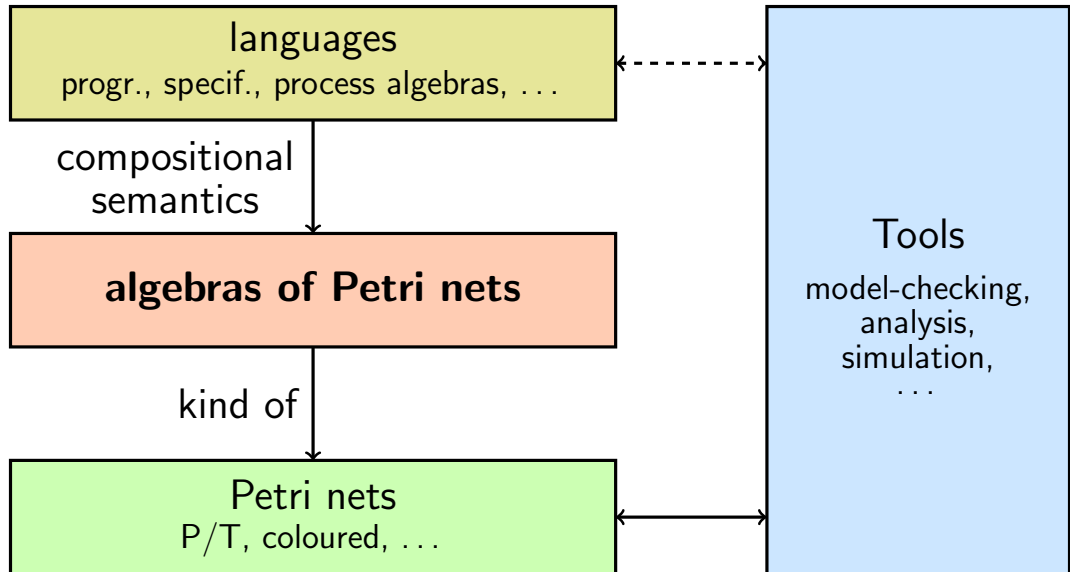
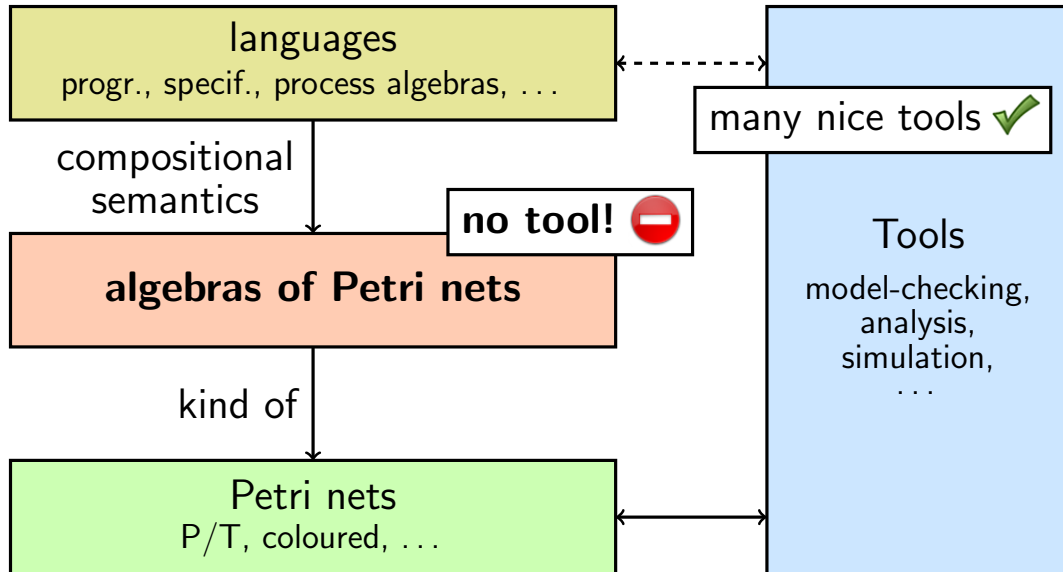

Quickly prototyping Petri nets tools with SNAKES

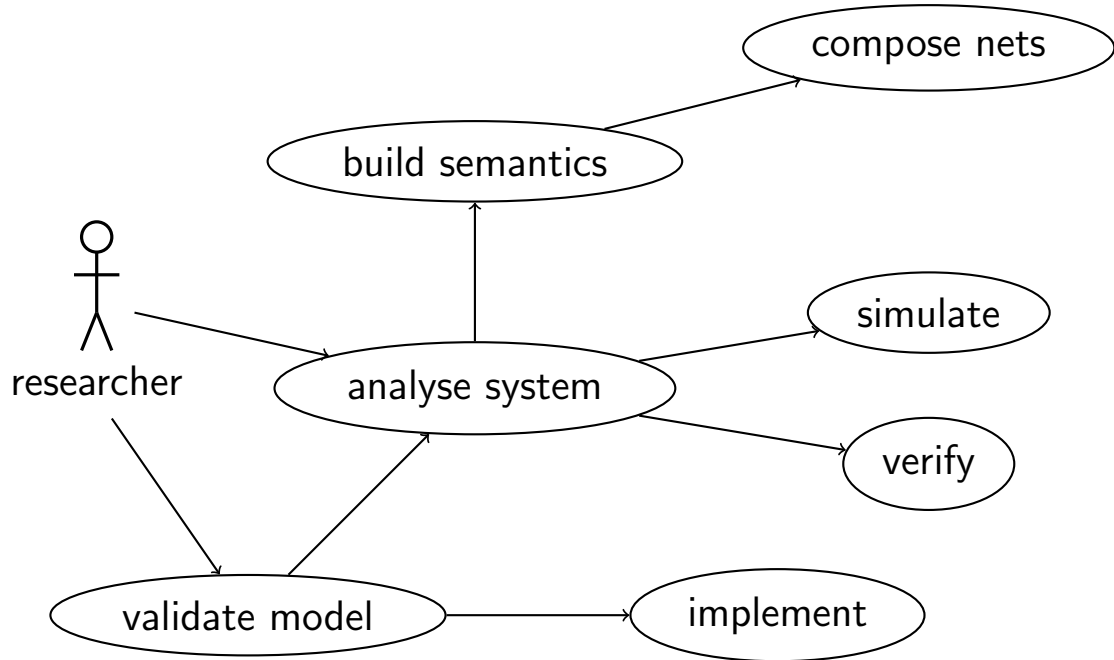
1. Petri Box Calculus and M-nets family



1. Petri Box Calculus and M-nets family



2. Expected use cases



3. SNAKES' main features

- ▶ built-in coloured Petri nets model (general and executable)
- ▶ flexible plugin system (quick implementation or prototyping)
- ▶ easy to use and portable

3. SNAKES' main features

- ▶ built-in coloured Petri nets model (general and executable)
- ▶ flexible plugin system (quick implementation or prototyping)
- ▶ easy to use and portable
- ▶ PBC/M-nets operations included (as plugins)
- ▶ PNML support (beta)
- ▶ graphical rendering (through GraphViz)

3. SNAKES' main features

- ▶ built-in coloured Petri nets model (general and executable)
- ▶ flexible plugin system (quick implementation or prototyping)
- ▶ easy to use and portable
- ▶ PBC/M-nets operations included (as plugins)
- ▶ PNML support (beta)
- ▶ graphical rendering (through GraphViz)
- ▶ programming library (no UI but an API)
- ▶ implemented in Python
- ▶ released under the GNU LGPL (non contagious free software)

4. Architecture

Core library with Petri net classes:

- ▶ places (typed by Boolean functions)
- ▶ transitions (guarded by Boolean functions)
- ▶ tokens (any Python objects)
- ▶ input arcs (values, variables, read arcs, ...)
- ▶ output arcs (as input + arbitrary expressions)
- ▶ ... (multisets, substitutions, ...)

4. Architecture

Core library with Petri net classes:

- ▶ places (typed by Boolean functions)
- ▶ transitions (guarded by Boolean functions)
- ▶ tokens (any Python objects)
- ▶ input arcs (values, variables, read arcs, ...)
- ▶ output arcs (as input + arbitrary expressions)
- ▶ ... (multisets, substitutions, ...)

Plugins: redefine/extend any part of the core library

5. General coloured Petri net model

Transition rule:

- ▶ bind variables on input arcs to actual tokens
- ▶ test guards
- ▶ evaluate output arcs
- ▶ check created tokens against output places
- ▶ consume and produce tokens

5. General coloured Petri net model

Transition rule:

- ▶ bind variables on input arcs to actual tokens
- ▶ test guards
- ▶ evaluate output arcs
- ▶ check created tokens against output places
- ▶ consume and produce tokens

Can be changed:

- ▶ Petri nets equipped with unbounded counters + compact state space
- ▶ Merlin's time Petri nets (almost finished)
- ▶ object Petri nets (use Petri net objects as tokens)
- ▶ ...

6. Actual use cases

- ▶ Petri net semantics of various formalisms
- ▶ modelling & verification of security protocols
(Security Protocol Language $\xrightarrow{\text{SNAKES}}$ Petri nets $\xrightarrow{\text{Helena}}$ model-checking)
- ▶ simulation of MINs (on-chip interconnexion networks)

6. Actual use cases

- ▶ Petri net semantics of various formalisms
- ▶ modelling & verification of security protocols
(Security Protocol Language $\xrightarrow{\text{SNAKES}}$ Petri nets $\xrightarrow{\text{Helena}}$ model-checking)
- ▶ simulation of MINs (on-chip interconnexion networks)
- ▶ prototyping compact state space for Petri nets with counters
(use Lash to handle counters)

6. Actual use cases

- ▶ Petri net semantics of various formalisms
- ▶ modelling & verification of security protocols
(Security Protocol Language $\xrightarrow{\text{SNAKES}}$ Petri nets $\xrightarrow{\text{Helena}}$ model-checking)
- ▶ simulation of MINs (on-chip interconnexion networks)
- ▶ prototyping compact state space for Petri nets with counters
(use Lash to handle counters)
- ▶ teaching support (put theory into practice)

6. Actual use cases

- ▶ Petri net semantics of various formalisms
- ▶ modelling & verification of security protocols
(Security Protocol Language $\xrightarrow{\text{SNAKES}}$ Petri nets $\xrightarrow{\text{Helena}}$ model-checking)
- ▶ simulation of MINs (on-chip interconnexion networks)
- ▶ prototyping compact state space for Petri nets with counters
(use Lash to handle counters)
- ▶ teaching support (put theory into practice)
- ▶ compositional Petri net modelling of biological processes
(regulatory networks)

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system ✓

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system ✓
- ▶ interactive simulation

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system ✓
- ▶ interactive simulation ✓

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system ✓
- ▶ interactive simulation ✓
- ▶ state space/fast simulation

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system ✓
- ▶ interactive simulation ✓
- ▶ state space/fast simulation ❌

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system ✓
- ▶ interactive simulation ✓
- ▶ state space/fast simulation 🛑
 - 💡 export net and use a model-checker/simulator

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system ✓✓
- ▶ interactive simulation ✓
- ▶ state space/fast simulation 🚫
 - 💡 export net and use a model-checker/simulator

Generic solutions (easy but limited):

- ▶ Psyco: kind of JIT (×4)
- ▶ Shed skin: compile *restricted* Python to C++ (×35)
- ▶ PyPy: *full* Python with JIT or compile to C (×65)

7. Performance issues

Depend on the usage:

- ▶ build the Petri net semantics of a system ✓
- ▶ interactive simulation ✓
- ▶ state space/fast simulation 🚫
 - 💡 export net and use a model-checker/simulator

Generic solutions (easy but limited):

- ▶ Psyco: kind of JIT (×4)
- ▶ Shed skin: compile restricted Python to C++ (×35)
- ▶ PyPy: full Python with JIT or compile to C (×65)

Real solutions (*i.e.*, from prototype to implementation):

- ▶ use external fast libraries (as Lash for nets with counters)
- ▶ profile prototype and implement critical parts in C/C++

8. Work in progress, future plans

▶ use it

8. Work in progress, future plans

- ▶ use it
- ▶ improve documentation (90%), unit test (50%) and API doc (50%)
- ▶ better PNML support

8. Work in progress, future plans

- ▶ use it
- ▶ improve documentation (90%), unit test (50%) and API doc (50%)
- ▶ better PNML support
- ▶ add Petri net variants (time PN, stochastic, objects, ...)

8. Work in progress, future plans

- ▶ use it
- ▶ improve documentation (90%), unit test (50%) and API doc (50%)
- ▶ better PNML support
- ▶ add Petri net variants (time PN, stochastic, objects, ...)
- ▶ export a C API (using introspection and Pyrex)
- ▶ bind the C API to other languages (using SWIG)

8. Work in progress, future plans

- ▶ use it
- ▶ improve documentation (90%), unit test (50%) and API doc (50%)
- ▶ better PNML support
- ▶ add Petri net variants (time PN, stochastic, objects, ...)
- ▶ export a C API (using introspection and Pyrex)
- ▶ bind the C API to other languages (using SWIG)
- ▶ ... (your feature here)

8. Work in progress, future plans

- ▶ use it
- ▶ improve documentation (90%), unit test (50%) and API doc (50%)
- ▶ better PNML support
- ▶ add Petri net variants (time PN, stochastic, objects, ...)
- ▶ export a C API (using introspection and Pyrex)
- ▶ bind the C API to other languages (using SWIG)
- ▶ ... (your feature here)





pommereau snakes



pommereau snakes





pommereau snakes



Thank you for your attention

Petri Box Calculus and M-nets family	1
Expected use cases	2
SNAKES' main features	3
Architecture	4
General coloured Petri net model	5
Actual use cases	6
Performance issues	7
Work in progress, future plans	8