P-Lingua: from user to developer in ∼1 hour

David Orellana-Martín

Research Group on Natural Computing
Dept. of Computer Science and Artificial Intelligence
Universidad de Sevilla, Seville, Spain

February 23, 2022

P-Lingua: Origins

- Specific purpose:

P-Lingua: Origins

- Specific purpose:
  - *Ad-hoc* simulators
  - Problem $\rightarrow$ Simulator

P-Lingua: Origins

- Specific purpose:
  - *Ad-hoc* simulators
  - Problem $\rightarrow$ Simulator (direct algorithm translation)
- Framework oriented:

P-Lingua: Origins

- Specific purpose:
  - *Ad-hoc* simulators
  - Problem $\rightarrow$ Simulator (direct algorithm translation)
- Framework oriented:
  - Within an *specific* framework
  - MetaPlab, Infobiotics Workbench, kPWorkbench...

P-Lingua: Origins

- Specific purpose:
  - *Ad-hoc* simulators
  - Problem $\rightarrow$ Simulator (direct algorithm translation)
- Framework oriented:
  - Within an *specific* framework
  - MetaPlab, Infobiotics Workbench, kPWorkbench... (abstraction of a type of P system)
- General purpose:

P-Lingua: Origins

- Specific purpose:
  - *Ad-hoc* simulators
  - Problem $\rightarrow$ Simulator (direct algorithm translation)
- Framework oriented:
  - Within an *specific* framework
  - MetaPlab, Infobiotics Workbench, kPWorkbench... (abstraction of a type of P system)
- General purpose:
  - Global scope (lots of MC frameworks)
  - P-Lingua framework, UPSimulator, formal frameworks...

P-Lingua: Origins

- Specific purpose:
  - *Ad-hoc* simulators
  - Problem $\rightarrow$ Simulator (direct algorithm translation)
- Framework oriented:
  - Within an *specific* framework
  - MetaPlab, Infobiotics Workbench, kPWorkbench... (abstraction of a type of P system)
- General purpose:
  - Global scope (lots of MC frameworks)
  - P-Lingua framework, UPSimulator, formal frameworks... (abstraction of the concept of *computation*)

P-Lingua framework

- Standard (*de-facto*) for specifying P systems and families of P systems (structured programming).

P-Lingua framework

- Standard (*de-facto*) for specifying P systems and families of P systems (structured programming).
- Different tools within the same framework:
  - Parsers

P-Lingua framework

- Standard (*de-facto*) for specifying P systems and families of P systems (structured programming).
- Different tools within the same framework:
  - Parsers
  - Simulators

P-Lingua framework

- Standard (*de-facto*) for specifying P systems and families of P systems (structured programming).
- Different tools within the same framework:
  - Parsers
  - Simulators
  - Output representations

P-Lingua framework

- Standard (*de-facto*) for specifying P systems and families of P systems (structured programming).
- Different tools within the same framework:
  - Parsers
  - Simulators
  - Output representations
  - pLinguaCore as a *standalone* library

P-Lingua framework

- Standard (*de-facto*) for specifying P systems and families of P systems (structured programming).
- Different tools within the same framework:
  - Parsers
  - Simulators
  - Output representations
  - pLinguaCore as a *standalone* library
  - Command line commands...
- `MeCoSim`

Versions of P-Lingua

- Different versions of P-Lingua:

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9 (deprecated)

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9 (deprecated)
  - P-Lingua 4.0

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9 (deprecated)
  - P-Lingua 4.0 (outdated)

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9 (deprecated)
  - P-Lingua 4.0 (outdated)
  - P-Lingua MeCoSim

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9 (deprecated)
  - P-Lingua 4.0 (outdated)
  - P-Lingua MeCoSim (Java, maintained by L. Valencia-Cabrera)

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9 (deprecated)
  - P-Lingua 4.0 (outdated)
  - P-Lingua MeCoSim (Java, maintained by L. Valencia-Cabrera)
  - P-Lingua 5.0

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9 (deprecated)
  - P-Lingua 4.0 (outdated)
  - P-Lingua MeCoSim (Java, maintained by L. Valencia-Cabrera)
  - P-Lingua 5.0 (C++, maintained by I. Pérez-Hurtado)

Versions of P-Lingua

- Different versions of P-Lingua:
  - P-Lingua 1.0, . . . , P-Lingua 3.9 (deprecated)
  - P-Lingua 4.0 (outdated)
  - **P-Lingua MeCoSim (Java, maintained by L. Valencia-Cabrera)**
  - P-Lingua 5.0 (C++, maintained by I. Pérez-Hurtado)

Parsers and simulators implemented

- Cell-like P systems
- Tissue-like P systems
- Spiking Neural P systems
- PDP systems
- Simple kernel P systems

References

- Sitio web de `P-Lingua` (wiki site)
- Introduction to `P-Lingua` (in Spanish)
- Case studies of different variants of P systems in the MeCoSim website

Hands on `.pli`!

Back to the slides. . .

- Lots of variants. . .

Back to the slides. . .

- Lots of variants. . .
- . . . But how are they implemented?

Back to the slides...

- Lots of variants...
- ...But how are they implemented?
- Subversion/Git repositories for different versions of P-Lingua

Back to the slides...

- Lots of variants...
- ...But how are they implemented?
- Subversion/Git repositories for different versions of P-Lingua
- Remember: **P-Lingua MeCoSim**

What is necessary?

- Name of variant (`@pcolonies`)

What is necessary?

- Name of variant (`@pcolonies`)
- Lexer

What is necessary?

- Name of variant (`@pcolonies`)
- Lexer (keywords)

What is necessary?

- Name of variant (`@pcolonies`)
- Lexer (keywords) (more precisely, keysymbols)

What is necessary?

- Name of variant (`@pcolonies`)
- Lexer (keywords) (more precisely, keysymbols)
- Parser

What is necessary?

- Name of variant (`@pcolonies`)
- Lexer (keywords) (more precisely, keysymbols)
- Parser (structure, objects, rules, programs. . .)

What is necessary?

- Name of variant (`@pcolonies`)
- Lexer (keywords) (more precisely, keysymbols)
- Parser (structure, objects, rules, programs. . .)
- Simulator

What is necessary?

- Name of variant (`@pcolonies`)
- Lexer (keywords) (more precisely, keysymbols)
- Parser (structure, objects, rules, programs. . .)
- Simulator (semantics of P colonies)

What is necessary?

- Name of variant (`@pcolonies`)
- Lexer (keywords) (more precisely, keysymbols)
- Parser (structure, objects, rules, programs. . .)
- Simulator (semantics of P colonies)
- Output (how we visualize the output of the system)

Reusable things: Lexer

- Arrows already defined (-->, <-->)

Reusable things: Lexer

- Arrows already defined (-->, <-->)
- Cells (tissue-like) $\sim$ Agents (P colonies)

Reusable things: Lexer

- Arrows already defined (-->, <-->)
- Cells (tissue-like) $\sim$ Agents (P colonies) (different permeability, same structure)

Reusable things: Lexer

- Arrows already defined (-->, <-->)
- Cells (tissue-like) $\sim$ Agents (P colonies) (different permeability, same structure)
- Symbols as objects

Reusable things: Parser

- Basic creation of a P colony (`@mu`, `@ms(h)`, ...)

Reusable things: Parser

- Basic creation of a P colony (`@mu`, `@ms(h)`, ...)
- Rules (not programs!) (rules can be defined in general for agents with a certain label)

Reusable things: Parser

- Basic creation of a P colony (`@mu`, `@ms(h)`, ...)
- Rules (not programs!) (rules can be defined in general for agents with a certain label) (evolution rules as in cell-like `[ a --> b ]'h`, communication rules as in tissue-like `[a]'h <--> [b]'0`)

Reusable things: Simulator

- Usual evolution of a P system (concept of *computation*)

Reusable things: Simulator

- Usual evolution of a P system (concept of *computation*)
- Semantics about limits of application of rules (non-deterministic, selection of rules/programs...)

Reusable things: Simulator

- Usual evolution of a P system (concept of *computation*)
- Semantics about limits of application of rules (non-deterministic, selection of rules/programs...)
- Priorities

Reusable things: Output

- Structure of the output REALLY similar to other variants (recalling...)

What to do?

- Define variants to add to the framework

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations)

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations) (initial variant was REALLY strict)

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations) (initial variant was REALLY strict) (programs)

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations) (initial variant was REALLY strict) (programs)
- What if...?

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations) (initial variant was REALLY strict) (programs)
- What if...? (to specify some details about the notion of *transition*)

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations) (initial variant was REALLY strict) (programs)
- What if...? (to specify some details about the notion of *transition*)
- Implement simulator (as general as possible)

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations) (initial variant was REALLY strict) (programs)
- What if...? (to specify some details about the notion of *transition*)
- Implement simulator (as general as possible)
- Non-determinism (optional in some systems)

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations) (initial variant was REALLY strict) (programs)
- What if...? (to specify some details about the notion of *transition*)
- Implement simulator (as general as possible)
- Non-determinism (optional in some systems)
- Adapt the output to programs...

What to do?

- Define variants to add to the framework
- Make the variant defined as general as possible (with not so much limitations) (initial variant was REALLY strict) **(programs)**
- What if...? (to specify some details about the notion of *transition*)
- **Implement simulator** (as general as possible)
- Non-determinism (optional in some systems)
- Adapt the output to programs...

Děkuji
Gràcies
谢谢
Obrigado
DANKE
THANKYOU
Gracias
Merci
Dankon
MULȚUMESC