Spot 2.0
A C++ library for model checking and ω-automata manipulation

Alexandre Duret-Lutz Alexandre Lewkowicz Amaury Fauchille
Étienne Renault Laurent Xu Thibaud Michaud

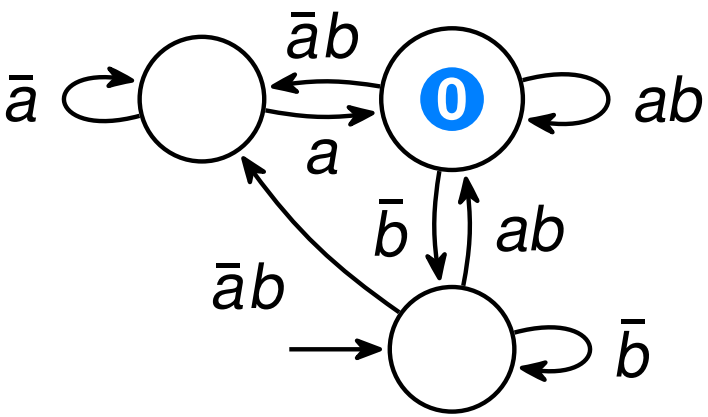
MeFoSyLoMa, 8 avril 2016, LRDE

A Generalized View of ω-Automata

ω-Automata in Spot have:

- a single initial state,
- transitions labeled by Boolean formulas over atomic prop.,
- acceptance marks (0, 1, 2, ...) that can label either states or transitions,
- an acceptance condition that tells which marks have to be seen infinitely often or finitely often for a run to be accepted.

Minimal Büchi automaton for $\mathbf{G}\,\mathbf{F}\,a \wedge \mathbf{G}\,\mathbf{F}\,b$:



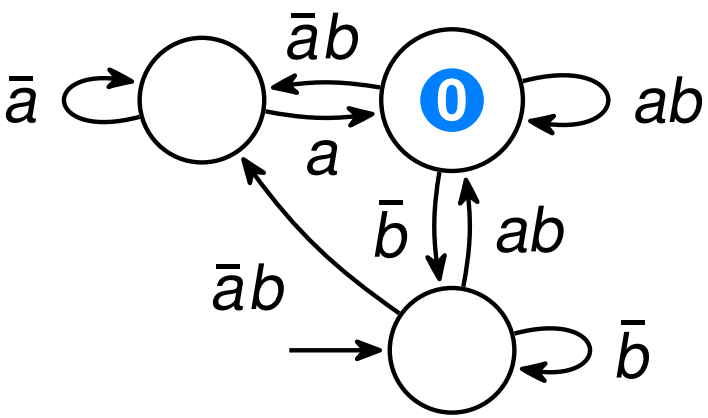
Inf(0)

A Generalized View of ω-Automata

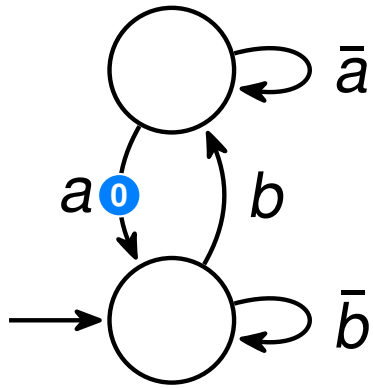
ω-Automata in Spot have:

- a single initial state,
- transitions labeled by Boolean formulas over atomic prop.,
- acceptance marks (0, 1, 2, ...) that can label either states or transitions,
- an acceptance condition that tells which marks have to be seen infinitely often or finitely often for a run to be accepted.

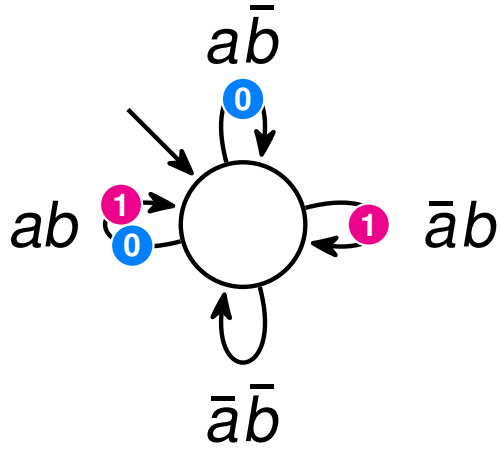
Minimal automata for $\mathbf{G}\,\mathbf{F}\,a \wedge \mathbf{G}\,\mathbf{F}\,b$:



Inf(0)



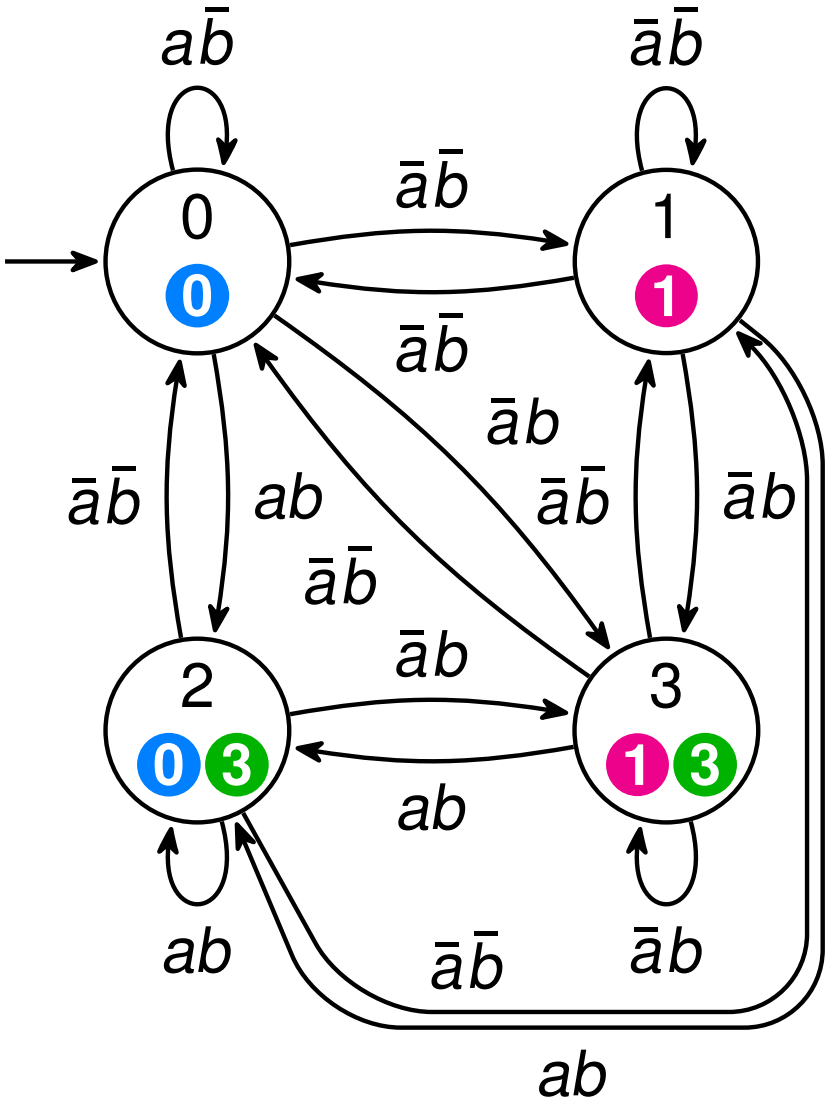
Inf(0)



Inf(0) $\wedge$ Inf(1)

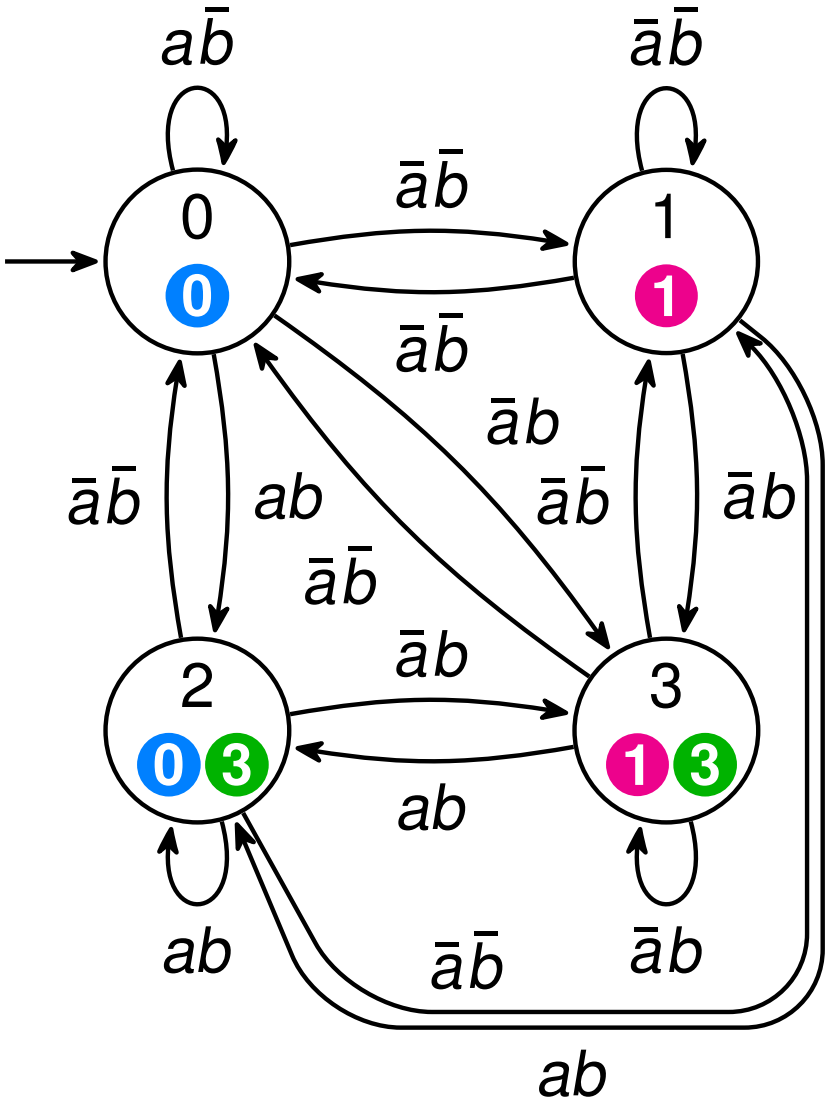
A Rabin Automaton for $\mathbf{G}\,\mathbf{F}\,a \rightarrow \mathbf{G}\,\mathbf{F}\,b$

$$\big(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)\big) \vee \big(\mathrm{Fin}(2) \wedge \mathrm{Inf}(3)\big)$$



A Rabin Automaton for $\mathbf{G}\,\mathbf{F}\,a \rightarrow \mathbf{G}\,\mathbf{F}\,b$

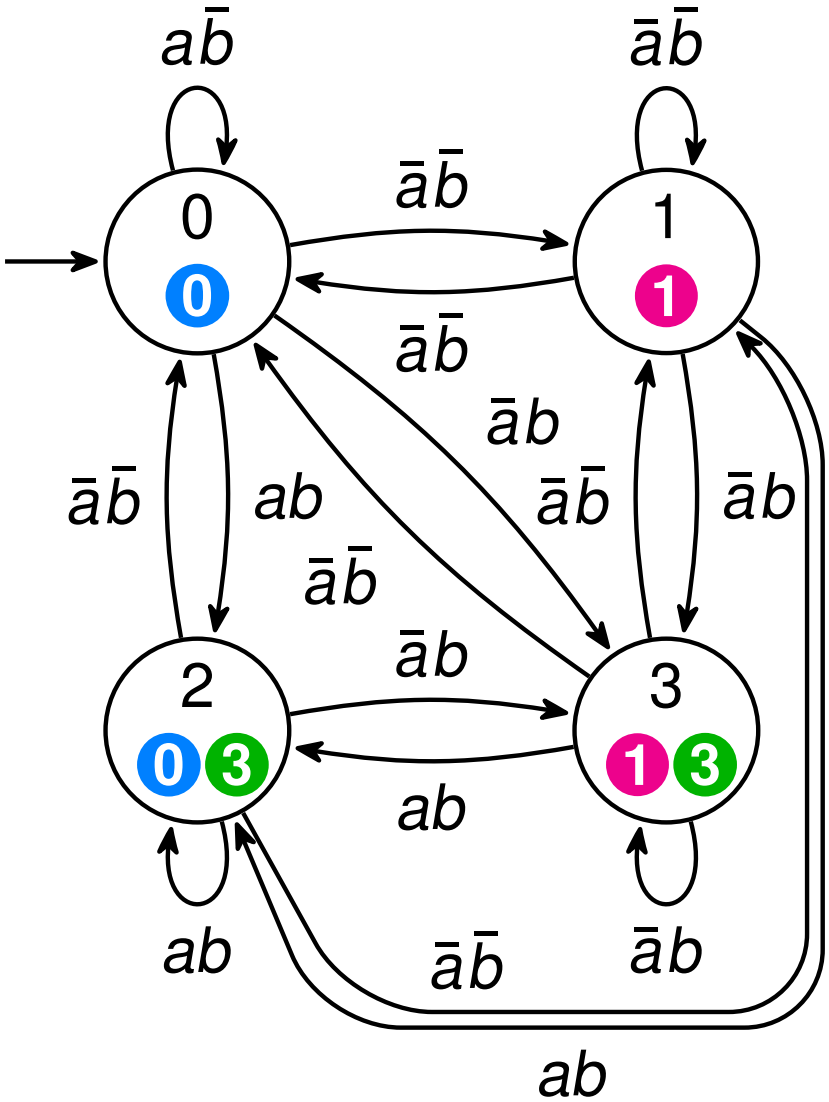
$$(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)) \vee (\mathrm{Fin}(2) \wedge \mathrm{Inf}(3))$$




```
HOA: v1
States: 4
Start: 0
AP: 2 "a" "b"
acc-name: Rabin 2
Acceptance: 4
  Fin( 0 )&Inf( 1 )|Fin( 2 )&Inf( 3 )
--BODY--
State: 0 { 0 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 1 { 1 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 2 { 0  3 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 3 { 1  3 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
--END--
```

A Rabin Automaton for $\mathbf{G}\,\mathbf{F}\,a \rightarrow \mathbf{G}\,\mathbf{F}\,b$

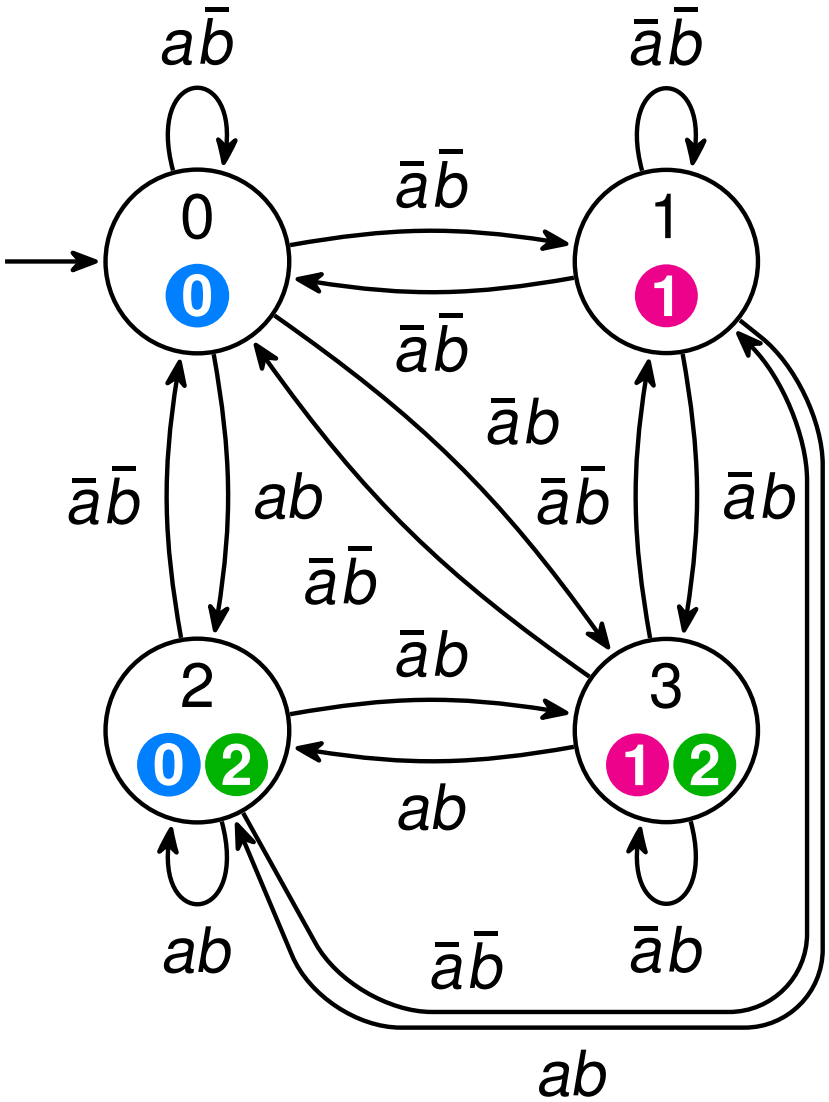
$(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)) \vee (\mathrm{Fin}(2) \wedge \mathrm{Inf}(3))$




```
HOA: v1
States: 4
Start: 0
AP: 2 "a" "b"
acc-name: Rabin 2
Acceptance: 4
  Fin( 0 )&Inf( 1 )|Fin( 2 )&Inf( 3 )
--BODY--
State: 0 { 0 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 1 { 1 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 2 { 0  3 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 3 { 1  3 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
--END--
```

An ω-Automaton for $\mathbf{G}\,\mathbf{F}\,a \rightarrow \mathbf{G}\,\mathbf{F}\,b$

$(\mathrm{Fin}(0) \wedge \mathrm{Inf}(1)) \vee \mathrm{Inf}(2)$

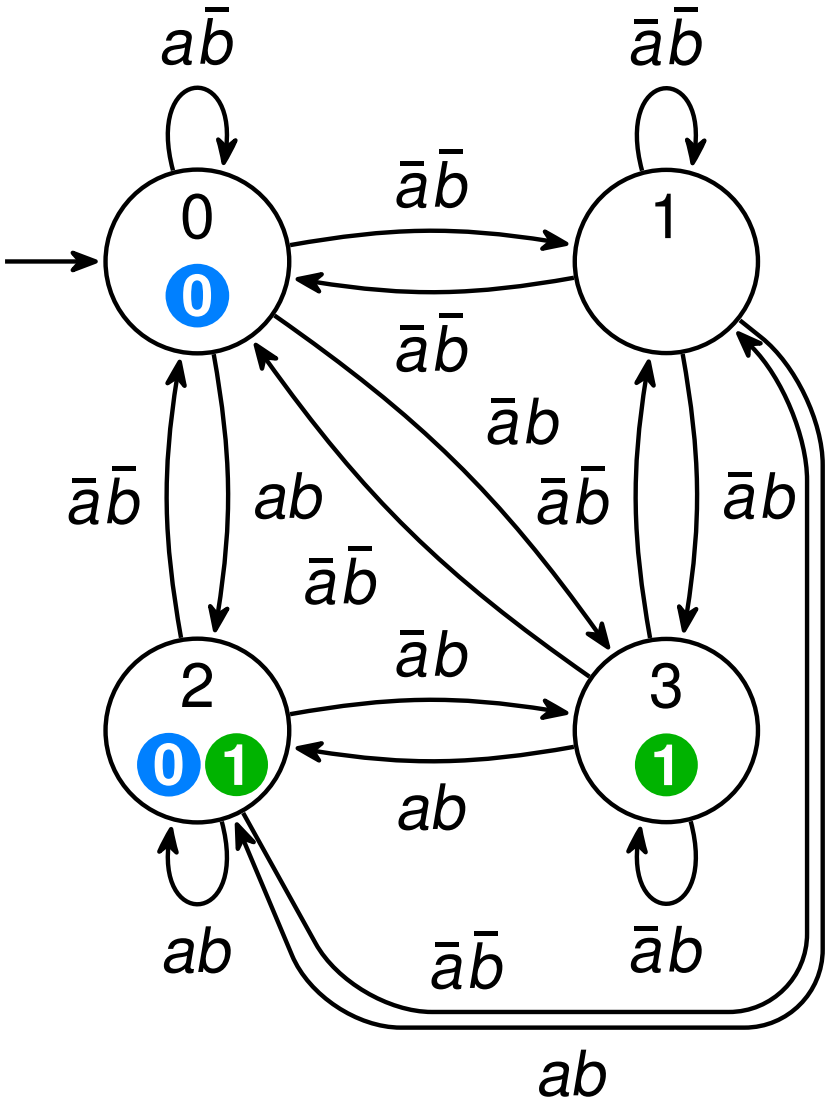



```
HOA: v1
States: 4
Start: 0
AP: 2 "a" "b"

Acceptance: 3
  Fin( 0 )&Inf( 1 )|Inf( 2 )
--BODY--
State: 0 { 0 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 1 { 1 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 2 { 0  2 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 3 { 1  2 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
--END--
```

A Streett Automaton for $\mathbf{G}\,\mathbf{F}\,a \rightarrow \mathbf{G}\,\mathbf{F}\,b$

Fin(0)∨Inf(1)




```
HOA: v1
States: 4
Start: 0
AP: 2 "a" "b"
acc-name: Streett 1
Acceptance: 2
  Fin( 0 )|Inf( 1 )
--BODY--
State: 0 { 0 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 1
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 2 { 0  1 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 3 { 1 }
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
--END--
```

Support for the Hanoi Omega-Automata Format

`http://adl.github.io/hoaf/support.html`

`ltl2dstar 0.5.3` output DRA or DSA, can also input BA

`ltl3ba 1.1.2` output BA, TGBA, or VWAA

`ltl3dra 0.2.2` output DRA, TGDRA or MMAA

`Rabinizer 3.1` output DRA, TDRA, GDRA, or TGDRA

`PRISM 4.3`

input deterministic automata for probabilistic model checking; (generalized) Rabin for MDP; any acceptance for CTMC/DTMC

`Spot` since `1.99.2`

can input/output anything that is not alternating; can convert from other formats; has several transformations

`jhoafparser` and `cpphoafparser`

two parsers with pretty printers, and convenient transformations

T. Babiak, F. Blahoudek, A. Duret-Lutz, J. Klein, J. Křetínský, D. Müller, D. Parker, and J. Strejček. The Hanoi Omega-Automata format. *CAV'15*

Spot's Services for Temporal Logic Formulas

For LTL formulas (or the linear fragment of PSL):

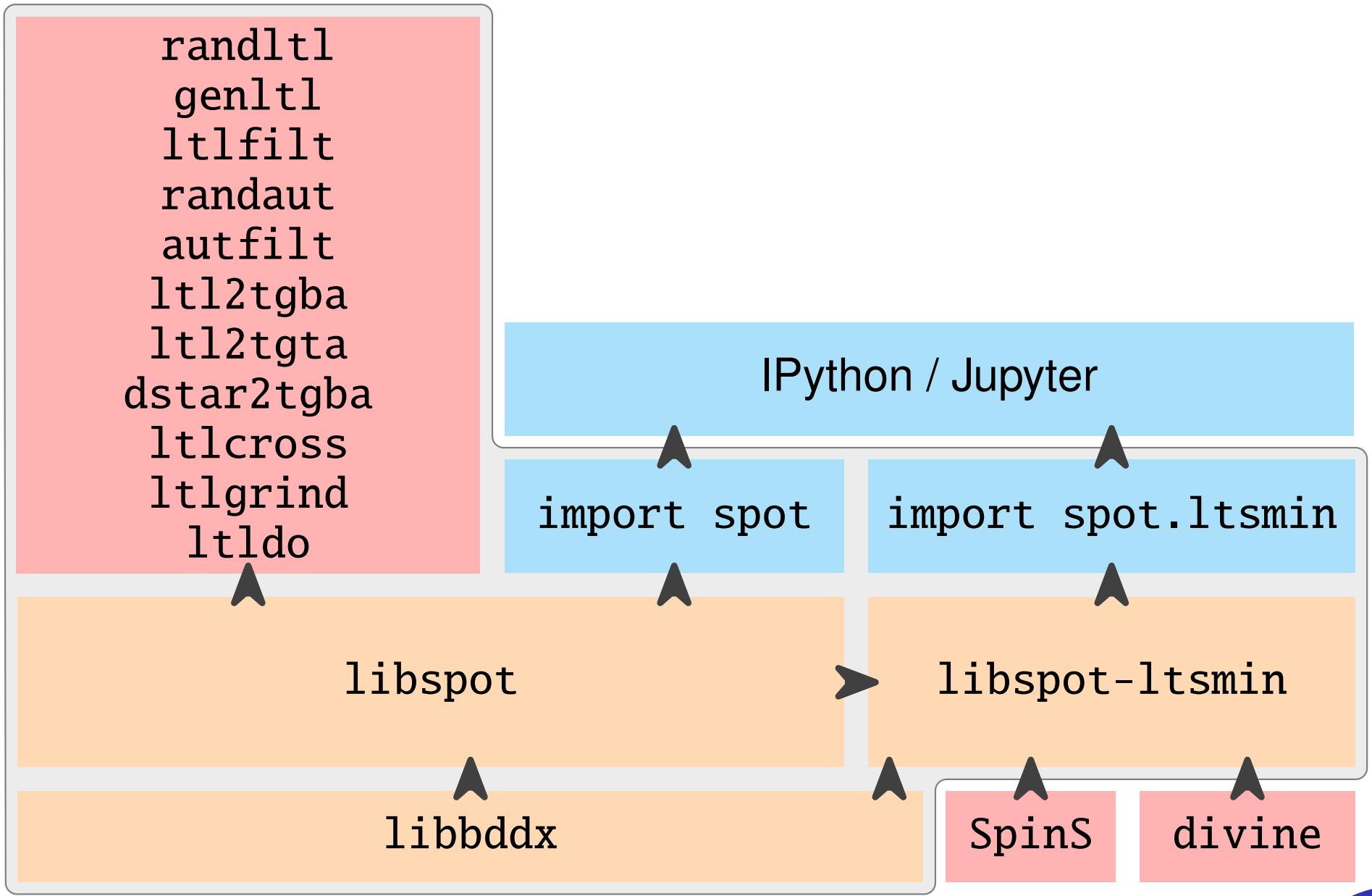
- Parsers, printers
- Simplifications, rewritings
- Implication, equivalence checks
- Stutter-invariance checks
- Filtering by properties
- Random formula generation
- Translation to Transition-based Generalized Büchi automata

Spot's Services for ω-automata

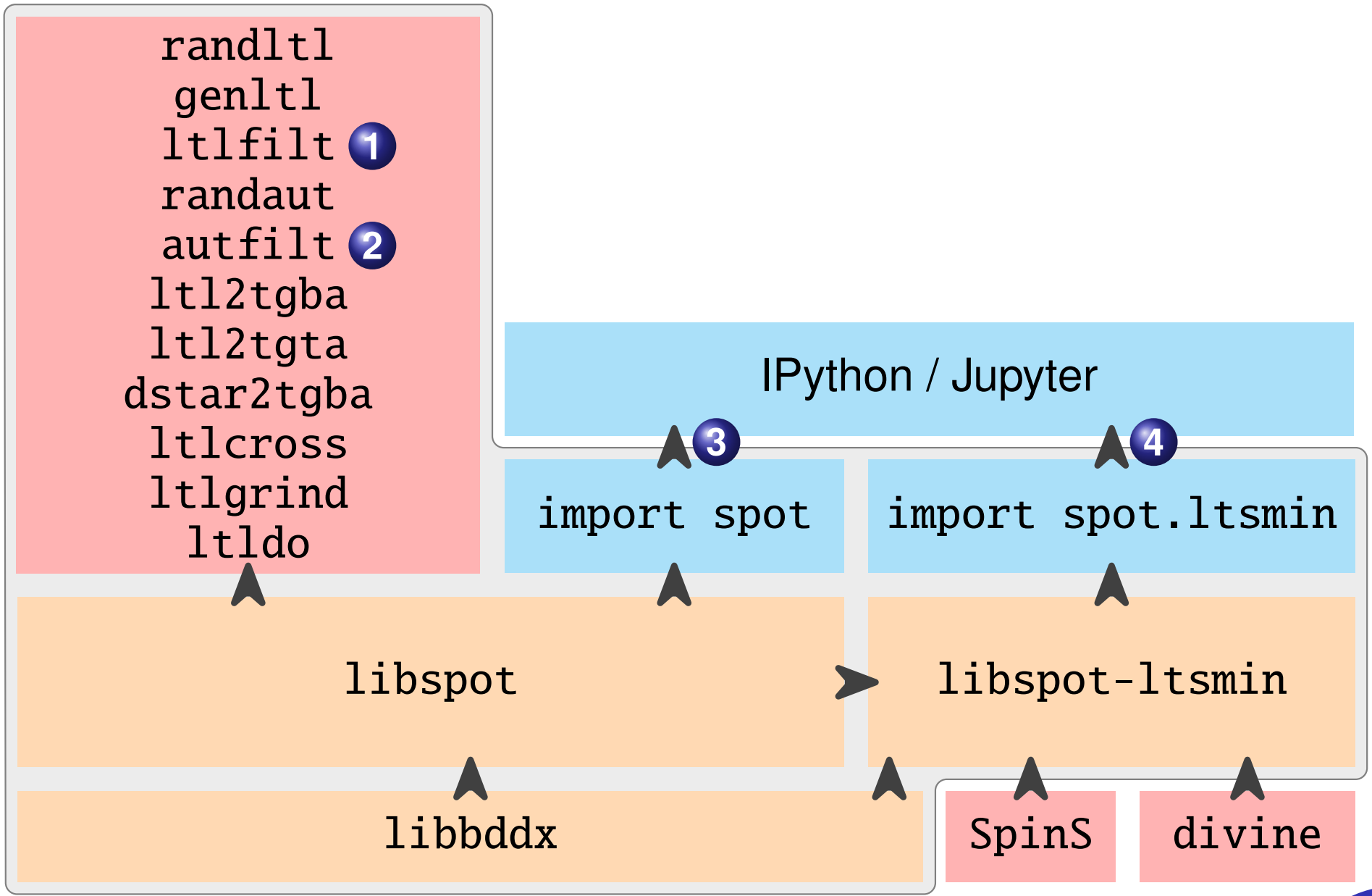
For ω-automata:

- Parsers, printers
- Simplifications: acceptance prunning, SCC-based mark simplifications, simulation-based reductions, minimization for obligation properties, ...
- Acceptance conversions
- Boolean operations:
  sum, product (on-the-fly if desired), complement
- Determinization (to Parity automata)
- Emptiness checks
- SAT-based minimization of deterministic automata
  (with arbitrary input and output acceptance)
- Random automata generation

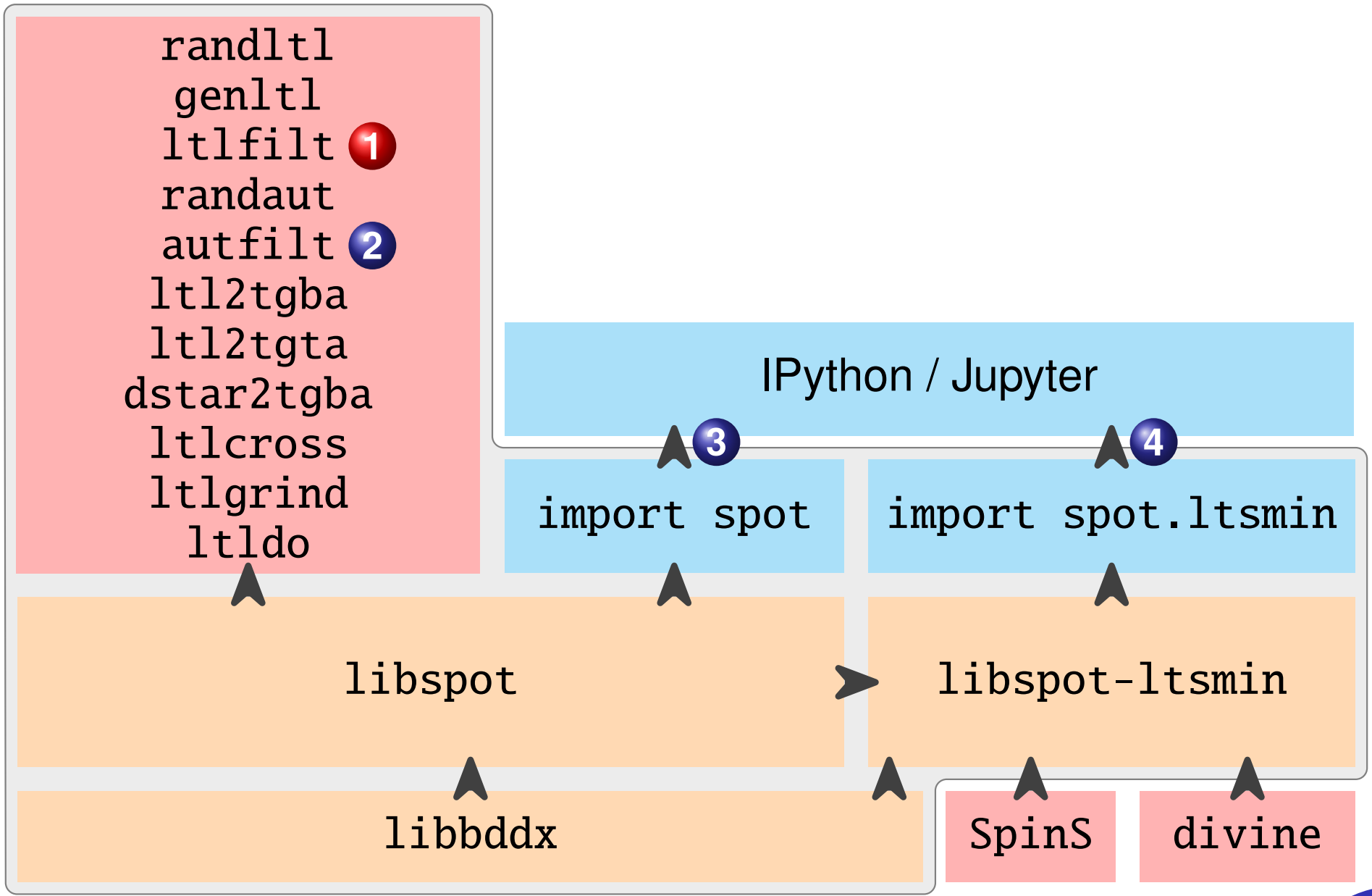
Spot's Architecture



Spot's Architecture



Spot's Architecture



`ltlfilt` — a Swiss-Army Knife for LTL Files

syntax conversion

formula transformations

formula filtering

`ltlfilt` — a Swiss-Army Knife for LTL Files

syntax conversion

syntax	input	output	example
Spot	(default)	(default)	`G(p0 -> (p1 R !p2))`
UTF-8	(default)	`-8`	`□(p0 → (p1 R ¬p2))`
Spin	(default)	`-s`	`[](p0 -> (p1 V !p2))`
Wring	(default)	`--wring`	`G(p0=1 -> (p1=1 R p2=0))`
Goal	(default)	n/a	`G(p0 --> (p1 R ~p2))`
LBT	`--lbt-input`	`-l`	`G i p0 V p1 !  p2`
LaTeX	n/a	`--latex`	`\G(p_0\limplies(p_1\R\lnot p_2))`

formula transformations

formula filtering

`ltlfilt` — a Swiss-Army Knife for LTL Files

syntax conversion

syntax	input	output	example
Spot	(default)	(default)	`G(p0 -> (p1 R !p2))`
UTF-8	(default)	`-8`	`□(p0 → (p1 R ¬p2))`
Spin	(default)	`-s`	`[](p0 -> (p1 V !p2))`
Wring	(default)	`--wring`	`G(p0=1 -> (p1=1 R p2=0))`
Goal	(default)	n/a	`G(p0 --> (p1 R ~p2))`
LBT	`--lbt-input`	`-l`	`G i p0 V p1 !  p2`
LaTeX	n/a	`--latex`	`\G(p_0\limplies(p_1\R\lnot p_2))`

formula transformations

`--boolean-to-isop`, `--exclusive-ap`, `--negate`, `--nnf`, `--relabel`, `--relabel-bool`, `--remove-wm`, `--remove-x`, `--simplify`, `--unabbreviate`

formula filtering

`ltlfilt` — a Swiss-Army Knife for LTL Files

syntax conversion

syntax	input	output	example
Spot	(default)	(default)	`G(p0 -> (p1 R !p2))`
UTF-8	(default)	`-8`	`□(p0 → (p1 R ¬p2))`
Spin	(default)	`-s`	`[](p0 -> (p1 V !p2))`
Wring	(default)	`--wring`	`G(p0=1 -> (p1=1 R p2=0))`
Goal	(default)	n/a	`G(p0 --> (p1 R ~p2))`
LBT	`--lbt-input`	`-l`	`G i p0 V p1 !  p2`
LaTeX	n/a	`--latex`	`\G(p_0\limplies(p_1\R\lnot p_2))`

formula transformations

`--boolean-to-isop`, `--exclusive-ap`, `--negate`, `--nnf`, `--relabel`, `--relabel-bool`, `--remove-wm`, `--remove-x`, `--simplify`, `--unabbreviate`

formula filtering

`--ap`, `--boolean`, `--bsize`, `--equivalent-to`, `--eventual`, `--guarantee`, `--implied-by`, `--imply`, `--ltl`, `--nox`, `--obligation`, `--safety`, `--size`, `--stutter-invariant`, `--syntactic-guarantee`, `--syntactic-obligation`, `--syntactic-persistence`, `--syntactic-recurrence`, `--syntactic-safety`, `--universal`, `--unique`, `--invert-match`

`ltlfilt` — Rewriting Formulas

Operator rewritings

```
$ ltlfilt --unabbreviate=iGF -f 'G(a -> Fb)'
0 R (!a | (1 U b))
```

`ltlfilt` — Rewriting Formulas

Operator rewritings

```
$ ltlfilt --unabbreviate=iGF -f 'G(a -> Fb)'
0 R (!a | (1 U b))
```

```
$ ltlfilt --unabbreviate=iGFR -f 'G(a -> Fb)'
(!a | (1 U b)) W 0
```

`ltlfilt` — Rewriting Formulas

Operator rewritings

```
$ ltlfilt --unabbreviate=iGF -f 'G(a -> Fb)'
0 R (!a | (1 U b))
```

```
$ ltlfilt --unabbreviate=iGFR -f 'G(a -> Fb)'
(!a | (1 U b)) W 0
```

```
$ ltlfilt --unabbreviate=iGFRW -f 'G(a -> Fb)'
!(1 U !(!a | (1 U b)))
```

`ltlfilt` — Rewriting Formulas

Operator rewritings

```
$ ltlfilt --unabbreviate=iGF -f 'G(a -> Fb)'
0 R (!a | (1 U b))
```

```
$ ltlfilt --unabbreviate=iGFR -f 'G(a -> Fb)'
(!a | (1 U b)) W 0
```

```
$ ltlfilt --unabbreviate=iGFRW -f 'G(a -> Fb)'
!(1 U !(!a | (1 U b)))
```

Simplifications

```
$ ltlfilt --simplify -f '!(1 U !(!a | (1 U b)))'
G(!a | Fb)
```

`ltlfilt` — Answering Simple Questions

Is a **U** (b **U** a) equivalent to b **U** a?

```
$ ltlfilt -f 'a U (b U a)' --equivalent-to 'b U a'
a U (b U a)
```

`ltlfilt` — Answering Simple Questions

Is a **U** (b **U** a) equivalent to b **U** a?

```
$ ltlfilt -f 'a U (b U a)' --equivalent-to 'b U a'
a U (b U a)
```

Which of these formulas are stutter-invariant?

```
$ ltlfilt -f 'G(a | X(a -> b))' -f 'G(a | X(a <-> b))' \
  --stutter-invariant
G(a | X(a -> b))
```

T. Michaud and A. Duret-Lutz. Practical stutter-invariance checks for ω-regular languages. *SPIN'15*

`ltlfilt` — Answering Simple Questions

Is $a \mathbf{U} (b \mathbf{U} a)$ equivalent to $b \mathbf{U} a$?

```
$ ltlfilt -f 'a U (b U a)' --equivalent-to 'b U a'
a U (b U a)
```

Which of these formulas are stutter-invariant?

```
$ ltlfilt -f 'G(a | X(a -> b))' -f 'G(a | X(a <-> b))' \
  --stutter-invariant
G(a | X(a -> b))
```

Give an **X**-free formula for $\mathbf{G}(a \vee \mathbf{X}(a \rightarrow b))$

```
$ ltlfilt -f 'G(a | X(a -> b))' --remove-x --simplify
G(a | (!a & (!a U (a & (!a | b))) & ((!b U a) | (b U a))) |
(b & (b U (!b & (!a | b))) & ((!a U !b) | (a U !b))) | ((!a |
b) & (G!a | Ga) & (G!b | Gb)) | (!b & ((!a U b) | (a U b))))
```

K. Etessami. A note on a question of Peled and Wilke regarding stutter-invariant LTL. *Information Processing Letters*, 75(6):261–263, 2000

`ltlfilt` — Random Generation with Constraints

Build 10 pathological safety formulas

```
$ randltl -n -1 --tree-size=10..13 a b |
  ltlfilt --simplify --safety --uniq |
  ltlfilt --invert-match --syntactic-safety -n 10
```

```
F(!a | Ga)
(!b & X((b W Xb) R b)) | (b & X((!b M X!b) U !b))
G(b U XXb)
(((!b & XGa) | (b & XF!a)) W b) R a
((b W a) M b) | (!a R X!a)
F(b | X!b)
G((b & XF!b) | (!b & XGb))
Xa U (Gb | Ga)
Fa R X!a
F(b | G!b)
```

`ltlfilt` — Random Generation with Constraints

Build 10 pathological safety formulas

```
$ randltl -n -1 --tree-size=10..13 a b |
  ltlfilt --simplify --safety --uniq |
  ltlfilt --invert-match --syntactic-safety -n 10
```

```
F(!a | Ga)
(!b & X((b W Xb) R b)) | (b & X((!b M X!b) U !b))
G(b U XXb)
(((!b & XGa) | (b & XF!a)) W b) R a
((b W a) M b) | (!a R X!a)
F(b | X!b)
G((b & XF!b) | (!b & XGb))
Xa U (Gb | Ga)
Fa R X!a
F(b | G!b)
```

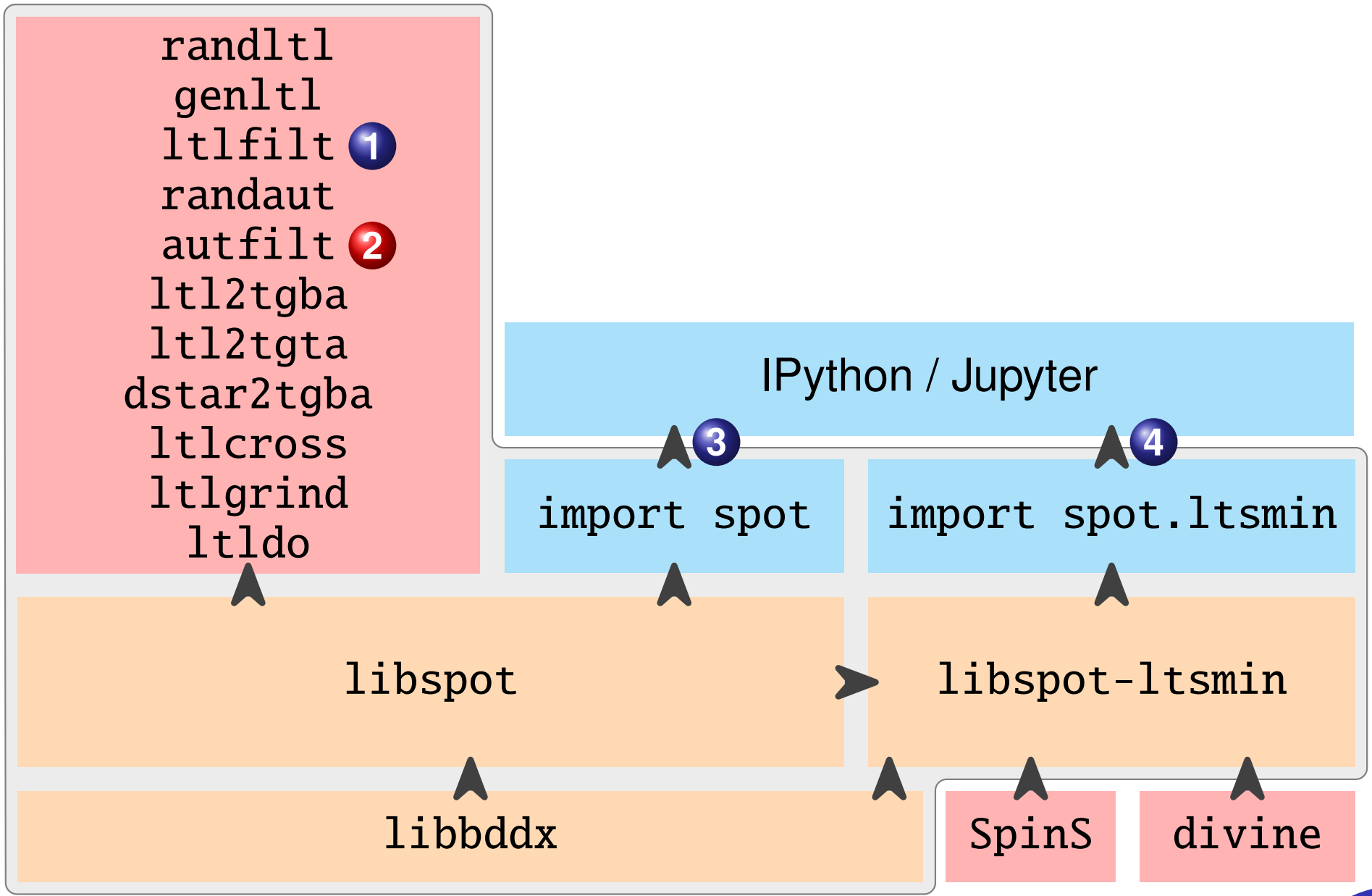
`ltlfilt` — Random Generation with Constraints

Build 10 pathological safety formulas not equivalent to ⊤ or ⊥

```
$ randltl -n -1 --tree-size=10..13 a b |
  ltlfilt --simplify --safety --uniq |
  ltlfilt --invert-match --syntactic-safety |
  ltlfilt --invert-match --equivalent-to=1 |
  ltlfilt --invert-match --equivalent-to=0
```

```
(!b & X((b W Xb) R b)) | (b & X((!b M X!b) U !b))
G(b U XXb)
(((!b & XGa) | (b & XF!a)) W b) R a
((b W a) M b) | (!a R X!a)
Xa U (Gb | Ga)
Fa R X!a
(a & (!a U b)) | (!a & (a R !b))
G(((a & b) | (!a & !b)) & (a M b)) R a
b & ((!b & F!a) | (b & Ga))
b & (!a | (b M Xb))
```

Spot's Architecture



`autfilt` — a Swiss-Army Knife for ω-Automata

format conversion

transformations

filtering

simplification objectives

output constraints

`autfilt` — a Swiss-Army Knife for ω-Automata

format conversion

HOA (rw), never claim (rw), LBTT (rw), DSTAR (r), dot (w)

transformations

filtering

simplification objectives

output constraints

`autfilt` — a Swiss-Army Knife for ω-Automata

format conversion

HOA (rw), never claim (rw), LBTT (rw), DSTAR (r), dot (w)

transformations

`--cleanup-acceptance`, `--cnf-acceptance`, `--complement`, `--complement-acceptance`, `--decompose-strength`, `--destut`, `--dnf-acceptance`, `--exclusive-ap`, `--instut`, `--keep-states`, `--mask-acc`, `--merge-transitions`, `--product`, `--product-or`, `--randomize`, `--remove-ap`, `--remove-dead-states`, `--remove-fin`, `--remove-unreachable-states`, `--sat-minimize`, `--separate-sets`, `--simplify-exclusive-ap`, `--strip-acceptance`

filtering

simplification objectives

output constraints

`autfilt` — a Swiss-Army Knife for ω-Automata

format conversion

HOA (rw), never claim (rw), LBTT (rw), DSTAR (r), dot (w)

transformations

`--cleanup-acceptance`, `--cnf-acceptance`, `--complement`, `--complement-acceptance`, `--decompose-strength`, `--destut`, `--dnf-acceptance`, `--exclusive-ap`, `--instut`, `--keep-states`, `--mask-acc`, `--merge-transitions`, `--product`, `--product-or`, `--randomize`, `--remove-ap`, `--remove-dead-states`, `--remove-fin`, `--remove-unreachable-states`, `--sat-minimize`, `--separate-sets`, `--simplify-exclusive-ap`, `--strip-acceptance`

filtering

`--acc-sets`, `--accept-word`, `--ap`, `--are-isomorphic`, `--edges`, `--equivalent-to`, `--included-in`, `--intersect`, `--is-complete`, `--is-deterministic`, `--is-empty`, `--is-inherently-weak`, `--is-terminal`, `--is-unambiguous`, `--is-weak`, `--reject-word`, `--states`, `--unique`, `--invert-match`

simplification objectives

output constraints

`autfilt` — a Swiss-Army Knife for ω-Automata

format conversion

HOA (rw), never claim (rw), LBTT (rw), DSTAR (r), dot (w)

transformations

`--cleanup-acceptance`, `--cnf-acceptance`, `--complement`, `--complement-acceptance`, `--decompose-strength`, `--destut`, `--dnf-acceptance`, `--exclusive-ap`, `--instut`, `--keep-states`, `--mask-acc`, `--merge-transitions`, `--product`, `--product-or`, `--randomize`, `--remove-ap`, `--remove-dead-states`, `--remove-fin`, `--remove-unreachable-states`, `--sat-minimize`, `--separate-sets`, `--simplify-exclusive-ap`, `--strip-acceptance`

filtering

`--acc-sets`, `--accept-word`, `--ap`, `--are-isomorphic`, `--edges`, `--equivalent-to`, `--included-in`, `--intersect`, `--is-complete`, `--is-deterministic`, `--is-empty`, `--is-inherently-weak`, `--is-terminal`, `--is-unambiguous`, `--is-weak`, `--reject-word`, `--states`, `--unique`, `--invert-match`

simplification objectives

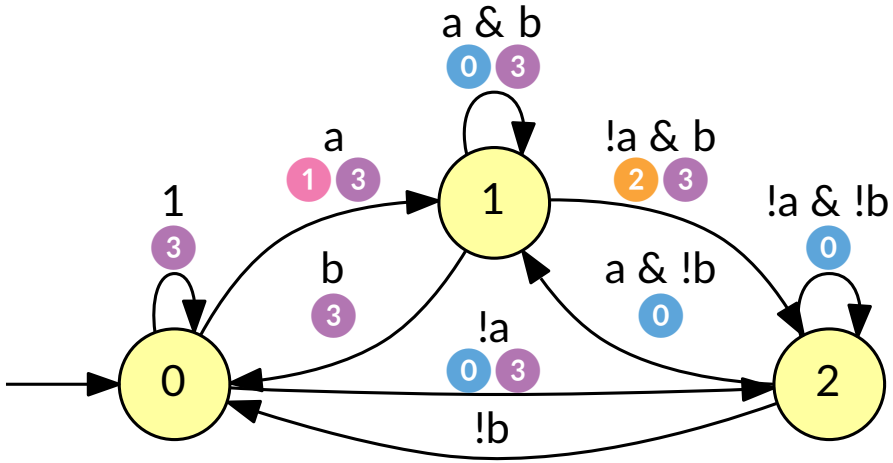
`--any`, `--small`, `--deterministic`

output constraints

`--ba`, `--complete`, `--generic`, `--monitor`, `--sbacc`, `--tgba`

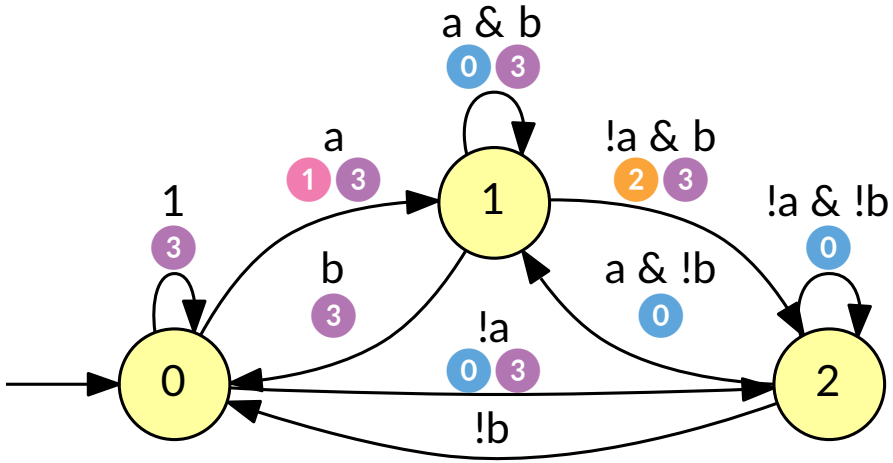
`autfilt` — Acceptance Transformations

(Fin(1) & Fin(3) & Inf(0)) | (Inf(2)&Inf(3)) | Inf(1)



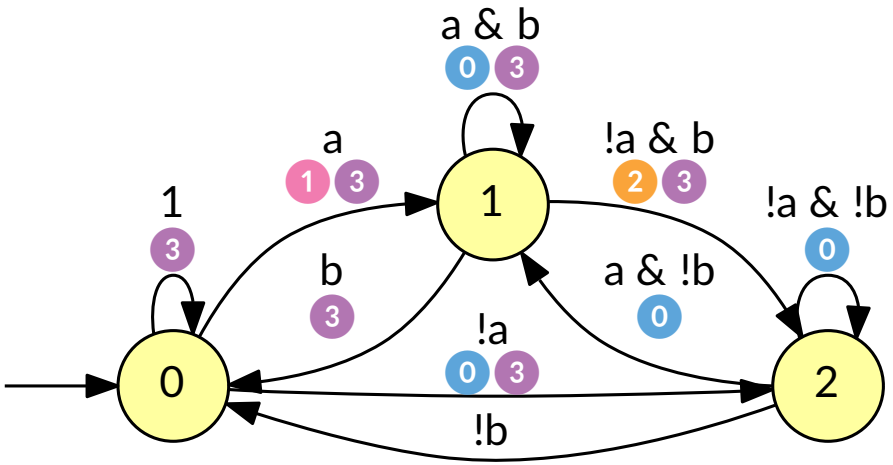
autfilt — Acceptance Transformations

(Fin(1) & Fin(3) & Inf(0)) | (Inf(2)&Inf(3)) | Inf(1)



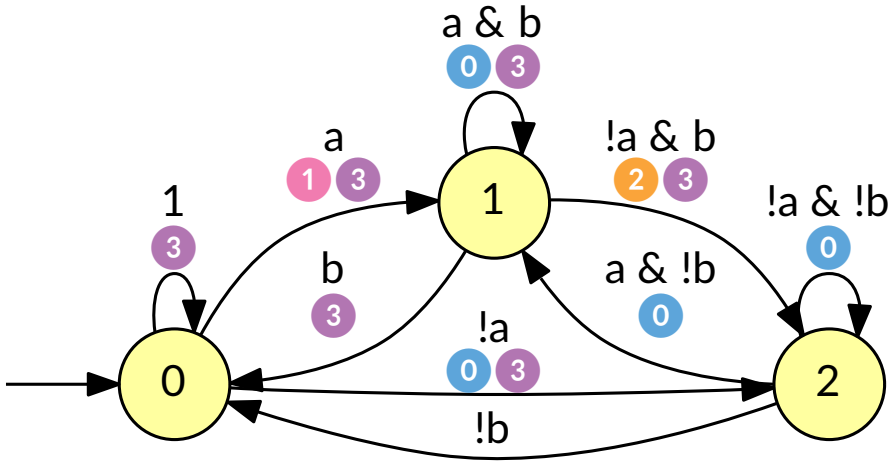

```
$ autfilt --cnf-acceptance example.hoa > output.hoa
```

(Inf(0) | Inf(1) | Inf(3)) & (Fin(3) | Inf(1) | Inf(2))



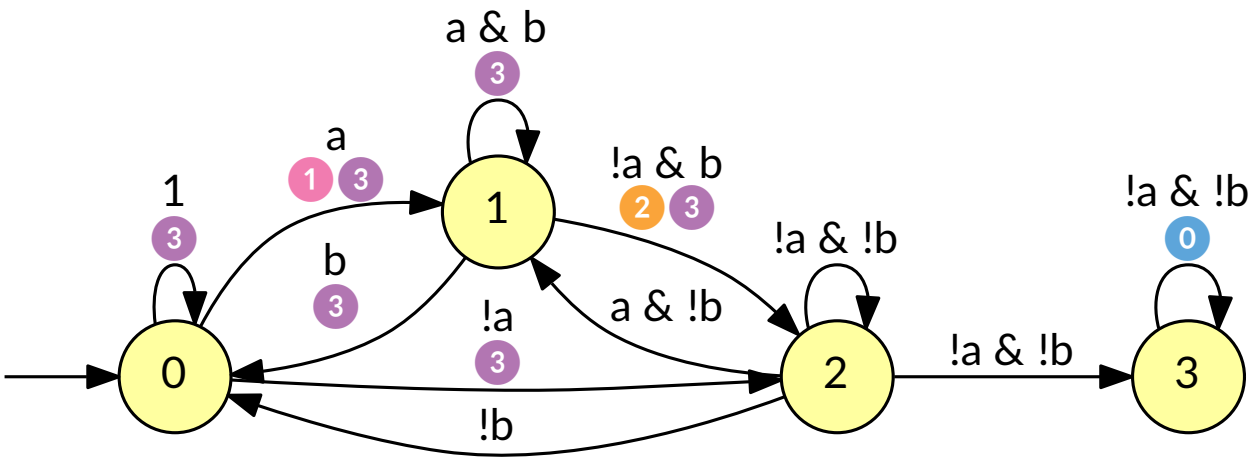
autfilt — Acceptance Transformations

(Fin(1) & Fin(3) & Inf(0)) | (Inf(2)&Inf(3)) | Inf(1)



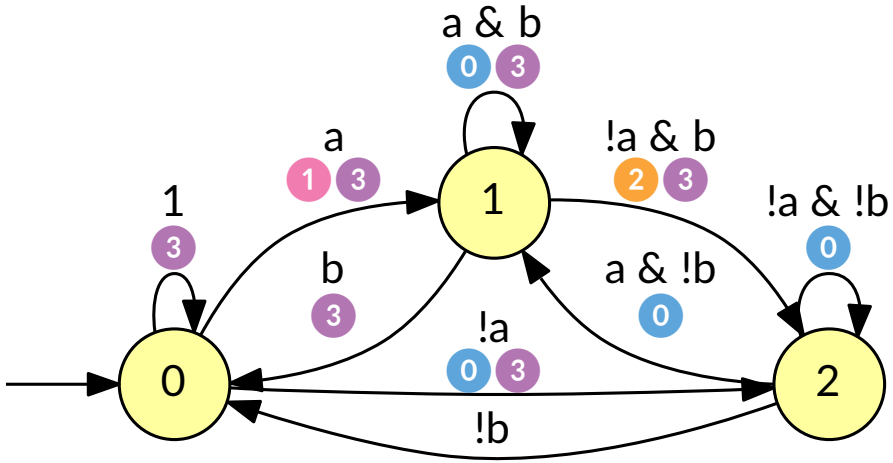

```
$ autfilt --remove-fin example.hoa > output.hoa
```

Inf(0) | Inf(1) | (Inf(2)&Inf(3))



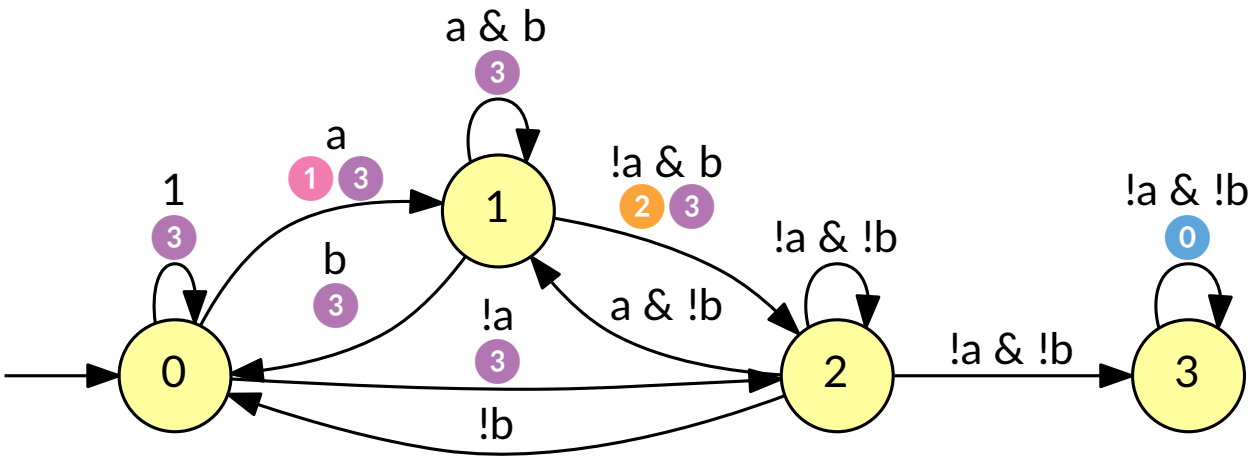
`autfilt` — Acceptance Transformations

(Fin(1) & Fin(3) & Inf(0)) | (Inf(2)&Inf(3)) | Inf(1)

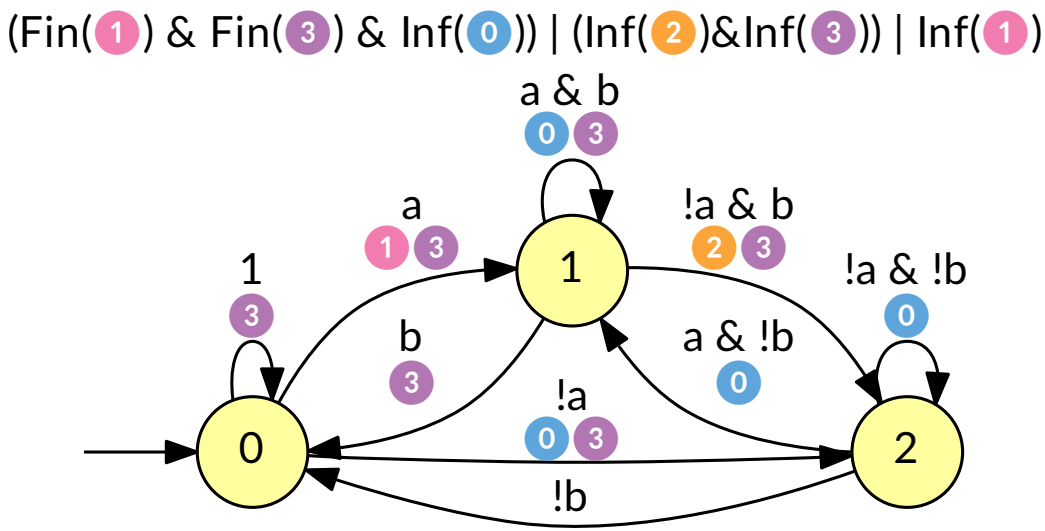



```
$ autfilt --remove-fin --cnf-acc example.hoa > output.hoa
```

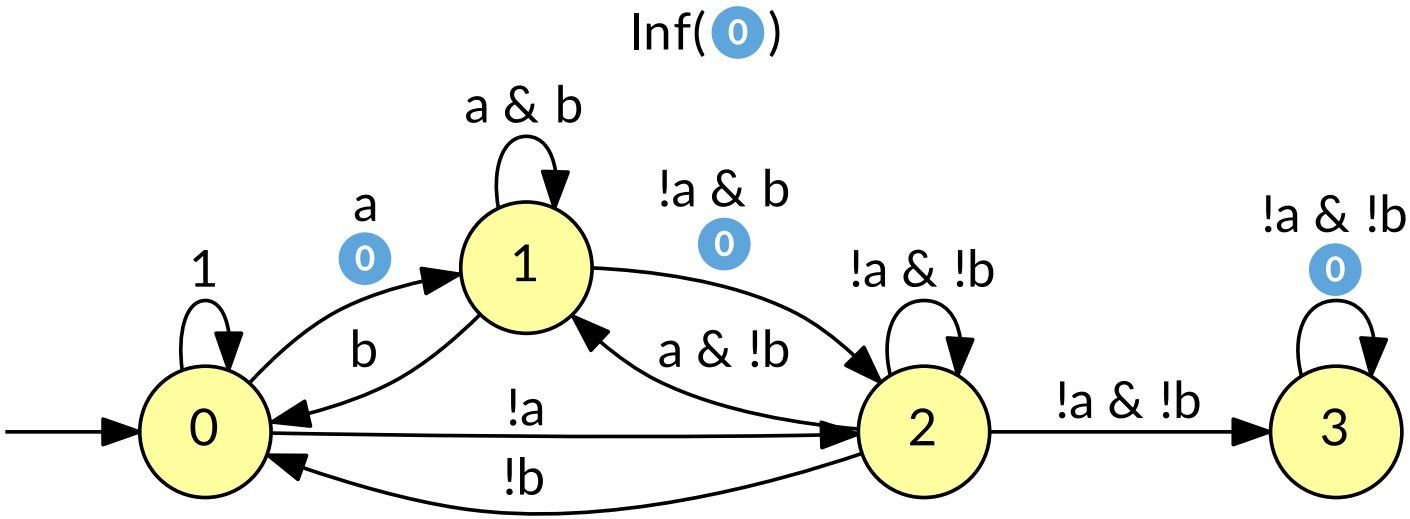
(Inf(0) | Inf(1) | Inf(2)) & (Inf(0) | Inf(1) | Inf(3))



autfilt — Acceptance Transformations

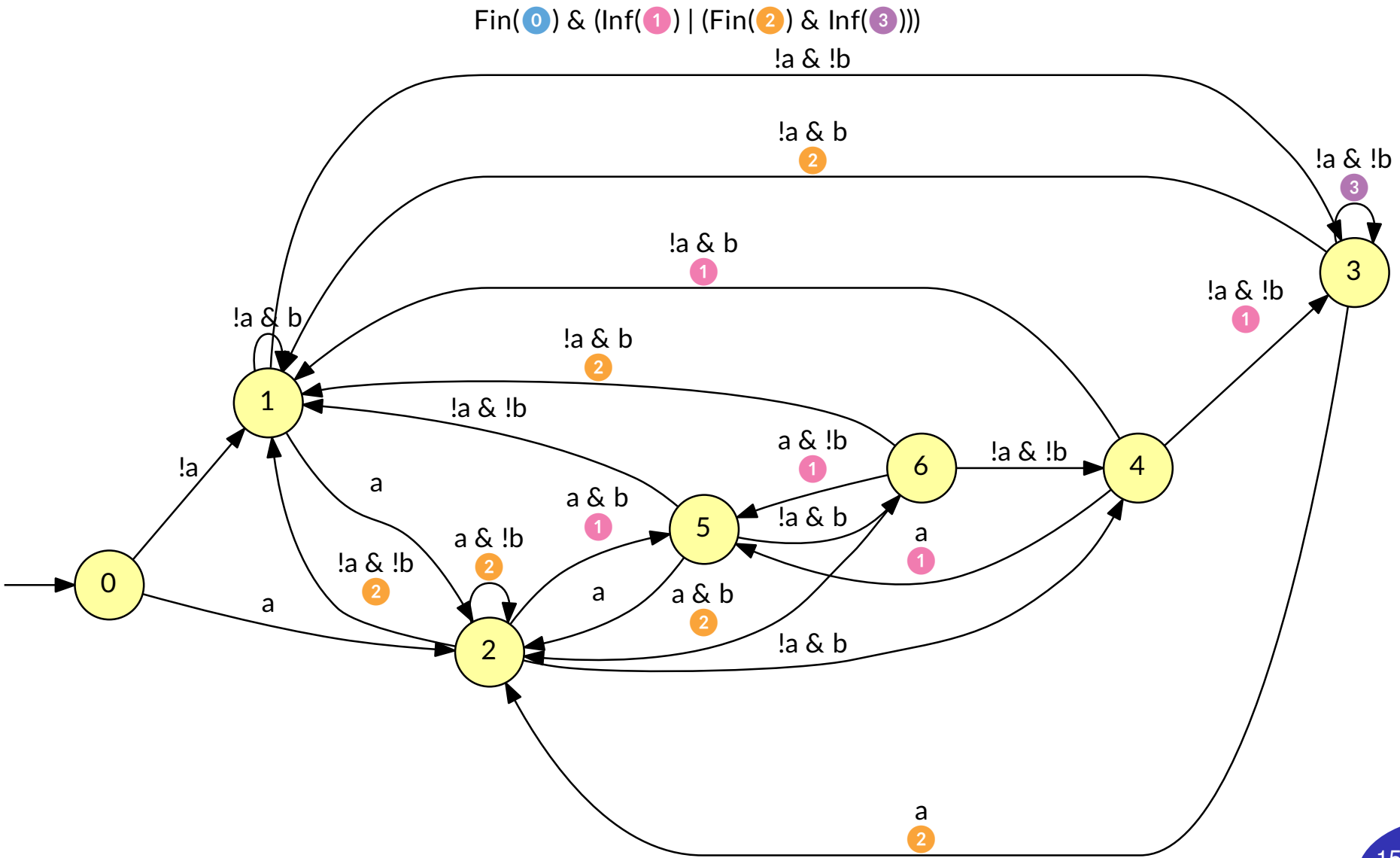



```
$ autfilt --tgba example.hoa > output.hoa
```



autfilt — Determinization

```
$ autfilt --deterministic example.hoa > output.hoa
```



Complex Pipelines via Named Automata (1/2)

The HOA output carry some extra information

```
$ ltl2tgba -f 'a U b'
HOA: v1
name: "a U b"
States: 2
Start: 1
AP: 2 "a" "b"
acc-name: Buchi
Acceptance: 1 Inf(0)
properties: trans-labels explicit-labels state-acc
properties: deterministic stutter-invariant terminal
--BODY--
State: 0 {0}
[t] 0
State: 1
[1] 0
[0&!1] 1
--END--
```

Complex Pipelines via Named Automata (1/2)

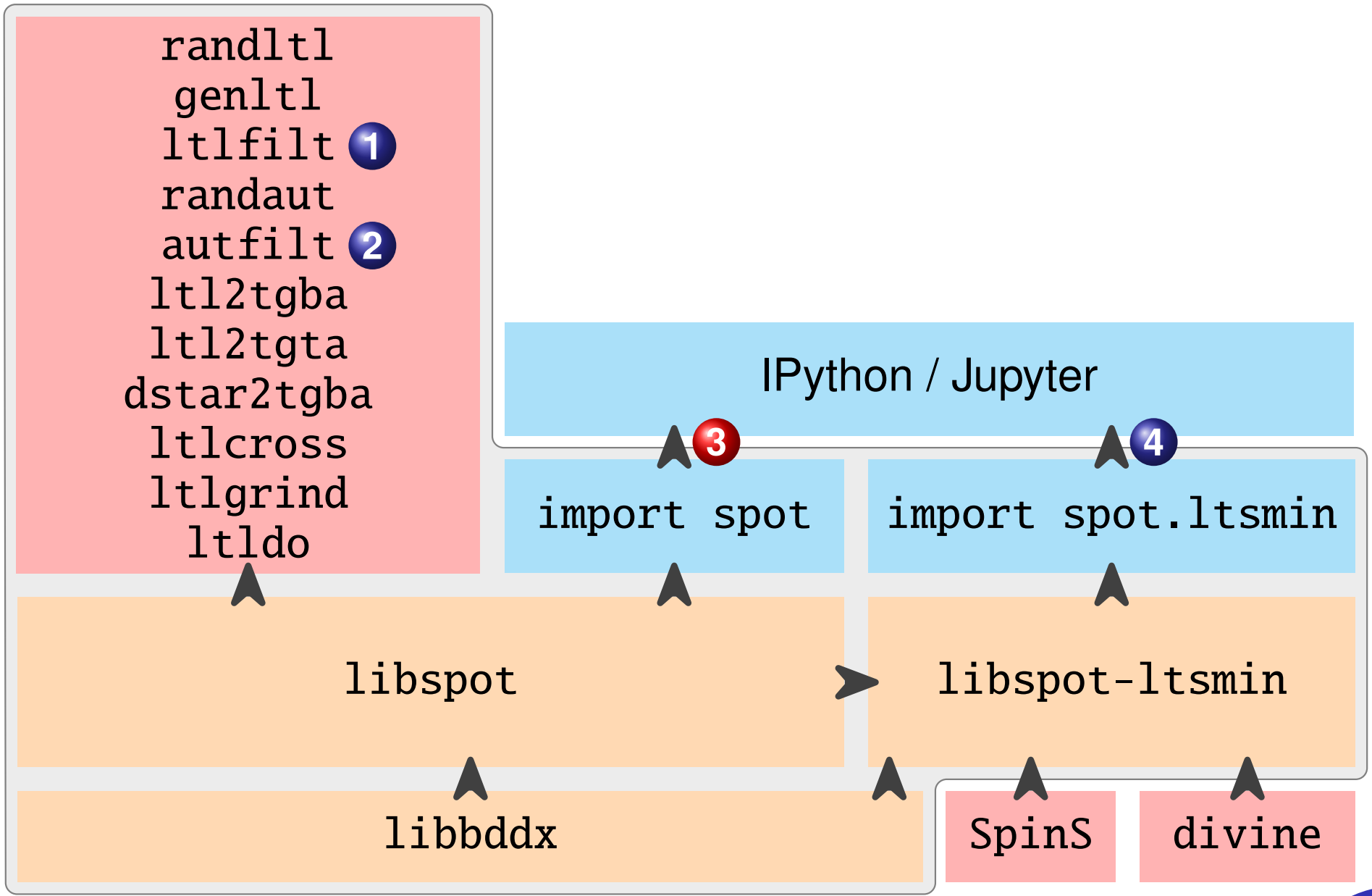
The HOA output carry some extra information

```
$ ltl2tgba -f 'a U b' | autfilt --stats=%M
a U b
```

Complex Pipelines via Named Automata (2/2)

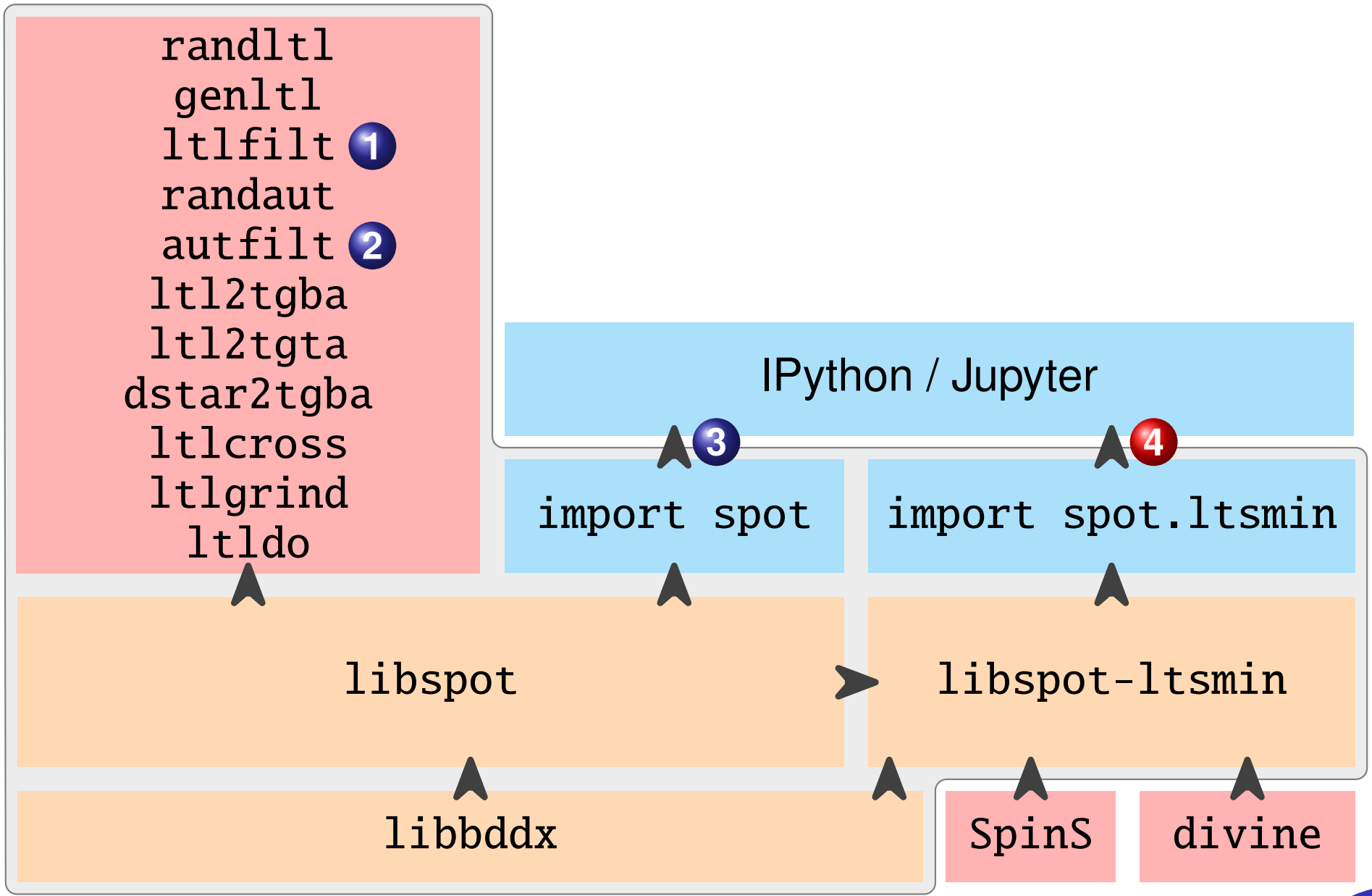
```
$ randltl -n -1 a b | ltlfilt --simplify --uniq |
  ltl2tgba -F- |
  autfilt --accept-word='a&!b;cycle{!a&!b}' \
          --accept-word='!a&!b;cycle{a&b}' \
          --reject-word='cycle{b}' --stats=%M -n 10
F!b
!b
F(!a & !b)
(!a & (XX!a | (!a W F!b))) R !b
F(Fb R !b)
Fa R F!b
Fa U !b
!b & X(!b W Ga)
Fb R F!b
XF!b U (!b & (!a | G!b))
```

Spot's Architecture

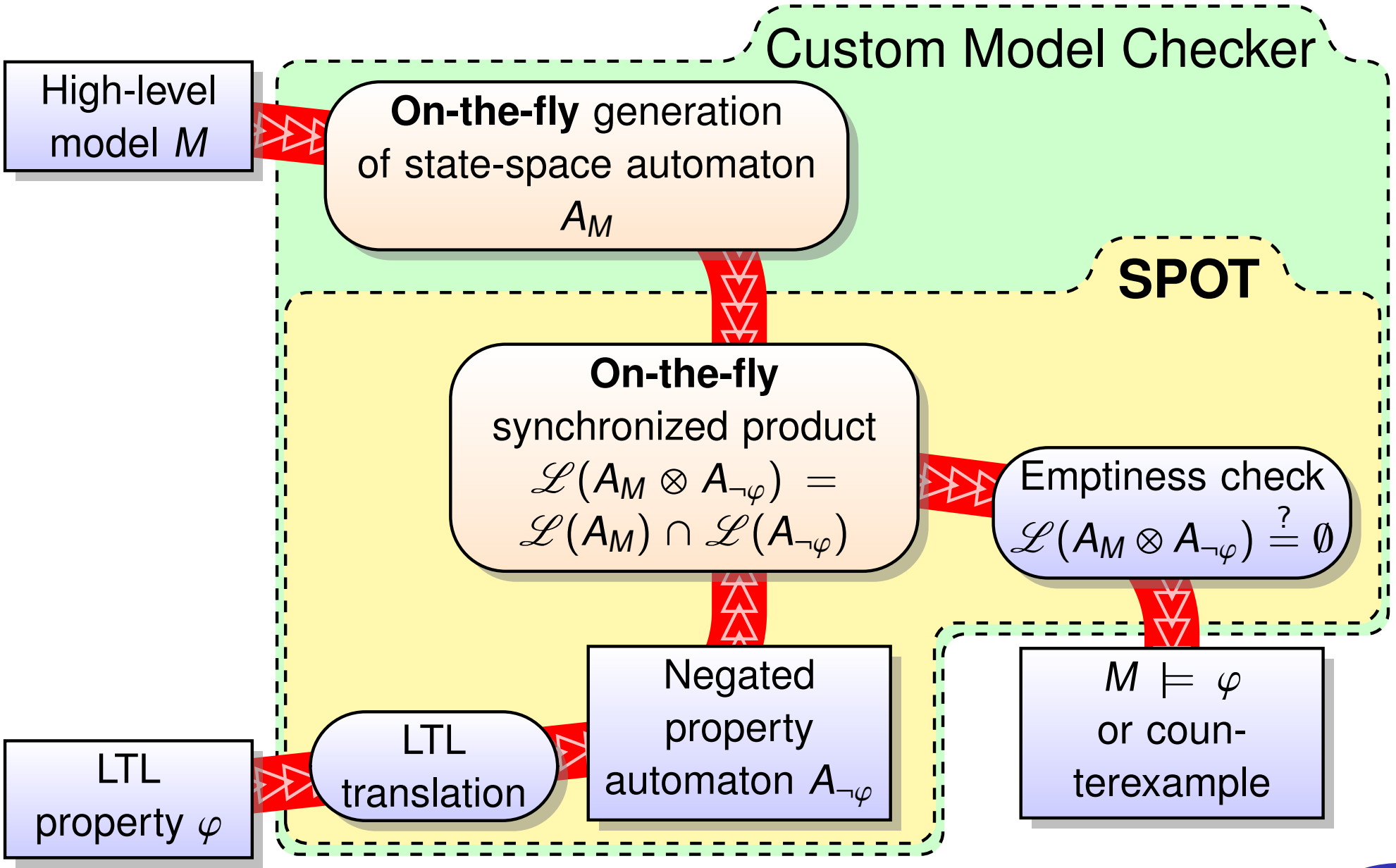


Live demo.

Spot's Architecture



Automata-theoretic LTL model checking



Live demo.

Conclusion & Availability

Summary:

- C++ library for model checking & ω-automata manipulation
- Support for generic acceptance conditions
- Shell commands & Python bindings
- Interface with several tools

Availability:

- `https://spot.lrde.epita.fr/` (new site)
- License GNU GPL v3+
- Source tarball or Debian packages
- Live Jupyter installation at `http://spot-sandbox.lrde.epita.fr/`

Future work:

- More uses of generic acceptance condition