P-Lingua 5: A tutorial

Ignacio Pérez-Hurtado, David Orellana-Martín

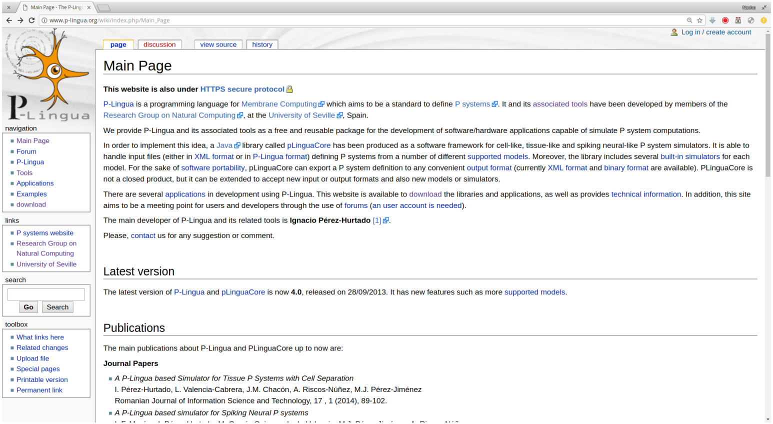
17th Brainstorming Week on Membrane Computing

5-8 February 2019, Sevilla, Spain

Introduction

P-Lingua: A programming language for membrane computing (https://p-lingua.org/)

- Presented for the first time in the 6th BWMC (2008)



Main Page - The P-Lingu ×

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

Log in / create account

page | discussion | view source | history

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

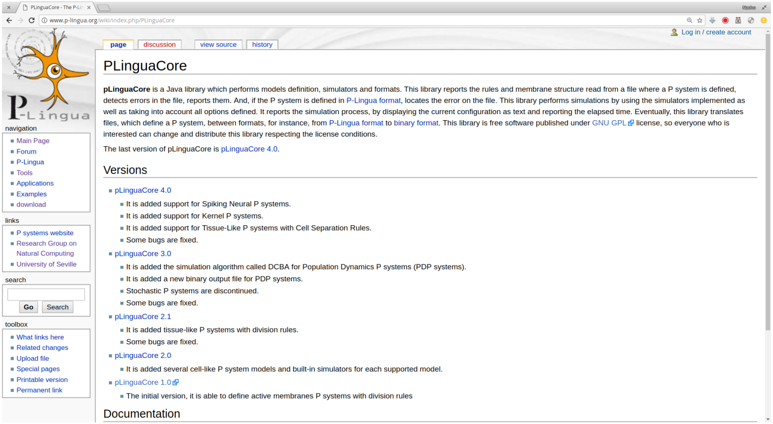
The main publications about P-Lingua and PLinguaCore up to now are:

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
  Romanian Journal of Information Science and Technology, 17 , 1 (2014), 89-102.
- *A P-Lingua based simulator for Spiking Neural P systems*

Introduction

pLinguaCore: A Java library to parse and simulate P-Lingua files


PLinguaCore - The P-Li

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

Log in / create account

page | discussion | view source | history

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

PLinguaCore

pLinguaCore is a Java library which performs 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. And, if the P system is defined in P-Lingua format, locates the error on 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, which define a P system, between formats, for instance, from P-Lingua format to binary format. This library is free software published under GNU GPL license, so everyone who is interested can change and distribute this library respecting the license conditions.

The last version of pLinguaCore is pLinguaCore 4.0.

Versions

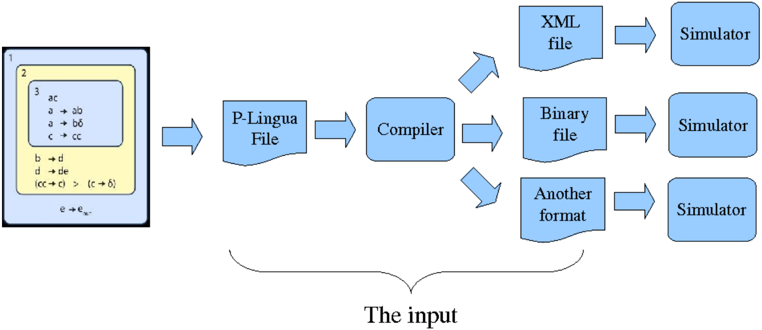
- pLinguaCore 4.0
  - It is added support for Spiking Neural P systems.
  - It is added support for Kernel P systems.
  - It is added support for Tissue-Like P systems with Cell Separation Rules.
  - Some bugs are fixed.
- pLinguaCore 3.0
  - It is added the simulation algorithm called DCBA for Population Dynamics P systems (PDP systems).
  - It is added a new binary output file for PDP systems.
  - Stochastic P systems are discontinued.
  - Some bugs are fixed.
- pLinguaCore 2.1
  - It is added tissue-like P systems with division rules.
  - Some bugs are fixed.
- pLinguaCore 2.0
  - It is added several cell-like P system models and built-in simulators for each supported model.
- pLinguaCore 1.0
  - The initial version, it is able to define active membranes P systems with division rules

Documentation

- pLinguaCore 1.0
  - Initial version
- pLinguaCore 2.0
  - Cell-like P systems with membrane division
  - Transition P systems
- pLinguaCore 2.1
  - Tissue-like P systems with cell-division
- pLinguaCore 3.0
  - PDP systems (several simulation algorithms, DCBA, Binomial...)
  - Tissue-like P systems with cell-separation
- pLinguaCore 4.0
  - Kernel P systems
  - Spiking Neural P systems
  - Tissue-like P systems with cell-separation

Introduction

pLinguaCore: A Java library to parse and simulate P-Lingua files



- Extending pLinguaCore for a new P system variant:
  - Decide a new name to identify the variant.
  - Extend the syntactic/semantic parser.
  - Implement code to generate output formats.
  - Implement one or more simulation algorithms.
- All is hard-coded in the library!

Diagram of the general software methodology to add a new variant in pLinguaCore:

Diagram of the general software methodology to add a new variant in pLinguaCore:

P-Lingua 5: An extension with semantic features

https://github.com/RGNC/plingua

- A new tool written from scratch in C/C++
- A generic compiler for the command-line:
  - Input: P-Lingua files
  - Output: P system definition in XML, JSON or binary format
- P system variants are defined as sets of rule patterns
- Rule patterns can be written in P-Lingua files
- Two derivation modes for rules:
  - Maximal parallel mode
  - Bounded parallel mode
- A C++ generic simulator for the command-line

```
!dam_evolution {
   ?[a -> v]'h;
   ?[a ->  ]'h;
}
!dam_send_in {
   a ?[ ]'h -> ?[b]'h;
}
!dam_send_out {
   ?[a]'h -> b ?[ ]'h;
}
!dam_dissolution {
   ?[a]'h -> b;
   ?[a]'h ->  ;
}
```

```
!dam_division {
   ?[a]'h -> ?[ ]'h ?[ ]'h;
   ?[a]'h -> ?[b]'h ?[ ]'h;
   ?[a]'h -> ?[ ]'h ?[b]'h;
   ?[a]'h -> ?[b]'h ?[c]'h;
}

@model(membrane_division) =
       dam_evolution,
       // evolution rules are maximally parallel
       @1{dam_send_in, dam_send_out, dam_dissolution, dam_division};
       // upper-bound for send_in, send_out, dissolution, division is 1
```

```
@model<membrane_division>
@include "membrane_division_model.pli"
def Sat(m,n)
{
 /* Initial configuration */
 @mu = [[]'2]'1;

 /* Initial multisets */
 @ms(2) = d{1};

 /* Set of rules */
 [d{k}]'2 --> +[d{k}]-[d{k}] : 1 <= k <= n;
```

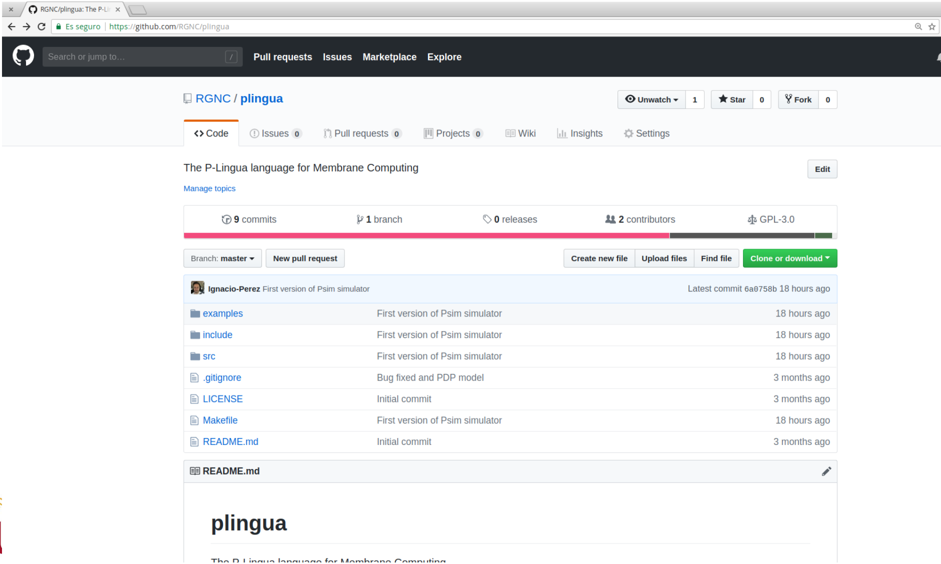
An extension of P-Lingua for semantic features

Example: Transition P systems

```
!transition_evolution /* Limited to rules with 3 inner membranes */
{
       [a -> v]'h;
       [a -> v, @d]'h;
   (?) [a -> v]'h;
   (?) [a -> v, @d]'h;
       [a [ ]'h1 --> v [w]'h1]'h;
       [a [ ]'h1 --> v [w]'h1]'h;
   (?) [a [ ]'h1 --> v [w]'h1]'h;
   (?) [a [ ]'h1 --> v [w]'h1]'h;
       [a [ ]'h1 [ ]'h2 --> v [w1]'h1 [w2]'h2]'h;
       [a [ ]'h1 [ ]'h2 --> v [w1]'h1 [w2]'h2]'h;
   (?) [a [ ]'h1 [ ]'h2 --> v [w1]'h1 [w2]'h2]'h;
   (?) [a [ ]'h1 [ ]'h2 --> v [w1]'h1 [w2]'h2]'h;
       [a [ ]'h1 [ ]'h2 [ ]'h3 --> v [w1]'h1 [w2]'h2 [w3]'h3]'h;
       [a [ ]'h1 [ ]'h2 [ ]'h3 --> v [w1]'h1 [w2]'h2 [w3]'h3]'h;
   (?) [a [ ]'h1 [ ]'h2 [ ]'h3 --> v [w1]'h1 [w2]'h2 [w3]'h3]'h;
   (?) [a [ ]'h1 [ ]'h2 [ ]'h3 --> v [w1]'h1 [w2]'h2 [w3]'h3]'h;
} @model(transition) = transition_evolution;
```

An extension of P-Lingua for semantic features

Let's see more examples from Github



The P-Lingua parser is able to recognize rules with a very flexible syntax:

$$p$$

$$u[v_1[v_{1,1}]_{h_{1,1}}^{\alpha_{1,1}} \ldots [v_{1,m_1}]_{h_{1,m_1}}^{\alpha_{1,m_1}}]_{h_1}^{\alpha_1} \ldots [v_n[v_{n,1}]_{h_{n,1}}^{\alpha_{n,1}} \ldots [v_{n,m_n}]_{h_{n,m_n}}^{\alpha_{n,m_n}}]_{h_n}^{\alpha_n}$$

$$\xrightarrow{q} \textit{ or } \xleftrightarrow{q}$$

$$w_0[w_1[w_{1,1}]_{g_{1,1}}^{\beta_{1,1}} \ldots [w_{1,r_1}]_{g_{1,r_1}}^{\beta_{1,r_1}}]_{g_1}^{\beta_1} \ldots [w_s[w_{s,1}]_{g_{s,1}}^{\beta_{s,1}} \ldots [w_{s,r_s}]_{g_{s,r_s}}^{\beta_{s,r_s}}]_{g_s}^{\beta_s}$$

where:

- p is a priority related to the rule given by a natural number, where a lower number means a higher rule priority.
- q is a probability related to the rule given by a real number in $[0, 1]$.
- $\alpha_i, \alpha_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m_i$ and $\beta_i, \beta_{i,j}, 1 \leq i \leq s, 1 \leq j \leq r_i$ are electrical charges.
- $h_i, h_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m_i$ and $g_i, g_{i,j}, 1 \leq i \leq s, 1 \leq j \leq r_i$ are membrane labels.
- $u, v_i, v_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m_i$ and $w_i, w_{i,j}, 1 \leq i \leq s, 1 \leq j \leq r_i$ are multisets of objects.

Next, there is a list of P-Lingua rule examples matching the general rule syntax:

- `a,b [ d,e*2 ]'h --> [f,g]'h :: q;` where q is the probability of the rule.
- `(p) [a]'h --> [b]'h;` where p is the priority of the rule.
- `[a --> b]'h;`, the left-hand side and right-hand side of evolution rules can be collapsed.
- `+[a]'h --> +[b]'h -[c]'h;` a division rule using electrical charges.
- `[a]'h --> ;` a dissolution rule.
- `a[ ]'h --> [b]'h;` a send-in rule.
- `[a]'h --> b[ ]'h;` a send-out rule.
- `[a --> #]'h;` the symbol # can be optionally used as empty multiset.
- `[a]'1 <--> [b]'0;` a symport/antiport rule in the tissue-like framework.

- The general rule syntax is very permissive
- We introduce a rule pattern matching language to define the subset of allowed rules

```
!rule-type-id
{
pattern1
pattern2
...
patternN
}
```

An extension of P-Lingua for semantic features

Derivation modes

- From an informal point of view, we can see a derivation mode as the way a step of a P system is performed.
- In this extension of P-Lingua, we provide two derivation modes
  - *Maximally parallel derivation mode (max)*. In this mode, we only take multisets of rules that are not extensible.
  - *Bounded-by-rule parallel derivation mode*. In this mode, an upper-bound is defined for multisets of rules that can be selected.
- A P system model can be defined in this new extension of P-Lingua by aggregating the allowed rule patterns and its corresponding derivation modes.

Derivation modes

- $bound_{B_1,\ldots,B_r}$

Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$

Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$

Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max)*.

Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$

Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*

Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$

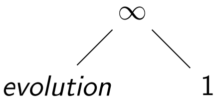
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$

∞

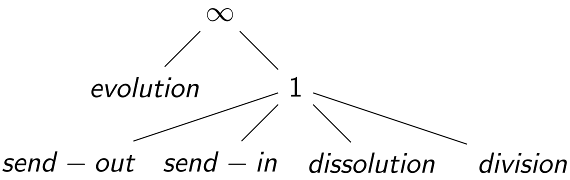
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



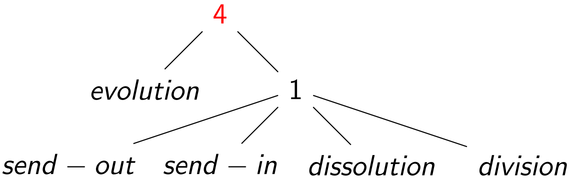
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



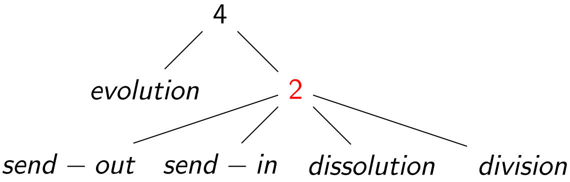
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



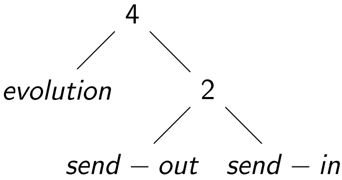
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



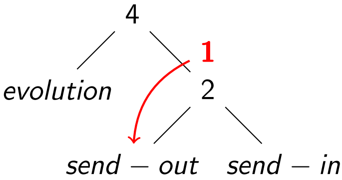
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



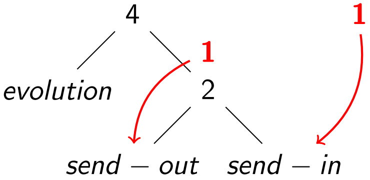
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



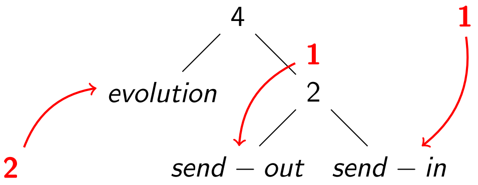
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



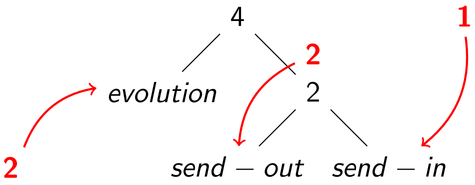
Derivation modes

- $bound_{B_1,\ldots,B_r}$
    - $B_i = j, j \in \{a, b, \ldots\}$
    - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
    - *Maximally parallel derivation mode (max).*
      $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
    - *Bounded-by-rule parallel derivation mode.*
      $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



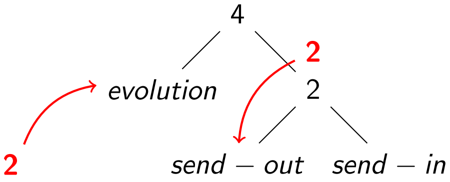
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



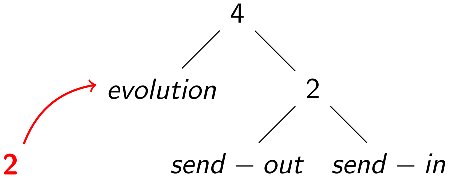
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



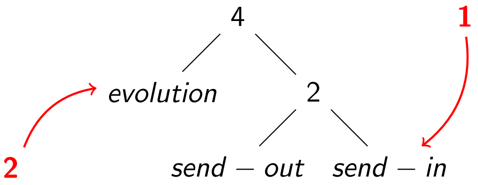
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



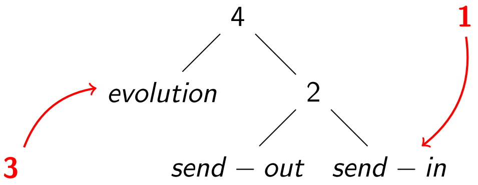
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



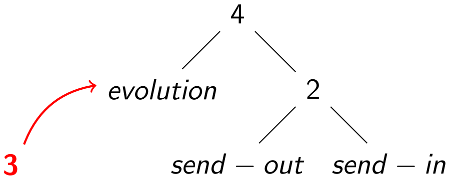
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



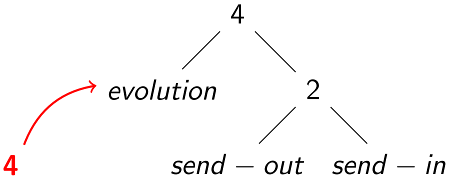
Derivation modes

- *bound*$_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $\textit{bound}_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $\textit{bound}_{\beta_4(evolution,\beta_2(send-out,send-in))}$



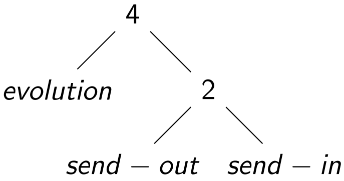
Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



Derivation modes

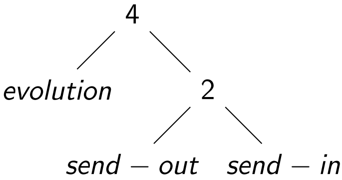
- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $\color{red}{bound}_{evolution,\color{blue}{\beta_1}(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $\color{red}{bound}_{\color{blue}{\beta_4}(evolution,\color{blue}{\beta_2}(send-out,send-in))}$



Not only **syntax**, but also **semantics** definition!

Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution, \beta_1(send-out, send-in, dissolution, division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution, \beta_2(send-out, send-in))}$

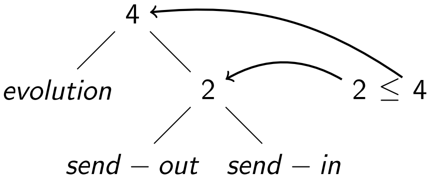


Not only **syntax**, but also **semantics** definition!

Take into account!

Derivation modes

- *$bound_{B_1,\ldots,B_r}$*
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$

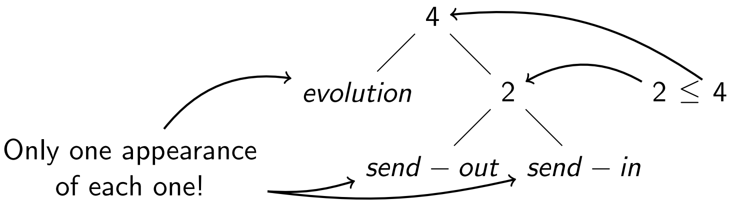


Not only **syntax**, but also **semantics** definition!

Take into account!

Derivation modes

- $bound_{B_1,\ldots,B_r}$
  - $B_i = j, j \in \{a, b, \ldots\}$
  - $B_i = \beta_n(B_1, \ldots, B_{r_i})$
  - *Maximally parallel derivation mode (max).*
    $bound_{evolution,\beta_1(send-out,send-in,dissolution,division)}$
  - *Bounded-by-rule parallel derivation mode.*
    $bound_{\beta_4(evolution,\beta_2(send-out,send-in))}$



Not only **syntax**, but also **semantics** definition!

Take into account!

P-Lingua 5: The command-line simulator

- A command-line simulator has been written in C++
- It reads the output generated by the P-Lingua compiler (XML/Json/binary file defining the P system)
- It optionally reads a file with the initial configuration
- It simulates the P system following the defined semantics in the file
- It outputs one computation until a halting state or a number of simulation steps
- It can be run in a non-randomized mode, where it outputs always the same computation for a given P system
- The final configuration is written to a file, the simulation can be re-started

Conclusions

- A new version of P-Lingua has been designed including rule patterns and semantic definitions
- a command-line compiler has been written from scratch in C/C++
- a command-line simulator tool is also provided
- hard-coding the definition of the P system variants is not longer necessary
- this tool allow the designers to "play" with experimental variants of P systems

Future work

- To refine the syntax for semantic ingredients in P-Lingua.
- To cover variants such as Spiking Neural P systems and Fuzzy Reasoning Spiking Neural P systems
- To write simulators for parallel architectures, such as multi-core processors, pthreads, GPUs, FPGAs...
- To design optimized simulation tools for interesting case studies

Thanks for your attention!