
SPIN'25
Growing an
ω-Automaton Library
Alexandre Duret-Lutz (EPITA, France)

What is Spot?

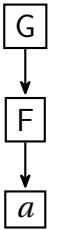
A platform for manipulation of LTL formulas and ω-automata. With three interfaces.

- C++17 library
- Command-line tools
- Python bindings

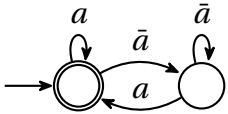
What is Spot?

A platform for manipulation of LTL formulas and ω-automata. With three interfaces.
E.g. convert an LTL formula into a neverclaim for Spin:

$\mathrm{GF}a$ ➡



➡



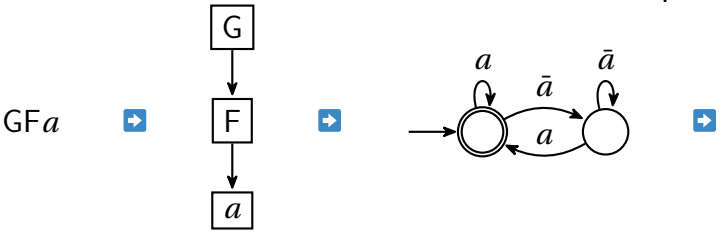
➡

```
never { /* GFa */
accept_init:
  if
  :: (a) -> goto accept_init
  :: (!(a)) -> goto T0_S1
  fi;
T0_S1:
  if
  :: (a) -> goto accept_init
  :: (!(a)) -> goto T0_S1
  fi;
}
```

- C++17 library
- Command-line tools
- Python bindings

What is Spot?

A platform for manipulation of LTL formulas and ω-automata. With three interfaces.
E.g. convert an LTL formula into a neverclaim for Spin:



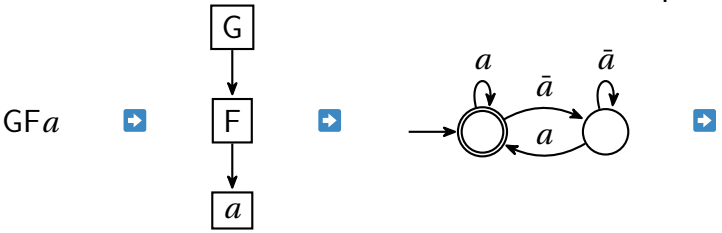

```
never { /* GFa */
accept_init:
  if
  :: (a) -> goto accept_init
  :: (!(a)) -> goto T0_S1
  fi;
T0_S1:
  if
  :: (a) -> goto accept_init
  :: (!(a)) -> goto T0_S1
  fi;
}
```

- C++17 library
- Command-line tools
- Python bindings

```
spot::parsed_formula pf =
  spot::parse_infix_psl("GFa");
if (pf.format_errors(std::cerr))
  exit(1);
spot::translator trans;
trans.set_type(spot::postprocessor::Buchi);
trans.set_pref(spot::postprocessor::SBAcc
               | spot::postprocessor::Small);
spot::twa_graph_ptr aut = trans.run(pf.f);
print_never_claim(std::cout, aut) << '\n';
```

What is Spot?

A platform for manipulation of LTL formulas and ω-automata. With three interfaces.
E.g. convert an LTL formula into a neverclaim for Spin:




```
never { /* GFa */
accept_init:
  if
  :: (a) -> goto accept_init
  :: (!(a)) -> goto T0_S1
  fi;
T0_S1:
  if
  :: (a) -> goto accept_init
  :: (!(a)) -> goto T0_S1
  fi;
}
```

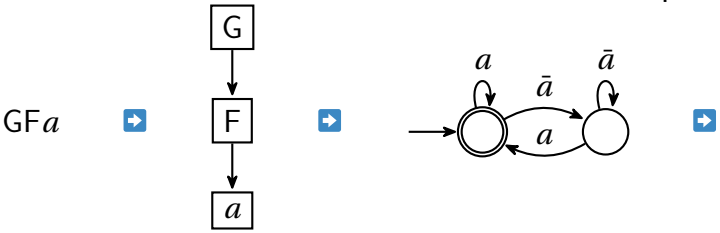
- C++17 library
- Command-line tools
- Python bindings

```
spot::parsed_formula pf =
  spot::parse_infix_psl("GFa");
if (pf.format_errors(std::cerr))
  exit(1);
spot::translator trans;
trans.set_type(spot::postprocessor::Buchi);
trans.set_pref(spot::postprocessor::SBAcc
               | spot::postprocessor::Small);
spot::twa_graph_ptr aut = trans.run(pf.f);
print_never_claim(std::cout, aut) << '\n';
```

```
% ltl2tgba --spin GFa
```

What is Spot?

A platform for manipulation of LTL formulas and ω-automata. With three interfaces.
E.g. convert an LTL formula into a neverclaim for Spin:




```
never { /* GFa */
accept_init:
  if
  :: (a) -> goto accept_init
  :: (!(a)) -> goto T0_S1
  fi;
T0_S1:
  if
  :: (a) -> goto accept_init
  :: (!(a)) -> goto T0_S1
  fi;
}
```

- C++17 library

```
spot::parsed_formula pf =
  spot::parse_infix_psl("GFa");
if (pf.format_errors(std::cerr))
  exit(1);
spot::translator trans;
trans.set_type(spot::postprocessor::Buchi);
trans.set_pref(spot::postprocessor::SBAcc
               | spot::postprocessor::Small);
spot::twa_graph_ptr aut = trans.run(pf.f);
print_never_claim(std::cout, aut) << '\n';
```

- Command-line tools

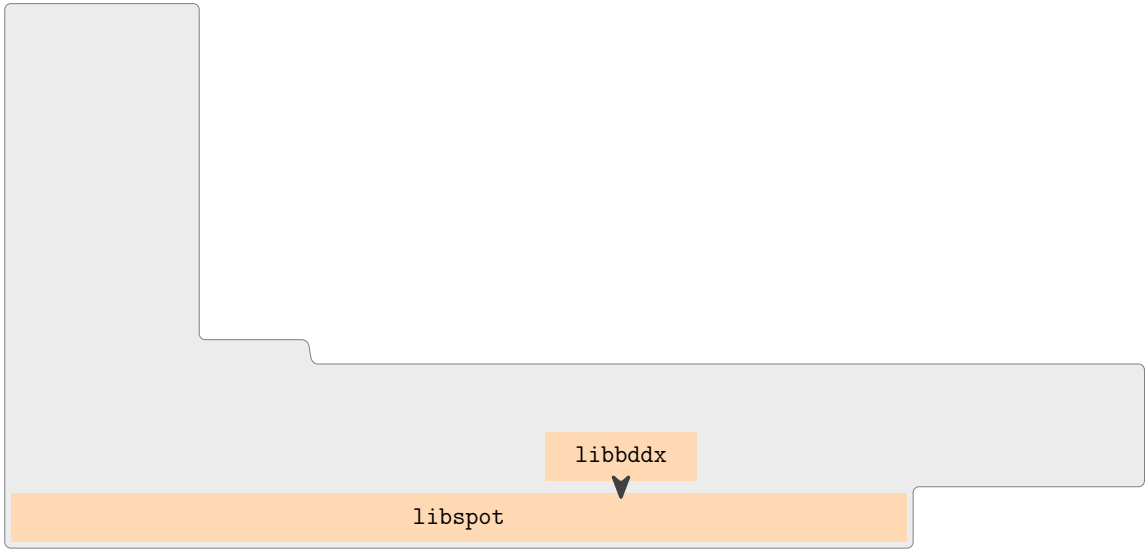
```
% ltl2tgba --spin GFa
```

- Python bindings

```
import spot
a = spot.translate('GFa', 'buchi', 'sbacc')
print(a.to_str('spin'))
```

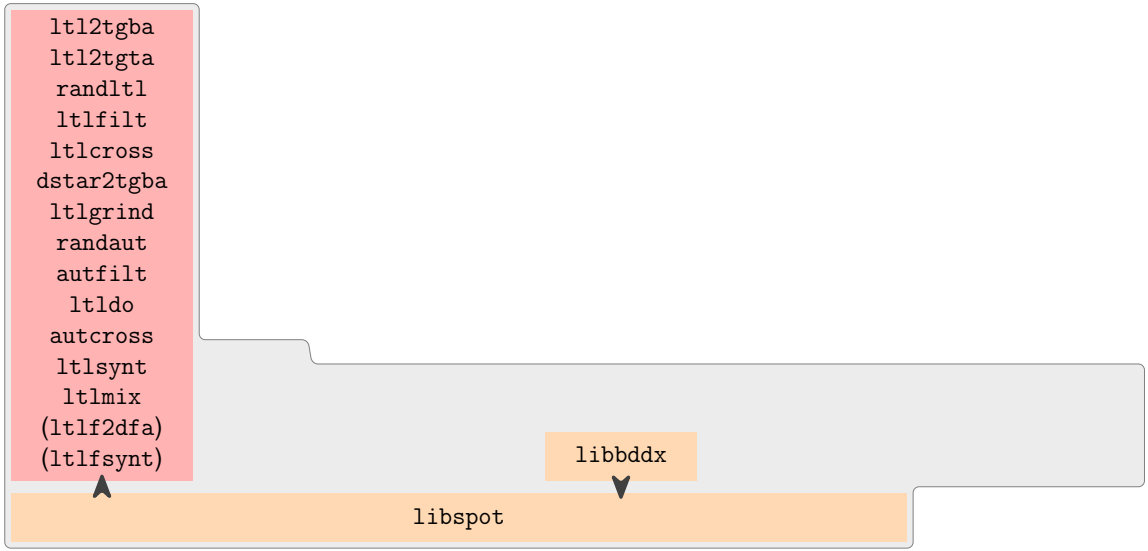
- With graphical representations in Jupyter

Architecture

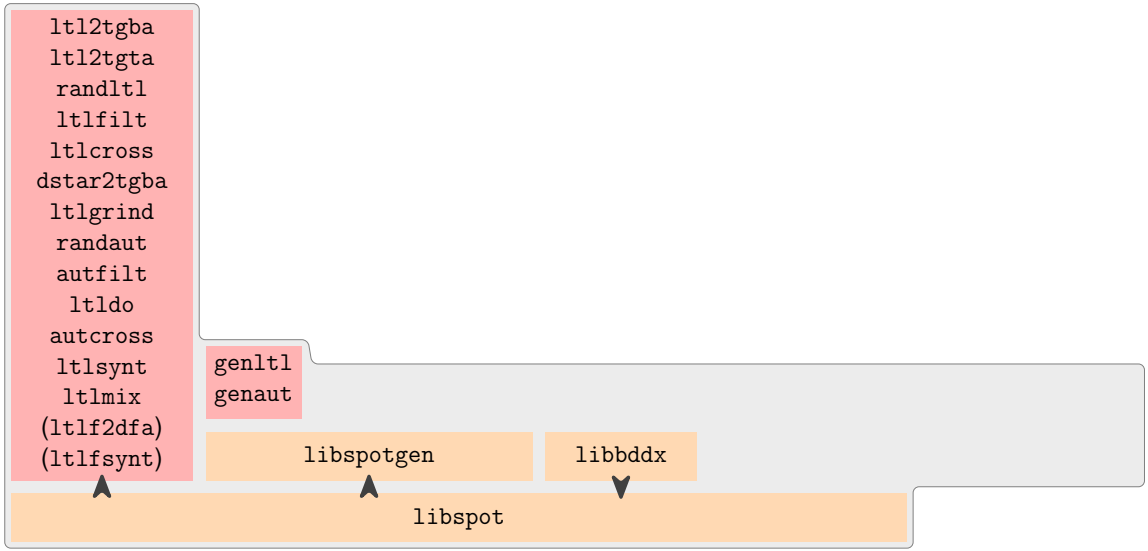


Lind-Nielsen. *BuDDy: A binary decision diagram package*, 1999. www

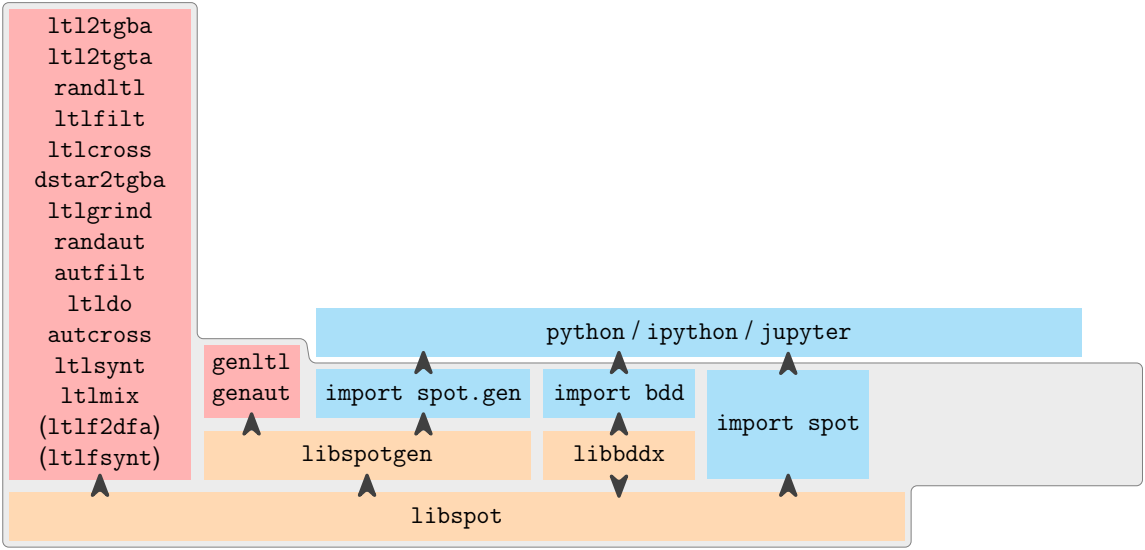
Architecture



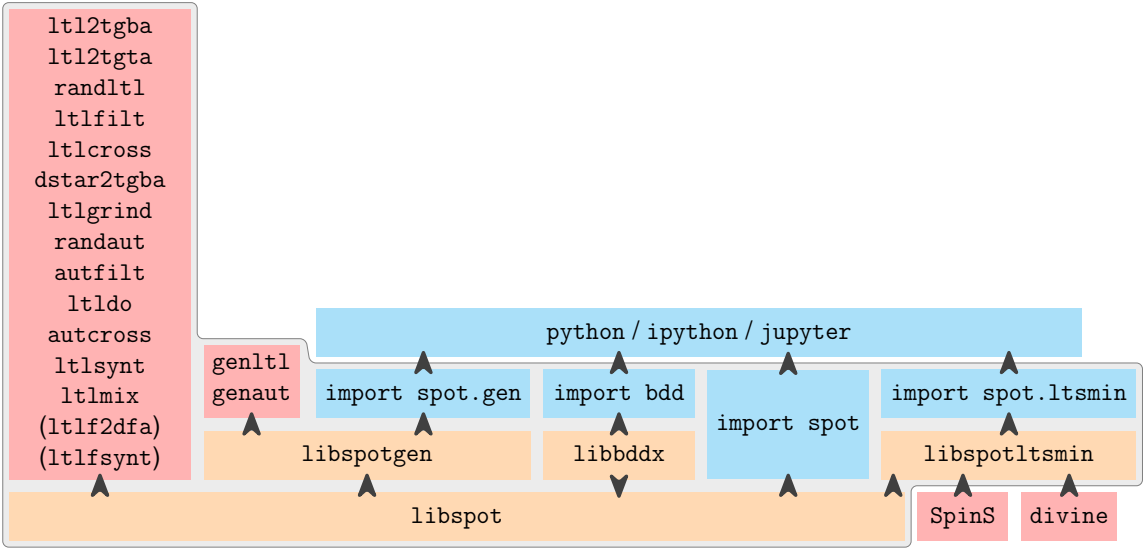
Architecture



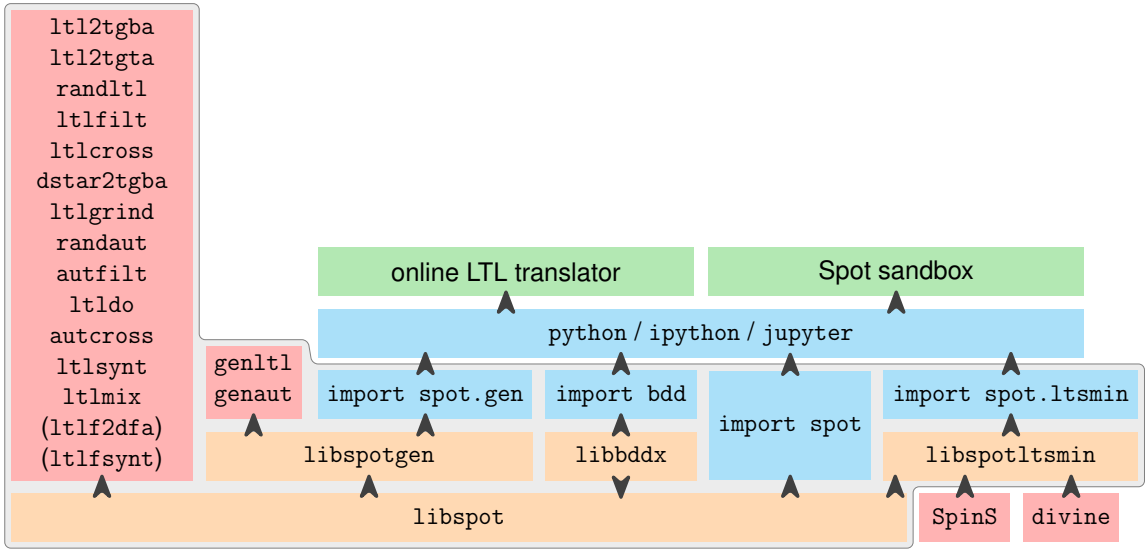
Architecture



Architecture



Architecture



The evolution

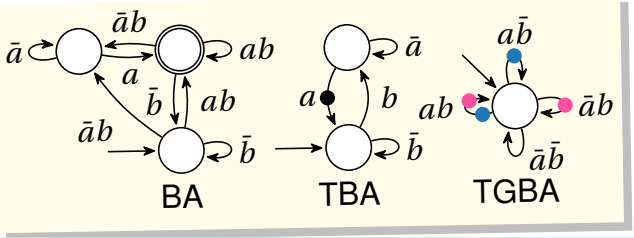
1. Model checking library (2003–2012)
   - using TGBA for model checking
   - pure C++ library
   - no tools (except for the test suite)

Duret-Lutz and Poitrenaud. *Spot: an extensible model checking library using transition-based generalized Büchi automata*. **MASCOTS'04**. doi

The evolution

1 Model checking library (2003–2012)

- using TGBA for model checking
- pure C++ library
- no tools (except for the test suite)



Duret-Lutz and Poitrenaud. *Spot: an extensible model checking library using transition-based generalized Büchi automata*. **MASCOTS'04**. doi

The evolution

1. Model checking library (2003–2012)
   - using TGBA for model checking
   - pure C++ library
   - no tools (except for the test suite)

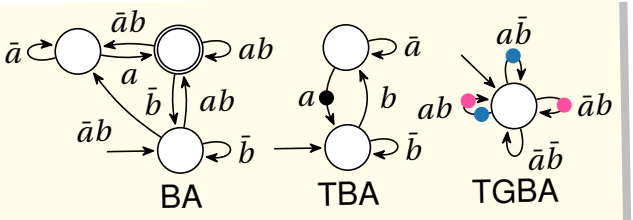


Fig. 7 illustrates how we have connected Spot to GreatSPN[3]. GreatSPN can produce a symbolic reachability graph (SRG) for a Colored Petri net by exploiting its symmetries [19]. We have implemented this interface as a `tgba` subclass whose methods simply delegate their work to the corresponding procedures of GreatSPN. From the point of view of Spot, an SRG appears as any other TGBA.

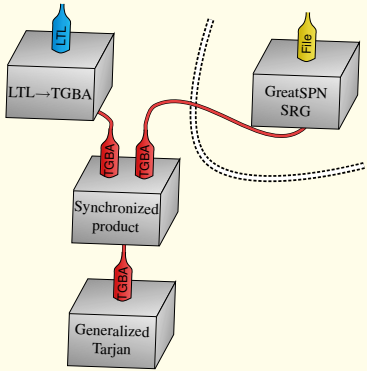


Fig. 7. Interfacing third party state graph generators.

Duret-Lutz and Poitrenaud. *Spot: an extensible model checking library using transition-based generalized Büchi automata*. **MASCOTS'04**. doi

The evolution

1 Model checking library (2003–2012)

- using TGBA for model checking
- pure C++ library
- no tools (except for the test suite)

2 Platform for LTL manip. and model checking (2012–2015)

- tools with LTL/PSL input
- no tool reading TGBA by lack of exchange format

The evolution

1 Model checking library (2003–2012)

- using TGBA for model checking
- pure C++ library
- no tools (except for the test suite)

2 Platform for LTL manip. and model checking (2012–2015)

- tools with LTL/PSL input
- no tool reading TGBA by lack of exchange format

3 Platform for LTL and ω-automata (2016–)

- HOA format
- more tools
- major rewrite
- Jupyter support


Duret-Lutz et al. *Spot 2.0 — a framework for LTL and ω-automata manipulation*. **ATVA'16**. doi


Babiak et al. *The Hanoi Omega-Automata format*. **CAV'15**. doi

The evolution

1 Model checking library (2003–2012)

- using TGBA for model checking
- pure C++ library
- no tools (except for the test suite)

2 Platform for LTL manip. and model checking (2012–2015)

- tools with LTL/PSL input
- no tool reading TGBA by lack of exchange format

3 Platform for LTL and ω-automata (2016–)

- HOA format
- more tools
- major rewrite
- Jupyter support

4 New application: Synthesis

- build on existing features
- improves existing features
- adds games, mealy machines

What I will present

1 Model checking library (2003–2012)

- A bug!

2 Platform for LTL manip. and model checking (2012–2015)

- command-line tools
- SAT-based minimization

3 Platform for LTL and ω-automata (2016–)

- Jupyter visualizations
- Emerson-Lei acceptance conditions

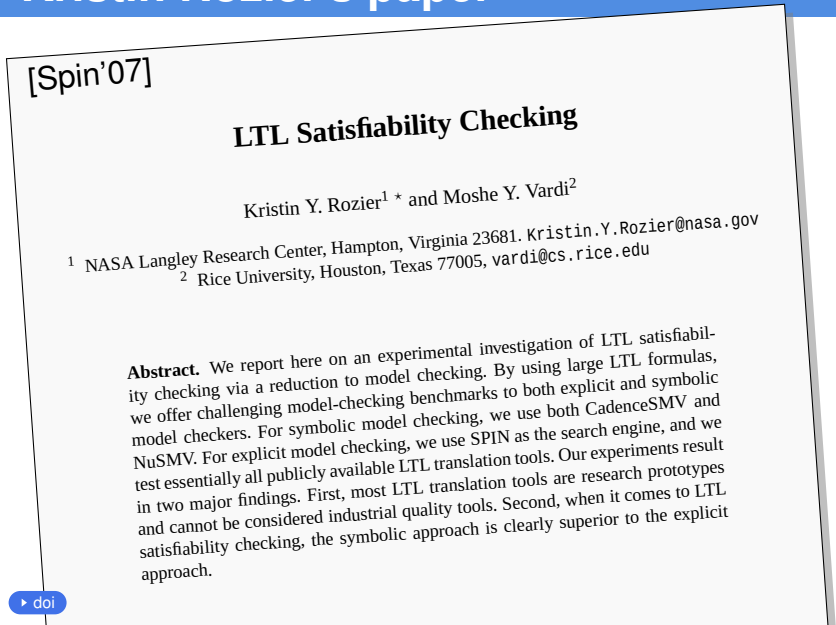
4 New application: Synthesis

- `ltlsynt`, ACD
- `ltlfsynt`, MTDFA

A memorable bug

(Reported in 2007; fixed in 2009.)

Kristin Rozier's paper



[Spin'07]

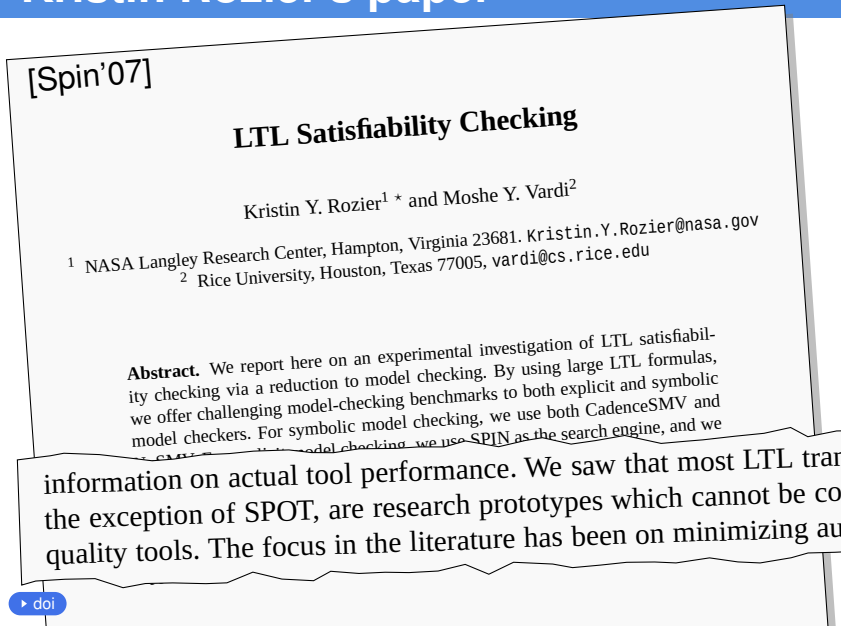
LTL Satisfiability Checking

Kristin Y. Rozier[1] ⋆ and Moshe Y. Vardi[2]

[1] NASA Langley Research Center, Hampton, Virginia 23681. Kristin.Y.Rozier@nasa.gov
[2] Rice University, Houston, Texas 77005, vardi@cs.rice.edu

Abstract. We report here on an experimental investigation of LTL satisfiability checking via a reduction to model checking. By using large LTL formulas, we offer challenging model-checking benchmarks to both explicit and symbolic model checkers. For symbolic model checking, we use both CadenceSMV and NuSMV. For explicit model checking, we use SPIN as the search engine, and we test essentially all publicly available LTL translation tools. Our experiments result in two major findings. First, most LTL translation tools are research prototypes and cannot be considered industrial quality tools. Second, when it comes to LTL satisfiability checking, the symbolic approach is clearly superior to the explicit approach.

Kristin Rozier's paper



[Spin'07]

LTL Satisfiability Checking

Kristin Y. Rozier[1] ⋆ and Moshe Y. Vardi[2]

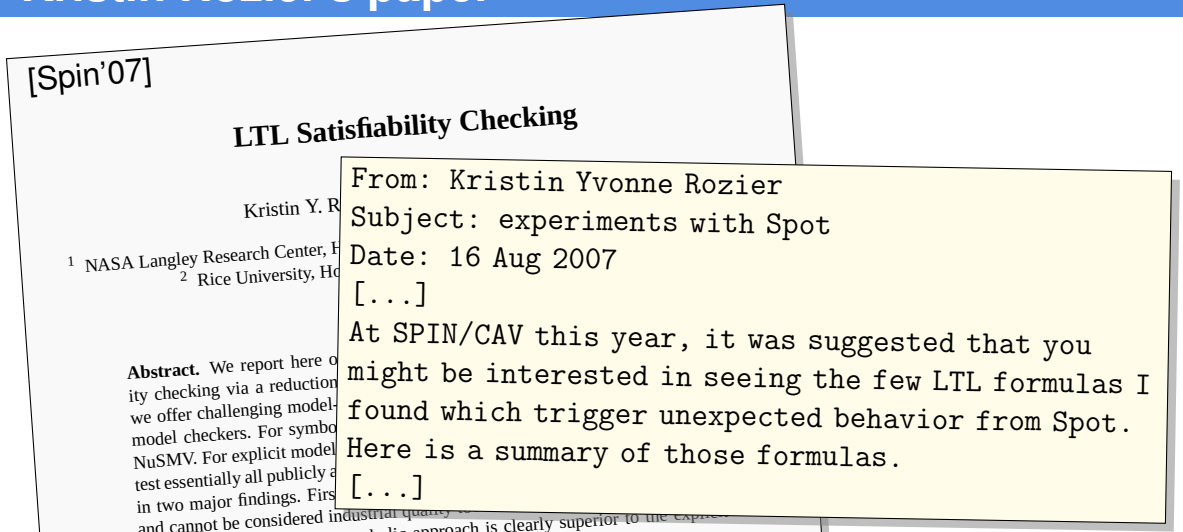
[1] NASA Langley Research Center, Hampton, Virginia 23681. Kristin.Y.Rozier@nasa.gov
[2] Rice University, Houston, Texas 77005, vardi@cs.rice.edu

Abstract. We report here on an experimental investigation of LTL satisfiability checking via a reduction to model checking. By using large LTL formulas, we offer challenging model-checking benchmarks to both explicit and symbolic model checkers. For symbolic model checking, we use both CadenceSMV and

information on actual tool performance. We saw that most LTL translation tools, with the exception of SPOT, are research prototypes which cannot be considered industrial quality tools. The focus in the literature has been on minimizing automata size, rather

doi

Kristin Rozier's paper



Triggering the bug with “counter formulas”

C_n is a 2-variable LTL formula describing a loop over all values of an n-bit counter.

Triggering the bug with “counter formulas”

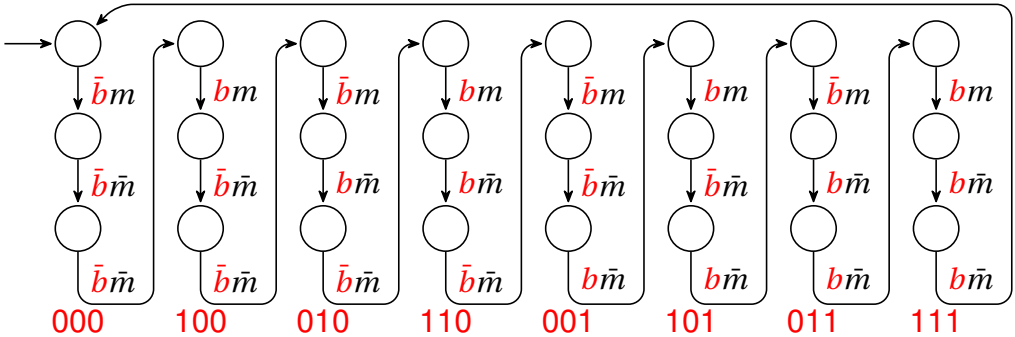
C_n is a 2-variable LTL formula describing a loop over all values of an n-bit counter.

$$C_3 = \neg b \wedge m \wedge \mathsf{G}(m \rightarrow \mathsf{X}(\neg m \wedge \mathsf{X}(\neg m \wedge \mathsf{X}m))) \wedge \mathsf{X}(\neg b \wedge \mathsf{X}\neg b) \wedge \mathsf{G}((\neg b \wedge m) \rightarrow \mathsf{X}(\mathsf{XX}b \wedge ((\neg m \wedge (b \leftrightarrow \mathsf{XXX}b)) \,\mathsf{U}\, m))) \wedge \mathsf{G}((b \wedge m) \rightarrow \mathsf{X}(\mathsf{XX}\neg b \wedge ((b \wedge \neg m \wedge \mathsf{XXX}\neg b) \,\mathsf{U}\, (m \vee (\neg b \wedge \neg m \wedge \mathsf{X}(\mathsf{XX}b \wedge ((\neg m \wedge (b \leftrightarrow \mathsf{XXX}b)) \,\mathsf{U}\, m)))))))$$

Triggering the bug with “counter formulas”

C_n is a 2-variable LTL formula describing a loop over all values of an n-bit counter.

$$C_3 = \neg b \wedge m \wedge \mathsf{G}(m \rightarrow \mathsf{X}(\neg m \wedge \mathsf{X}(\neg m \wedge \mathsf{X}m))) \wedge \mathsf{X}(\neg b \wedge \mathsf{X}\neg b) \wedge \mathsf{G}((\neg b \wedge m) \rightarrow \mathsf{X}(\mathsf{XX}b \wedge ((\neg m \wedge (b \leftrightarrow \mathsf{XXX}b)) \,\mathsf{U}\, m))) \wedge \mathsf{G}((b \wedge m) \rightarrow \mathsf{X}(\mathsf{XX}\neg b \wedge ((b \wedge \neg m \wedge \mathsf{XXX}\neg b) \,\mathsf{U}\, (m \vee (\neg b \wedge \neg m \wedge \mathsf{X}(\mathsf{XX}b \wedge ((\neg m \wedge (b \leftrightarrow \mathsf{XXX}b)) \,\mathsf{U}\, m)))))))$$

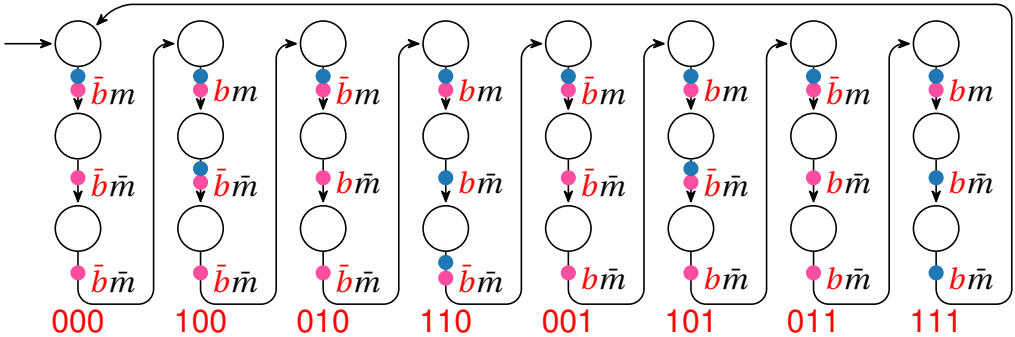


The mininal automaton for C_n has $n2^n$ states.

Triggering the bug with “counter formulas”

C_n is a 2-variable LTL formula describing a loop over all values of an n-bit counter.

$$C_3 = \neg b \wedge m \wedge \mathsf{G}(m \rightarrow \mathsf{X}(\neg m \wedge \mathsf{X}(\neg m \wedge \mathsf{X}m))) \wedge \mathsf{X}(\neg b \wedge \mathsf{X}\neg b) \wedge \mathsf{G}((\neg b \wedge m) \rightarrow \mathsf{X}(\mathsf{XX}b \wedge ((\neg m \wedge (b \leftrightarrow \mathsf{XXX}b)) \,\mathsf{U}\, m))) \wedge \mathsf{G}((b \wedge m) \rightarrow \mathsf{X}(\mathsf{XX}\neg b \wedge ((b \wedge \neg m \wedge \mathsf{XXX}\neg b) \,\mathsf{U}\, (m \vee (\neg b \wedge \neg m \wedge \mathsf{X}(\mathsf{XX}b \wedge ((\neg m \wedge (b \leftrightarrow \mathsf{XXX}b)) \,\mathsf{U}\, m)))))))$$



The mininal automaton for C_n has $n2^n$ states.

Triggering the bug with “counter formulas”

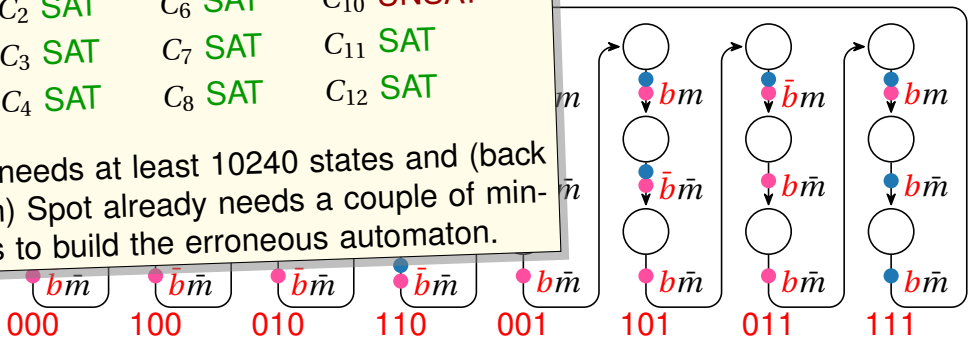
C_n is a 2-variable LTL formula describing a loop over all values of an n-bit counter.

Kristin’s issue with Spot 0.3:

C_1 SAT	C_5 SAT	C_9 SAT
C_2 SAT	C_6 SAT	C_{10} UNSAT
C_3 SAT	C_7 SAT	C_{11} SAT
C_4 SAT	C_8 SAT	C_{12} SAT

C_{10} needs at least 10240 states and (back then) Spot already needs a couple of minutes to build the erroneous automaton.

$n))) \wedge \mathsf{X}(\neg b \wedge \mathsf{X}\neg b) \wedge \mathsf{G}((\neg b \wedge m) \rightarrow$
$\wedge \mathsf{G}((b \wedge m) \rightarrow \mathsf{X}(\mathsf{XX}\neg b \wedge ((b \wedge \neg m \wedge$
$((\neg m \wedge (b \leftrightarrow \mathsf{XXX}b)) \,\mathsf{U}\, m)))))))$



The mininal automaton for C_n has $n2^n$ states.

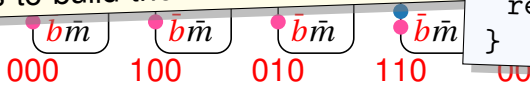
Triggering the bug with “counter formulas”

C_n is a 2-variable LTL formula describing a loop over all values of an n-bit counter.

Kristin’s issue with Spot 0.3:

C_1 SAT	C_5 SAT	C_9 SAT
C_2 SAT	C_6 SAT	C_{10} UNSAT
C_3 SAT	C_7 SAT	C_{11} SAT
C_4 SAT	C_8 SAT	C_{12} SAT

C_{10} needs at least 10240 states and
then) Spot already needs a couple o
utes to build the erroneous automato



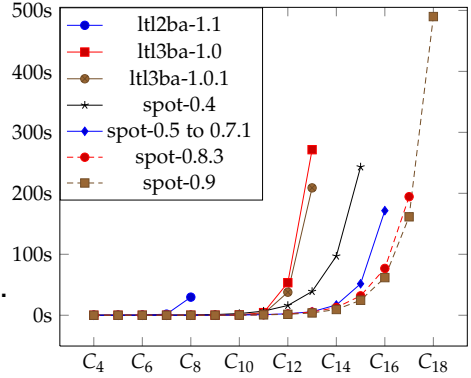
The one-character fix!

```
bool operator<(const formula* left,
               const formula* right)
{
  size_t l = left->hash();
  size_t r = right->hash();
- if (1 != r)
+ if (l != r)
    return l < r;
  return left->dump() < right->dump();
}
```

The mininal automaton for C_n has $n2^n$ states.

The aftermath

- I'm now very picky about the font I use in my terminal and editor. (Iosevka is great! ▸ www)
- Realized that the representation of LTL formulas in Spot 0.4 was really inefficient.
- Divided translation time by ≈ 3 after fixing the bug, by improving formula representations.
- Today C_n can be generated by Spot's `genltl` tool.



▸ benchmark from 2012

Command-line Tools

Command-line Tools

Objectives:

- Easy access to Spot's algorithms
- Providing convenient tools for day-to-day experiments
- Support streaming (a.k.a., pipelining) and scripting (i.e., useful exit codes)
- Have coherent options among tools
- Have good defaults
- Have good error reporting

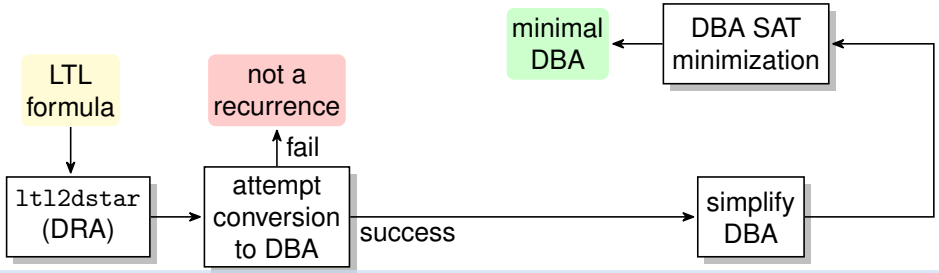
SAT-based minimization of deterministic TGBA

From LTL to Minimal D[T][G]BA

Output: DBA. (Ehlers' setup.)

From LTL to Minimal D[T][G]BA

Output: DBA. (Ehlers' setup.)

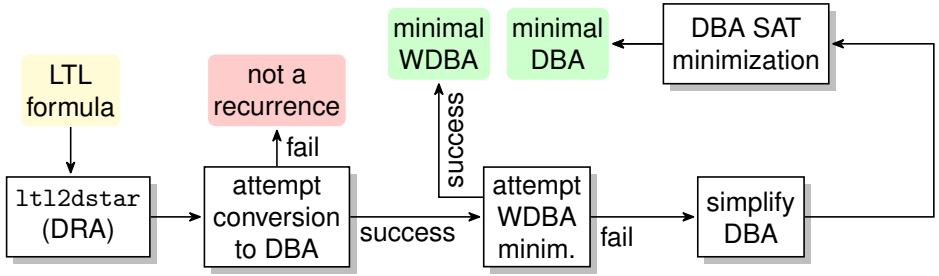


Klein and Baier. *On-the-fly stuttering in the construction of determ. ω-automata*. **CIAA'07**. doi

Krishnan, Puri, and Brayton. *Determ. ω-automata vis-a-vis determ. Büchi automata*. **ISAAC'94**. doi

From LTL to Minimal D[T][G]BA

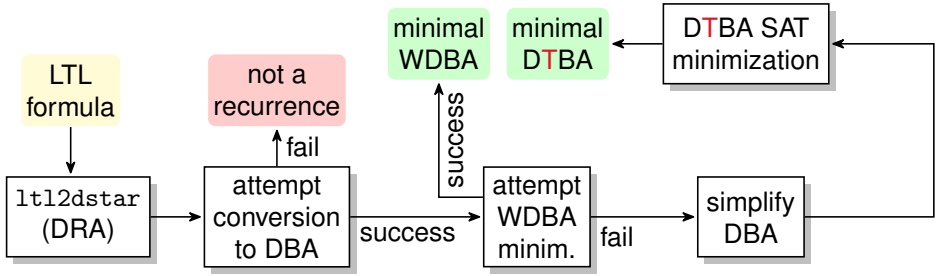
Output: DBA.



Dax, Eisinger, and Klaedtke. *Mechanizing the powerset construction for restricted classes of ω-automata*. **ATVA'07**. doi

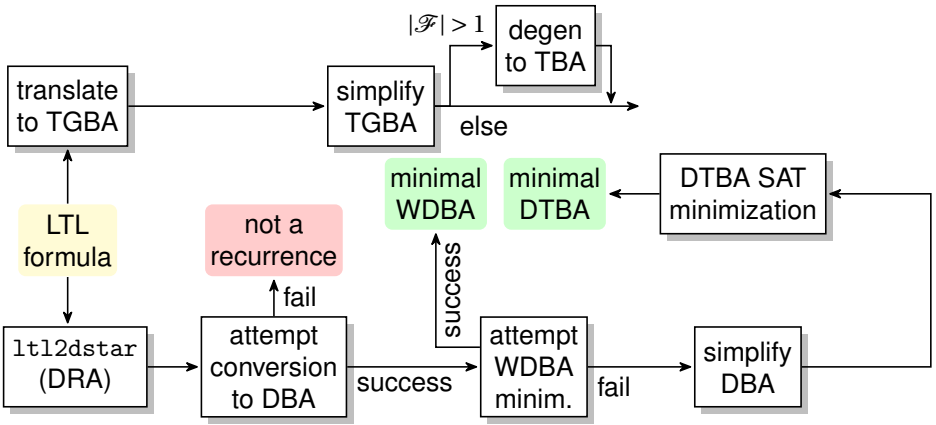
From LTL to Minimal D[T][G]BA

Output: DTBA.



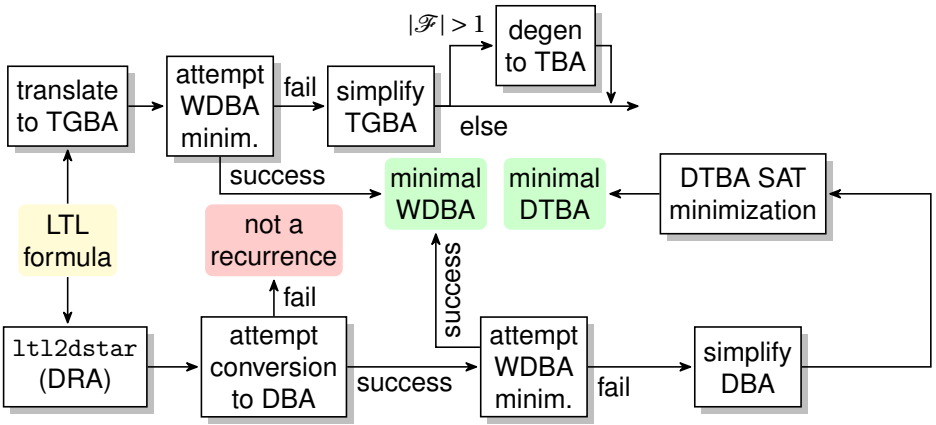
From LTL to Minimal D[T][G]BA

Output: DTBA.



From LTL to Minimal D[T][G]BA

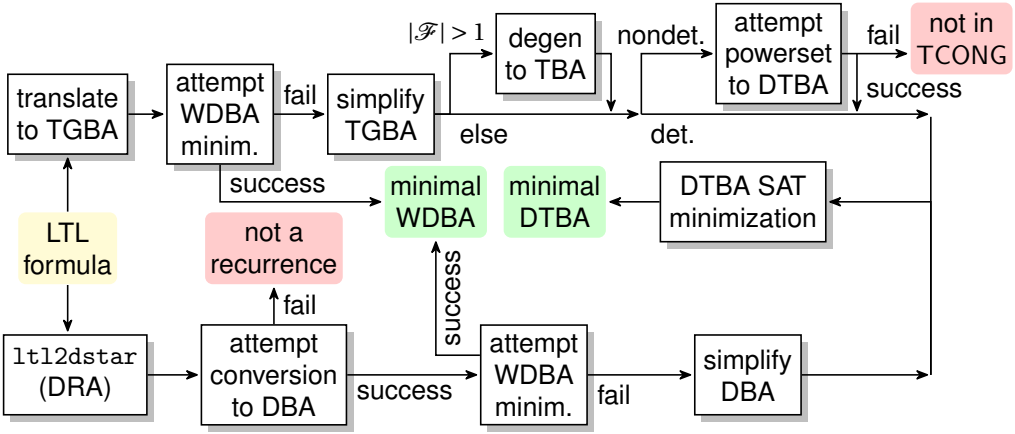
Output: DTBA.



Dax, Eisinger, and Klaedtke. *Mechanizing the powerset construction for restricted classes of ω-automata*. **ATVA'07**. doi

From LTL to Minimal D[T][G]BA

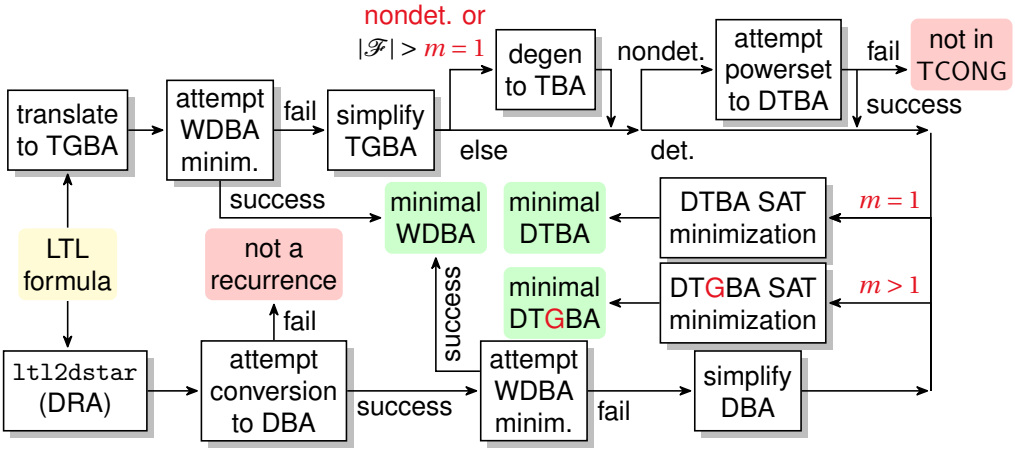
Output: DTBA.



Dax, Eisinger, and Klaedtke. *Mechanizing the powerset construction for restricted classes of ω-automata.* **ATVA'07.** doi

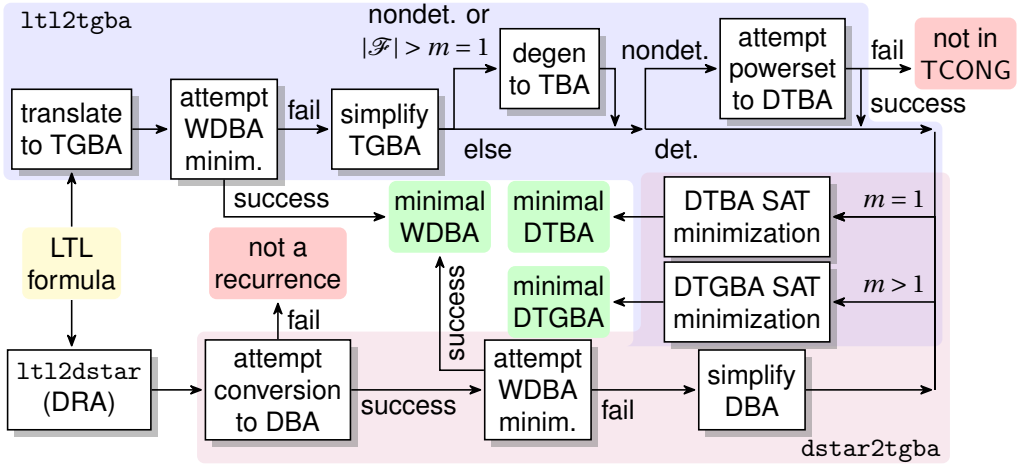
From LTL to Minimal D[T][G]BA

Output: DTGBA ($m > 1$) or DTBA ($m = 1$).



From LTL to Minimal D[T][G]BA

Output: DTGBA ($m > 1$) or DTBA ($m = 1$). Our setup.



Baarir and Duret-Lutz. *Mechanizing the minimization of deterministic generalized Büchi automata.* **FORTE'14.** doi

Spot 2
HOA + Jupyter $\Longrightarrow$ major rewrite

The Hanoi Omega-Automata Format

The Hanoi Omega-Automata Format

Tool support at publication

`ltl2dstar 0.5.3`	`PRISM 4.3`	`jhoafparser`
`ltl3ba 1.1.2`	`Rabinizer 3.1`	`cpphoafparser`
`ltl3dra 0.2.2`	`Spot 1.99.2`	

The Hanoi Omega-Automata Format

Original motivations

- Unify output formats for different tools/acceptance conditions
- Allow **new** acceptance conditions

positive Boolean formulas of $\mathrm{Inf}(x)$ and $\mathrm{Fin}(y)$ terms, a.k.a. Emerson-Lei acceptance

Tool support at publication

`ltl2dstar 0.5.3`
`ltl3ba 1.1.2`
`ltl3dra 0.2.2`
`PRISM 4.3`
`Rabinizer 3.1`
`Spot 1.99.2`
`jhoafparser`
`cpphoafparser`

The Hanoi Omega-Automata Format

Original motivations

- Unify output formats for different tools/acceptance conditions
- Allow **new** acceptance conditions

positive Boolean formulas of $\mathrm{Inf}(x)$ and $\mathrm{Fin}(y)$ terms, a.k.a. Emerson-Lei acceptance

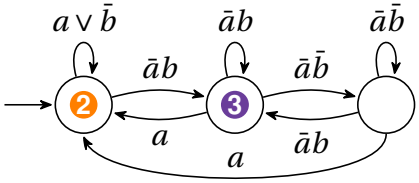
Tool support at publication

`ltl2dstar 0.5.3`
`ltl3ba 1.1.2`
`ltl3dra 0.2.2`
`PRISM 4.3`
`Rabinizer 3.1`
`Spot 1.99.2`
`jhoafparser`
`cpphoafparser`

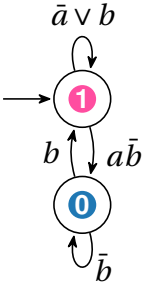
Resulting challenge

Can we build tools that process automata with arbitrary acceptance conditions?

Generic Intersections and Unions

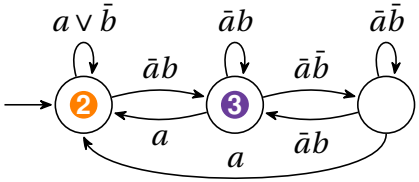


$\mathsf{Fin}(2) \vee \mathsf{Inf}(3)$

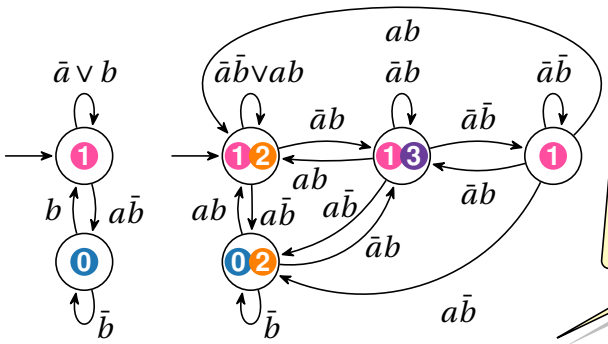


$\mathsf{Fin}(0) \vee \mathsf{Inf}(1)$

Generic Intersections and Unions



$\mathsf{Fin}(2) \vee \mathsf{Inf}(3)$

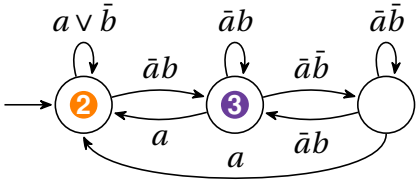


The intersection of two Streett automata is a Streett automaton

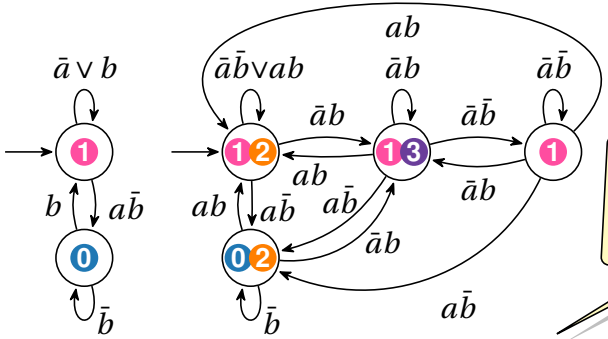
$\mathsf{Fin}(0) \vee \mathsf{Inf}(1)$

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

Generic Intersections and Unions



$\mathsf{Fin}(2) \vee \mathsf{Inf}(3)$

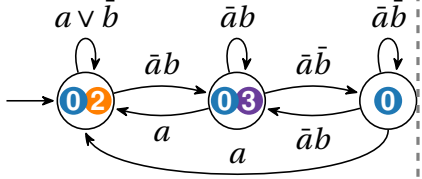


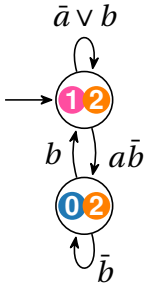
The union of two Streett automata is... nothing special

$\mathsf{Fin}(0) \vee \mathsf{Inf}(1)$

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

Generic Intersections and Unions

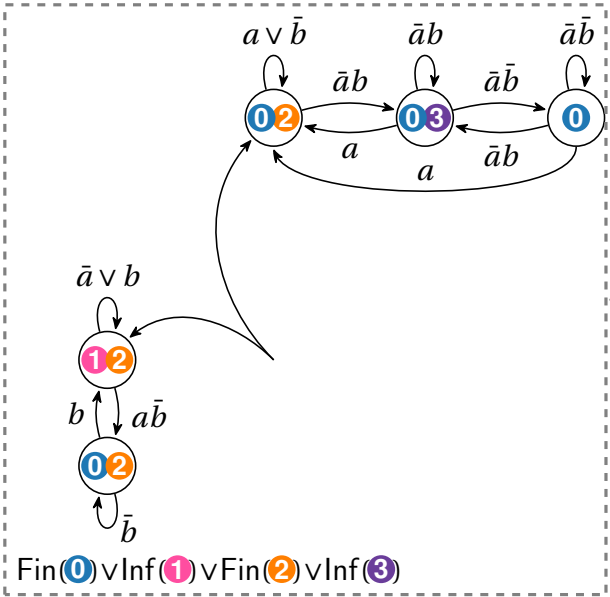




$\mathsf{Fin}(0) \vee \mathsf{Inf}(1) \vee \mathsf{Fin}(2) \vee \mathsf{Inf}(3)$

Union using non-deterministic initial states

Generic Intersections and Unions



Intersection using universal initial states

Other Operations

automata simplifications

Most of Spot's simplifications have been generalized already.

Other Operations

automata simplifications

Most of Spot's simplifications have been generalized already.

complementation

Trivial on any deterministic ω-automata.

What about non-deterministic ω-automata?

Other Operations

automata simplifications

Most of Spot's simplifications have been generalized already.

complementation

Trivial on any deterministic ω-automata.

What about non-deterministic ω-automata?

emptiness check

Easy for "Fin-less acceptance".

Generic Emptiness Check implemented since Spot 2.7.

Other Operations

automata simplifications

Most of Spot's simplifications have been generalized already.

complementation

Trivial on any deterministic ω-automata.

What about non-deterministic ω-automata?

emptiness check

Easy for "Fin-less acceptance".

Generic Emptiness Check implemented since Spot 2.7.

acceptance conversions

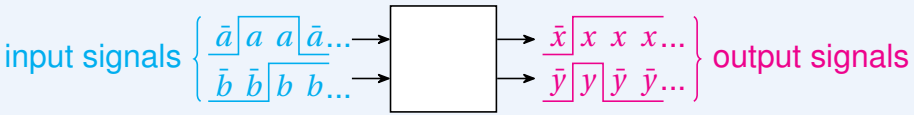
Many are implemented.

(Jupyter demo)


Reactive Synthesis

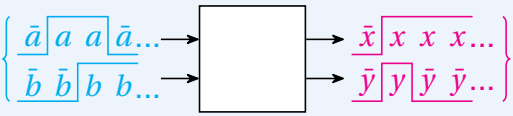
Reactive Synthesis in a Nutshell

A **reactive controller** produces output as a reaction to its input



Reactive Synthesis in a Nutshell

A **reactive controller** produces output as a reaction to its input

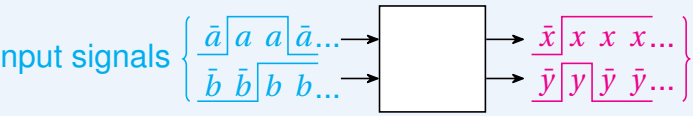


The reactive synthesis problems

Given a specification relating input signals and output signals over time:
Realizability: decide if a controller exist; **Synthesis:** construct it.

Reactive Synthesis in a Nutshell

A **reactive controller** produces output as a reaction to its input



The reactive synthesis problems

Given a specification relating input signals and output signals over time:
Realizability: decide if a controller exist; **Synthesis:** construct it.

LTL Synthesis Competition ▸ www

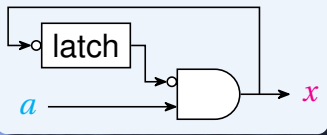
- the specification is an LTL formula, over ω-words such as "$\bar{a}\bar{b}\bar{x}\bar{y}; a\bar{b}xy; abx\bar{y}; \ldots$"
- the controller should be an **And-Inverter Graph**

Reactive Synthesis Example

1. LTL Spec.

$a \leftrightarrow \mathsf{F}(x)$

6. Output AIG



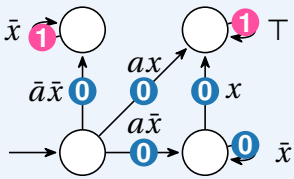
Reactive Synthesis Example

1. LTL Spec.

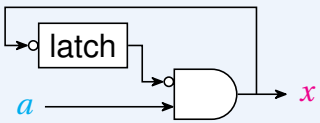
$a \leftrightarrow \mathsf{F}(x)$

2. DPA

$\mathsf{Fin}(0) \vee \mathsf{Inf}(1)$



6. Output AIG

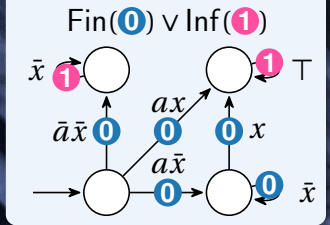


Reactive Synthesis Example

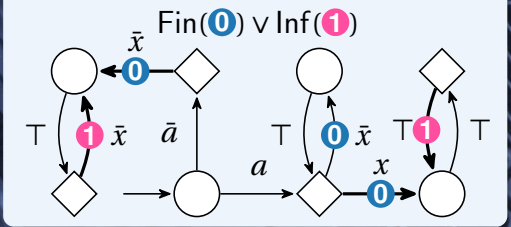
1. LTL Spec.

$a \leftrightarrow \mathsf{F}(x)$

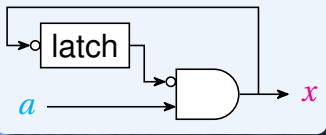
2. DPA



3. Parity Game



6. Output AIG

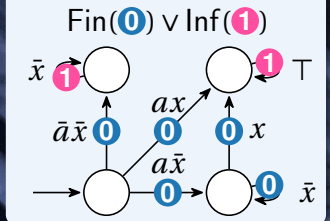


Reactive Synthesis Example

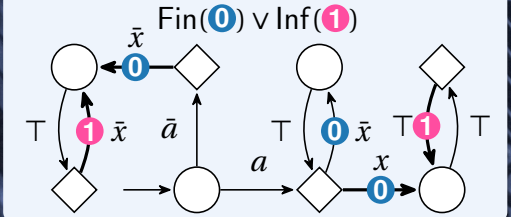
1. LTL Spec.

$a \leftrightarrow \mathsf{F}(x)$

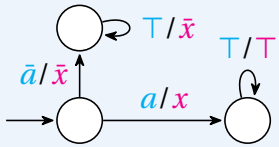
2. DPA



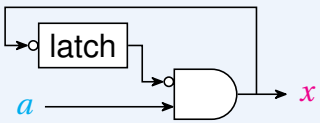
3. Parity Game



4. Winning Strategy = Mealy



6. Output AIG

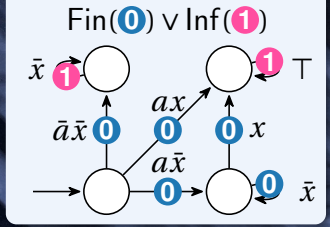


Reactive Synthesis Example

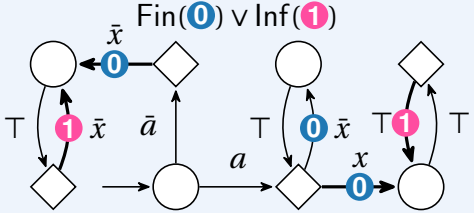
1. LTL Spec.

$a \leftrightarrow \mathsf{F}(x)$

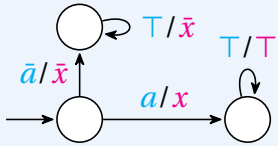
2. DPA



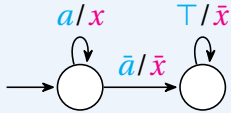
3. Parity Game



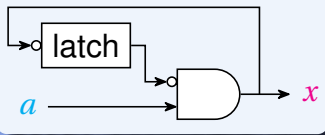
4. Winning Strategy = Mealy



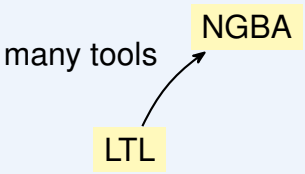
5. Simplify Mealy



6. Output AIG

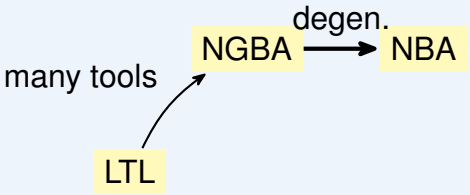


How to turn an LTL formula into a DPA?



DPA

How to turn an LTL formula into a DPA?

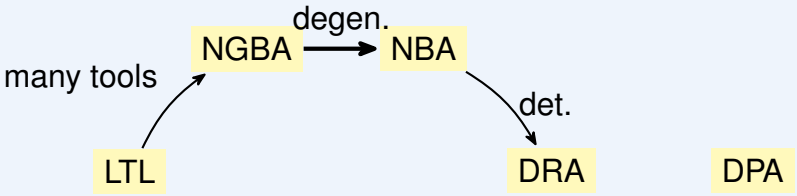


DPA

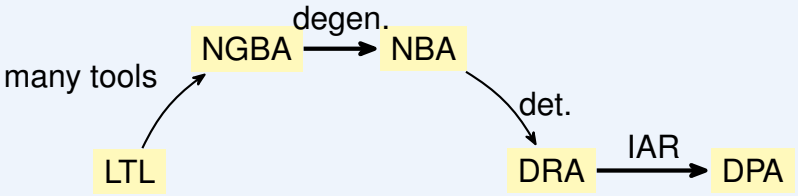
Gastin and Oddoux. *Fast LTL to Büchi automata translation*. **CAV'01**. doi

Babiak et al. *Compositional approach to suspension and other improvements to LTL translation*. **SPIN'13**. doi

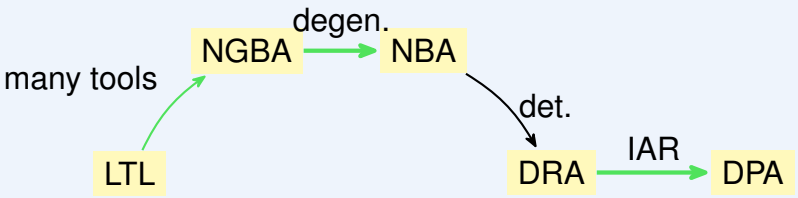
How to turn an LTL formula into a DPA?



How to turn an LTL formula into a DPA?

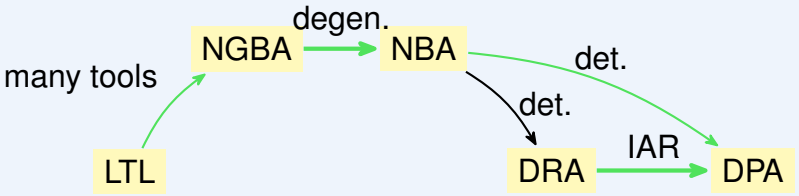


How to turn an LTL formula into a DPA?



Procedures implemented in Spot.

How to turn an LTL formula into a DPA?

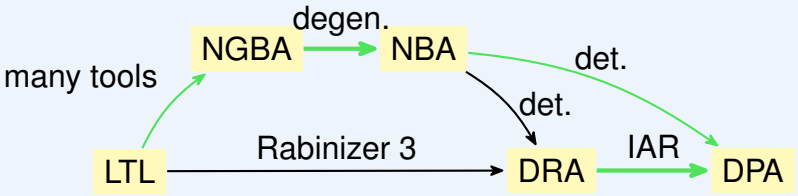


Procedures implemented in Spot.

Piterman. *From nondeterministic Büchi and Streett automata to deterministic parity automata.* **LICS'06.** doi

Redziejowski. *An improved construction of deterministic omega-automaton using derivatives.* **Fundamenta Informaticae**, 2012. doi

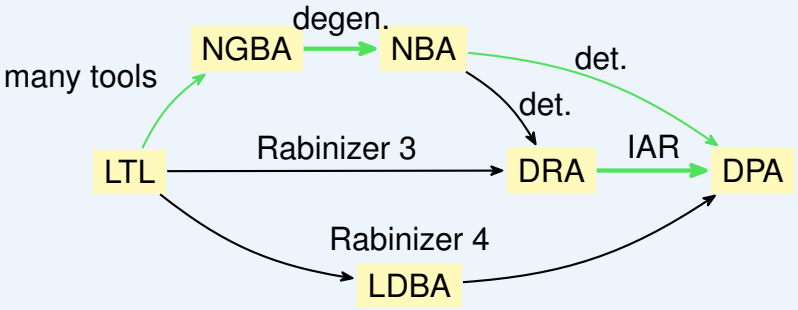
How to turn an LTL formula into a DPA?



Procedures implemented in Spot.

Komárková and Křetínský. *Rabinizer 3: Safraless translation of LTL to small deterministic automata*. **ATVA'14**. doi

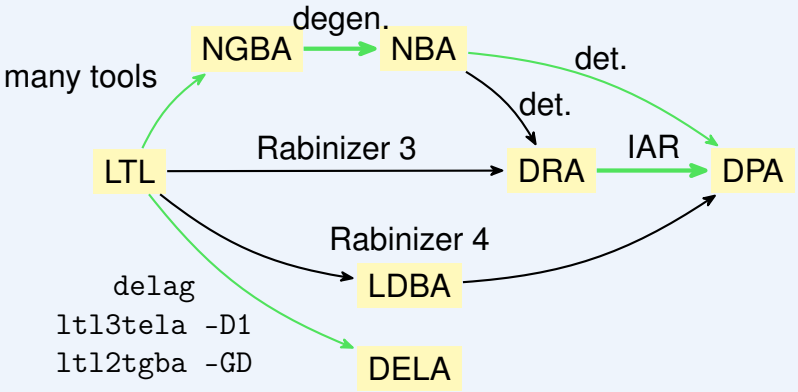
How to turn an LTL formula into a DPA?



Procedures implemented in Spot.

Esparza et al. *From LTL and limit-deterministic Büchi automata to deterministic parity automata.* TACAS'17. doi

How to turn an LTL formula into a DPA?

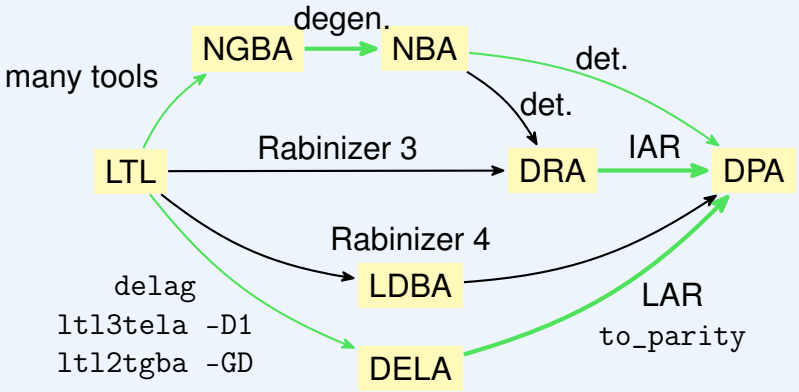


Procedures implemented in Spot.

Müller and Sickert. *LTL to deterministic Emerson-Lei automata*. **GandALF'17**. doi

Major et al. *ltl3tela: LTL to small deterministic or nondeterministic Emerson-Lei automata*. **ATVA'19**. doi

How to turn an LTL formula into a DPA?

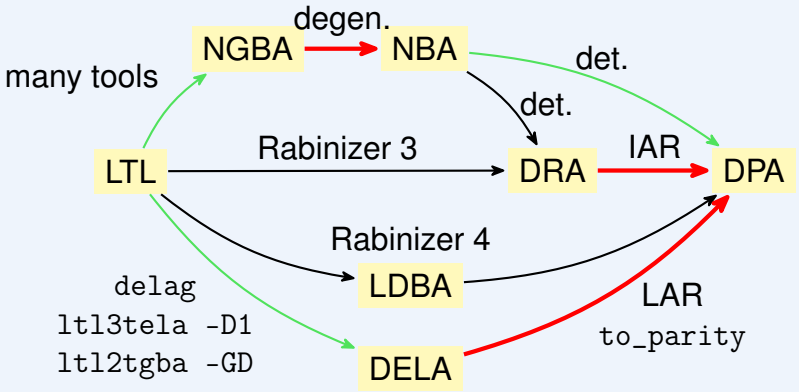


Procedures implemented in Spot.

Renkin, Duret-Lutz, and Pommellet. *Practical "paritizing" of Emerson-Lei automata.* **ATVA'20**. doi

Löding. *Optimal bounds for transformations of ω-automata.* **FSTTCS'99**. doi

How to turn an LTL formula into a DPA?



Procedures implemented in Spot.

These paritization procedures are all improved by ACD!

Casares, Colcombet, and Fijalkow. *Optimal transformations of games and automata using Muller conditions*. **ICALP'21**. doi

Casares et al. *Practical applications of the Alternating Cycle Decomposition*. **TACAS'22**. doi

(Jupyter demo)

What I did present

1 Model checking library (2003–2012)

- A bug!

2 Platform for LTL manip. and model checking (2012–2015)

- command-line tools
- SAT-based minimization

3 Platform for LTL and ω-automata (2016–)

- Jupyter visualizations
- Emerson-Lei acceptance conditions

4 New application: Synthesis

- `ltlsynt`, ACD
- `ltlfsynt`, MTDFA