Membrane Computing
Crash Course

Agustín Riscos-Núñez
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
Universidad de Sevilla

14th Brainstorming Week on Membrane Computing
Tutorial session
February 1, 2016, Sevilla, Spain
1. Introduction

2. “In silico” Membrane Computing: P-Lingua

3. HPC simulators: GPU Computing

4. MeCoSim

5. Modelling framework

6. Final comments
1 Introduction

2 “In silico” Membrane Computing: P-Lingua

3 HPC simulators: GPU Computing

4 MeCoSim

5 Modelling framework

6 Final comments
Inspiration

Processes taking place in the compartmental structure of a cell.

Molecular Cell Biology
Darnell, Lodish & Baltimore, 1990

©Michael Levitt 90
Membrane Computing

Thomson Institute for Scientific Information (ISI)

- Seminal paper\textsuperscript{a} awarded as a \textit{Fast Breaking Paper} (feb. 2003).
- Declared by ISI as a \textit{Fast Emerging Research Front in Computer Science} (nov. 2003).


It has developed quickly into a vigorous scientific discipline.

- More than 60 PhD theses
- Brainstorming Week on Membrane Computing (14th edition).
Basic References


References of Applications


Membrane Systems

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

Figure: A P system
Membrane Systems

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

Figure: A P system
A membrane system generating the set \( \{ n^2 : n \geq 1 \} \).
### An example (II)

<table>
<thead>
<tr>
<th>Step</th>
<th>Membrane 1</th>
<th>Membrane 2</th>
<th>Membrane 3</th>
<th>Membrane 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>af</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>ab’ (f^2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>(ab'^2 f^{2^2})</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>(ab'^3 f^{2^3})</td>
<td></td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>(m)</td>
<td></td>
<td></td>
<td></td>
<td>(ab'^m f^{2^m})</td>
</tr>
<tr>
<td>(m+1)</td>
<td></td>
<td>(b'(m+1) f^{2^{m+1}})</td>
<td>dissolved</td>
<td></td>
</tr>
<tr>
<td>(m+2)</td>
<td></td>
<td>(b^{m+1} f^{2^m})</td>
<td>dissolved</td>
<td></td>
</tr>
<tr>
<td>((m+2)+1)</td>
<td></td>
<td>(b^{m+1} f^{2^{m-1}})</td>
<td>dissolved</td>
<td>(c^{m+1})</td>
</tr>
<tr>
<td>((m+2)+2)</td>
<td></td>
<td>(b^{m+1} f^{2^{m-2}})</td>
<td>dissolved</td>
<td>(c^{2(m+1)})</td>
</tr>
<tr>
<td>((m+2)+3)</td>
<td></td>
<td>(b^{m+1} f^{2^{m-3}})</td>
<td>dissolved</td>
<td>(c^{3(m+1)})</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>((m+2)+m)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2m+3)</td>
<td>(ab^{m+1})</td>
<td>dissolved</td>
<td>dissolved</td>
<td>(c^{m(m+1)})</td>
</tr>
</tbody>
</table>

A. Riscos-Núñez (Univ. Sevilla)  
Tutorial on Membrane Computing  
BWMC 2016 – Sevilla
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, . . .
Introduction

“In silico” Membrane Computing: P-Lingua

HPC simulators: GPU Computing

MeCoSim

Modelling framework

Final comments
**Implementation**

Not yet, but . . .

<table>
<thead>
<tr>
<th><strong>In vitro / In vivo</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Artificial or synthetic membranes / capsides</td>
<td></td>
</tr>
<tr>
<td>Micro reactors</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>In silico</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciobanu, Guo (2003)</td>
<td></td>
</tr>
<tr>
<td>Petreska, Teuscher (2003)</td>
<td></td>
</tr>
<tr>
<td>Nguyen, Kearney, Gioiosa (2006)</td>
<td></td>
</tr>
<tr>
<td>Ciobanu, Ipate (2013)</td>
<td></td>
</tr>
</tbody>
</table>
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)*
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

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 simulating 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). P-LinguaCore 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] [2].

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
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

\[
\begin{align*}
[a \rightarrow a, b]_1 \\
[b]_2 &\rightarrow [c]_2^+ \\
[c]_2^+ &\rightarrow [d]_2 [e]_2^-
\end{align*}
\]

@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;
}

A. Riscos-Núñez (Univ. Sevilla)
Tutorial on Membrane Computing
BWMC 2016 – Sevilla
Example 2: Transition P systems

\[ a_i \rightarrow a_{i+1}(\text{here}) \ b_i(\text{in}_2) \ 1 \leq i \leq 10 \]
\[ [a_i \ [ ]_2]_1 \rightarrow [a_{i+1} \ [b_i]_2]_1 \ 1 \leq i \leq 10 \]

@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;
Example 3: Probabilistic P systems

\[
\begin{align*}
[a]_2 & \xrightarrow{0.4} b[c]_2 \\
[a]_2 & \xrightarrow{0.4} [d]_2 \\
[a]_2 & \xrightarrow{0.2} e[ ]_2
\end{align*}
\]

@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
1 Introduction

2 “In silico” Membrane Computing: P-Lingua

3 HPC simulators: GPU Computing

4 MeCoSim

5 Modelling framework

6 Final comments
GPU computing

- **Graphics Processor Unit (GPU)**
- **Data-parallel** computing model:
  - SPMD programming model (*Same Program for Multiple Data*)
  - Shared memory system
- New programming languages: CUDA and OpenCL
- A GPU features **thousands of cores**
NVIDIA’s technology

- CUDA programming model
  - 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).

---

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

<table>
<thead>
<tr>
<th>P system model</th>
<th>FLEXIBLE</th>
<th>AD HOC (SAT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P systems with active membranes</td>
<td>PCUDA</td>
<td>PCUDASAT</td>
</tr>
<tr>
<td>Tissue P systems with cell division</td>
<td></td>
<td>TSPCUUDASAT</td>
</tr>
<tr>
<td>Population Dynamics P systems</td>
<td>ABCD-GPU</td>
<td></td>
</tr>
<tr>
<td>Enzymatic Numerical P systems</td>
<td>ENPS-GPU</td>
<td></td>
</tr>
</tbody>
</table>
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
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
\[ u \ [ \nu \ ]_h^{\alpha} \xrightarrow{fr} u' \ [ \nu' \ ]_h^{\beta} \]

Environment rules
\[ (a) e_j \xrightarrow{fr} (b) e_k \]

Algorithms for probabilistic behaviour
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
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


1. Filter: block charges (F1); block objs. (F2); dummy objs. (F3)
   - Loop over rows (object, region)
     - for each element: \( \div \) by row sum and \( \times \) by obj. multiplicity
   - Loop over columns (blocks)
     - number of applications \( \equiv \) minimum

2. Loop over blocks (\( \mathcal{X} \)): maximize applications

3. Loop over blocks: \( Multinomial\ distib. \) \( \Rightarrow \) rule applications
Execution stage

Execution (for BBB, DNDP, DCBA)

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

- Add $n \cdot \text{RHS}(r)$
- Update charges
Example. Proportional Objects distribution

Current configuration

Two active blocks

- $a^2[b]_2$
- $a^4[c^2]_3$
Selection phase 1: distribution

**Static value:** Inverse of occurrences on LHS

<table>
<thead>
<tr>
<th></th>
<th>$a^2[b]_2$</th>
<th>$a^4[c^2]_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a, 1)$</td>
<td>$1/2$</td>
<td>$1/4$</td>
</tr>
<tr>
<td>$(b, 2)$</td>
<td>$1$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$(c, 3)$</td>
<td>$\emptyset$</td>
<td>$1/2$</td>
</tr>
</tbody>
</table>
Selection phase 1: distribution

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

<table>
<thead>
<tr>
<th></th>
<th>$a^2[b]_2$</th>
<th>$a^4[c^2]_3$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a, 1)$</td>
<td>$1/2 \cdot 2/3$</td>
<td>$1/4 \cdot 1/3$</td>
<td>$3/4$</td>
</tr>
<tr>
<td>$(b, 2)$</td>
<td>$1 \cdot 1$</td>
<td>$\emptyset$</td>
<td>$1$</td>
</tr>
<tr>
<td>$(c, 3)$</td>
<td>$\emptyset$</td>
<td>$1/2 \cdot 1$</td>
<td>$1/2$</td>
</tr>
</tbody>
</table>
## Step 9: Multiplicity in current configuration

<table>
<thead>
<tr>
<th></th>
<th>$a^2[b]_2$</th>
<th>$a^4[c^2]_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a, 1)$</td>
<td>$1/2 \cdot 2/3 \cdot 100$</td>
<td>$1/4 \cdot 1/3 \cdot 100$</td>
</tr>
<tr>
<td>$(b, 2)$</td>
<td>$1 \cdot 1 \cdot 100$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$(c, 3)$</td>
<td>$\emptyset$</td>
<td>$1/2 \cdot 1 \cdot 100$</td>
</tr>
</tbody>
</table>
### Selection phase 1: distribution

<table>
<thead>
<tr>
<th></th>
<th>$a^2[b]_2$</th>
<th>$a^4[c^2]_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a, 1)$</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>$(b, 2)$</td>
<td>100</td>
<td>∅</td>
</tr>
<tr>
<td>$(c, 3)$</td>
<td>∅</td>
<td>50</td>
</tr>
<tr>
<td><strong>MIN</strong></td>
<td><strong>33 times</strong></td>
<td><strong>8 times</strong></td>
</tr>
</tbody>
</table>

**Objects consumed:**
- $66 + 32 = 98$ copies of $a$
- 33 copies of $b$
- 16 copies of $c$
Selection phase 1: distribution
Second iteration (accuracy $A = 2$)

<table>
<thead>
<tr>
<th></th>
<th>$a^2[b]_2$</th>
<th>$a^4[c^2]_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a, 1)$</td>
<td>1/2</td>
<td>1/4</td>
</tr>
<tr>
<td>$(b, 2)$</td>
<td>1</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$(c, 3)$</td>
<td>$\emptyset$</td>
<td>1/2</td>
</tr>
</tbody>
</table>
Selection phase 1: distribution
Second iteration (accuracy $A = 2$)

<table>
<thead>
<tr>
<th></th>
<th>$a^2[b]_2$</th>
<th>$a^4[c^2]_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a,1)$</td>
<td>1/2</td>
<td>1/4</td>
</tr>
<tr>
<td>$(b,2)$</td>
<td>1</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$(c,3)$</td>
<td>$\emptyset$</td>
<td>1/2</td>
</tr>
</tbody>
</table>

(Filter 2)
Selection phase 1: distribution
Second iteration (accuracy $A = 2$)

<table>
<thead>
<tr>
<th></th>
<th>$a^2[b]_2$</th>
<th>$a^4[c^2]_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(a, 1)$</td>
<td>$1/2$</td>
<td>$1/4$</td>
</tr>
<tr>
<td>$(b, 2)$</td>
<td>$1$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$(c, 3)$</td>
<td>$\emptyset$</td>
<td>$1/2$</td>
</tr>
</tbody>
</table>

(FILTER 2)

null row
### Selection phase 1: distribution

Second iteration (accuracy $A = 2$)

<table>
<thead>
<tr>
<th></th>
<th>$a^2[b]_2$</th>
<th>(a, 1)</th>
<th>1/2 · 1 · 2 = 1</th>
<th>1/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b, 2)</td>
<td></td>
<td>1 · 1 · 67 = 67</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**MIN** 1 more time
1 Introduction

2 “In silico” Membrane Computing: P-Lingua

3 HPC simulators: GPU Computing

4 MeCoSim

5 Modelling framework

6 Final comments
MeCoSim (Membrane Computing Simulator)

- 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**) and **virtual experiments**
  - 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)
Main functionalities

- **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/

MeCoSim Membrane Computing Simulator

MeCoSim installation

Java 1.7 required; included in Path

For Windows and Unix, a MeCoSim shortcut will be created in your desktop.
For Mac OS X users, run from MeCoSim dir: "CreateShortcut_Mac.command".

Get Started
About MeCoSim software

- Automatically updated whenever it runs
- **Extensible**: plugins architecture (Java / non-Java extensions allowed)
- **Export** option for releasing end-user applications

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)

<table>
<thead>
<tr>
<th>App Name</th>
<th>PLingua file path</th>
<th>Data file path</th>
<th>Current simulator</th>
<th>Sims</th>
<th>Cycles</th>
<th>Steps</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>model/example1.pli</td>
<td>data/example1_1.ec2</td>
<td>dndp4</td>
<td>1</td>
<td>1</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>
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
Understanding MeCoSim philosophy
Simulations, Cycles, Steps

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, ...

(c) 2011 Research Group on Natural Computing, http://www.gcn.us.es
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

1 Introduction

2 “In silico” Membrane Computing: P-Lingua

3 HPC simulators: GPU Computing

4 MeCoSim

5 Modelling framework

6 Final comments
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

REAL-LIFE PROCESS
(e.g. an ecosystem)

DATA

Carrying out studies/experiments

MODEL

VALIDATION

Compare results

Run virtual experiments

Simulator

VALIDATED MODEL

Success

Fail

Inspiration

Inspiration
Modeling ecosystems

Virtual Experiments

1. Expert suggests virtual experiments.
2. Run virtual experiments using a simulator.
3. Check results.
4. Filter hypotheses.
5. Select hypotheses.
6. Validate model.
7. Conduct new experiments.
8. Generate new knowledge.
1 Introduction

2 “In silico” Membrane Computing: P-Lingua

3 HPC simulators: GPU Computing

4 MeCoSim

5 Modelling framework

6 Final comments
Some studies within the RGNC

Partition (\(A_M\))

SAT (tissue)

Licence CC by Richard Bartz

Photo by Amy Benson, U.S. Geological Survey

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

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.