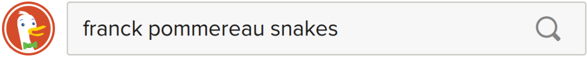
SNAKES: a flexible high-level Petri nets library

Franck Pommereau

IBISC, University of Évry, France

PETRI NETS 2015 — June 25th



franck pommereau snakes

How to implement a fine idea defined in a nice paper?

How to implement a fine idea defined in a nice paper?

- implement Petri nets

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods

How to implement a fine idea defined in a nice paper?

- implement Petri nets
    - places
    - transitions
    - arcs
    - markings
    - implement a bunch of related methods
    - colours
        - data model
        - language
        - expressions evaluation
        - interface with Petri net

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule
- implement auxiliary but required features
  - draw Petri nets and/or state spaces
  - implement save/load Petri nets and/or state spaces

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule
- implement auxiliary but required features
  - draw Petri nets and/or state spaces
  - implement save/load Petri nets and/or state spaces
- implement Petri nets executions
  - transition firing
  - interactive simulation
  - fast simulation
  - state space

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule
- implement auxiliary but required features
  - draw Petri nets and/or state spaces
  - implement save/load Petri nets and/or state spaces
- implement Petri nets executions
  - transition firing
  - interactive simulation
  - fast simulation
  - state space
  - accessibility analysis
  - interface with a model-checker

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule
- implement auxiliary but required features
  - draw Petri nets and/or state spaces
  - implement save/load Petri nets and/or state spaces
- implement Petri nets executions
  - transition firing
  - interactive simulation
  - fast simulation
  - state space
  - accessibility analysis
  - interface with a model-checker
- and so on

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule
- implement auxiliary but required features
  - draw Petri nets and/or state spaces
  - implement save/load Petri nets and/or state spaces
- implement Petri nets executions
  - transition firing
  - interactive simulation
  - fast simulation
  - state space
  - accessibility analysis
  - interface with a model-checker
- and so on
- and so on
- and so on
- and so on

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule
- implement auxiliary but required features
  - draw Petri nets and/or state spaces
  - implement save/load Petri nets and/or state spaces
- implement Petri nets executions
  - transition firing
  - interactive simulation
  - fast simulation
  - state space
  - accessibility analysis
  - interface with a model-checker
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on

How to implement a fine idea defined in a nice paper?

- implement Petri nets
    - places
    - transitions
    - arcs
    - markings
    - implement a bunch of related methods
    - colours
        - data model
        - language
        - expressions evaluation
        - interface with Petri net
- implement various extensions
    - custom arcs
    - custom places
    - custom firing rule
- implement auxiliary but required features
    - draw Petri nets and/or state spaces
    - implement save/load Petri nets and/or state spaces
- implement Petri nets executions
    - transition firing
    - interactive simulation
    - fast simulation
    - state space
    - accessibility analysis
    - interface with a model-checker
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule
- implement auxiliary but required features
  - draw Petri nets and/or state spaces
  - implement save/load Petri nets and/or state spaces
- implement Petri nets executions
  - transition firing
  - interactive simulation
  - fast simulation
  - state space
  - accessibility analysis
  - interface with a model-checker
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- implement your idea

How to implement a fine idea defined in a nice paper?

- implement Petri nets
  - places
  - transitions
  - arcs
  - markings
  - implement a bunch of related methods
  - colours
    - data model
    - language
    - expressions evaluation
    - interface with Petri net
- implement various extensions
  - custom arcs
  - custom places
  - custom firing rule
- implement auxiliary but required features
  - draw Petri nets and/or state spaces
  - implement save/load Petri nets and/or state spaces
- implement Petri nets executions
  - transition firing
  - interactive simulation
  - fast simulation
  - state space
  - accessibility analysis
  - interface with a model-checker
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- and so on
- implement your idea

How to implement a fine idea defined in a nice paper?

- grab SNAKES
- implement your idea

How to implement a fine idea defined in a nice paper?

- grab SNAKES
- make quick customisation

  (time Petri nets $\leq$ 100 LoC / nets-within-nets $\leq$ 30 LoC)
- implement your idea

Outline

Introducing SNAKES

Efficient model-checking

Interfacing with other languages

Typical use cases

Conclusion

Outline

Introducing SNAKES

Efficient model-checking

Interfacing with other languages

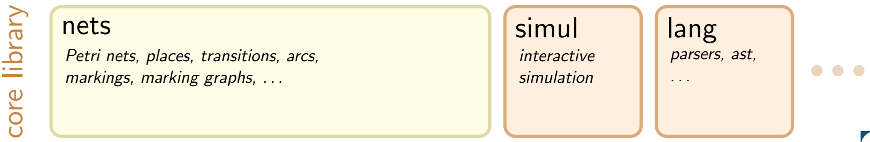
Typical use cases

Conclusion

SNAKES in a nutshell

- SNAKES is a Python library
    - free software (GNU LGPL)
    - 82k lines of code
- define and manipulate Petri nets
    - very generic definition
    - various extensions provided by default
      (read arcs, whole-place arcs, inhibitor arcs, . . .)
    - others are easy to add (timed nets, nets-within-nets, . . .)
- annotations are Python expressions
  tokens are Python objects (even SNAKES' net objects)
- every net can be executed (*i.e.*, transitions can be fired)
- limited PNML support
- extensible with plugins

Architecture



core library

nets

Petri nets, places, transitions, arcs, markings, marking graphs, . . .

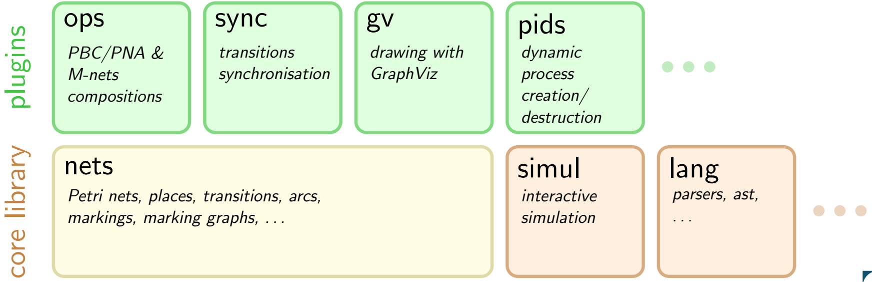
simul

interactive simulation

lang

parsers, ast, . . .

Architecture



plugins

ops

PBC/PNA & M-nets compositions

sync

transitions synchronisation

gv

drawing with GraphViz

pids

dynamic process creation/ destruction

. . .

core library

nets

Petri nets, places, transitions, arcs, markings, marking graphs, . . .

simul

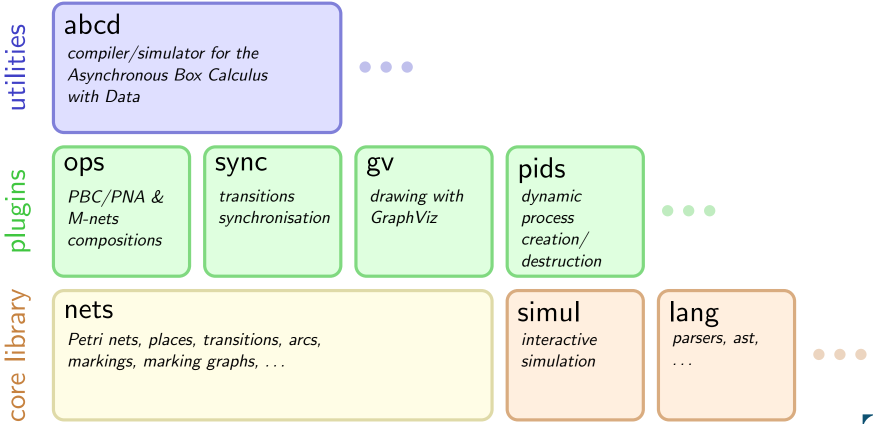
interactive simulation

lang

parsers, ast, . . .

. . .

Architecture



utilities

abcd
compiler/simulator for the Asynchronous Box Calculus with Data

. . .

plugins

ops
PBC/PNA & M-nets compositions

sync
transitions synchronisation

gv
drawing with GraphViz

pids
dynamic process creation/ destruction

. . .

core library

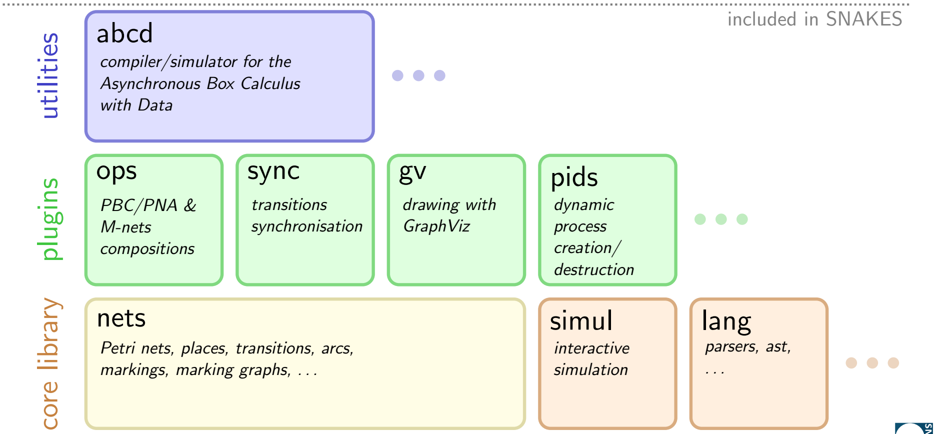
nets
Petri nets, places, transitions, arcs, markings, marking graphs, . . .

simul
interactive simulation

lang
parsers, ast, . . .

. . .

Architecture



included in SNAKES

utilities

abcd

compiler/simulator for the Asynchronous Box Calculus with Data

. . .

plugins

ops

PBC/PNA & M-nets compositions

sync

transitions synchronisation

gv

drawing with GraphViz

pids

dynamic process creation/ destruction

. . .

core library

nets

Petri nets, places, transitions, arcs, markings, marking graphs, . . .

simul

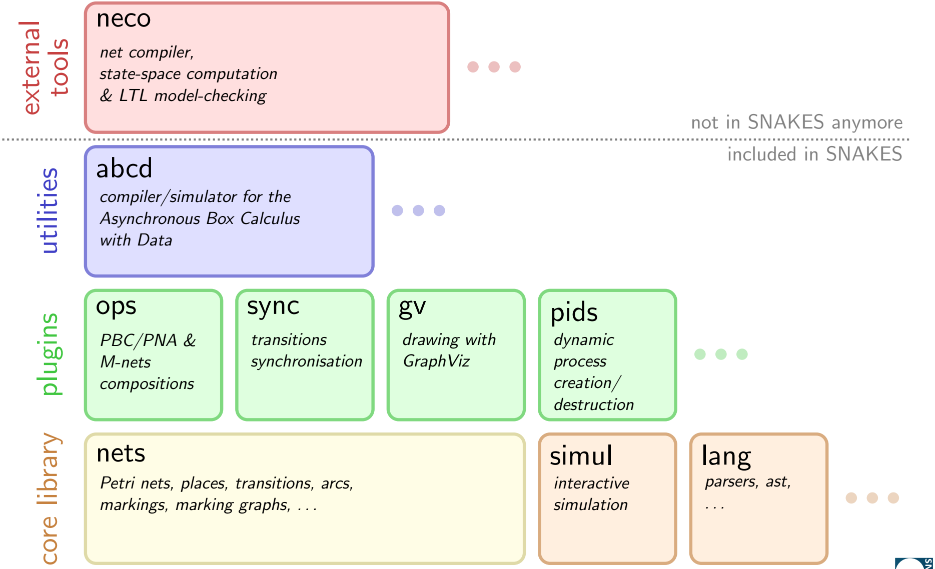
interactive simulation

lang

parsers, ast, . . .

. . .

Architecture



external tools

neco

net compiler,
state-space computation
& LTL model-checking

. . .

not in SNAKES anymore

included in SNAKES

utilities

abcd

compiler/simulator for the
Asynchronous Box Calculus
with Data

. . .

plugins

ops

PBC/PNA & M-nets compositions

sync

transitions synchronisation

gv

drawing with GraphViz

pids

dynamic process creation/ destruction

. . .

core library

nets

Petri nets, places, transitions, arcs, markings, marking graphs, . . .

simul

interactive simulation

lang

parsers, ast, . . .

. . .

Hello world

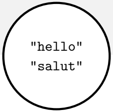
```
from snakes.nets import *
```

Hello world

```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
```

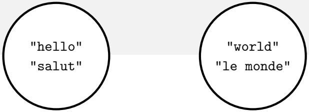
Hello world



```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
```

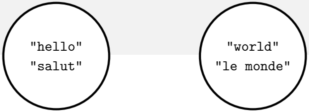
Hello world

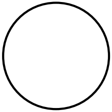


```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
pn.add_place(Place("world", ["world", "le monde"]))
```

Hello world

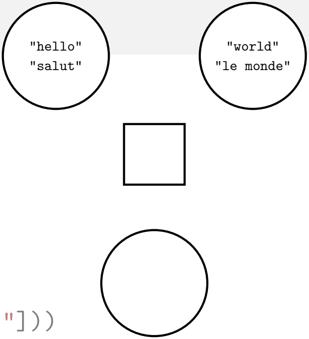




```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
pn.add_place(Place("world", ["world", "le monde"]))
pn.add_place(Place("sentence"))
```

Hello world

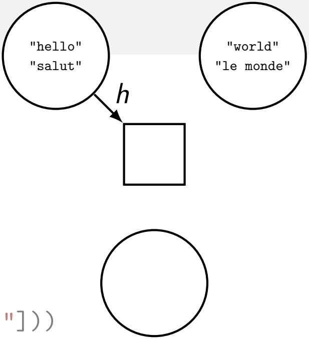


```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
pn.add_place(Place("world", ["world", "le monde"]))
pn.add_place(Place("sentence"))

pn.add_transition(Transition("concat"))
```

Hello world

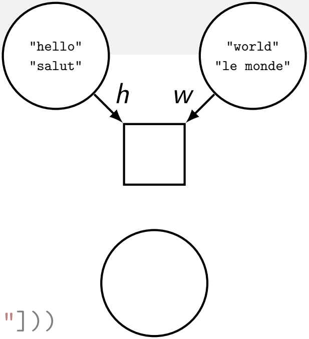


```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
pn.add_place(Place("world", ["world", "le monde"]))
pn.add_place(Place("sentence"))

pn.add_transition(Transition("concat"))
pn.add_input("hello", "concat", Variable("h"))
```

Hello world

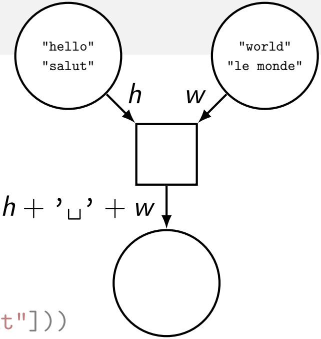


```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
pn.add_place(Place("world", ["world", "le monde"]))
pn.add_place(Place("sentence"))

pn.add_transition(Transition("concat"))
pn.add_input("hello", "concat", Variable("h"))
pn.add_input("world", "concat", Variable("w"))
```

Hello world

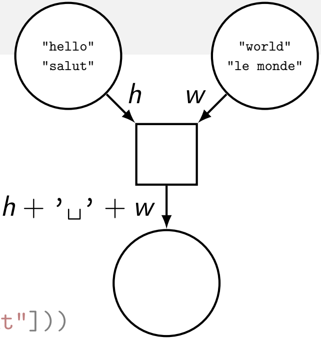


```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
pn.add_place(Place("world", ["world", "le monde"]))
pn.add_place(Place("sentence"))

pn.add_transition(Transition("concat"))
pn.add_input("hello", "concat", Variable("h"))
pn.add_input("world", "concat", Variable("w"))
pn.add_output("sentence", "concat", Expression("h + ’ ’ + w"))
```

Hello world



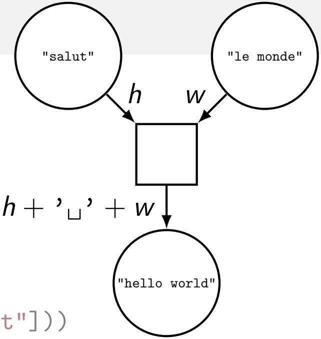
```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
pn.add_place(Place("world", ["world", "le monde"]))
pn.add_place(Place("sentence"))

pn.add_transition(Transition("concat"))
pn.add_input("hello", "concat", Variable("h"))
pn.add_input("world", "concat", Variable("w"))
pn.add_output("sentence", "concat", Expression("h + ' ' + w"))

modes = pn.transition("concat").modes()    # returns 4 modes
```

Hello world



```
from snakes.nets import *

pn = PetriNet("hello world in SNAKES")
pn.add_place(Place("hello", ["hello", "salut"]))
pn.add_place(Place("world", ["world", "le monde"]))
pn.add_place(Place("sentence"))

pn.add_transition(Transition("concat"))
pn.add_input("hello", "concat", Variable("h"))
pn.add_input("world", "concat", Variable("w"))
pn.add_output("sentence", "concat", Expression("h + ' ' + w"))

modes = pn.transition("concat").modes()    # returns 4 modes
pn.transition("concat").fire(modes[2])
```

Outline

Introducing SNAKES

Efficient model-checking

Interfacing with other languages

Typical use cases

Conclusion

But… Isn't Python slow as hell?

Python is fast

- for building and manipulating Petri nets (even large ones)
- for firing transitions interactively
- for calling CPU-intensive routines from an external library

But. . . Isn’t Python slow as hell?

Python is fast

- for building and manipulating Petri nets (even large ones)
- for firing transitions interactively
- for calling CPU-intensive routines from an external library

Python is slow

- for computing large state-spaces
- for running complex algorithms
- SNAKES is even slower (not optimised for speed)

But. . . Isn't Python slow as hell?

Python is fast

- for building and manipulating Petri nets (even large ones)
- for firing transitions interactively
- for calling CPU-intensive routines from an external library

Python is slow

- for computing large state-spaces
- for running complex algorithms
- SNAKES is even slower (not optimised for speed)

So, can we use SNAKES for model-checking?

Fast analysis with Neco

- Łukasz Fronc's companion tool
  `https://code.google.com/p/neco-net-compiler/`
- Neco compiles SNAKES' Petri nets into fast native code
  - per-net optimised marking structure
  - per-transition optimised firing
  - no magic ⇒ cannot optimise arbitrary Python code
- reads PNML, ABCD, or net objects
- process-symmetries reductions (plugin `pids`)
- state space exploration and LTL model-checking (using SPOT)
- awarded at the *model-checking contest* 2013

Outline

Introducing SNAKES

Efficient model-checking

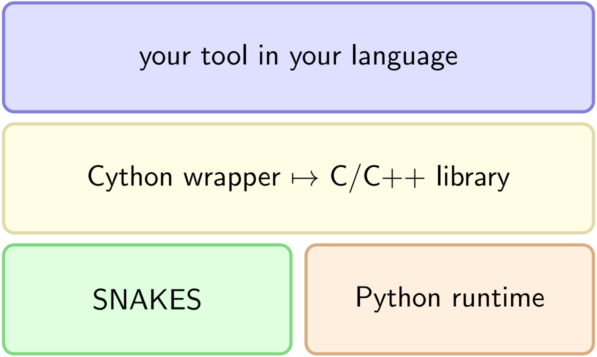
Interfacing with other languages

Typical use cases

Conclusion

SNAKES out of Python

Cython = Python + type annotations ⇒ generates optimised C/C++



your tool in your language

Cython wrapper ↦ C/C++ library

SNAKES

Python runtime

Outline

Introducing SNAKES

Efficient model-checking

Interfacing with other languages

Typical use cases

Conclusion

Some SNAKES users

- Sam Sanjabi's post-doc (2010) and Samira Chaou's PhD (2013)
  - ABCD modelling of peer-to-peer storage systems
  - security analysis (model-checking, simulation & stats)
- Michaël Guedj's PhD (2012)
  - ABCD modelling of security protocols (Alice-Bob kind)
  - BSP-parallel CTL$\ast$ model-checking (algorithm and scalability study)
- Viet Van Pham's PhD (2014)
  - semantics of π-graphs, analysis of open reconfigurable systems
  - reachability testing, simulation and LTL model-checking (using Neco)
- Mourad Amziani's PhD (2015)
  - modelling of elasticity mechanisms in cloud systems (nets-within-nets)
  - safety analysis (reachability testing)
- support to Petri net research
  - several other PhD around the world (few information available)
  - prototyping, experiments, methods validation, ...
- numerous master students' projects, teaching, tutorials, etc.

Some SNAKES users

- Sam Sanjabi's post-doc (2010) and Samira Chaou's PhD (2013)
  - ABCD modelling of peer-to-peer storage systems
  - security analysis (model-checking, simulation & stats)
- Michaël Guedj's PhD (2012)
  - ABCD modelling of security protocols (Alice-Bob kind)
  - BSP-parallel CTL∗ model-checking (algorithm and scalability study)
- Viet Van Pham's PhD (2014)
  - semantics of π-graphs, analysis of open reconfigurable systems
  - reachability testing, simulation and LTL model-checking (using Neco)
- Mourad Amziani's PhD (2015)
  - modelling of elasticity mechanisms in cloud systems (nets-within-nets)
  - safety analysis (reachability testing)
- support to Petri net research
  - several other PhD around the world (few information available)
  - prototyping, experiments, methods validation, . . .
- numerous master students' projects, teaching, tutorials, etc.

Some SNAKES users

- Sam Sanjabi's post-doc (2010) and Samira Chaou's PhD (2013)
  - ABCD modelling of peer-to-peer storage systems
  - security analysis (model-checking, simulation & stats)
- Michaël Guedj's PhD (2012)
  - ABCD modelling of security protocols (Alice-Bob kind)
  - BSP-parallel CTL∗ model-checking (algorithm and scalability study)
- Viet Van Pham's PhD (2014)
  - semantics of π-graphs, analysis of open reconfigurable systems
  - reachability testing, simulation and LTL model-checking (using Neco)
- Mourad Amziani's PhD (2015)
  - modelling of elasticity mechanisms in cloud systems (nets-within-nets)
  - safety analysis (reachability testing)
- support to Petri net research
  - several other PhD around the world (few information available)
  - prototyping, experiments, methods validation, . . .
- numerous master students' projects, teaching, tutorials, etc.

Some SNAKES users

- Sam Sanjabi's post-doc (2010) and Samira Chaou's PhD (2013)
  - ABCD modelling of peer-to-peer storage systems
  - security analysis (model-checking, simulation & stats)
- Michaël Guedj's PhD (2012)
  - ABCD modelling of security protocols (Alice-Bob kind)
  - BSP-parallel CTL∗ model-checking (algorithm and scalability study)
- Viet Van Pham's PhD (2014)
  - semantics of π-graphs, analysis of open reconfigurable systems
  - reachability testing, simulation and LTL model-checking (using Neco)
- Mourad Amziani's PhD (2015)
  - modelling of elasticity mechanisms in cloud systems (nets-within-nets)
  - safety analysis (reachability testing)
- support to Petri net research
  - several other PhD around the world (few information available)
  - prototyping, experiments, methods validation, . . .
- numerous master students' projects, teaching, tutorials, etc.

Some SNAKES users

- Sam Sanjabi's post-doc (2010) and Samira Chaou's PhD (2013)
  - ABCD modelling of peer-to-peer storage systems
  - security analysis (model-checking, simulation & stats)
- Michaël Guedj's PhD (2012)
  - ABCD modelling of security protocols (Alice-Bob kind)
  - BSP-parallel CTL$*$ model-checking (algorithm and scalability study)
- Viet Van Pham's PhD (2014)
  - semantics of π-graphs, analysis of open reconfigurable systems
  - reachability testing, simulation and LTL model-checking (using Neco)
- Mourad Amziani's PhD (2015)
  - modelling of elasticity mechanisms in cloud systems (nets-within-nets)
  - safety analysis (reachability testing)
- support to Petri net research
  - several other PhD around the world (few information available)
  - prototyping, experiments, methods validation, . . .
- numerous master students' projects, teaching, tutorials, etc.

Some SNAKES users

- Sam Sanjabi's post-doc (2010) and Samira Chaou's PhD (2013)
    - ABCD modelling of peer-to-peer storage systems
    - security analysis (model-checking, simulation & stats)
- Michaël Guedj's PhD (2012)
    - ABCD modelling of security protocols (Alice-Bob kind)
    - BSP-parallel CTL$*$ model-checking (algorithm and scalability study)
- Viet Van Pham's PhD (2014)
    - semantics of π-graphs, analysis of open reconfigurable systems
    - reachability testing, simulation and LTL model-checking (using Neco)
- Mourad Amziani's PhD (2015)
    - modelling of elasticity mechanisms in cloud systems (nets-within-nets)
    - safety analysis (reachability testing)
- support to Petri net research
    - several other PhD around the world (few information available)
    - prototyping, experiments, methods validation, . . .
- numerous master students' projects, teaching, tutorials, etc.

Outline

Introducing SNAKES

Efficient model-checking

Interfacing with other languages

Typical use cases

Conclusion

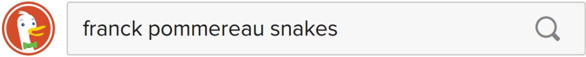
Ongoing and future work

- necessary conditions to reach version 1.0 (current: 0.9.17)
  - replace PNML support with more generic output (GrML, JSON)
  - recover through GrML ↦ PNML (using third-party tool CosyVerif)
  - integrate Neco through a plugin
  - fill a few holes in the documentation
  - minor code cleanup and simplification

Ongoing and future work

- necessary conditions to reach version 1.0 (current: 0.9.17)
  - replace PNML support with more generic output (GrML, JSON)
  - recover through GrML ↦ PNML (using third-party tool CosyVerif)
  - integrate Neco through a plugin
  - fill a few holes in the documentation
  - minor code cleanup and simplification
- other needs and ideas
  - better interactive/fast simulation, coupled with statistical analysis
  - genericity w.r.t. annotation language (use compilation approach)
  - major code cleanup and simplification
  - integrate with other tools (GUIs, analysers, etc.)
  - more inputs/outputs (using third-party tools)
  - automate API generation to other languages
  - extend ABCD with a Petri net syntax
  - add processes to ABCD
  - win the lottery and hire engineers

Thank you. Questions?



franck pommereau snakes