Contributions to LTL and $\omega$-Automata for Model Checking

Alexandre Duret-Lutz
LRDE/EPITA
10 February 2017

Javier Esparza  Technische Universität München  reviewer
Radu Mateescu  INRIA Grenoble  reviewer
Moshe Y. Vardi  Rice University, Houston, Texas  reviewer
Rüdiger Ehlers  Universität Bremen  examiner
Stephan Merz  INRIA Nancy & LORIA  examiner
Jaco van de Pol  University of Twente  examiner
Fabrice Kordon  Univ. Pierre & Marie Curie, Paris  examiner
Live demo

In [62]: @interact(n=IntSlider(1, 1, 150, 5))
   def uncover_state_space(n):
       return lift_display(k, n)

   n  1

...  

In [63]: spot.stats_reachable(k).states

Out[63]: 352

In [64]: def model_check(model, f):
   f = spot.formula(f)
   ss = model.kripke(spot.atomic_prop_collect(f))
   nf = spot.formula_Not(f).translate()
   return ss.intersecting_run(nf)

In [65]: lift_display(model_check(m, 'G("req[1]" -> F("p==1" && "cabin.open")'))

Out[65]:

...
Automata-Theoretic LTL Model Checking

High-level model $M$

State-space generation

State-space automaton $A_M$

Synchronized product

$L(A_M \otimes A_{\neg \varphi}) = L(A_M) \cap L(A_{\neg \varphi})$

Negated property automaton $A_{\neg \varphi}$

Product automaton $A_M \otimes A_{\neg \varphi}$

Emptiness check

$L(A_M \otimes A_{\neg \varphi}) \neq \emptyset$

LTL property $\varphi$

LTL translation

M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. LICS’86
Automata-Theoretic LTL Model Checking

High-level model \( M \)

State-space generation

State-space automaton \( A_M \)

Product automaton \( A_M \otimes A_{\neg \varphi} \)

Synchronized product

\[ \mathcal{L}(A_M \otimes A_{\neg \varphi}) = \mathcal{L}(A_M) \cap \mathcal{L}(A_{\neg \varphi}) \]

Emptiness check

\[ \mathcal{L}(A_M \otimes A_{\neg \varphi}) \neq \emptyset \]

LTL property \( \varphi \)

LTL translation

Negated property automaton \( A_{\neg \varphi} \)

\( M \models \varphi \) or counterexample

M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *LICS’86*
Automata-Theoretic LTL Model Checking

- **High-level model** $M$
- **On-the-fly** generation of state-space automaton $A_M$
- **On-the-fly** synchronized product
  \[ L(A_M \otimes A_{\neg \varphi}) = L(A_M) \cap L(A_{\neg \varphi}) \]
- **Emptiness check**
  \[ L(A_M \otimes A_{\neg \varphi}) \neq \emptyset \]
- **LTL property** $\varphi$
- **LTL translation**
- **Negated property automaton** $A_{\neg \varphi}$
- **M. Y. Vardi and P. Wolper.** An automata-theoretic approach to automatic program verification. *LICS’86*
Automata-Theoretic LTL Model Checking

Custom Model Checker

Option 1: Translate your model to feed some existing model checker.
Option 2: Write a model checker from scratch.
Option 3: Adapt a model checker to your formalism.
Option 4: Use a model-checking framework designed for extensibility.

High-level model $M$

On-the-fly generation of state-space automaton $A_M$

On-the-fly synchronized product $L(A_M \otimes A_{\neg \varphi}) = L(A_M) \cap L(A_{\neg \varphi})$

Emptiness check $L(A_M \otimes A_{\neg \varphi}) \neq \emptyset$

Negated property automaton $A_{\neg \varphi}$

LTL property $\varphi$

LTL translation

MASCOTS’04

Motivation: Supporting Research

Spot should offer a set of **efficient** and **reusable** blocks for model checking and **related tasks**.
Motivation: Supporting Research

Spot should offer a set of **efficient** and **reusable** blocks for model checking and **related tasks**.

**Efficient:**
- Implement state-of-the-art algorithms
- Improve them
- Propose new algorithms

**Reusable:**
- Multiple interfaces (C++/Python/Shell)
- Documented
- Tested

**Related tasks:**
- LTL and \( \omega \)-automata toolbox
- Glue between third-party tools
Spot should offer a set of **efficient** and **reusable** blocks for model checking and **related tasks**.

Efficient:
- Implement state-of-the-art algorithms
- Improve them
- Propose new algorithms

Reusable:
- Multiple interfaces (C++/Python/Shell)
- Documented
- Tested

Related tasks:
- LTL and $\omega$-automata toolbox
- Glue between third-party tools
Contributions

- High-level model $M$
- On-the-fly generation of state-space automaton $A_M$

On-the-fly synchronized product
$$L(A_M \otimes A_{\neg \phi}) = L(A_M) \cap L(A_{\neg \phi})$$

Emptiness check
$$L(A_M \otimes A_{\neg \phi}) \neq \emptyset$$

- LTL property $\varphi$
- LTL translation
- Negated property automaton $A_{\neg \varphi}$

- $M \models \varphi$ or counterexample
Contributions

**High-level model** $M$

**On-the-fly** generation of state-space automaton $A_M$

**Plugging Spot with various tools**

**On-the-fly** synchronized product

$L(A_M \otimes A_{\neg \varphi}) = L(A_M) \cap L(A_{\neg \varphi})$

**Emptiness check**

$(A_M \otimes A_{\neg \varphi}) \neq \emptyset$

**Testing automata**

**Generic acceptance**

**Decomposition**

**Union-find**

**Parallelization**

**Stutter checks**

**Simplifications**

**Classification**

**Many improvements**

**PSL translation**

**Negated property automaton** $A_{\neg \varphi}$

**SAT-based minimization**

**LTL property $\varphi$**

**LTL translation**

**Hybrid model checking**

**Provisos**

**Dynamic and on-the-fly generation** of an automaton $D$ such that

$L(D) = \emptyset \iff L(A_M \otimes A_{\neg \varphi}) = \emptyset$
Contributions

High-level model $M$

On-the-fly generation of state-space automaton $A_M$

Option 1 Translate your model to feed some existing model checker.
Option 2 Write a model checker from scratch.
Option 3 Adapt a model checker to your formalism.
Option 4 Use a model-checking framework designed for extensibility.

On-the-fly state-space generation

On-the-fly generation of state-space automaton $A_M$

Negated property automaton $A_{\neg \varphi}$

LTL property $\varphi$

LTL translation

SAT-based minimization

Stutter checks

Simplifications

Classification

Many improvements

PSL translation

Hybrid model checking

Provisos

Emptiness check

Generic acceptance

Decomposition

Testing automata

Parallelization

Union-find

Plugging Spot with various tools

SAT-based minimization

5 / 27
Context & Motivation

LTL to Büchi

Generalized Acceptance

Tooling for improvement

Closing remarks

Spot has a very good translator, combining several improved procedures.

Named acceptances are a hindrance. Generic algorithms are more elegant.

Spot: groundwork for research + tools for experimenting, testing, finding interesting cases.
Büchi Variations on $G F a \land G F b$
Büchi Variations on $G F a \land G F b$

Büchi

\[
\begin{align*}
&\overset{\bar{a}}{1} & \overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} & \overset{ab}{2} & \overset{ab}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{\bar{b}}{0} \\
&\overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} & \overset{\bar{b}}{2} & \overset{ab}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{\bar{a}}{1} & \overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} \\
&\overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} & \overset{\bar{b}}{2} & \overset{ab}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{\bar{a}}{1} & \overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} \\
&\overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} & \overset{\bar{b}}{2} & \overset{ab}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{\bar{a}}{1} & \overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} \\
&\overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} & \overset{\bar{b}}{2} & \overset{ab}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{\bar{a}}{1} & \overset{\bar{a}b}{\overset{a}{\rightleftharpoons}} \\
\end{align*}
\]

$\text{Inf}(\text{0})$

generalized Büchi

\[
\begin{align*}
&\overset{\bar{a}}{1} & \overset{\bar{a}}{0} & \overset{\bar{b}}{1} & \overset{\bar{b}}{0} \\
&\overset{\bar{b}}{0} & \overset{a}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{b}{0} & \overset{\bar{b}}{1} \\
&\overset{\bar{b}}{0} & \overset{a}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{b}{0} & \overset{\bar{b}}{1} \\
&\overset{\bar{b}}{0} & \overset{a}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{b}{0} & \overset{\bar{b}}{1} \\
&\overset{\bar{b}}{0} & \overset{a}{\overset{\bar{b}}{\rightleftharpoons}} & \overset{b}{0} & \overset{\bar{b}}{1} \\
\end{align*}
\]

$\text{Inf}(\text{0}) \land \text{Inf}(\text{1})$

Only useful when mixing accepting & rejecting cycles.
Büchi Variations on $G F a \land G F b$

### Büchi

- **State-based**
  - Initial State: 0
  - Transitions:
    - $\bar{a}$ from 1 to 2
    - $\bar{ab}$ from 2 to 0
    - $ab$ from 2 to 1
    - $\bar{b}$ from 0 to 2
    - $\bar{b}$ from 2 to 0
  - $\text{Inf}(0)$

- **Transition-based**
  - Initial State: 0
  - Transitions:
    - $\bar{a}$ from 1 to 0
    - $a$ from 0 to 1
    - $b$ from 1 to 0
    - $\bar{b}$ from 0 to 1
  - $\text{Inf}(0)$

### Generalized Büchi

- **State-based**
  - Initial State: 0
  - Transitions:
    - $\bar{a}$ from 1 to 0
    - $a$ from 0 to 1
    - $b$ from 1 to 0
    - $\bar{b}$ from 0 to 1
  - $\text{Inf}(0) \land \text{Inf}(1)$

- **Transition-based**
  - Initial State: 0
  - Transitions:
    - $a \bar{b}$ from 0 to 1
    - $b \bar{a}$ from 1 to 0
    - $\bar{a} \bar{b}$ from 0 to 1
  - $\text{Inf}(0) \land \text{Inf}(1)$
Büchi Variations on $GF_a \land GF_b$

**Büchi**

- State-based
  - Transition diagram:
    - States: 1, 2, 0
    - Transitions: $\bar{a}$, $\bar{ab}$, $ab$, $a$, $b$
    - Equation: $\text{Inf}(0)$

- Transition-based
  - Transition diagram:
    - States: 1, 0
    - Transitions: $\bar{a}$, $a$, $b$
    - Equation: $\text{Inf}(0)$

**Generalized Büchi**

- State-based
  - Transition diagram:
    - States: 1, 0
    - Transitions: $\bar{a}$, $a$, $b$
    - Equation: $\text{Inf}(0) \land \text{Inf}(1)$

- Transition-based
  - Transition diagram:
    - States: 1, 0
    - Transitions: $\bar{ab}$, $ab$
    - Equation: $\text{Inf}(0) \land \text{Inf}(1)$

*Only useful when mixing accepting & rejecting cycles*
## Comparison of Some “LTL to Büchi” Translators

Results summed over 178 formulas from the literature.

<table>
<thead>
<tr>
<th>Translator</th>
<th>nd. time</th>
<th>automaton size</th>
<th>product size</th>
</tr>
</thead>
<tbody>
<tr>
<td>spin (11×☠️)</td>
<td>162 220.7s 1440 1236 46033</td>
<td>259313 9433430</td>
<td></td>
</tr>
<tr>
<td>ltl2ba</td>
<td>169 0.3s 1000 801 29974</td>
<td>190898 5616566</td>
<td></td>
</tr>
<tr>
<td>modella</td>
<td>109 18.5s 1244 577 23474</td>
<td>210494 4033414</td>
<td></td>
</tr>
<tr>
<td>trans</td>
<td>119 0.5s 957 398 16798</td>
<td>172246 3276714</td>
<td></td>
</tr>
<tr>
<td>ltl3ba</td>
<td>115 0.7s 829 307 14322</td>
<td>155220 2913043</td>
<td></td>
</tr>
<tr>
<td>ltl2tgba -s</td>
<td>49 1.9s 666 102 10346</td>
<td>129419 2399328</td>
<td></td>
</tr>
<tr>
<td>ltl2tgba -Ds</td>
<td>44 1.9s 671 96 10456</td>
<td>129804 2401471</td>
<td></td>
</tr>
</tbody>
</table>
From LTL to Büchi Automata

LTL form.

LTL rewritings

Core translation

Post-processings

Büchi automaton

\( G(Fa \land Fb) \)

\( G(Fa \land Fb) \)

\( \text{Inf}(0) \land \text{Inf}(1) \)

\( \text{Inf}(0) \land \text{Inf}(1) \)

\( \text{Inf}(0) \land \text{Inf}(1) \)

Lots of rewritings (e.g., \( f U G f \equiv G f \))

Implication-based rewritings (e.g., if \( f \rightarrow g \) then \( f U g \equiv g \))

Syntactic or automata-based

Couvreur’s translation, plus:

Improved determinism

Improved translation of persistent formulas

Improved translation of \( G \)-subformulas

\( \text{TGBA} \)

\( \text{SCC} \)

Simpl.

Fwd/bwd simul.

Degen.

Fwd/bwd simul.

BA

determinize and minimize

Obligation properties

Remove:

- Useless SCCs
- Useless acc. sets
- BDD signatures
- Improves det.

SCC-aware degeneralization

Secret weapon — only implemented in Spot!

Requires: product, emptiness check, DBA complementation, NFA determinization, DFA minimization, SCC enumeration...
From LTL to Büchi Automata

LTL form. LTL rewritings Core translation Post-processings Büchi automaton

- lots of rewritings (e.g. \( f U G f \equiv G f \))
- implication-based rewritings (e.g., if \( f \rightarrow g \) then \( f U g \equiv g \))
syntactic or automata-based
From LTL to Büchi Automata

LTL form. ➔ LTL rewritings ➔ Core translation ➔ Post-processings ➔ Büchi automaton

Couvreur’s translation, plus:
- Improved determinism
- Improved translation of persistent formulas
- Improved translation of $G$-subformulas

J.-M. Couvreur. On-the-fly verification of temporal logic. *FM’99*

From LTL to Büchi Automata

- LTL form
  - LTL rewritings
  - Core translation
  - Post-processings
  - Büchi automaton

- TGBA
  - SCC simpl.

- BA
  - forward/backward simulation
  - determinize and minimize obligation properties

- BA
  - best

- Secret weapon — only implemented in Spot!
From LTL to Büchi Automata

- LTL form.
- LTL rewritings
- Core translation
- Post-processings
- Büchi automaton

- TGBA
- SCC simpl.

- fwd/bwd simul.
- degen.
- fwd/bwd simul.

- determinize and minimize obligation properties

- Remove:
  - useless SCCs
  - useless acc. sets

- BDD signatures
- improves det.

- SCC-aware degeneralization

T. Babiak, T. Badie, A. Duret-Lutz, M. Křetínský, and J. Strejček. Compositional approach to suspension and other improvements to LTL translation. *SPIN’13*
From LTL to Büchi Automata

LTL form. \rightarrow \quad LTL rewritings \rightarrow \quad Core translation \rightarrow \quad Post-processings \rightarrow \quad Büchi automaton

TGBA \rightarrow \quad SCC simpl. \rightarrow \quad fwd/bwd simul. \rightarrow \quad degen. \rightarrow \quad fwd/bwd simul. \rightarrow \quad best \rightarrow \quad BA

determinize and minimize obligation properties

Secret weapon — only implemented in Spot!


C. Dax, J. Eisinger, and F. Klaedtke. Mechanizing the powerset construction for restricted classes of $\omega$-automata. ATVA’07
From LTL to Büchi Automata

LTL form. \rightarrow LTL rewritings \rightarrow Core translation \rightarrow Post-processings \rightarrow Büchi automaton

TGBA \rightarrow SCC simpl. \rightarrow fwd/bwd simul. \rightarrow degen. \rightarrow fwd/bwd simul. \rightarrow best \rightarrow BA

determinize and minimize obligation properties

Secret weapon — only implemented in Spot!
Requires: product, emptiness check, DBA complementation, NFA determinization, DFA minimization, SCC enumeration...
The Temporal Hierarchy

<table>
<thead>
<tr>
<th>Deterministic Büchi</th>
<th>Weak Büchi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recurrence</td>
<td>Persistence</td>
</tr>
<tr>
<td>$\forall GF p_i \lor FG q_i$</td>
<td>$FG p$</td>
</tr>
<tr>
<td>Safety</td>
<td>Guarantee</td>
</tr>
<tr>
<td>$GF p$</td>
<td>$FP$</td>
</tr>
</tbody>
</table>

Reactivity

Obligation

Safety

Guarantee

Transition-based and generalized Büchi acceptance useful. Compare:

$\inf(0) \lor \inf(1)$

Automata for $GF a \land GF b$.

More complex acceptance conditions (Rabin, Streett, Parity, generalized-Rabin, etc.) compete here.

Z. Manna and A. Pnueli. A hierarchy of temporal properties. *PODC’90*

I. Černá and R. Pelánek. Relating hierarchy of temporal properties to model checking. *MFCS’03*
Comparison of Some “LTL to Büchi” Translators

Results summed over 178 formulas from the literature.

<table>
<thead>
<tr>
<th>Translator</th>
<th>nd.</th>
<th>time</th>
<th>automaton size</th>
<th>product size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nd.</td>
<td>time</td>
<td>st.</td>
<td>nd.st.</td>
</tr>
<tr>
<td>spin (11×)</td>
<td>162</td>
<td>220.7s</td>
<td>1440</td>
<td>1236</td>
</tr>
<tr>
<td>ltl2ba</td>
<td>169</td>
<td>0.3s</td>
<td>1000</td>
<td>801</td>
</tr>
<tr>
<td>modella</td>
<td>109</td>
<td>18.5s</td>
<td>1244</td>
<td>577</td>
</tr>
<tr>
<td>trans</td>
<td>119</td>
<td>0.5s</td>
<td>957</td>
<td>398</td>
</tr>
<tr>
<td>ltl3ba</td>
<td>115</td>
<td>0.7s</td>
<td>829</td>
<td>307</td>
</tr>
<tr>
<td>ltl2tgba -s</td>
<td>49</td>
<td>1.9s</td>
<td>666</td>
<td>102</td>
</tr>
<tr>
<td>ltl2tgba -Ds</td>
<td>44</td>
<td>1.9s</td>
<td>671</td>
<td>96</td>
</tr>
</tbody>
</table>

Spot
The Temporal Hierarchy

Reactivity
\( \wedge G F p_i \lor F G q_i \)

25

Recurrence
\( G F p \)

10

Persistence
\( F G p \)

25

Obligation
\( \wedge G p_i \lor F q_i \)

8

Safety
\( G p \)

52

Guarantee
\( F p \)

52

Weak Büchi

Deterministic Büchi

Terminal Büchi

10%

6%

Z. Manna and A. Pnueli. A hierarchy of temporal properties. *PODC’90*

I. Černá and R. Pelánek. Relating hierarchy of temporal properties to model checking. *MFCS’03*
The Temporal Hierarchy

- **Reactivity**: $\wedge GFp_i \lor FGq_i$
- **Recurrence**: $GFp$
- **Persistence**: $FGp$
- **Obligation**: $\wedge Gp_i \lor Fq_i$
- **Safety**: $Gp$
- **Guarantee**: $Fp$
- **Deterministic Büchi**
- **Weak Büchi**
- **Terminal Büchi**

- 25 cases of Recurrence
- 10 cases of Reactivity
- 25 cases of Persistence
- 8 cases of Obligation
- 52 cases of Safety
- 52 cases of Guarantee
- 6 cases of Monitor

66% of the 178 formulas are obligations

Good for the secret weapon!
66% of the 178 formulas are obligations

80% are persistences
Good for the improved translation!

66% of the 178 formulas are obligations
Good for the secret weapon!
Büchi Variations on $GF\ a \land GF\ b$

**Büchi**

- State-based
  - Transition labeled $\bar{a}$
  - Transition labeled $\bar{a}b$
  - Transition labeled $ab$
  - Transition labeled $a$
  - Transition labeled $\bar{b}$
  - Transition labeled $ab$
  - Transition labeled $\bar{b}$

**generalized Büchi**

- Transition labeled $\bar{a}$
- Transition labeled $a$
- Transition labeled $\bar{b}$
- Transition labeled $b$

- Only useful when mixing accepting & rejecting cycles

- Inf($0$) ∧ Inf($1$)
- Inf($0$)
- Inf($0$)
The Temporal Hierarchy

Reactivity
\[ \land G F p_i \lor F G q_i \]

Persistency
\[ \lor G F p_i \land F q_i \]

Obligation
\[ \land G p_i \lor F q_i \]

Safety
\[ \land G p \]

Guarantee
\[ \lor F p \]

Deterministic Büchi

Transition-based and generalized Büchi acceptance useful. Compare:

Automata for \( G F a \land G F b \).
The Temporal Hierarchy

- **Reactivity**: $\bigwedge \ G F p_i \lor F G q_i$
- **Recurrence**: $G F p$
- **Obligation**: $\bigwedge \ G p_i \lor F q_i$
- **Safety**: $G p$
- **Persistence**: $F G p$
- **Guarantee**: $F p$
- **Deterministic Büchi**
- **Weak Büchi**
- **Terminal Büchi**

**Transition-based and generalized co-Büchi** acceptance useful.

Automata for $FG \bar{a} \lor FG \bar{b}$.
The Temporal Hierarchy

Reactivity
\[ \land G F p_i \lor F G q_i \]

Recurrence
\[ G F p \]

Persistence
\[ F G p \]

Obligation
\[ \land G p_i \lor F q_i \]

Safety
\[ G p \]

Guarantee
\[ F p \]

More complex acceptance conditions (Rabin, Streett, Parity, generalized-Rabin, etc.) compete here.
Spot has a very good translator, combining several improved procedures.

Named acceptances are a hindrance. Generic algorithms are more elegant.

Spot: groundwork for research + tools for experimenting, testing, finding interesting cases.
Original motivations

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

Tool support at publication

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

Resulting challenge

Can we build tools that process automata with arbitrary acceptance conditions?
The Hanoi Omega-Automata Format

Original motivations

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

Tool support at publication

<table>
<thead>
<tr>
<th>ltl2dstar 0.5.3</th>
<th>Rabinizer 3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ltl3ba 1.1.2</td>
<td>PRISM 4.3</td>
</tr>
<tr>
<td>ltl3dra 0.2.2</td>
<td>Spot 1.99.2</td>
</tr>
<tr>
<td>jhoafparser</td>
<td>cpphoafparser</td>
</tr>
</tbody>
</table>

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

### Original motivations

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

### Tool support at publication

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ltl2dstar</td>
<td>0.5.3</td>
</tr>
<tr>
<td>ltl3ba</td>
<td>1.1.2</td>
</tr>
<tr>
<td>ltl3dra</td>
<td>0.2.2</td>
</tr>
<tr>
<td>Rabinizer</td>
<td>3.1</td>
</tr>
<tr>
<td>PRISM</td>
<td>4.3</td>
</tr>
<tr>
<td>Spot</td>
<td>1.99.2</td>
</tr>
<tr>
<td>jhoafparser</td>
<td></td>
</tr>
<tr>
<td>cpphoafparser</td>
<td></td>
</tr>
</tbody>
</table>

**positive Boolean formulas of** $\text{Inf}(x)$ **and** $\text{Fin}(y)$ **terms**
The Hanoi Omega-Automata Format

Original motivations

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

Tool support at publication

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ltl2dstar</td>
<td>0.5.3</td>
</tr>
<tr>
<td>ltl3ba</td>
<td>1.1.2</td>
</tr>
<tr>
<td>ltl3dra</td>
<td>0.2.2</td>
</tr>
<tr>
<td>Rabinizer</td>
<td>3.1</td>
</tr>
<tr>
<td>PRISM</td>
<td>4.3</td>
</tr>
<tr>
<td>Spot</td>
<td>1.99.2</td>
</tr>
<tr>
<td>jhoafparser</td>
<td></td>
</tr>
<tr>
<td>cpphoafparser</td>
<td></td>
</tr>
</tbody>
</table>

Resulting challenge

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

The intersection of two Streett automata is a Streett automaton,\(\text{Fin}(0) \lor \text{Inf}(1)\).

The union of two Streett automata is... nothing special. Union using non-deterministic initial states. Intersection using universal initial states.
Generic Intersections and Unions

The intersection of two Streett automata is a Streett automaton.

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

Intersection using universal initial states

Union using non-deterministic initial states
Generic Intersections and Unions

The intersection of two Streett automata is a Streett automaton.

\[ \text{Fin}(0) \lor \text{Inf}(1) \]

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

\[ (\text{Fin}(0) \lor \text{Inf}(1)) \lor (\text{Fin}(2) \lor \text{Inf}(3)) \]
The intersection of two Streett automata is a Streett automaton.

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

Union using non-deterministic initial states.

Intersection using universal initial states.

Union using non-deterministic initial states.
Generic Intersections and Unions

Intersection using universal initial states

The intersection of two Streett automata is a Streett automaton

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

Intersection using universal initial states

\[
\begin{align*}
\bar{a} \lor b & \quad \bar{a}b & \quad \bar{a}b \\
0 & \quad 0 & \quad 0 \\
\bar{a} \lor b & \quad \bar{a}b & \quad \bar{a}b \\
1 & \quad 0 & \quad 0 \\
b & \quad a\bar{b} & \\
0 & \quad 0 \\
\bar{b} &
\end{align*}
\]

\[
\begin{align*}
\text{Fin}(0) \lor \text{Inf}(1) \lor \text{Fin}(2) \lor \text{Inf}(3)
\end{align*}
\]
Other Operations

automata simplifications
Most of Spot’s simplifications have been generalized already.

complementation
Trivial on any deterministic $\omega$-automata.
What about non-deterministic $\omega$-automata?
Other Operations

automata simplifications
Most of Spot’s simplifications have been generalized already.

complementation
Trivial on any deterministic $\omega$-automata.
What about non-deterministic $\omega$-automata?

emptiness check
Easy for “Fin-less acceptance”.
More generic algorithms in the works.
(NP-complete in the general case.)
automata simplifications
Most of Spot’s simplifications have been generalized already.

complementation
Trivial on any deterministic \( \omega \)-automata.
What about non-deterministic \( \omega \)-automata?

emptiness check
Easy for “Fin-less acceptance”.
More generic algorithms in the works.
(NP-complete in the general case.)

acceptance conversions
Used before “non-generic” algorithms:

\[
\begin{align*}
\text{CNF} & \equiv \text{Gen. Streett} \equiv \text{Streett} \\
\text{any} & \equiv \text{Fin-less} \equiv \text{Gen. Büchi} \equiv \text{Büchi} \\
\text{DNF} & \equiv \text{Gen. Rabin} \equiv \text{Rabin}
\end{align*}
\]
0 Context & Motivation

1 LTL to Büchi

Spot has a very good translator, combining several improved procedures.

2 Generalized Acceptance

Named acceptances are a hindrance. Generic algorithms are more elegant.

3 Tooling for improvement

Spot: groundwork for research + tools for experimenting, testing, finding interesting cases.

4 Closing remarks
How to test an LTL translator?

By comparing results of multiple translators.

Spot has been using LBTT (LTL-to-Büchi Translator Testbench) since 2003 in its test-suite.

How to test an LTL translator?

By comparing results of multiple translators.

Spot has been using LBTT (LTL-to-Büchi Translator Testbench) since 2003 in its test-suite.

- LBTT is no longer maintained (last release in 2005),
- LBTT is restricted to LTL,
- LBTT is restricted generalized Büchi acceptance.
Spot has been using LBTT (*LTL-to-Büchi Translator Testbench*) since 2003 in its test-suite.
- LBTT is no longer maintained (last release in 2005),
- LBTT is restricted to LTL,
- LBTT is restricted generalized Büchi acceptance.

*ltlcross* is Spot’s replacement of LBTT. It supports:
- linear fragment of PSL (since Spot 1.0),
- arbitrary acceptance conditions (since Spot 1.99.1),
- weak alternating automata (since Spot 2.3),
- optional determinization-based checks (since Spot 1.99.8).
Reusing Algorithms to Improve Other Tools

LTL form. → LTL rewritings → Core translation → Post-processings → automaton

- \texttt{ltl2tgba}

21 / 27
Reusing Algorithms to Improve Other Tools

ltl2tgba

LTL form. → LTL rewritings → Core translation → Post-processings → automaton

- ltlfilt --simplify
- autfilt --small
- autfilt --det

An obligation. The secret weapon did all the work.
Reusing Algorithms to Improve Other Tools

LTL form. → LTL rewritings → Core translation → Post-processings → automaton

- ltl2tgba
- ltlfilt --simplify
- autfilt --small
- autfilt --det

$ ltl3dra -f 'F!p0 && Xp0 && (Gp1 || XF!p0)' | grep States:
States: 7

$ ltl3dra -f 'F!p0 && Xp0 && (Gp1 || XF!p0)' | autfilt --det | grep States:
States: 2
Reusing Algorithms to Improve Other Tools

ltl2tgba

LTL form. → LTL rewritings → Core translation → Post-processing → automaton

ltlfilt --simplify

autfilt --small

autfilt --det

$ ltl3dra -f 'F!p0 && Xp0 && (Gp1 || XF!p0)' | grep States:
States: 7

$ ltl3dra -f 'F!p0 && Xp0 && (Gp1 || XF!p0)' | autfilt --det | grep States:
States: 2

An obligation. The secret weapon did all the work.
Let us not reinvent the wheel every time a new tool is made.

**Diagram:**
- LTL form.
- LTL rewritings
- Core translation
- Post-processings
- Automaton
Building New Tools Efficiently

Let us not reinvent the wheel every time a new tool is made.

**ltl3hoa**

LTL form. → **LTL rewritings** → **New translation** → **Post-processings** → automaton

Tool by Juraj Major; https://github.com/jurajmajor/ltl3hoa.
Let us not reinvent the wheel every time a new tool is made.

**ltl3hoa**

- **LTL form.**
- **LTL rewritings**
- **New translation**
- **Post-processings**
- **Automaton**

Outputs non-deterministic $\omega$-automata with generic acceptance.

Configured to preserve generic acceptance.

Tool by Juraj Major; [https://github.com/jurajmajor/ltl3hoa.](https://github.com/jurajmajor/ltl3hoa)
Building New Tools Efficiently

Let us not reinvent the wheel every time a new tool is made.

ltl3hoa

LTL form.
LTL rewritings
New translation
Post-processings
automaton

Outputs non-deterministic $\omega$-automata with generic acceptance.

Configured to preserve generic acceptance.

gen. Büchi automaton
Degen.
Semi-determinize
Post-processings
Büchi automaton

Seminator

Tool by Mikulas Klokokca; https://github.com/mklokokcka/seminator.
SAT-based minimization of det. $\omega$-automata

Using a SAT solver, we can mine for cases where our routines (or other tools) could be improved.

R. Ehlers. Minimising deterministic Büchi automata precisely using SAT solving. $SAT'10$

S. Baarir and A. Duret-Lutz. SAT-based minimization of deterministic $\omega$-automata. $LPAR'15$
Using a SAT solver, we can mine for cases where our routines (or other tools) could be improved.

```bash
$ ltl2dstar -f 'GFa->GFb' | grep 'States:\|Accept'
States: 4
Acceptance: 4 (Fin(0) & Inf(1)) | (Fin(2) & Inf(3))

$ ltl2dstar -f 'GFa->GFb' | autfilt --det | grep 'States:\|Accept'
States: 4
Acceptance: 4 (Fin(0) & Inf(1)) | (Fin(2) & Inf(3))

$ ltl2dstar -f 'GFa->GFb' | autfilt -S --sat-minimize | grep 'States:\|Accept'
States: 3
Acceptance: 4 (Fin(0) & Inf(1)) | (Fin(2) & Inf(3))
```

S. Baarir and A. Duret-Lutz. SAT-based minimization of deterministic ω-automata. *LPAR’15*
Spot has a very good translator, combining several improved procedures.

Named acceptances are a hindrance. Generic algorithms are more elegant.

Spot: groundwork for research + tools for experimenting, testing, finding interesting cases.
Future Directions

Generalizing Algorithms

**Problem:** algorithms dedicated to “named acceptance conditions”

**Goal:** algorithms working for larger classes of acceptance

**Examples:**
- Can we unify emptiness checks for various acceptance conditions? (Also consider model checking under fairness hypothesis.)
- Is there a determinization procedure that unifies existing ones?

Expand into related territories

**Goal:** build up on the features we support

**Examples:**
- Alternating automata could be used for games, satisfiability, synthesis.
- With little effort, Spot would be a very nice tool for teaching.
Research Statistics (since 2007)

**Publications**
- 3 journal papers
- 22 conference papers (6×ATVA, 4×SPIN, 3×TACAS, 2×LPAR, 2×CIAA, CAV, FORTE, VeCOS, SUMO, FSMNLP)

**Supervision**
- 2 PhD students
- 17+ Epita students

**Development**
- 44 releases of Spot
- 15 releases of Vaucanson

**Citations**
- Over 150 references to Spot

**Reviewing**
- 17 conference papers
- 5 journal papers
- examiner of 2 PhD thesis

**Dissemination**
- 7 invited talks

**Collaboration programs**
- 1 French-Taiwanese ANR
- 1 French-Czech PHC
Co-authors (since 2007)

Co-authors of papers...

Co-authors of software (not already/yet listed above)...
A Rabin Automaton for $GFa \rightarrow GFb$

$(\text{Fin}(0) \land \text{Inf}(1)) \lor (\text{Fin}(2) \land \text{Inf}(3))$
A Rabin Automaton for $GFa \rightarrow GFb$

$$\left(\text{Fin}(0) \land \text{Inf}(1)\right) \lor \left(\text{Fin}(2) \land \text{Inf}(3)\right)$$

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 $\text{GF } a \rightarrow \text{GF } b$

$$(\text{Fin}(0) \land \text{Inf}(1)) \lor (\text{Fin}(2) \land \text{Inf}(3))$$

HOA: v1
States: 4
Start: 0
AP: 2 "a" "b"
acc-name: Rabin 2
Acceptance: 4

$\text{Fin}(0) \& \text{Inf}(1) \lor \text{Fin}(2) \& \text{Inf}(3)$

--BODY--
State: 0 \{ \textcolor{blue}{0} \}
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 1 \{ \textcolor{green}{1} \}
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 2 \{ \textcolor{green}{0} \textcolor{blue}{3} \}
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2
State: 3 \{ \textcolor{green}{1} \textcolor{blue}{3} \}
[!0&!1] 1 [0&!1] 0 [!0&1] 3 [0&1] 2

--END--
An $\omega$-Automaton for $GFa \rightarrow GFb$

$$(\text{Fin}(0) \land \text{Inf}(1)) \lor \text{Inf}(2)$$

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

Acceptance: 3
$\text{Fin}(0) \land \text{Inf}(1) \lor \text{Inf}(2)$

---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 $G F a \rightarrow G F b$

$\text{Fin(0)} \lor \text{Inf(1)}$

HOA: v1
States: 4
Start: 0
AP: 2 "a" "b"
acc-name: Streett 1
Acceptance: 2

--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--
Code metrics

Code, Comments and Blank Lines

Commits per Month

Graphs provided by https://www.openhub.net/p/spot
Spot’s Architecture

```
import spot
import spot.ltsmin
```

libspot

libspot-ltsmin

libbddie

IPython / Jupyter

Import spot

Import spot.ltsmin

libspot

libspot-ltsmin

SpinS

divine
<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>$f$</td>
</tr>
<tr>
<td>all</td>
<td>$t$</td>
</tr>
<tr>
<td>Buchi</td>
<td>$\text{Inf}(0)$</td>
</tr>
<tr>
<td>gen. Buchi 2</td>
<td>$\text{Inf}(0) \land \text{Inf}(1)$</td>
</tr>
<tr>
<td>gen. Buchi 3</td>
<td>$\text{Inf}(0) \land \text{Inf}(1) \land \text{Inf}(2)$</td>
</tr>
<tr>
<td>co-Buchi</td>
<td>$\text{Fin}(0)$</td>
</tr>
<tr>
<td>gen. co-Buchi 2</td>
<td>$\text{Fin}(0) \lor \text{Fin}(1)$</td>
</tr>
<tr>
<td>Rabin 1</td>
<td>$\text{Fin}(0) \land \text{Inf}(1)$</td>
</tr>
<tr>
<td>Rabin 2</td>
<td>$(\text{Fin}(0) \land \text{Inf}(1)) \lor (\text{Fin}(2) \land \text{Inf}(3))$</td>
</tr>
<tr>
<td>Streett 1</td>
<td>$\text{Fin}(0) \lor \text{Inf}(1)$</td>
</tr>
<tr>
<td>Streett 2</td>
<td>$(\text{Fin}(0) \lor \text{Inf}(1)) \land (\text{Fin}(2) \lor \text{Inf}(3))$</td>
</tr>
<tr>
<td>gen. Rabin 3 1 0 2</td>
<td>$(\text{Fin}(0) \land \text{Inf}(1)) \lor \text{Fin}(2) \lor (\text{Fin}(3) \land \text{Inf}(4) \land \text{Inf}(5))$</td>
</tr>
<tr>
<td>parity min odd 5</td>
<td>$\text{Fin}(0) \land (\text{Inf}(1) \lor (\text{Fin}(2) \land (\text{Inf}(3) \lor \text{Fin}(4))))$</td>
</tr>
<tr>
<td>parity max even 5</td>
<td>$\text{Inf}(4) \lor (\text{Fin}(3) \land (\text{Inf}(2) \lor (\text{Fin}(1) \land \text{Inf}(0))))$</td>
</tr>
</tbody>
</table>
Acceptance Transformations (example)

\[(\text{Fin}(1) \land \text{Fin}(3) \land \text{Inf}(0)) \lor (\text{Inf}(2) \land \text{Inf}(3)) \lor \text{Inf}(1)\]

![Diagram of a state machine](attachment:image.png)
Acceptance Transformations (example)

\[(\text{Fin}(1) \& \text{Fin}(3) \& \text{Inf}(0)) | (\text{Inf}(2)\&\text{Inf}(3)) | \text{Inf}(1)\]

$$\text{a} \& \text{b}$$

\[
\begin{array}{c}
0 \\
1 \\
2
\end{array}
\]

$$\text{a}$$

$$\text{!a} \& \text{b}$$

$$\text{!a} \& \text{!b}$$

\[
\begin{array}{c}
0 \\
1 \\
2
\end{array}
\]

$$\text{b}$$

$$\text{a} \& \text{!b}$$

$$\text{!b}$$

\[
\begin{array}{c}
0 \\
1 \\
2
\end{array}
\]

$\text{a} \& \text{b}$

$$\text{!a} \& \text{b}$$

$$\text{!a} \& \text{!b}$$

\[
\begin{array}{c}
0 \\
1 \\
2
\end{array}
\]

\[
\begin{array}{c}
\text{Inf}(0) | \text{Inf}(1) | \text{Inf}(3) \end{array}
\]

\[
\begin{array}{c}
\text{Fin}(3) | \text{Inf}(1) | \text{Inf}(2)
\end{array}
\]

$\text{auffilt} --\text{cnf-acceptance} \text{ example.hoa} > \text{ output.hoa}$
Acceptance Transformations (example)

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

$ autfilt --remove-fin example.hoa > output.hoa$
Acceptance Transformations (example)

\[(\text{Fin}(1) \& \text{Fin}(3) \& \text{Inf}(0)) | (\text{Inf}(2) \& \text{Inf}(3)) | \text{Inf}(1)\]

$\text{autfilt} --\text{remove-fin} --\text{cnf-acc} \text{ example.hoa} > \text{ output.hoa}$
Acceptance Transformations (example)

\[(\text{Fin}(\text{a}) \land \text{Fin}(\text{b}) \land \text{Inf}(\text{c})) \lor (\text{Inf}(\text{d}) \land \text{Inf}(\text{e})) \lor \text{Inf}(\text{f})]\]

$\texttt{autfilt} --tgba \text{example.hoa} > \text{output.hoa}$
Determinization (example)

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

Fin(❶) & (Inf(❶) | (Fin(❷) & Inf(❸)))

0
1
!a
2
a
!a & b
a
3
!a & !b
!a & !b
❷
a & !b
❷
4
!a & b
5
a & b
❶
!a & b
❷
a
❷
!a & !b
❸
!a & b
❶ !a & !b
❶
a
❶
!a & !b
a
6
!a & b
!a & b
❷
a & b
❷
!a & !b

Diagram representation of the automaton.
From LTL to Minimal D[T][G]BA

Output: DBA. (Ehlers’ setup.)

R. Ehlers. Minimising deterministic Büchi automata precisely using SAT solving. *SAT’10*

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

Output: DBA.
From LTL to Minimal D[T][G]BA

Output: DTBA.
From LTL to Minimal D[T][G]BA

Output: DTBA.

- **LTL formula**
  - `ltl2dstar (DRA)`
  - attempt conversion to DBA
    - fail
    - success
    - attempt WDBA
      - success
      - fail
      - minimal WDBA
    - minimal DTBA
  - not a recurrence
    - fail
    - success
  - simplify TGBA
    - fail
    - |F| > 1
      - degen to TBA
    - else
      - DTBA SAT minimization
- simplify TGBA
  - minimal DTBA
    - DTBA SAT minimization
  - DTBA SAT minimization
    - minimal DTBA
    - minimal WDBA
    - WDBA minim.
    - WDBA minim.
    - DBA
    - DBA
    - DBA
    - DBA
From LTL to Minimal D[T][G]BA

Output: DTBA.

LTL formula

translate to TGBA

ltl2dstar (DRA)

attempt WDBA minim.

success

not a recurrence

fail

attempt conversion to DBA

success

attempt WDBA minim.

success

minimize TGBA

else

|F| > 1
degen to TBA

|F| > 1

nondet. or

|F| >
m

m = 1

else

attempt powerset to DTBA

not in TCONG

fail

success

nondet.
det.

DTBA SAT minimization

DTBA SAT minimization

minimal DTBA

success

fail

simplify DBA

DBA SAT minimization

minimal DBA

WDBA success

not a recurrence

fail

success

WDBA minim.

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

Output: DTBA.

LTL formula

ltl2dstar (DRA)

translate to TGBA

translate to TGBA

attempt WDBA minim.

attempt WDBA minim.

simplify TGBA

simplify TGBA

|F| > 1
degen to TBA
degen to TBA

|F| > 1
degen to TBA
degen to TBA

nondet.

nondet.

det.
det.

attempt powerset to DTBA

attempt powerset to DTBA

not in TCONG

success

success

success

success

attempt WDBA

attempt WDBA

minimal WDBA

minimal WDBA

not a recurrence

not a recurrence

success

success

success

success

fail

fail

fail

fail

success

success

success

success

DBA SAT minimization

DBA SAT minimization

minimal WDBA

minimal WDBA

DTBA SAT minimization

DTBA SAT minimization

DBA

DBA

success

success

success

success

not a recurrence

not a recurrence

fail

fail
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.

- **ltl2tgba**: translate to TGBA
  - attempt WDBA minim.
    - fail
    - success
- **ltl2dstar (DRA)**: attempt conversion to DBA
  - success
  - fail
- **degen to TBA**: nondet. or $|F| > m = 1$
  - det.
  - nondet.
- **powerset to DTBA**: attempt
- **simplify TGBA**: else
  - det.
  - nondet.
- **simplify DBA**: fail
  - success
- **success 
  - minimal WDBA
  - not a recurrence
- **success 
  - minimal DTBA 
  - DTGBA SAT minimization
  - $m = 1$
  - not in TCONG 
  - success 
  - minimal DTGBA 
  - DTBA SAT minimization 
  - $m > 1$
  - fail 
  - det. 
  - success 

**dstar2tgba**: simplify DBA
ltlcross — basic operations

- Take a list of formulas (LTL/PSL) from file, stdin, or arguments.
- Take a list of translators $T_1, T_2, \ldots$ given as arguments.
- For any formula $\varphi$ and its negation, run all translators:
  \[
  P_i = T_i(\varphi) \quad N_i = T_i(\neg \varphi)
  \]
- Perform the three checks of LBTT:
  - intersection tests: $\mathcal{L}(N_i \otimes P_j) = \emptyset \quad \mathcal{L}(P_i \otimes N_j) = \emptyset$
  - cross-comparison tests (S is a random state-space)
    \[
    \mathcal{L}(P_i \otimes S) = \emptyset \iff \mathcal{L}(P_j \otimes S) = \emptyset \\
    \mathcal{L}(N_i \otimes S) = \emptyset \iff \mathcal{L}(N_j \otimes S) = \emptyset
    \]
  - consistency check: $\text{states}(P_i \otimes S)_{|S} \cup \text{states}(N_i \otimes S)_{|S} = S$
- Additional intersection tests in ltlcross (Spot 1.2):
  \[
  \mathcal{L}(P_i \otimes \overline{P_j}) = \emptyset \text{ if } P_j \text{ is deterministic}
  \]
  \[
  \mathcal{L}(N_i \otimes \overline{N_j}) = \emptyset \text{ if } N_j \text{ is deterministic}
  \]
- Once all formulas have been processed, optionally output detailed statistics in a CSV file.