Membrane Computing Crash Course

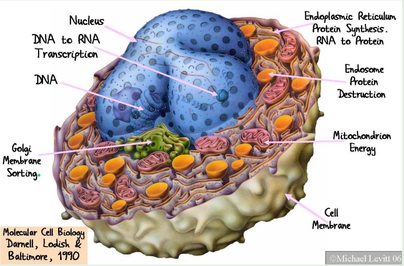
Agustín Riscos-Núñez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla

14^{th} Brainstorming Week on Membrane Computing
Tutorial session
February 1, 2016, Sevilla, Spain

Inspiration

Processes taking place in the **compartmental** structure of a cell.



Membrane Computing

Gh. Păun (oct. 1998 – feb. 2000)

Thomson Institute for Scientific Information (**ISI**)

- Seminal paper[a] awarded as a *Fast Breaking Paper* (feb. 2003).
- Declared by ISI as a *Fast Emerging Research Front in Computer Science* (nov. 2003).

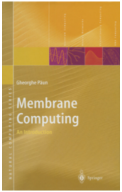
[a] Gh. Păun. Membrane Computing. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for Computer Science-TUCS Report* Nr. 208, 1998.

It has developed quickly into a vigorous scientific discipline.

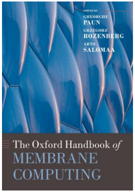
- ⋆ More than 60 PhD theses
- ⋆ International Conference on Membrane Computing (16th edition).
- ⋆ Brainstorming Week on Membrane Computing (14th edition).
- ⋆ Asian Conference on Membrane Computing (4th edition).

Basic References

Gh. Păun. **Membrane Computing. An Introduction**, Springer, Natural Computing Series, 2002.

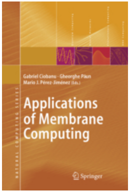


Gh. Păun, G. Rozenberg, A. Salomaa. **The Oxford Handbook of Membrane Computing**, Oxford University Press, 2010.

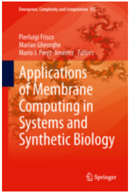


References of Applications

G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez (eds.) **Applications of Membrane Computing** Natural Computing Series, Springer, 2006.



P. Frisco, M. Gheorghe, M. J. Pérez-Jiménez (eds.) **Applications of Membrane Computing in Systems and Synthetic Biology** Series: Emergence, Complexity and Computation, Springer, 2014.



Membrane Systems

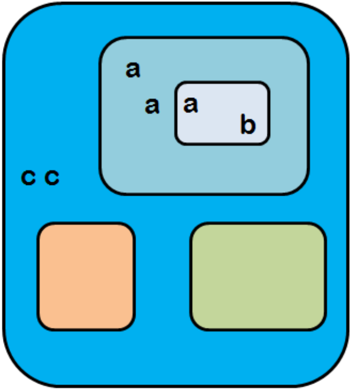


Figure : A P system

- Multisets of objects
- Membranes (regions)
- Rules
  - Objects
  - Membranes
- Environment

Membrane Systems

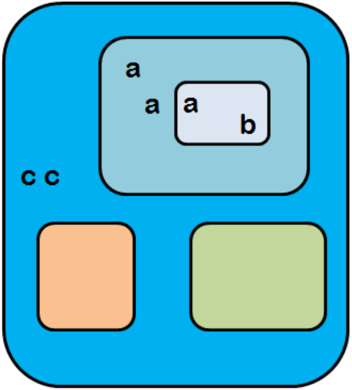
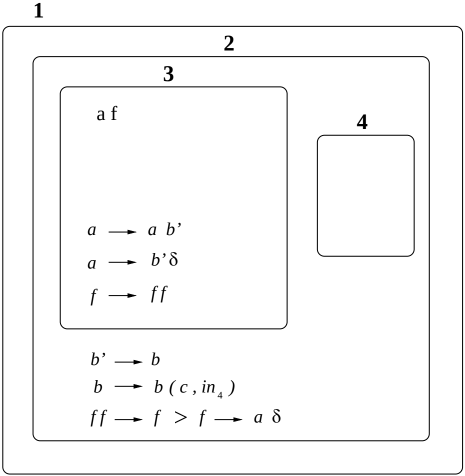


Figure : A P system

- Machine oriented model.
- Non-deterministic devices.
- Two levels of parallelism (objects & membranes).
- Global clock.

An example (I)

A membrane system generating the set $\{n^2 : n \geq 1\}$.



An example (II)

Step	Membrane 1	Membrane 2	Membrane 3	Membrane 4
0			af	
1			$ab' f^2$	
2			$ab'^2 f^{2^2}$	
3			$ab'^3 f^{2^3}$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m			$ab'^m f^{2^m}$	
$m+1$		$b'^{(m+1)} f^{2^{m+1}}$	*dissolved*	
$m+2$		$b^{m+1} f^{2^m}$	*dissolved*	
$(m+2)+1$		$b^{m+1} f^{2^{m-1}}$	*dissolved*	c^{m+1}
$(m+2)+2$		$b^{m+1} f^{2^{m-2}}$	*dissolved*	$c^{2(m+1)}$
$(m+2)+3$		$b^{m+1} f^{2^{m-3}}$	*dissolved*	$c^{3(m+1)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(m+2)+m$		$b^{m+1} f^{2^{m-m}}$	*dissolved*	$c^{m(m+1)}$
$2m+3$	ab^{m+1}	*dissolved*	*dissolved*	$c^{(m+1)(m+1)}$

Diversity of definitions

Syntax

Objects

- strings, arrays, spikes, ...

Membranes

- tree-like / tissue-like structure
- labels, charges, proteins, ...

Diversity of definitions

Semantics

Rules

- selecting which **types**
  (e.g. forbidding dissolution, using only communication, . . .)
- controlling **applicability**
  (e.g. priorities, permitting / forbidding conditions,
  alternatives to maximal parallelism, . . .)

Diversity of interpretations

- *Generative* devices: fixed initial configuration, we collect the outputs of all the non-deterministic computations.
- *Computing* devices: given an input (encoded somehow), compute the resulting output multiset.
- *Decision* tools: special objects *yes* and *no*, s.t. their presence / absence in the output decides whether the given input was accepted by the P system or not.
- *Simulation* tools: no halting configuration, the output is the computation.

Diversity of interpretations

- *Generative* devices: fixed initial configuration, we collect the outputs of all the non-deterministic computations.
- *Computing* devices: given an input (encoded somehow), compute the resulting output multiset.
- *Decision* tools: special objects *yes* and *no*, s.t. their presence / absence in the output decides whether the given input was accepted by the P system or not.
- *Simulation* tools: no halting configuration, the output is the computation.

Diversity of interpretations

- *Generative* devices: fixed initial configuration, we collect the outputs of all the non-deterministic computations.
- *Computing* devices: given an input (encoded somehow), compute the resulting output multiset.
- *Decision* tools: special objects *yes* and *no*, s.t. their presence / absence in the output decides whether the given input was accepted by the P system or not.
- *Simulation* tools: no halting configuration, the output is the computation.

Diversity of interpretations

- *Generative* devices: fixed initial configuration, we collect the outputs of all the non-deterministic computations.
- *Computing* devices: given an input (encoded somehow), compute the resulting output multiset.
- *Decision* tools: special objects *yes* and *no*, s.t. their presence / absence in the output decides whether the given input was accepted by the P system or not.
- *Simulation* tools: no halting configuration, the output is the computation.

Main research directions

- Theoretical Foundations
  - **Universality** results
    Generative / accepting power equivalent to . . .
  - What if . . . ?
  - Formalization
- Computational Complexity
  - **Efficient** solutions to **hard** problems
  - **P conjecture**
- Practical Approach
  - **Simulators**
  - **Modelling**
  - Generative music, Robot control, Model checking, . . .

Implementation

Not yet, but . . .

In vitro / In vivo

- Artificial or synthetic membranes / capsides
- Micro reactors

In silico

- Ciobanu, Guo (2003)
- Petreska, Teuscher (2003)
- Nguyen, Kearney, Gioiosa (2006)
- Ciobanu, Ipate (2013)

Why Simulator Software?

Applications of simulators

- Pedagogical tools
- Support research in Membrane Computing
  - provide *experimental* validation
  - debugging assistant
- **Running virtual experiments**

- Efficient in practice!

Historical overview

Almost from the very beginning

First software simulators

- Prolog (Malita, 2000).
- Visual C++ (Ciobanu, Paraschiv, 2001).
- Haskell (Arroyo, Luengo, Baranda, de Mingo, 2002).
- Scheme (Balbontín, Pérez, Sancho, 2002).

Stochastic bio-processes

- *Cyto-Sim* (Sedwards & Mazza, Bioinformatics 2007)
- *MetaPlab* / *MpTheory* (Castellini & Manca, WMC 2008)
- *BioSimWare* (Besozzi et al, CMC 2010)
- *Infobiotics Workbench* (Blakes et al, Bioinformatics 2011)

Historical overview (cont.)

Towards *in silico* implementation

Parallel / Distributed simulation

- Microcontrollers (UPM team, WMC 2006)
- GPU (Martínez-del-Amor et al, BWMC 2009)
- FPGA (Verlan & Quirós, CMC 2012)
- Big Data (UAM team, IWANN 2011)
  (Ciobanu & Ipate, CMC 2013)

Software developed at RGNC

In this talk

- P-Lingua + *pLinguaCore*
- PMCGPU
- MeCoSim

Starting point

A “programming language” to define P systems

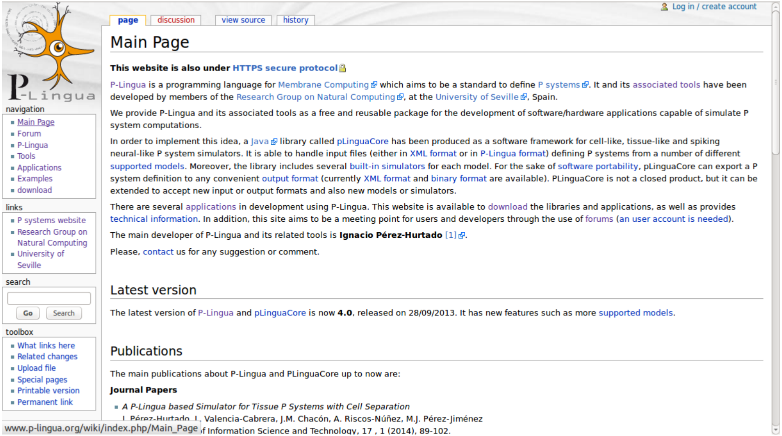
- A *standard* language for the MC community
- Unambiguous syntax and semantics

Heterogeneous simulators

- Analogous structure
- Specific inputs by means of *ad-hoc* compilations

P-Lingua wiki: Everyone's invited!

http://www.p-lingua.org



Log in / create account

page | discussion | view source | history

P-Lingua

navigation
- Main Page
- Forum
- P-Lingua
- Tools
- Applications
- Examples
- download

links
- P systems website
- Research Group on Natural Computing
- University of Seville

search

Go | Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Main Page

This website is also under HTTPS secure protocol

P-Lingua is a programming language for Membrane Computing which aims to be a standard to define P systems. It and its associated tools have been developed by members of the Research Group on Natural Computing, at the University of Seville, Spain.

We provide P-Lingua and its associated tools as a free and reusable package for the development of software/hardware applications capable of simulate P system computations.

In order to implement this idea, a Java library called pLinguaCore has been produced as a software framework for cell-like, tissue-like and spiking neural-like P system simulators. It is able to handle input files (either in XML format or in P-Lingua format) defining P systems from a number of different supported models. Moreover, the library includes several built-in simulators for each model. For the sake of software portability, pLinguaCore can export a P system definition to any convenient output format (currently XML format and binary format are available). PLinguaCore is not a closed product, but it can be extended to accept new input or output formats and also new models or simulators.

There are several applications in development using P-Lingua. This website is available to download the libraries and applications, as well as provides technical information. In addition, this site aims to be a meeting point for users and developers through the use of forums (an user account is needed).

The main developer of P-Lingua and its related tools is **Ignacio Pérez-Hurtado** [1].

Please, contact us for any suggestion or comment.

Latest version

The latest version of P-Lingua and pLinguaCore is now **4.0**, released on 28/09/2013. It has new features such as more supported models.

Publications

The main publications about P-Lingua and PLinguaCore up to now are:

Journal Papers

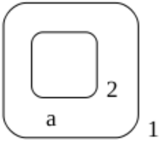
- *A P-Lingua based Simulator for Tissue P Systems with Cell Separation*
  I. Pérez-Hurtado, L. Valencia-Cabrera, J.M. Chacón, A. Riscos-Núñez, M.J. Pérez-Jiménez
  ...f Information Science and Technology, 17 , 1 (2014), 89-102.

www.p-lingua.org/wiki/index.php/Main_Page

P-Lingua: A language to define P systems

- Language close to scientific notation
- Standard, modular and parametric
- Desacoupled from its applications
- Many supported classes of P systems: *cell–like*, *tissue–like*, *spiking*, *kernel*
- Extensible (next release coming soon!)

Example 1: Active membranes with division rules



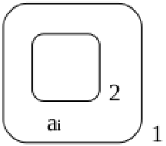
$$[a \longrightarrow a, b]_1$$

$$b[\;]_2 \longrightarrow [c]_2^+$$

$$[c]_2^+ \longrightarrow [d]_2\;[e]_2^-$$

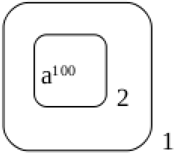
```
@model<membrane_division>
def main()
{
 @mu = [[]’2]’1;
 @ms(1) = a;
 [a --> a,b]’1;
 b[]’2 --> +[c]’2;
 +[c]’2 --> [d]’2 -[e]’2;
}
```

Example 2: Transition P systems



$$a_i \longrightarrow a_{i+1}(here)\, b_i(in_2) \;\; 1 \leq i \leq 10$$
$$[a_i\ [\ ]_2]_1 \longrightarrow [a_{i+1}\ [b_i]_2]_1 \;\; 1 \leq i \leq 10$$

```
@model<transition>
def main()
{
 @mu = [[]'2]'1;
 @ms(1) = a{1};
 [a{i} []'2]'1 --> [a{i+1} [b{i}]'2]'1 : 1<=i<=10;
}
```



$$[a]_2 \xrightarrow{0.4} b[c]_2$$
$$[a]_2 \xrightarrow{0.4} [d]_2$$
$$[a]_2 \xrightarrow{0.2} e[\,]_2$$

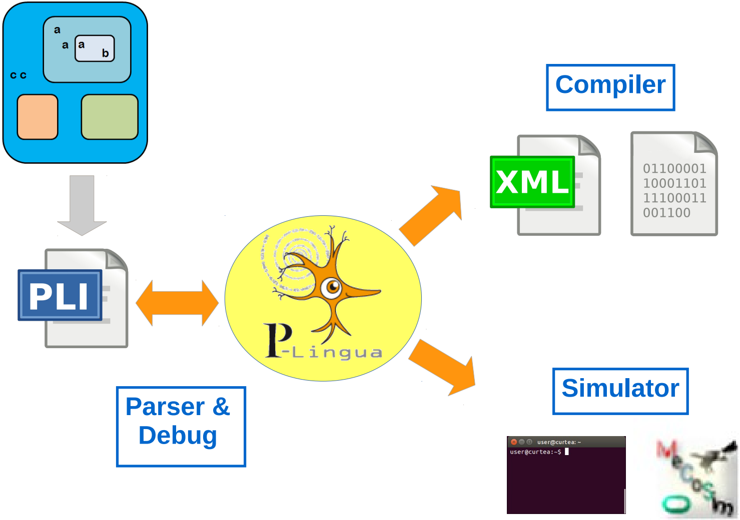
```
@model<probabilistic>
def main()
{
 @mu = [[]'2]'1;
 @ms(2) = a*100;
 [a]'2 --> b[c]'2:: 0.4;
 [a]'2 --> [d]'2 :: 0.4;
 [a]'2 --> e[]'2 :: 0.2;
}
```

Example 4: Tissue P systems

```
@model<tissue_psystems>
def main()
{
@mu = [[]'1 []'2]'0;
@ms(0) = a;
@ms(1) = b*5;
@ms(2) = c*10,d;
[b]'1 <--> [c*2]'2;
[c]'1 <--> [a]'0;
[d]'2  --> [e]'2 [f]'2;
}
```

pLinguaCore functionalities

Free software (GNU GPL license)



pLinguaCore

Java library for parsing, exporting and simulating

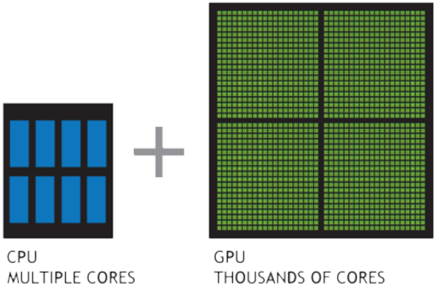
- Errors detection in P-Lingua files
- Able to export to other file formats
  - producing input for external simulators
- “Batteries” included (simulation algorithms)
- Easy to use it within other Java applications

Error example: a division rule in “membrane creation”

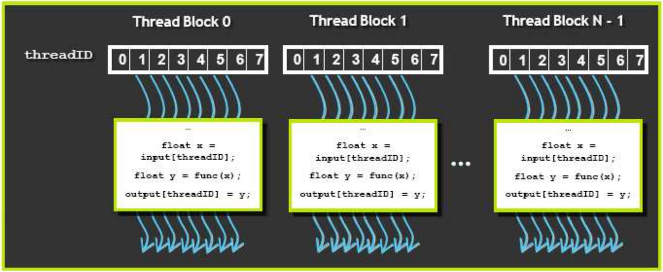
```
Semantics error: The rule doesn't match the
 "membrane_creation" specification in line 38 : 2--28
 Division rules are not allowed
```

GPU computing

- Graphics Processor Unit (GPU)
- Data-parallel computing model:
  - SPMD programming model (***S**ame **P**rogram for **M**ultiple **D**ata*)
  - Shared memory system
- New programming languages: CUDA and OpenCL
- A GPU features thousands of cores



- CUDA programming model[1]
  - Heterogeneous model: CPU (host) + GPU (device).
  - All threads execute the same code (kernel) in parallel.
  - Three-level hierarchy of threads (grid, blocks, threads).
  - Memory hierarchy (global, shared within block).



[1] W.-M. Hwu, D. Kirk. Programming massively parallel processors, Morgan Kaufmann, 2010.

Why is the GPU interesting for simulating P systems?

- Interesting properties:
  - High level of parallelism *(from 16 to 2880 cores)*
  - Shared memory system *(easily synchronized)*
  - Scalability *(multi-GPU systems)*
  - Cheap technology *(cost and maintenance)*

- NVIDIA's Tesla GPUs at RGNC
  - Tesla C1060: *240 cores, 4 GB memory.*
  - Tesla K40: *2880 cores, 12 GB memory.*
  - GeForce GTX 780 Ti: *2880 cores, 3 GB memory.*

PMCGPU project

PMCGPU project (GPL version 3):
`http://sourceforge.net/projects/pmcgpu`

P system model	**FLEXIBLE**	**AD HOC (SAT)**
P systems with active membranes	PCUDA	PCUDASAT
Tissue P systems with cell division		TSPCUDASAT
Population Dynamics P systems	ABCD-GPU	
Enzymatic Numerical P systems	ENPS-GPU	

Design guidelines I

Ingredients representation

- Multisets of objects:
  - Each object should be easily indexed by threads
  - While trying to decrease waste of space
- Charges:
  - Discriminate rules in disjoint sets by the charge
  - Reduce space on selection of rules
- Membrane structure:
  - If a thread block per membrane, just two levels in hierarchy
  - Synchronize each level
- Rule cooperation degree:
  - Define how to control object competition
  - It would require extra synchronization steps

Design guidelines II

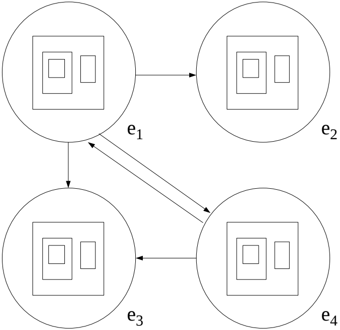
Work assignation

- Thread blocks:
  - Independent data chunks. E.g. membranes, environments, etc.
- Threads:
  - Small information unit. E.g. rules, rule blocks, etc.
  - Recommended between 64 and 512.
  - Should be synchronized with other threads (SIMD).

Synchronization

- Separated Selection and Execution stages
- Temporal copies of configuration to join them

Example. Population Dynamics P systems



Skeleton rules

$$u\ [\ v\ ]_h^{\alpha} \xrightarrow{f_r} u'\ [\ v'\ ]_h^{\beta}$$

Environment rules

$$(a)_{e_j} \xrightarrow{f_r} (b)_{e_k}$$

- Algorithms for probabilistic behaviour

Simulation algorithms

*Rules are applied in a maximal parallel way **according to their probabilities***

General scheme

1. **Selection** process:
   decides which rules to apply and how many times
2. **Execution** process:
   updates the configuration according to rules RHS

BBB: Binomial Block-based

Selection

Loop over **all** blocks (⤨)

- Loop over **all*** rules (⤨)
  - choose randomly the number of applications (*Binomial distrib. on the **remaining** objects*)
  - * the last rule takes it all

DNDP: Direct Non-deterministic Distribution with Probabilities

First Selection (consistency)

Loop over **all** rules (⤨)

- If rule is consistent with previous ones (otherwise discard)
  - choose randomly the number of applications (*Binomial distrib. on the **total** available objects*)

Second Selection (maximality)

Loop over selected rules (ordered by probabilities)

- apply as many times as possible

DCBA: Direct distribution based on Consistent Blocks Algorithm

Selection: 1. Distribution; 2. Maximality; 3. Probability

1. Filter: block charges (F1); block objs. (F2); dummy objs. (F3)

Loop over rows (object,region)

- for each element: $/$ by row sum and $*$ by obj. multiplicity

Loop over columns (blocks)

- number of applications $\equiv$ minimum

2. Loop over blocks (⤮): maximize applications

3. Loop over blocks: (*Multinomial distrib.*) $\Rightarrow$ rule applications

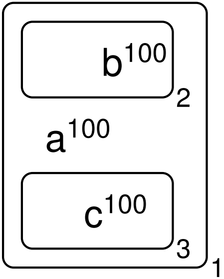
Execution stage

Execution (for BBB, DNDP, DCBA)

Loop over selected rules $\langle r, n \rangle$

- Add $n\cdot$ RHS(r)
- update charges

Example. Proportional Objects distribution



Current configuration

Two active blocks

- $a^2[b]_2$
- $a^4[c^2]_3$

Static value: Inverse of occurrences on LHS

	$a^2[b]_2$	$a^4[c^2]_3$	
$(a, 1)$	$1/2$	$1/4$	
$(b, 2)$	1	$\emptyset$	
$(c, 3)$	$\emptyset$	$1/2$	

Steps 7,8: Divide by the sum of the row

	$a^2[b]_2$	$a^4[c^2]_3$	
$(a, 1)$	$1/2 \cdot 2/3$	$1/4 \cdot 1/3$	$3/4$
$(b, 2)$	$1 \cdot 1$	$\emptyset$	1
$(c, 3)$	$\emptyset$	$1/2 \cdot 1$	$1/2$

Step 9: Multiplicity in current configuration

	$a^2[b]_2$	$a^4[c^2]_3$	
$(a,1)$	$1/2 \cdot 2/3 \cdot 100$	$1/4 \cdot 1/3 \cdot 100$	
$(b,2)$	$1 \cdot 1 \cdot 100$	$\emptyset$	
$(c,3)$	$\emptyset$	$1/2 \cdot 1 \cdot 100$	

Selection phase 1: distribution

	$a^2[b]_2$	$a^4[c^2]_3$	
$(a, 1)$	33	8	
$(b, 2)$	100	$\emptyset$	
$(c, 3)$	$\emptyset$	50	
MIN	33 *times*	8 *times*	

Objects consumed:

- $66 + 32 = 98$ copies of a
- 33 copies of b
- 16 copies of c

Selection phase 1: distribution

Second iteration (accuracy $A = 2$)

	$a^2[b]_2$	$a^4[c^2]_3$	
$(a, 1)$	$1/2$	$1/4$	
$(b, 2)$	1	$\emptyset$	
$(c, 3)$	$\emptyset$	$1/2$	

Selection phase 1: distribution

Second iteration (accuracy $A = 2$)

	$a^2[b]_2$	$a^4[c^2]_3$	
$(a, 1)$	$1/2$	$1/4$	
$(b, 2)$	1	$\emptyset$	
$(c, 3)$	$\emptyset$	$1/2$	

(FILTER 2)

Selection phase 1: distribution

Second iteration (accuracy $A = 2$)

	$a^2[b]_2$	$a^4[c^2]_3$	
$(a, 1)$	$1/2$	$1/4$	
$(b, 2)$	1	$\emptyset$	
$(c, 3)$	$\emptyset$	$1/2$	

← null row

(FILTER 2)

Selection phase 1: distribution

Second iteration (accuracy $A = 2$)

	$a^2[b]_2$	
$(a, 1)$	$1/2 \cdot 1 \cdot 2 = 1$	$1/2$
$(b, 2)$	$1 \cdot 1 \cdot 67 = 67$	1
MIN	1 *more time*	

MeCoSim (Membrane Computing Simulator)

MeCoSim

- **General purpose customizable** GUI to control and enhance the functionalities of *pLinguaCore*

- Required features: **flexibility**, **extensibility**.
  - Development of ad-hoc visual applications for different ecosystems in RGNC.
  - Detection of general needs.

MeCoSim (Membrane Computing Simulator)

Goals

- For **P systems designers**:
  - Visual analysis: alphabet, membrane structure, multisets, graphs
  - Support for parsing, debugging and different simulation algorithms (*pLinguaCore*)
  - Delivery of end-user applications
- For **end-users**: custom applications (**black boxes**)
  - set the inputs, run **virtual experiments** and get results

Custom applications

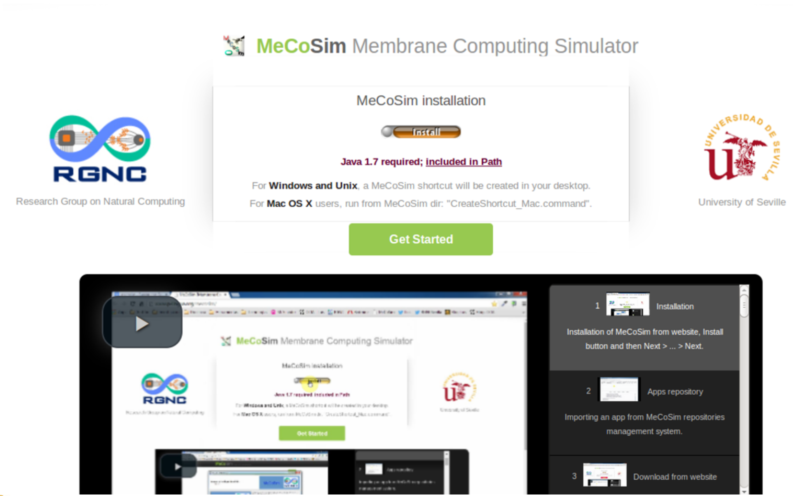
Definition of a custom application

- Visual **arrangement**
- **Input tables** to introduce data
- **Parameters** generation for the model/solution
- **Outputs** to show (tables/charts/graphs)

- **Modelling** and edition of solutions (P-Lingua files)
- **Debugging**
- **Visualization** of *alphabet*, *membrane structure* and *multisets*
- **Virtual experimentation** by **simulating** (halting or number of steps)

Getting the software

http://www.p-lingua.org/mecosim/



About MeCoSim software

- Automatically updated whenever it runs
- **Extensible**: **plugins** architecture (Java / non-Java extensions allowed)
- **Export** option for releasing **end-user applications**
- `http://www.p-lingua.org/mecosim/doc/_downloads/MeCoSimUserManual.pdf`

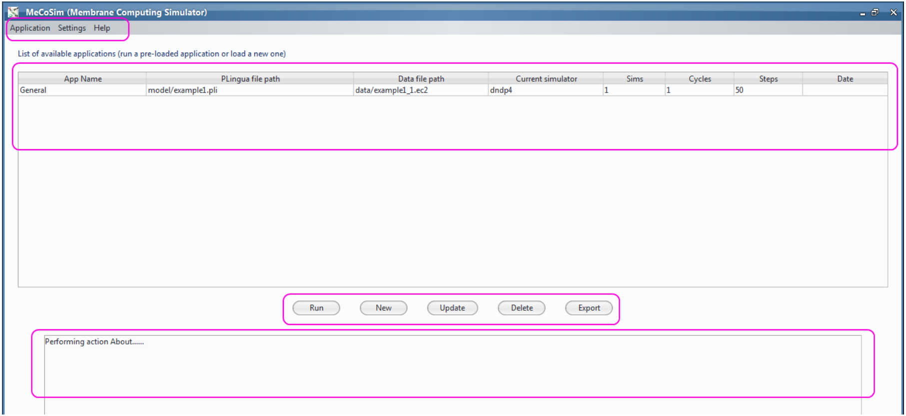
System of repositories

Built-in repositories management

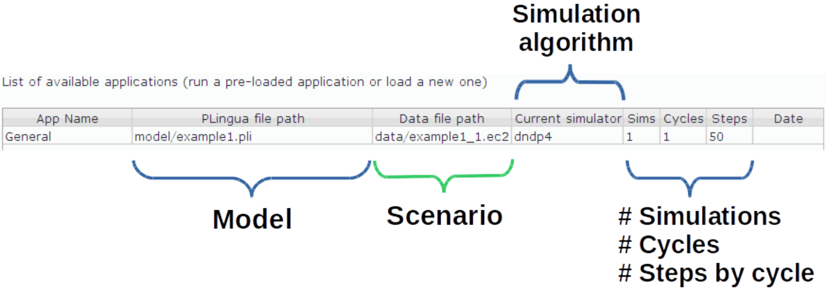
- Plugins
- Apps
- Models
- Scenarios

First glance at MeCoSim

Elements of the main window



Understanding MeCoSim philosophy
Applications



List of available applications (run a pre-loaded application or load a new one)

App Name	PLingua file path	Data file path	Current simulator	Sims	Cycles	Steps	Date
General	model/example1.pli	data/example1_1.ec2	dndp4	1	1	50	

Understanding MeCoSim philosophy

Application, Model, Scenario

Application

- Customized GUI for given model and scenario (`.XLS` file)
- Ready for virtual experimentation (end-user)

Model

- P system definition (`.PLI` file)
- might use parameters

Scenario

- Initial configuration
- Parameter values (if any)

Simulation Algorithm

- for each model, at least one simulation algorithm in *pLinguaCore*
- `“Simulation -> Options -> Simulation Algorithm”`
- can be connected to an external simulator

Simulations

- number of repetitions (if probabilistic behaviour)

Cycles

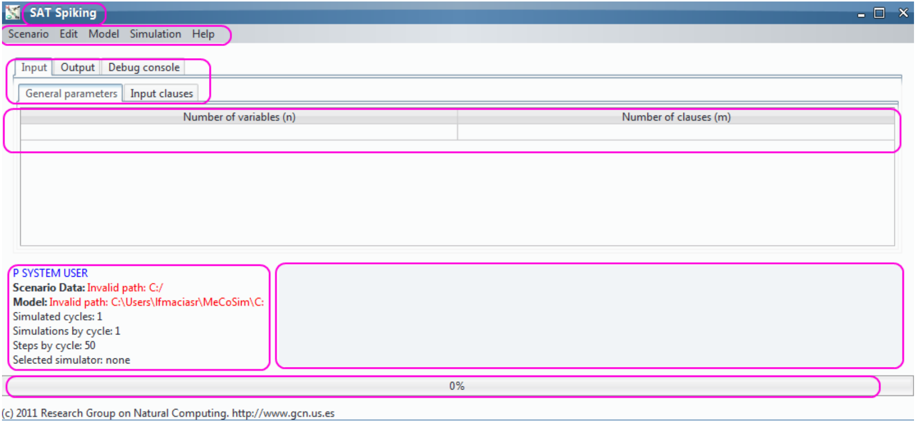
- halting condition (number of cycles)

Steps

- a cycle is the time unit of interest when studying a biological phenomenon (30 min, 1 week, 25 years, etc.)
- for each cycle, several P system steps might be required

Example: Custom app window

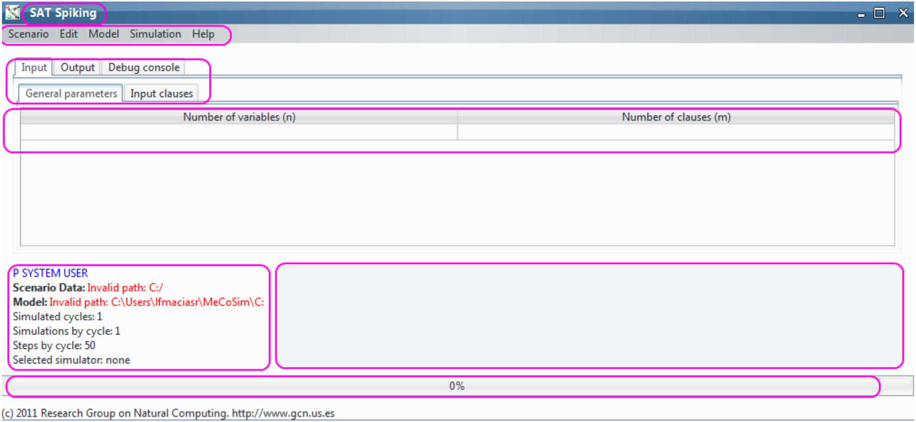
App: Spiking Neural P systems solving SAT



1 **Menu bar**: Scenario, Edit, Model, Simulation, Help

Example: Custom app window

App: Spiking Neural P systems solving SAT

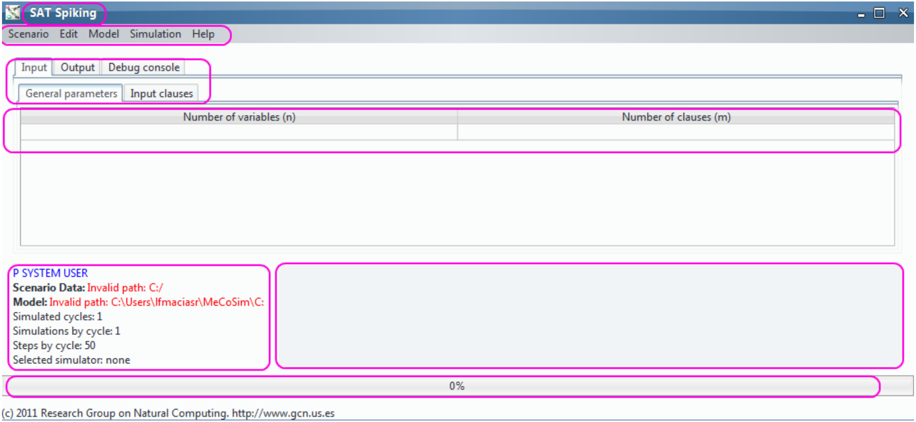


2 **Tabs**: where input and output data are placed

3 **Tables** / **Charts**

Example: Custom app window

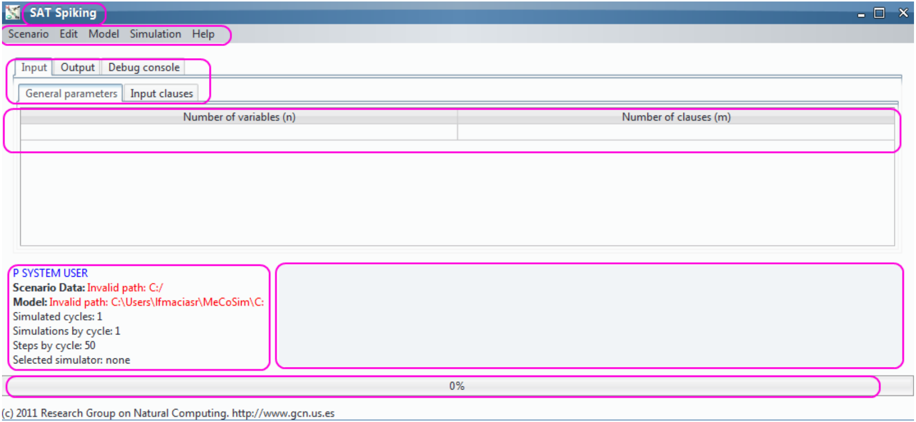
App: Spiking Neural P systems solving SAT



4 **Application info**: user type, scenario, model, ...

Example: Custom app window

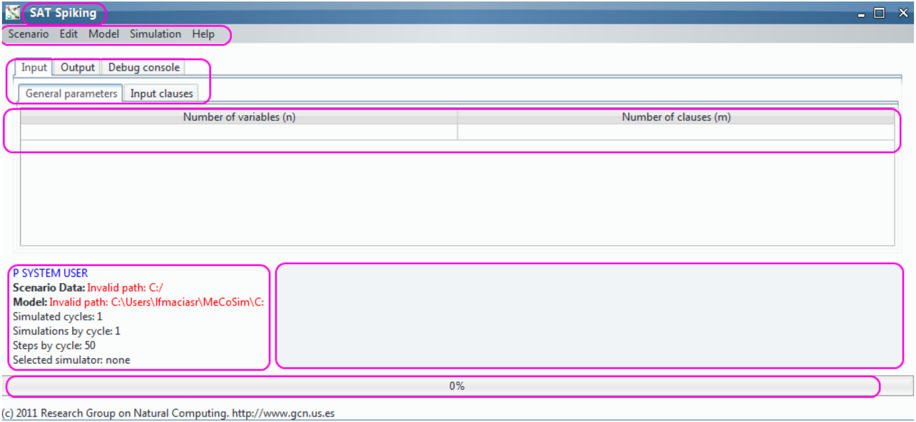
App: Spiking Neural P systems solving SAT



5 **Output console**: shows messages related to the simulation

Example: Custom app window

App: Spiking Neural P systems solving SAT



6 **Progress bar**: shows simulation progress

Modeling

What to Model

- Relevant ingredients
- Relevant features
- Focus on the Dynamics

Why?

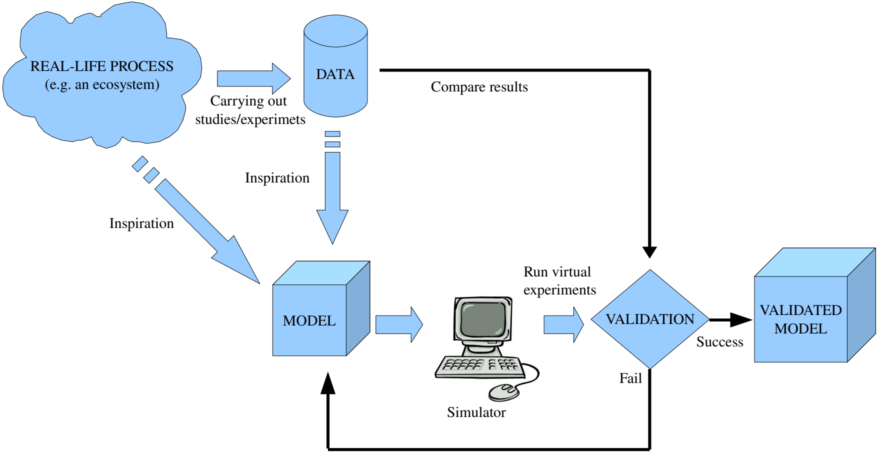
- Analyze / Understand
- Predict / Control

Requirements

- Keep it simple
- Simulation tools (Validation)
- Relevant, Readable, Extensible, Tractable

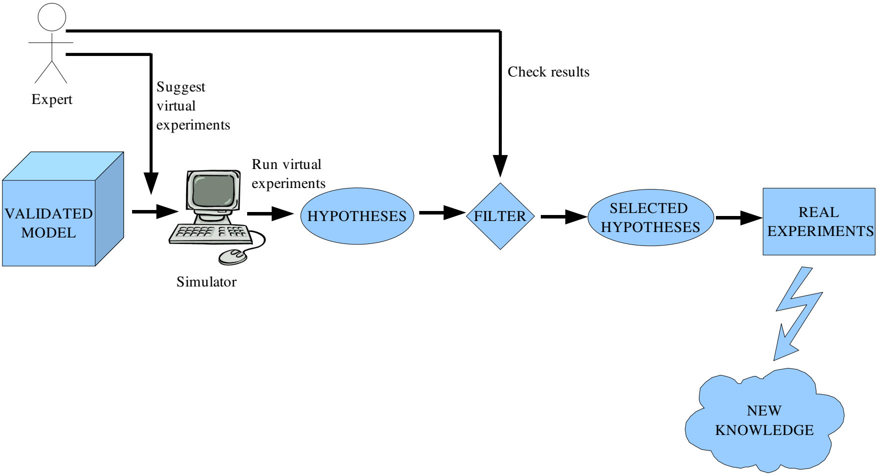
Modeling ecosystems

Validation process



Modeling ecosystems

Virtual Experiments

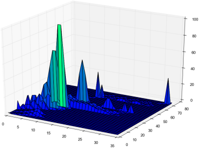


Some studies within the RGNC

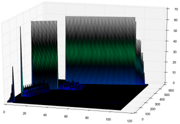
Licence CC by Richard Bartz

photo by Olivier Bareau on Flickr

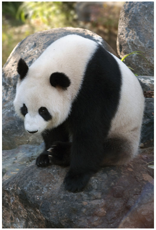
Photo by Amy Benson, U.S. Geological Survey



Partition ($\mathcal{AM}$)

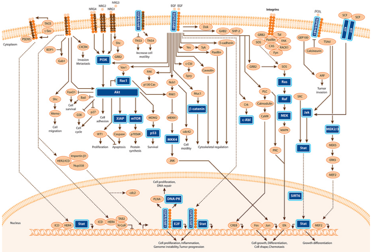


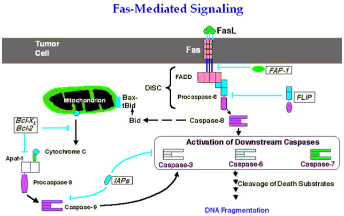
SAT (tissue)

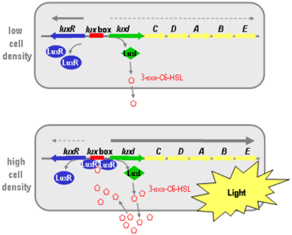


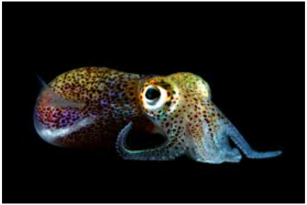
Licence CC by Bilby

Some studies within the RGNC (cont.)









Future (joint) work

Collaborative repositories

- Definitions: P-Lingua
- Bibliography: P page
- (Free) software
- Applications
  - Examples / Case studies
  - Simulation algorithms

Future (joint) work

Please join in!

- **Theoretical foundations**
- **Computational complexity**
- **Applications**
- **Simulators**
- **Implementation**

Thanks for your attention!

Acknowledgements

project TIN2012-37434, funded by the Spanish government, co-financed by EU FEDER funds.

NVIDIA GPU Research Center at Universidad de Sevilla.