Growing an ω-automaton library

Alexandre Duret-Lutz

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

Abstract. This is an **invited talk** about the past, present, and future of Spot, a C++ library for the manipulation of linear-time temporal logic and ω-automata, with applications to model checking and reactive synthesis. Today, most of the features of Spot revolve around its support for transition-based Emerson-Lei automata. I discuss key developments in the history of Spot, and recently introduced features that I find exciting.

1 Present: What is Spot?

Today, Spot is presented as a C++17 library for the manipulation of linear-time temporal logic (LTL) and ω-automata. Calling Spot a C++ library is a bit reductive, because users can actually use three interfaces: the C++ library, the Python bindings, and a set of command-line tools. The Python bindings are typically used for interactive exploration, prototyping, and scripting. C++ is for production code, where efficiency matters. The command-line tools are designed to follow the Unix philosophy of using text streams as input/output to enable composition of tools using pipes. These tools may be used for scripting tasks, or help with benchmarking algorithms. Spot also offers certain specialized tools (like `ltlcross` and `autcross`) that help to test LTL or automata-based algorithms.

2 A Brief History in Four Steps

2.1 Infancy

Spot started as a Master and then PhD project. In 1999, Jean-Michel Couvreur had published two algorithms related to the automata approach for LTL model checking [5]: a translation from LTL to Transition-based Generalized Büchi Automaton (TGBA), and an emptiness check for TGBA. Model checking is traditionally done using Büchi automata (BA), where accepting runs have to visit infinitely often a state that belongs to some acceptance set. TGBAs generalize those automata by using accepting transitions instead of accepting states, and by using multiple acceptance sets. As TGBAs can be more compact than BAs, we wanted to explore their use for model checking.

The automata-theoretic approach to model checking can be described in four steps: (1) translate some LTL specification φ into an automaton $\mathcal{A}_{\neg\varphi}$ that recognizes any behavior not allowed by φ, (2) build a Kripke structure S representing

the state-space of the model that you want to check against φ, (3) build the product $\mathcal{A}_{\neg\varphi} \times S$ representing all behaviors of S that are disallowed by φ, and finally (4) ensure that this product is empty.

Spot used the definition of TGBA as an abstract interface between all steps, in a way that allowed on-the-fly computations. Steps (1) and (4) correspond to Couvreur's algorithms, and step (3) is easy to implement. That leaves us with step (2), which depends on the language used to express the model. Since some colleagues were working on Petri-Net tools, we built a couple of interfaces to expose the reachabilty graphs of Petri-Net constructed by those tools using Spot's TGBA interface. Spot was built as a library to ease the combination of such user-supplied implementation of step (2) with the rest of the steps it provides [8].

Back them, Spot did not offer any command-line tools to its users, but it had a couple of such tools in its test-suite. For instance one tool was called `ltl2tgba` because it was meant to test step (1). However, as a testing tool, it grew a "*junk drawer*" interface: a confusing set of cryptic options added without any logic, for the purpose of testing new features as they were implemented.

Of course, people started using this tool nonetheless, because it had some options to translate LTL into Büchi automata and print them in useful formats. But the set of options made it very difficult to use for people who knew the kind of output they wanted but not how to obtain it.

2.2 Spot 1.0: Introducing Command-line Tools for Users

Seeing people benchmarking `ltl2tgba` (the testing tool) without always using the right options was a strong motivation to implement some command-line tools with a *sane* user-interface. The old `ltl2tgba` tool was eventually renamed `ikwiad` ("*I Know What I Am Doing*") to make sure people would switch to the brand new `ltl2tgba`. This one has a more friendly interface where you specify the type of output you want, and let tool figure out what algorithms to chain to obtain that. Other useful tools were provided along the way, like `genltl` (for generating lists of LTL formulas used in various benchmarks), `ltlfilt` (for filtering streams for formulas, or `ltlcross` (for testing LTL translators) [6].

In order to interface with other tools, Spot already had parsers for several automaton formats. For some work on SAT-based minimization of Deterministic TBA [1] we had to write *yet another* parser to read Rabin automata produced by `ltl2dstar` [10]. Meanwhile, other teams were working on building automata with different acceptance conditions, and were inventing their own format, making it difficult to combine or compare tools. At ATVA'13 in Hanoi, we gathered with a few tool authors, and started to discuss the need of some common and flexible format. We contacted more people after the conference to draft and implement what would eventually be called the *Hanoi Omega Automaton (HOA) Format* [2].

2.3 Spot 2.0: Building support for Emerson-Lei acceptance

The HOA format represents ω-automata in which the acceptance conditions is an arbitrary Boolean formula over atoms like $\mathsf{Fin}(0)$ (0 should be seen finitely

often) or $\mathsf{Inf}(❺)$ (❺ should be seen infinitely often). Such acceptance conditions, also known as Emerson-Lei conditions, can represent all traditional ω-automata acceptances, but give freedom to introduce new acceptance conditions when needed. For example, the product of two automatas with acceptance conditions α_1 and α_2 (assuming, w.l.o.g, disjoint colors) can be built with a simple synchronous product, setting the resulting acceptance to $\alpha_1 \wedge \alpha_2$.

Supporting HOA required a complete redesign of Spot, yielding version 2 [7]. Leaving the world of TGBAs, Spot started to provide algorithms that output more complex acceptance conditions (such as Safra-based determinization). It also opened many research objectives, to build algorithms that support generic acceptance conditions. Notable algorithms are our generic emptiness check [3], and our implementation of the Alternating Cycle Decomposition [4].

A second noteworthy feature of Spot 2.0 was its support for Jupyter's rich display system. Combined with Spot Python's bindings, it allows calling Spot's algorithms in Jupyter notebooks and getting some visual result. Such notebooks are great for prototyping algorithms, interactive exploration, teaching...

2.4 A Shift Towards Reactive Synthesis

With the support for HOA, Spot has grown into an ω-automata library, where model checking was just one *possible* application. In 2018, Maximilien Colange introduced a new application for Spot: LTL Reactive Synthesis. A new tool was added for this purpose (`ltlsynt` [11]), and its development had ripple effects: several algorithms (e.g., our Safra-based determinization) were seriously improved, but also new concepts such as games, Mealy machines, and And Inverter Graphs (AIG) were eventually introduced in Spot [9].

3 Upcoming features

Recently, there have been a lot of developments to apply Reactive Synthesis to LTL_f (LTL over finite traces). Doing this efficiently required the introduction of DFAs represented using Multi-Terminal Binary Decision Diagrams, similar to the DFA representation of Mona. We plan to try to generalize such a representation to automata with Emerson-Lei acceptance in the future.

References

1. Baarir, S., Duret-Lutz, A.: Mechanizing the minimization of deterministic generalized Büchi automata. In: Proceedings of the 34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE'14). Lecture Notes in Computer Science, vol. 8461, pp. 266–283. Springer (Jun 2014). https://doi.org/10.1007/978-3-662-43613-4_17
2. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J., Müller, D., Parker, D., Strejček, J.: The Hanoi Omega-Automata format. In: Proceedings of the 27th International Conference on Computer Aided Verification (CAV'15). Lecture Notes in Computer Science, vol. 9206, pp. 479–486. Springer (Jul 2015). https://doi.org/10.1007/978-3-319-21690-4_31

3. Baier, C., Blahoudek, F., Duret-Lutz, A., Klein, J., Müller, D., Strejček, J.: Generic emptiness check for fun and profit. In: Proceedings of the 17th International Symposium on Automated Technology for Verification and Analysis (ATVA'19). Lecture Notes in Computer Science, vol. 11781, pp. 445–461. Springer (Oct 2019). https://doi.org/10.1007/978-3-030-31784-3_26
4. Casares, A., Duret-Lutz, A., Meyer, K.J., Renkin, F., Sickert, S.: Practical applications of the Alternating Cycle Decomposition. In: Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'22). Lecture Notes in Computer Science, vol. 13244, pp. 99–117 (Apr 2022). https://doi.org/10.1007/978-3-030-99527-0_6
5. Couvreur, J.M.: On-the-fly verification of temporal logic. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM'99). Lecture Notes in Computer Science, vol. 1708, pp. 253–271. Springer-Verlag (Sep 1999). https://doi.org/10.1007/3-540-48119-2_16
6. Duret-Lutz, A.: Manipulating LTL formulas using Spot 1.0. In: Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13). Lecture Notes in Computer Science, vol. 8172, pp. 442–445. Springer, Hanoi, Vietnam (Oct 2013). https://doi.org/10.1007/978-3-319-02444-8_31
7. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16). Lecture Notes in Computer Science, vol. 9938, pp. 122–129. Springer (Oct 2016). https://doi.org/10.1007/978-3-319-46520-3_8
8. Duret-Lutz, A., Poitrenaud, D.: Spot: an extensible model checking library using transition-based generalized Büchi automata. In: Proceedings of the 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'04). pp. 76–83. IEEE Computer Society, Volendam, The Netherlands (Oct 2004). https://doi.org/10.1109/MASCOT.2004.1348184
9. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From Spot 2.0 to Spot 2.10: What's new? In: Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22). Lecture Notes in Computer Science, vol. 13372, pp. 174–187. Springer (Aug 2022). https://doi.org/10.1007/978-3-031-13188-2_9
10. Klein, J., Baier, C.: On-the-fly stuttering in the construction of determ. ω-automata. In: Holub, J., Žďárek, J. (eds.) Proceedings of the 12th International Conference on the Implementation and Application of Automata (CIAA'07). Lecture Notes in Computer Science, vol. 4783, pp. 51–61. Springer (2007). https://doi.org/10.1007/978-3-540-76336-9_7
11. Michaud, T., Colange, M.: Reactive synthesis from LTL specification with Spot. In: Proceedings of the 7th Workshop on Synthesis, SYNT@CAV 2018. Electronic Proceedings in Theoretical Computer Science (2018)