Teaching LTL and ω-Automata with Spot

Alexandre Duret-Lutz
EPITA Research Laboratory (LRE), Paris, France

Abstract

Spot is a mature, open-source C++/Python library and toolset for Linear Temporal Logic (LTL) and ω-automata manipulation. While Spot is routinely used as a research and verification back-end, its rich visualization capabilities and Python interface also make it an attractive platform for *teaching* the connections between temporal logic formulas and the ω-automata that give them their semantics.

This demonstration showcases two complementary, zero-install entry points into Spot that are suitable for educational settings: **1.** a web application that lets students type LTL or PSL formulas and immediately see the resulting automaton, explore formula simplifications, compare formula equivalence/implication, and navigate Manna & Pnueli's temporal hierarchy. **2.** Jupyter notebooks that combine narrative explanations, live Python code, and inline automaton drawings, enabling students to experiment interactively and instructors to build assignments around concrete, executable examples. Additionally, we can discuss the use of the command-line tools of Spot to generate random examples suitable for preparing a series of exercises.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Formal languages and automata theory; Applied computing → Education

Keywords and phrases linear temporal logic, ω-automata, interactive tool, Jupyter notebooks, education

Digital Object Identifier 10.4230/LIPIcs...

Category Demo

Supplementary Material Online tool: `https://spot.lre.epita.fr/app/`
Interactive notebooks: `https://spot.lre.epita.fr/tut.html`
Source code: `https://gitlab.lre.epita.fr/spot/spot`

1 LTL and Spot

Linear Temporal Logic (LTL) [18] and ω-automata are cornerstones of formal methods education. LTL appears in virtually every graduate course on program verification, and the translation of an LTL formula into a Büchi automaton is an important step to better understand the logic and its many applications, like model checking [19] or reactive synthesis [17].

Several textbooks present the theoretical foundations [e.g. 5, 6], but understanding deepens considerably when students can *experiment*: vary a formula, see the resulting automaton change, and build intuition about which formulas yield small deterministic automata and which do not.

Spot in a nutshell. Spot [8, 9] is an open-source library (GPL v3) for the manipulation of LTL formulas and ω-automata. It supports:

- LTL and a subset of PSL [1], with multiple input syntaxes;
- ω-automata with arbitrary (Emerson–Lei) acceptance conditions, in five standard formats (HOA [4], never claims, LBTT, DSTAR, PGsolver);
- a comprehensive set of formula and automaton algorithms: simplification, equivalence and implication testing, checking stutter-invariance [16], translation to automata (generalized Büchi, Rabin, Streett, parity, ...), simulation-based reduction [3] and SAT-based minimization [2], and more.
- recent support for LTLf (LTL over finite traces) and DFA represented using multi-terminal binary decision diagrams [10].

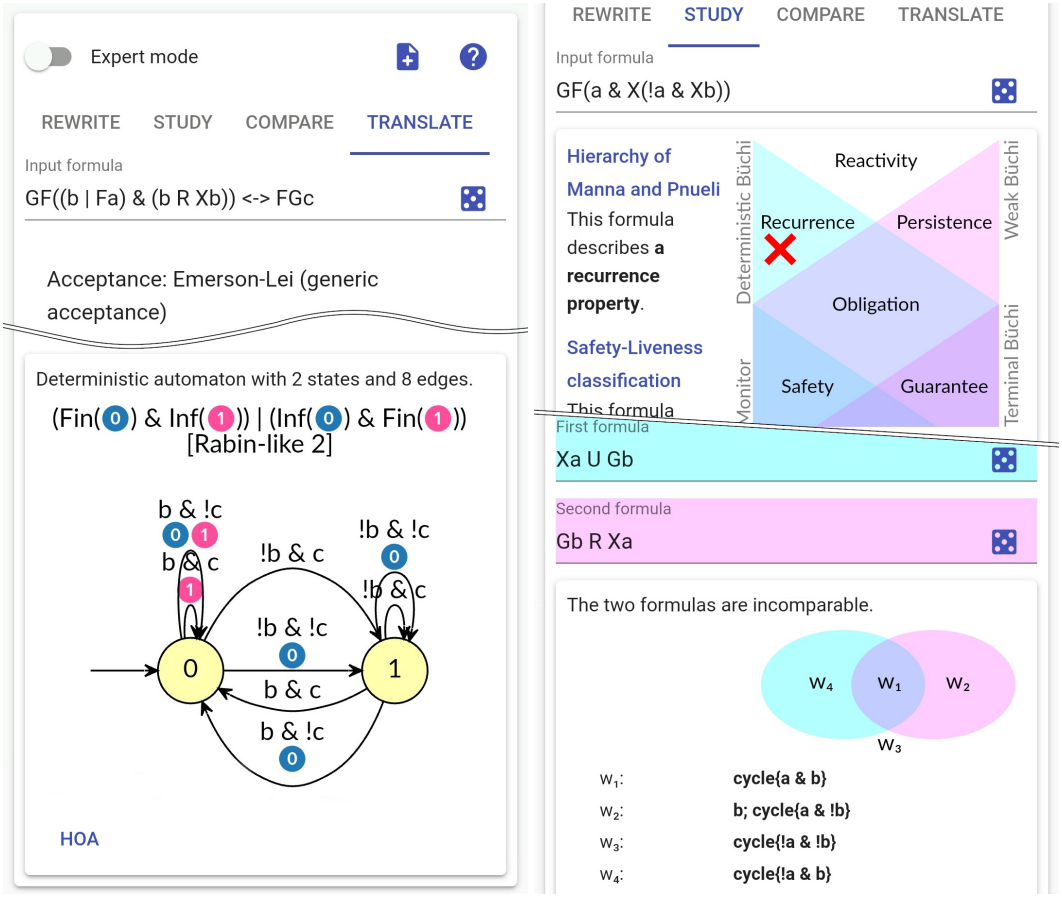


Figure 1 Spot's online LTL toolset [9].

Beyond the C++ library, Spot exposes all algorithms through command-line tools and Python bindings, and generates automaton pictures rendered via Graphviz.

2 The Online LTL Toolset

Spot's online LTL toolset, available at `https://spot.lre.epita.fr/app/`, is a single-page web application that requires no installation. Figure 1 shows a collage of three different interfaces:

- The left is an example of the TRANSLATE tab, where an LTL formula can be converted to ω-automata with various acceptance conditions.
- The STUDY tab in the upper right computes various characteristics of LTL formulas.
- The bottom right example shows the COMPARE tab, where two formulas can be compared, returning distinguishing ω-words when they exist.

A fourth tab, not illustrated here, is dedicated to the simplification of formulas.

This tool can be used by students (but also researchers) to improve their understanding of LTL, PSL, and ω-automata.

Because the tool runs in a standard web browser, it is suitable for classroom use on any device, including tablets. Internet access is required to query the server, where Spot is installed.

The source code for the client and the server of this web application can be found at `https://gitlab.lre.epita.fr/spot/spot-web-app/` for anyone willing to run it locally, or extend it.

Here are a few examples of activities that this tool can be used for:

- Translating a temporal logic formula to an ω-automaton is of course a way to understand the semantics of a given formula.
- Comparing two formulas, and showing examples of words that distinguish them, is a great way to work on finding equivalent formulas. For instance, if a student is tasked with finding a simpler formula equivalent to $(a \mathsf{R} (a \mathsf{U} b)) \mathsf{W} b$, the student could translate this formula into an automaton, and recognize that this looks like the automaton for $a \mathsf{U} b$ and use the COMPARE tab to verify. Or the student could play with the formula in the COMPARE tab, and discover that $(a \mathsf{R} (a \mathsf{U} b)) \mathsf{W} b \equiv (a \mathsf{U} b) \mathsf{W} b \equiv a \mathsf{U} b$.
- Being able to choose between multiple acceptance conditions in the TRANSLATE tab can help illustrate the expressiveness and succinctness of the different types of ω-automata.
- These different types of automata can also be related to the classical hierarchy of Manna and Pnueli [15]. For any given formula, the STUDY tab will point to the most precise class the formula belongs to, and this can be used to select the most appropriate type of ω-automaton to build.
- The syntactic-future hierarchy is the syntactic counterpart of the Manna and Pnueli hierarchy [11, Fig. 1]. It classifies LTL formulas according to their nesting of strong operators (F, U, M) and weak operators (G, W, R). The two hierarchies are related in the sense that if an LTL property belongs to some class of the Manna-Pnueli hierarchy, it can be represented by a formula in the corresponding class of the syntactic-future hierarchy. Sometimes, you can consider a formula such as $a \mathsf{W} \mathsf{F}(b)$ that is an Obligation property in the Manna-Pnueli hierarchy (corresponding to the syntactic class Δ_1) but that syntactically belongs to class Π_2 (which would be the Recurrence class of the Manna-Pnueli hierarchy). The tool points out that the discrepancy indicates that there exists an equivalent formula in class Δ_1 (where strong and weak operators may not be nested). Playing with the COMPARE tab can help figure out that $a \mathsf{W} \mathsf{F}(b) \equiv \mathsf{G}(a) \vee \mathsf{F}(b)$.
- A property of LTL formulas that is interesting from the point of view of model checking is that of "*stutter invariance*" [12] because it allows model checker to apply optimizations known as "*partial order reductions*" [5, Chap. 8] [6, Chap. 10]. A formula φ is stutter-invariant if an ω-word w is accepted by φ if and only if any ω-word w' derived from w by duplicating some letters or removing some duplicate letters is also accepted by φ. It is well known that X-free formulas are stutter-invariant, but this is not a necessary condition. For instance the formula $\mathsf{F}(a \wedge \mathsf{X}(\neg a \wedge b))$ is stutter-invariant. The STUDY tab of the tool will tell if a formula is stutter-invariant, and when that is not the case, it will provide some example words w and w' showing why that is not the case [16].

3 Jupyter Notebooks

For courses where students write code (or for researchers exploring Spot programmatically), Spot provides a rich Python API together with a gallery of Jupyter [13] notebooks at `https://spot.lre.epita.fr/tut.html`.

Figure 2 shows an example where the `spot.contains()` function is first used to check inclusion between the languages represented by two LTL formulas. Then a `contains()` function is defined to demonstrate how such an inclusion check could be implemented using non-deterministic Büchi automata, highlighting a counterexample annotated on those automata.

To help interpret automata and their connection to logic, a feature recently introduced in Spot allows relabeling the states of a deterministic automaton that recognizes a known formula. Figure 3 demonstrates this on an automaton produced by Owl [14].

```
In [2]: f = spot.formula("G(p -> Fq)")
        g = spot.formula("Gp -> Fq")
        display(f, g)
```

$G(p \rightarrow Fq)$

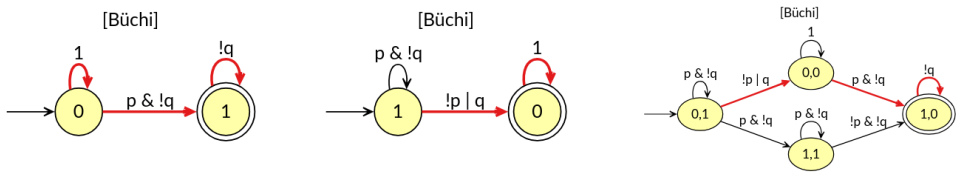
$Gp \rightarrow Fq$

```
In [3]: spot.contains(f, g)  # f ⊃ g ?
Out[3]: False
```

```
In [4]: def contains(f, g):
            f_neg = spot.translate(spot.formula.Not(f))
            g_pos = spot.translate(g)
            prod = spot.product(f_neg, g_pos)
            r = prod.accepting_run()
            if r is None:
                return True
            display(spot.twa_word(r))
            r.highlight(5)
            r.project(f_neg).highlight(5)
            r.project(g_pos, True).highlight(5)
            display_inline(f_neg, g_pos, prod, per_row=3)
            return False
```

```
In [5]: contains(f, g)
```

$\neg p \vee q; p \wedge \neg q; \mathrm{cycle}\{\neg q\}$




```
Out[5]: False
```

Figure 2 Using Spot in a Jupyter Notebook

More advanced scenarios, such as LTL model checking[1] or LTL synthesis[2], can also be covered.

For courses where a local Python/Jupyter installation is impractical, *spot-sandbox*[3] provides a pre-configured Jupyter instance with Spot already installed. Students connect through a browser and start experimenting immediately. The Docker image serving *spot-sandbox* can also be downloaded to be run locally.

4 Command-line Tools

For teaching activities, it is often useful to look for examples satisfying certain criteria. For instance, we might be looking for examples of formulas that are equivalent to $a \mathbin{U} b$, or some formula that belongs to a syntactic class above its property class, or formulas that translate to automata with fewer than 3 states, etc.

Spot's command-line tools make it easy to brute-force such examples by generating random formulas, and retaining only those matching the criteria [7]. For instance, here is a list of formulas equivalent to $a \mathbin{U} b$, not using X or xor; this command simply generates an infinite list (`-n-1`) of random formulas, and then filters the formulas that are equivalent to $a \mathbin{U} b$ and stops after 10 have been found.

```
% randltl --ltl-prio=X=0,xor=0 -n-1 a b | ltlfilt --equivalent-to='a␣U␣b' -n10
```

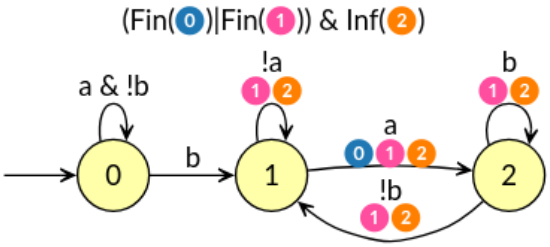
```
a U b
b W (a U b)
```

[1] https://spot.lre.epita.fr/ipynb/atva16-fig2b.html
[2] https://spot.lre.epita.fr/ipynb/synthesis.html
[3] http://spot-sandbox.lre.epita.fr/

```
In [1]: import spot
        spot.setup()

In [2]: def ltl2dela(f):
            f = spot.formula(f)
            return spot.automaton(f"owl ltl2dela -f {f:q} |")

In [3]: display(aut := ltl2dela(f := "(a U b) & (GFa -> FGb)"))
```




```
In [4]: spot.match_states_decorate(aut, f)
        display(aut)
```

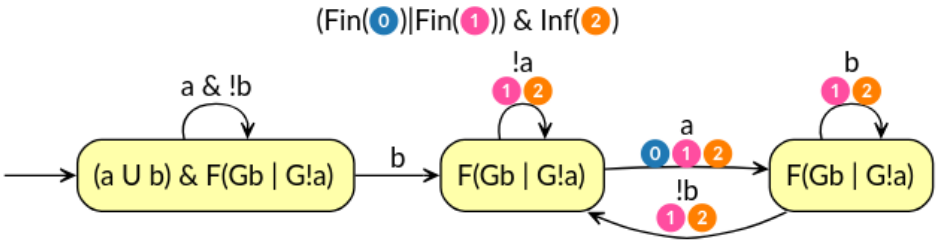


Figure 3 Running Owl's `ltl2dela` tool, and then relabeling the states of the produced automaton by the LTL formulas they recognize.

```
((a | Fb) -> b) M (a U b)
a U (Gb W b)
b U (a U b)
(b W (a & Fb)) W b
a U (b | (b & ((0 R b) R Ga)))
(a <-> Fb) U b
b M (b M (a U b))
b M (a | b)
```

The next command looks for 10 formulas that are stutter-invariant despite containing an X operator. The command also excludes formulas that are equivalent to ⊤ or ⊥ (option `-v` negates the matches), and simplifies formulas to also ignore formulas where the X are easy to remove.

```
% randltl -n-1 a b | grep X | ltlfilt --stutter-invariant | \
ltlfilt -v --equivalent-to=0 | ltlfilt -v --equivalent-to=1 | \
ltlfilt --simplify | grep X | head
```

```
b | (a & XF(b R a)) | (!a & XG(!b U !a))
G(b U (!b & X!b))
(b | X!b) W a
Xa W (b | ((!b M F!a) R !a))
!b & G(G!a | XFa)
```

```
G(!b & X(!b W XG!a))
b | (a & XF!a)
Ga | G(!a & X(!a U !b))
a | (((!b & Xb) | (b & X!b)) M !b)
GF(Ga | (b R X!b))
```

The following command looks for 10 obligation formulas that cannot be detected as obligations by looking at their syntactic classes. The option `--stats=%h,%f` asks `ltlfilt` to output a CSV file where the first column is a letter indicating the class in the Manna and Pnueli hierarchy (`O` is for *obligation*), and the second column is a formula. After grepping for lines that start with `O`, `ltlfilt` is used again to extract the first 10 lines of the second column (`-/2` designates the second column of standard input).

```
% randltl -n-1 a b | \
ltlfilt --simplify -v --syntactic-obligation --stats=%h,%f | \
grep '^O,' | ltlfilt -F-/2 --stats=%f -n10
```

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

5 Conclusion

Spot offers several interfaces for teaching LTL and ω-automata: a web application for quick browser-based exploration; Jupyter notebooks that weave narrative, live code, and inline automaton drawings; and command-line tools for easy chaining of commands with pipes. All three interfaces expose the same underlying library.

Spot is actively maintained under a GPL v3 license. However, most new features are currently driven by research needs rather than pedagogical ones. Instructors who find functionality missing or workflows that could be streamlined for classroom use are warmly encouraged to open feature requests on the issue tracker at `https://gitlab.lre.epita.fr/spot/spot`, so that teaching use cases can be better prioritized alongside research use cases.

References

1 *Property Specification Language Reference Manual v1.1*. Accellera, June 2004. URL `http://www.eda.org/vfv/`.

2 Souheib Baarir and Alexandre Duret-Lutz. SAT-based minimization of deterministic ω-automata. In *Proc. of LPAR'15*, volume 9450 of *LNCS*, pages 79–87, 2015. doi: 10.1007/978-3-662-48899-7_6.

3 Tomáš Babiak, Thomas Badie, Alexandre Duret-Lutz, Mojmír Křetínský, and Jan Strejček. Compositional approach to suspension and other improvements to LTL translation. In *Proceedings of the 20th International SPIN Symposium on Model Checking of Software (SPIN'13)*, volume 7976 of *Lecture Notes in Computer Science*, pages 81–98. Springer, July 2013. doi: 10.1007/978-3-642-39176-7_6.

4 Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi Omega-Automata format. In *Proc. of CAV'15*, volume 9206 of *LNCS*, pages 479–486, 2015. doi: 10.1007/978-3-319-21690-4_31.

5 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

6 Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999. ISBN 978-0-262-03270-4.

7 Alexandre Duret-Lutz. Manipulating LTL formulas using Spot 1.0. In *Proc. of ATVA'13*, volume 8172 of *LNCS*, pages 442–445, 2013. doi: 10.1007/978-3-319-02444-8_31.

8 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proc. of ATVA'16*, volume 9938 of *LNCS*, pages 122–129, 2016. doi: 10.1007/978-3-319-46520-3_8.

9 Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, August 2022. doi: 10.1007/978-3-031-13188-2_9.

10 Alexandre Duret-Lutz, Shufang Zhu, Nir Piterman, Giuseppe De Giacomo, and Moshe Y. Vardi. Engineering an LTLf synthesis tool. In *Proceedings of the 29th International Conference on Implementation and Applications of Automata (CIAA'25)*, volume 15981 of *Lecture Notes in Computer Science*, pages 129–147. Springer, September 2025. doi: 10.1007/978-3-032-02602-6_10.

11 Javier Esparza, Rubén Rubio, and Salomon Sickert. Efficient normalization of linear temporal logic. *Journal of the ACM*, 71(2), April 2024. ISSN 0004-5411. doi: 10.1145/3651152.

12 Kousha Etessami. A note on a question of Peled and Wilke regarding stutter-invariant LTL. *Information Processing Letters*, 75(6):261–263, 2000. doi: 10.1016/S0020-0190(00)00113-7.

13 Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter development team. Jupyter notebooks — a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Scmidt, editors, *Proceedings of 20th International Conference on Electronic Publishing: Positioning and Power in Academic Publishing: Players, Agents and Agendas (ELPUB'16)*, pages 87–90. IOS Press, 2016. doi: 10.3233/978-1-61499-649-1-87.

14 Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In *Proc. of ATVA'18*, volume 11138 of *LNCS*, pages 543–550, 2018. doi: 10.1007/978-3-030-01090-4_34.

15 Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing (PODC'90)*, pages 377–410, New York, NY, USA, 1990. ACM. doi: 10.1145/93385.93442.

16 Thibaud Michaud and Alexandre Duret-Lutz. Practical stutter-invariance checks for ω-regular languages. In *Proceedings of the 22th International SPIN Symposium on Model Checking of Software (SPIN'15)*, volume 9232 of *Lecture Notes in Computer Science*, pages 84–101. Springer, August 2015. doi: 10.1007/978-3-319-23404-5_7.

17 Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In E. Allen Emerson and Kedar S. Namjoshi", editors, *Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, volume 3855 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2006. doi: 10.1007/11609773_24.

18 Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977. doi: 10.1109/SFCS.1977.32.

19 Moshe Y. Vardi. Automata-theoretic model checking revisited. In *Proceedings of the 8th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'07)*, volume 4349 of *Lecture Notes in Computer Science*. Springer, 2007. doi: 10.1007/978-3-540-69738-1_10.