lingua 1.0 provided an Integrated Development Environment (IDE) [3], which eased the way people could use P-Lingua 1.0. For P-Lingua 2.0, a new IDE, called pLinguaPlugin, is being developed. Such an application is integrated into the Eclipse platform [13], so it makes the most of Eclipse's capabilities to provide a framework for translating, developing and testing P systems. It aims to be user-friendly and useful for P system researchers.

8.1 Acknowledgement

The authors acknowledge the valuable assistance given by Mario J. Perez-Jimenez whose vast experience and human quality was essential for us in taking our first steps in scientific research. The authors also acknowledge the support of the project TIN2006–13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

Bibliography

[1] M. Cardona, M.A. Colomer, M.J. Pérez–Jiménez, D. Sanuy, A. Margalida. Modeling Ecosystems Using P Systems: The Bearded Vulture, a Case Study. *Lecture Notes in Computer Science*, 5391, 137–156, 2009

[2] M. Cardona, M.A. Colomer, A. Margalida, I. Pérez–Hurtado, M.J. Pérez–Jiménez, D. Sanuy. P System Based Model of an Ecosystem of the Scavenger Birds, *Proceedings of the 7th Brainstorming Week on Membrane Computing*, Vol. I, 65–80, *in press*.

[3] D. Díaz–Pernil, I. Pérez–Hurtado, M.J. Pérez–Jiménez, A. Riscos–Núñez. A P-Lingua programming environment for Membrane Computing, *Proceedings of the 9th Workshop on Membrane Computing*, 155–172, 2008.

[4] M. García–Quismondo, R. Gutiérrez–Escudero, I. Pérez–Hurtado, M.J. Pérez–Jiménez. P-Lingua 2.0: New Features and First Applications, *Proceedings of the 7th Brainstorming Week on Membrane Computing*, Vol. I, 141–168, *in press*.

[5] M.A. Gibson and J. Bruck. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels, *J. Phys. Chem.*, 104, 1876–1889, 2000.

[6] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions, *J. Phys. Chem.*, 81, 2340–2361, 1977.

[7] Gh. Păun. Membrane computing. An introduction, Springer–Verlag, Berlin, 2002.

[8] Gh. Păun. Computing with Membranes, *Journal of Computer and System Sciences* 61(1) 108–143, 2000.

[9] Gh. Păun. P systems with active membranes. *Journal of Automata, Languages and Combinatorics*, 6, 1, 75–90, 2001.

[10] F.J. Romero–Campero. P Systems, a Computational Modelling Framework for Systems Biology, Doctoral Thesis, University of Seville, Department of Computer Science and Artificial Intelligence, 2008.

[11] The GNU General Public License: `http://www.gnu.org/copyleft/gpl.html`

[12] The GNU Lesser General Public License: `http://www.gnu.org/copyleft/lgpl.html`

[13] The Eclipse Project: `http://www.eclipse.org`

[14] The P-Lingua website: `http://www.p-lingua.org`

Manuel García-Quismondo Fernández was born in June 11, 1985. He got his degree in *Ingeniería Técnica en Informática de Sistemas* in the University of Sevilla in June 2007. Currently, he is about to get another degree, this time in *Ingeniería en Informática*, at the same university. Since September 2008, he has been a granted student at the Department of Computer Science and Artificial Intelligence. He has developed a software application called *pLinguaPlugin* and co-developed another one called *pLinguaCore*, directed by Agustín Riscos-Nuñez and Ignacio Pérez-Hurtado.

Rosa Gutiérrez-Escudero was born in August 16, 1984. She received her degree in Computer Science in June 2008 from the University of Sevilla. Since September 2008, she has been a PhD student at the Department of Computer Science and Artificial Intelligence of the University of Sevilla (Spain). She is also a member of the Research Group on Natural Computing in the same University. Her main research interests within the Membrane Computing area are computer simulation and Complexity Theory.

Miguel A. Martínez-del-Amor was born in July 10, 1984. He received his degree in Computer Science from the University of Murcia (Spain) in June 2008. Currently, he is a PhD student at the Department of Computer Science and Artificial Intelligence in the University of Sevilla (Spain). He is also a member of the Research Group on Natural Computing at the same University, and his main research interest is to joint Membrane Computing and High Performance Computing by using efficient computer simulations.

Enrique Orejuela-Pinedo was born in June 7, 1979. He received his degree in Biology in 2005, from the University of Sevilla (Spain). He has cooperated as internal student at the Departament of Ecology and Vegetal Biology in the University of Sevilla. Currently, he is a PhD student at the Department of Computer Science and Artificial Intelligence in the University of Sevilla. He is also a member of the Research Group on Natural Computing at the same University, and his main research interests are Natural Computing and Membrane Computing, specially computational models of ecosystems.

Ignacio Pérez-Hurtado was born in September 21, 1977. He received his degree in Computer Science in October 2003. He was systems analyst in a company for three years. Since September 2006, he has been a PhD student at the Department of Computer Science and Artificial Intelligence in the University of Sevilla (Spain). He is an associate professor at the same department. He is also

a member of the Research Group on Natural Computing at the said University, and his main research interests within the membrane computing are computer simulation, models for biological processes and Complexity Theory.