LTL Model Checking with Neco

Łukasz Fronc[1] Alexandre Duret-Lutz[2]

IBISC, Université d'Évry/Paris-Saclay
fronc@ibisc.univ-evry.fr

LRDE, EPITA, Kremlin-Bicêtre, France
adl@lrde.epita.fr

ATVA'13, 2013-10-16

http://code.google.com/p/neco-net-compiler/

What is Neco?

A Petri net compiler — transforms Petri nets into libraries

- works with high-level Petri nets — colored PN annotated by Python
  - based on SNAKES, a Python library for Petri nets

F. Pommereau. Quickly prototyping Petri nets tools with SNAKES. *Petri net newsletter*, October 2008

What is Neco?

A Petri net compiler — transforms Petri nets into libraries

- works with high-level Petri nets — colored PN annotated by Python
  - based on SNAKES, a Python library for Petri nets
- produces optimized code...
- ...for explicit model-checking
- expressivity compromise

With a set of command-line tools:

- `neco-compile` — PN compiler
- `neco-explore` — minimal exploration tool

Ł. Fronc and F. Pommereau. Building Petri nets tools around Neco compiler. *PNSE'13*

What is Neco?

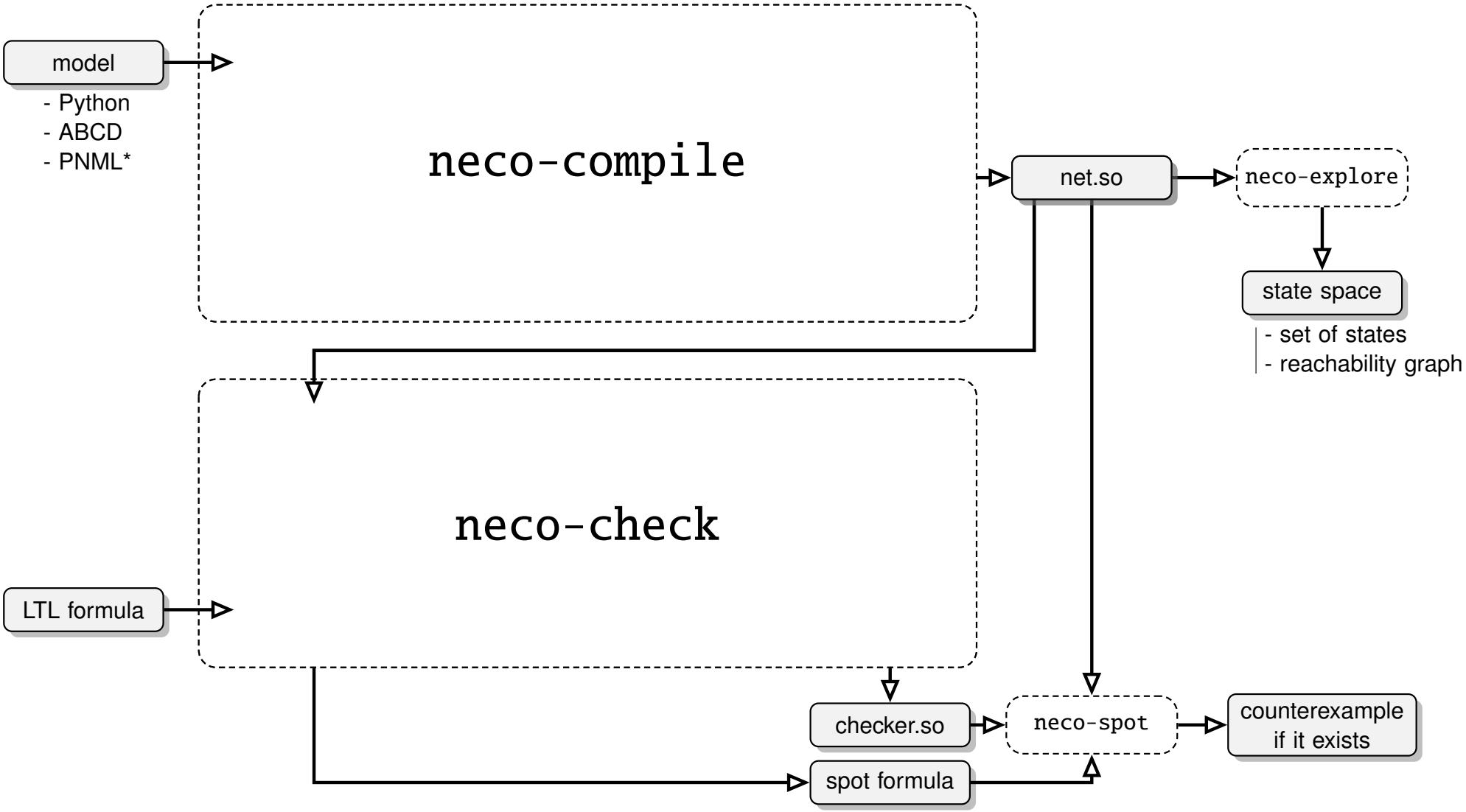
A Petri net compiler — transforms Petri nets into libraries

- works with high-level Petri nets — colored PN annotated by Python
  - based on SNAKES, a Python library for Petri nets
- produces optimized code...
- ...for explicit model-checking
- expressivity compromise

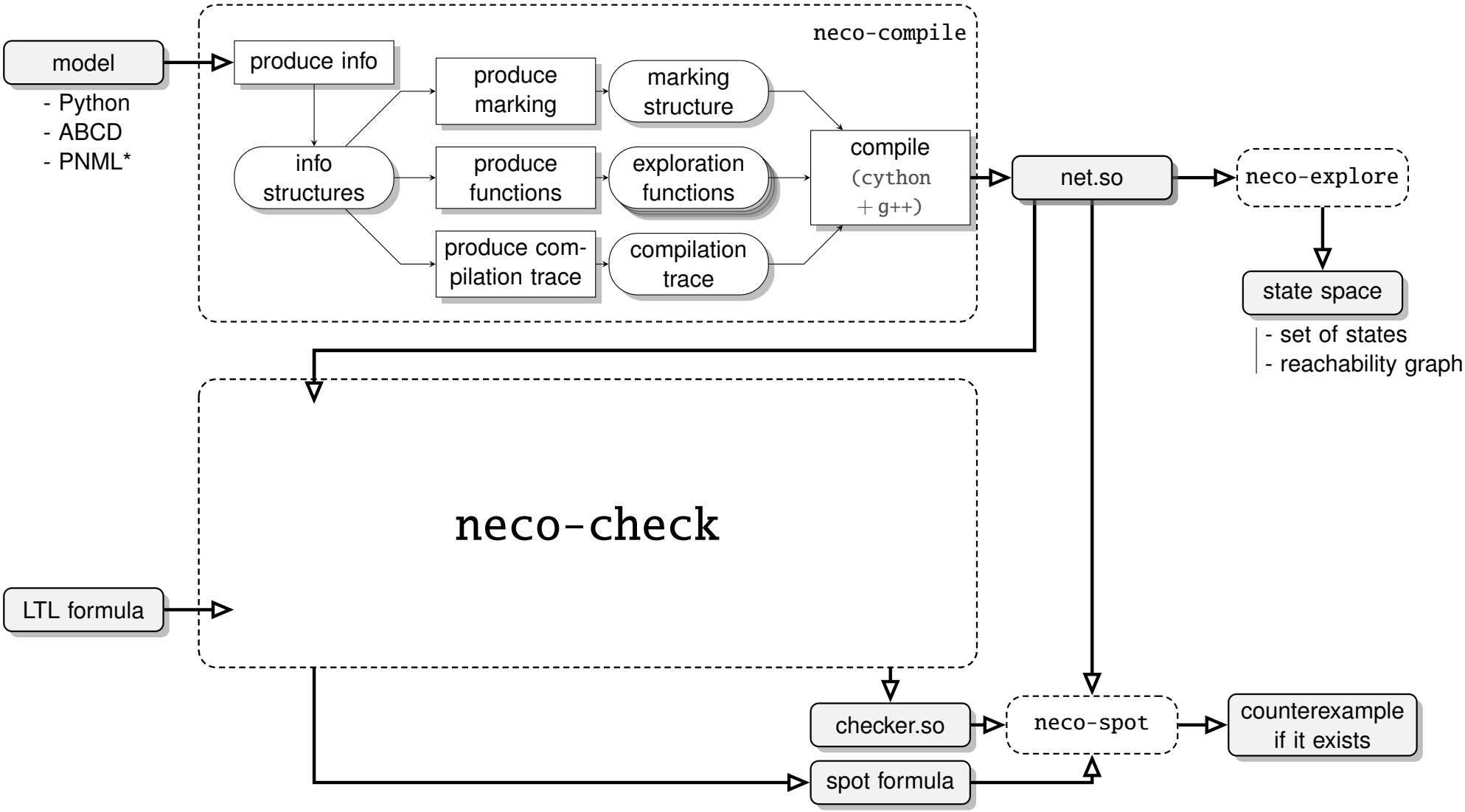
With a set of command-line tools:

- `neco-compile` — PN compiler
- `neco-explore` — minimal exploration tool
- `neco-check` new! — LTL-adapter compiler
- `neco-spot` new! — LTL model-checker

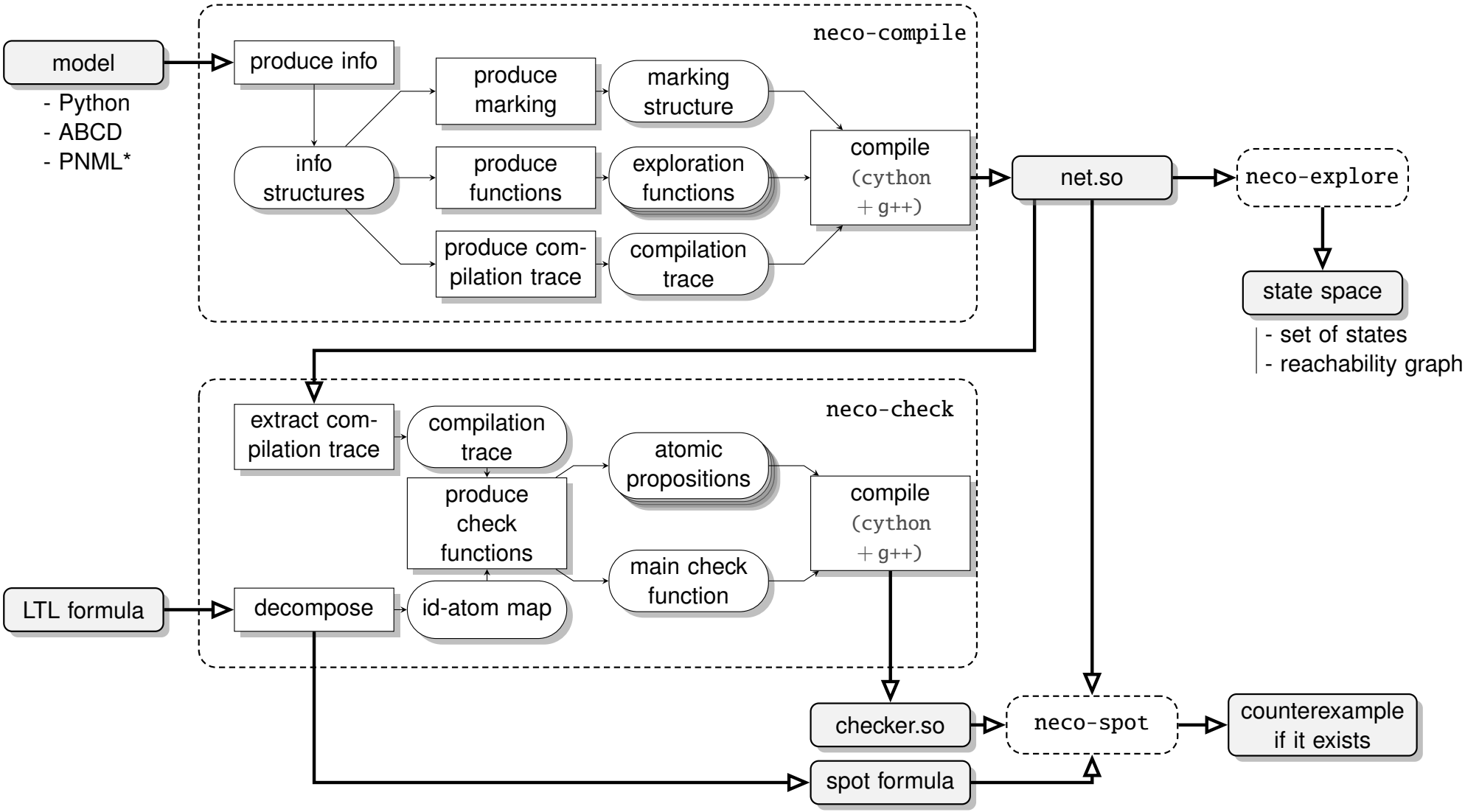
Neco's Workflow



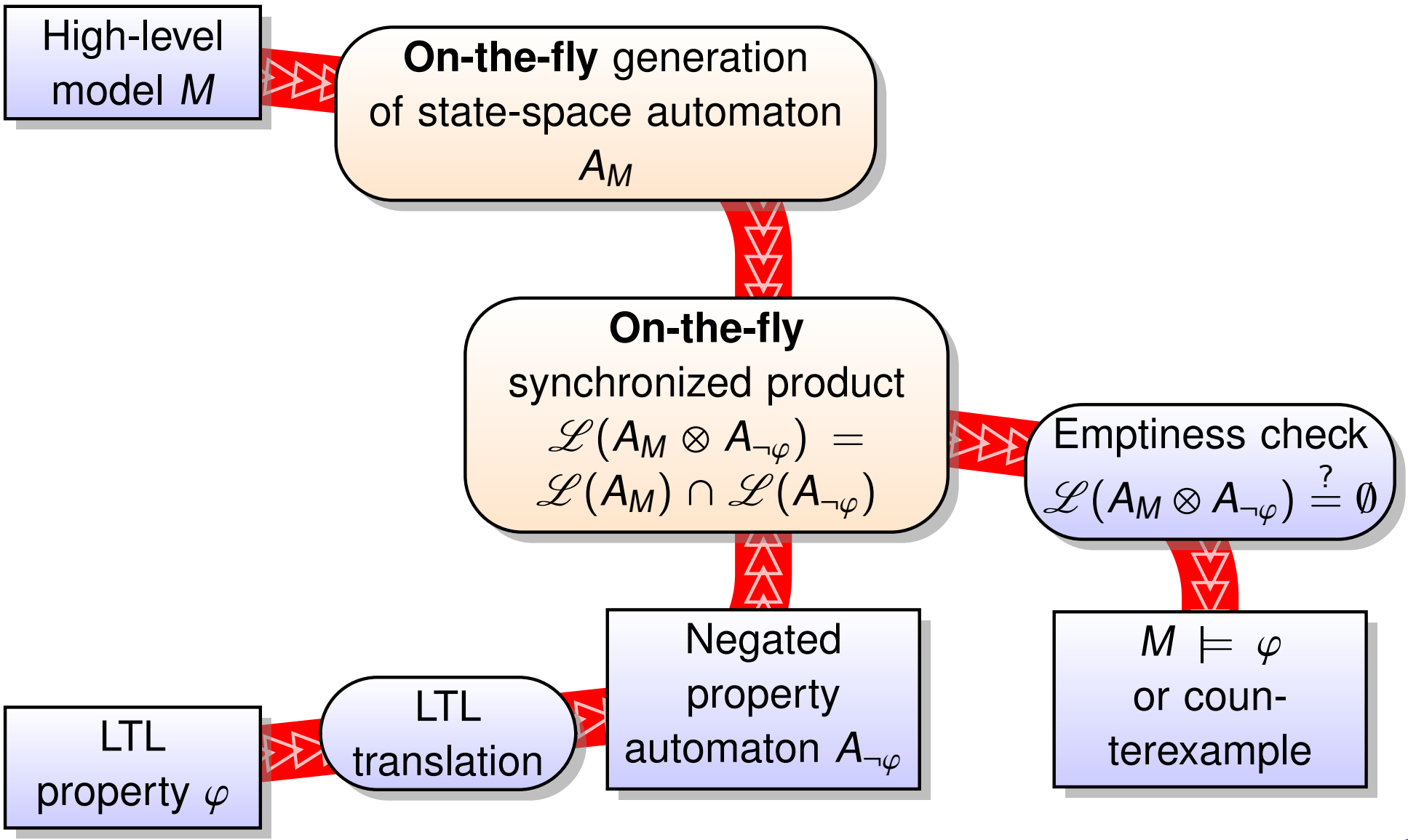
Neco's Workflow



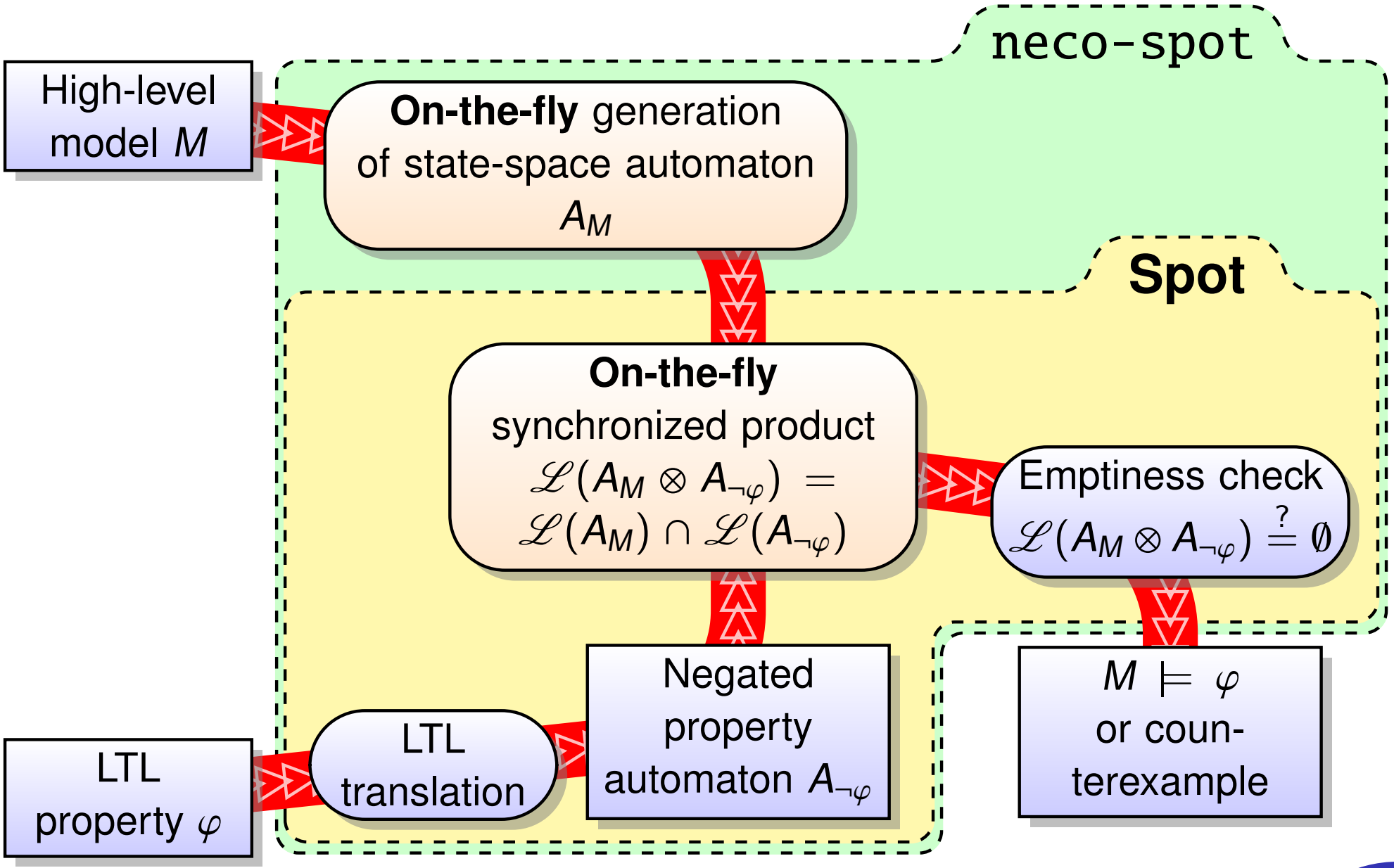
Neco's Workflow



Spot as a library for explicit model-checking



Spot as a library for explicit model-checking

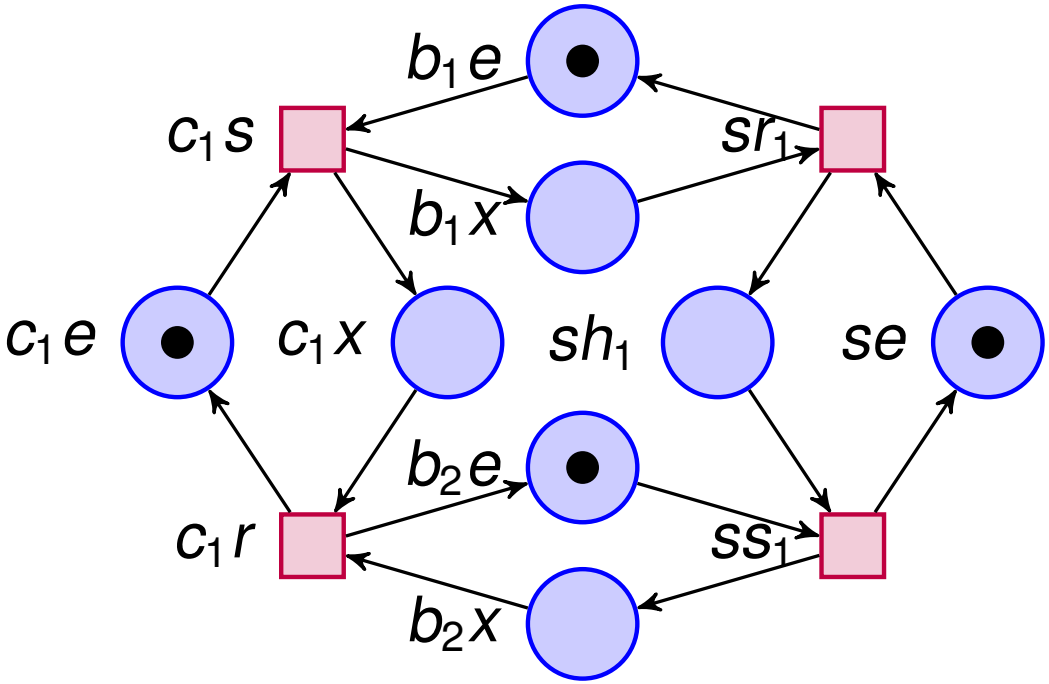


neco-spot

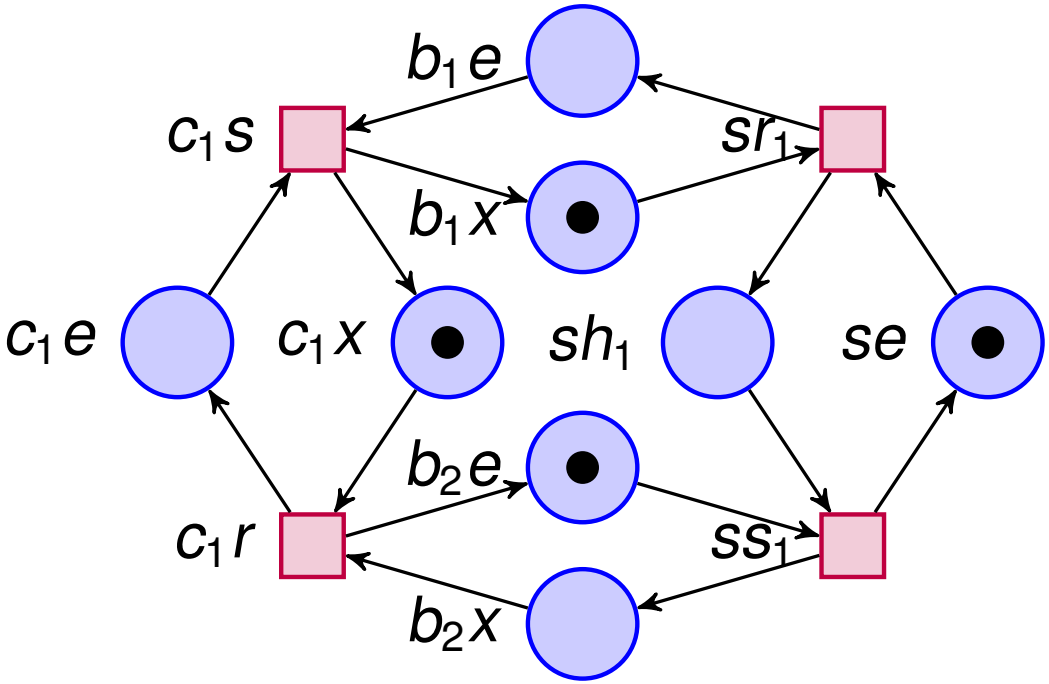
- A wrapper of `net.so` and `checker.so` that presents the reachability graph as a subclass of `spot::kripke`:
  - `get_init_state()` — initial state
  - `succ_iter(s)` — iterator over the successors of state s
  - `state_condition(s)` — value of atomic propositions for s

  This interface supports on-the-fly exploration.
- Translate the formula into a generalized Büchi automaton:
  - `spot::translator::run(f)` — includes many optimizations
- Synchronize reachability graph and formula automaton:
  - `spot::tgba_product(model, prop)` — on-the-fly
- Check the product for emptiness:
  - `spot::emptiness_check::check()`
- Optionally compute a counterexample:
  - `spot::emptiness_check_result::accepting_run()`

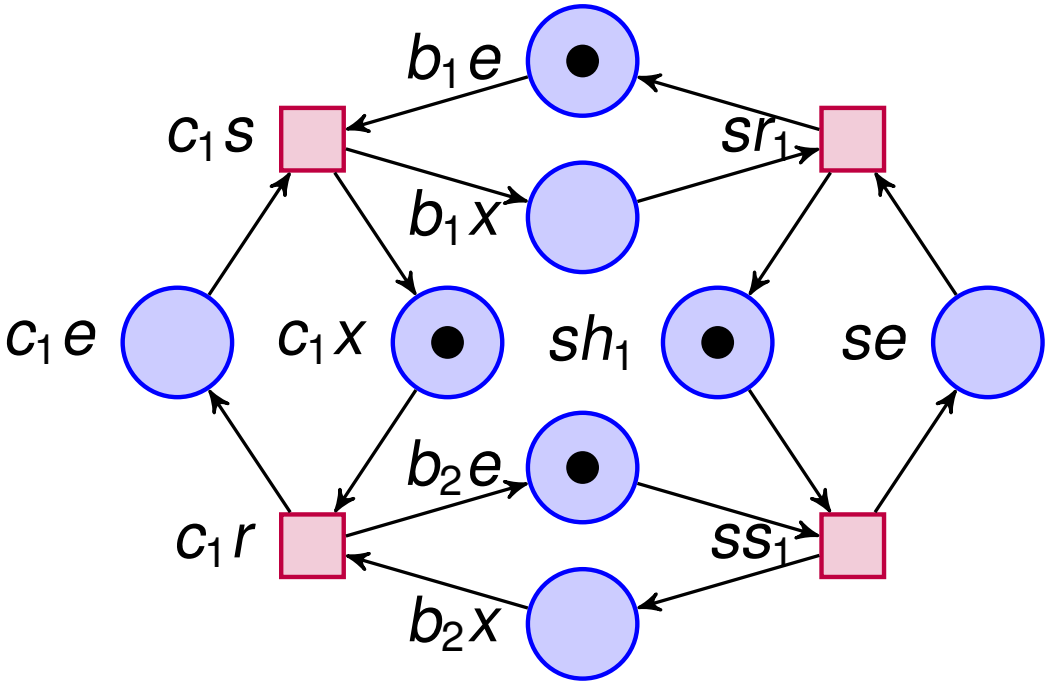
Demo



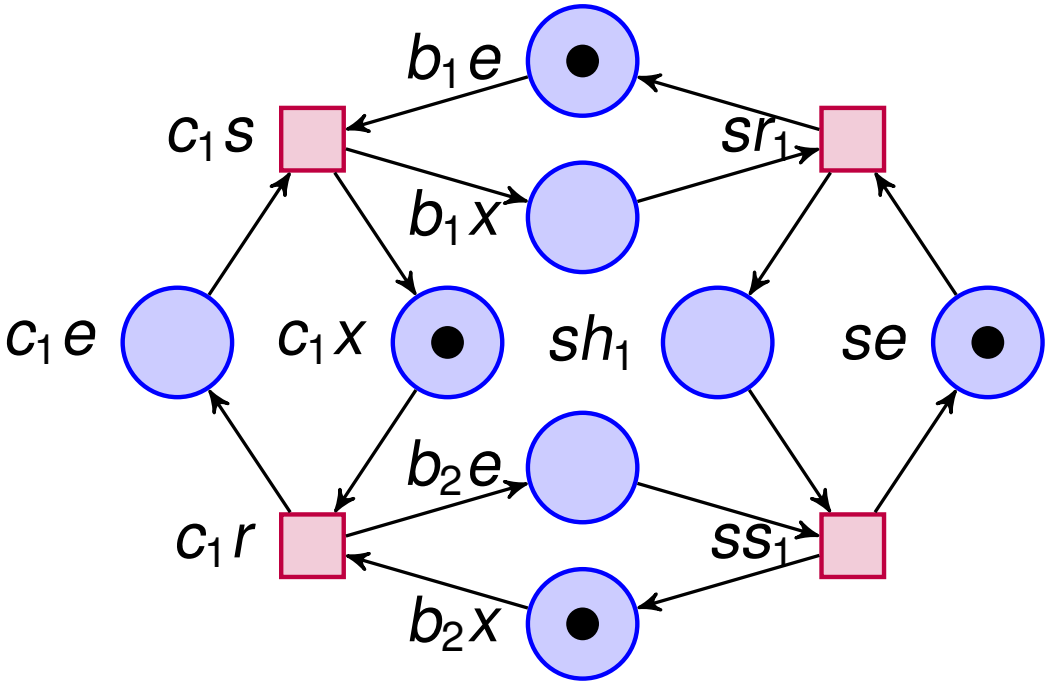
Demo



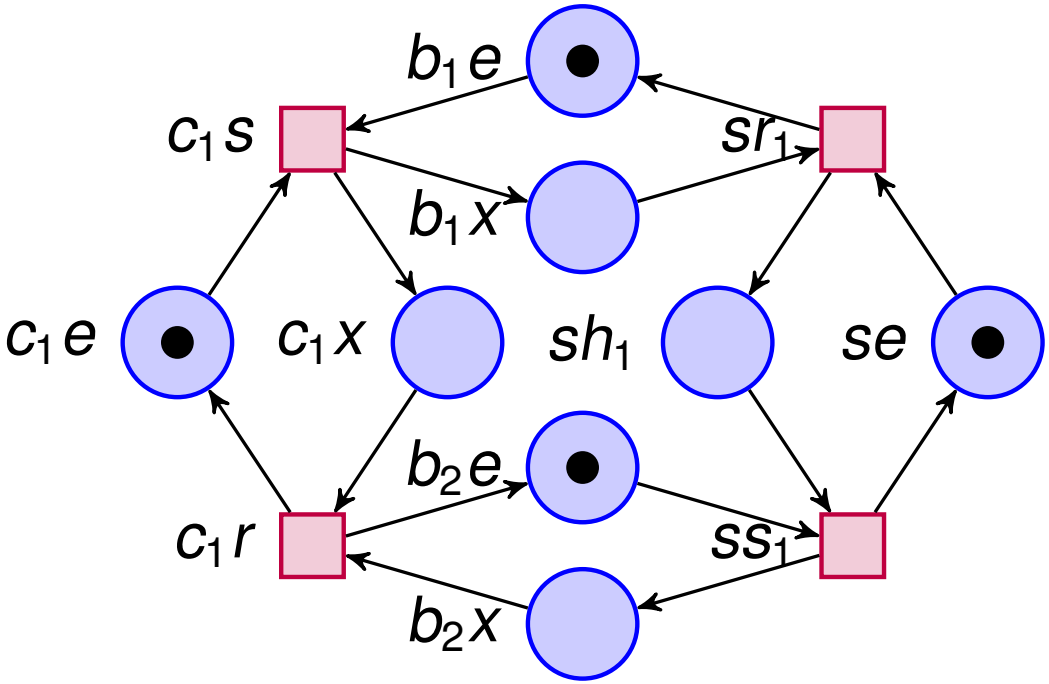
Demo



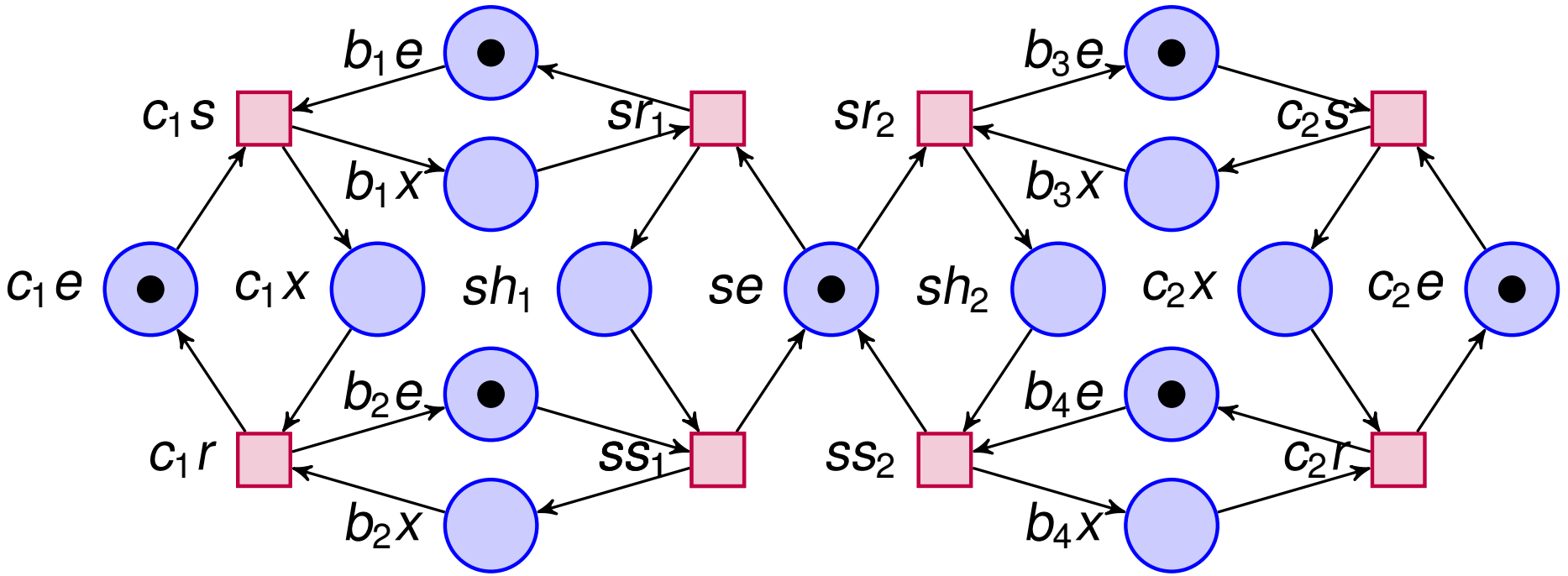
Demo



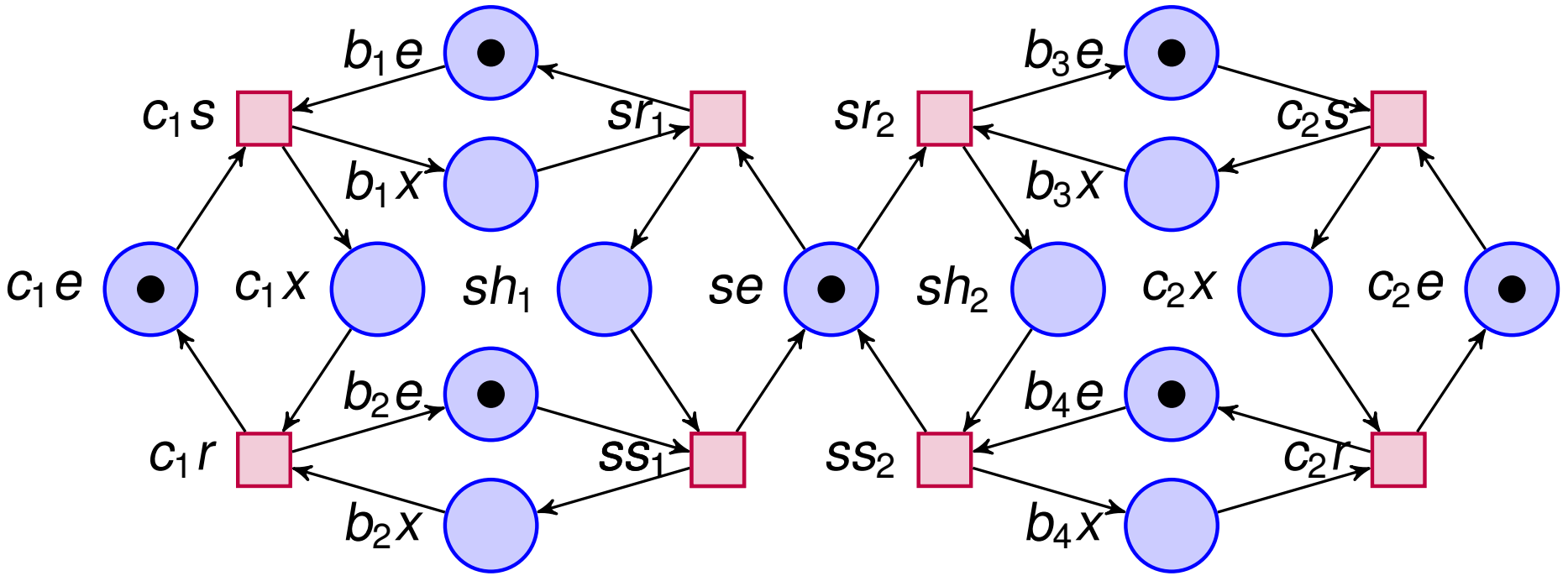
Demo



Demo


$b_1 e$
$c_1 s$
sr_1
$b_1 x$
$c_1 e$
$c_1 x$
sh_1
se
$b_2 e$
$c_1 r$
ss_1
$b_2 x$
$b_3 e$
sr_2
$c_2 s$
$b_3 x$
sh_2
$c_2 x$
$c_2 e$
$b_4 e$
ss_2
$c_2 r$
$b_4 x$

Demo



Running a model-checking task

```
$ neco-compile --module cs.py -lcython
$ neco-check --formula 'G ((marking(c1e)=[dot] and
    X marking(c1x)=[dot]) -> X F(marking(c1e)=[dot]))'
$ neco-spot neco_formula
```

Availability

Neco and Spot are free software.
Documentation and installation instructions can be found at

- `http://code.google.com/p/neco-net-compiler/` and
- `http://spot.lip6.fr/`