Machine Learning in LTL Synthesis

October 18, 2024

1 Introduction

In safety-critical systems, such as the controllers of rockets or pacemakers, secure software is essential as failures can have severe financial or even deadly consequences. To ensure the reliability of such systems, we often resort to verification, as ordinary software testing can yield false positives. Verification involves taking a formal program specification, typically expressed in a suitable logic, and checking that the program adheres to this specification. Note that this process provides a rigorous mathematical proof, ensuring that the program behaves as intended under *every possible* circumstance.

Reactive Systems Often the specification includes uncontrollable inputs to our system, which is to model interaction with the outside world (the *environment*). We call these systems *reactive* and verifying them amounts to proving that they adhere to the specification, *no matter how the uncontrollable inputs look like*. The classical example for a reactive system is an arbiter for the distribution of a critical resource. This is a system that receives *requests* from the outside world and can issue *grants*, i.e. reacting to a request by giving access to the critical resource. Further, these grants need to be issued *correctly* (no two parties may have access to the critical resource at the same time), but also *fairly* (every request will eventually be answered by a grant).

Linear Temporal Logic (LTL) A particularly common logic for program specifications is linear temporal logic [10], which is interpreted over discrete timesteps in each of which some *atomic propositions* are either true or false. Using appropriate propositions, this allows us to specify "good" and "bad" execution paths of a program. For example, the specification for our arbiter from above (for two parties) would look like the following: $\mathbf{G}(\neg(g_1 \wedge g_2)) \wedge \mathbf{G}(r_1 \rightarrow \mathbf{F}(g_1)) \wedge \mathbf{G}(r_2 \rightarrow \mathbf{F}(g_2))$, where r_i and g_i denote requests from - and grants to party i, and $\mathbf{G}$, $\mathbf{F}$ are LTL operators that denote something holds *always* and *at some point*, respectively. In particular, the first part of the formula means "It never holds that g_1 and g_2 are true at the same time", and the latter two parts mean "It always holds that if a request for party i, at some point in the future there is a grant to party i". In this example we'd call r_1, r_2 the uncontrollable *environment variables*, and g_1, g_2 the controllable *system variables*.

LTL Synthesis Synthesis is a prominent variant of verification, which was incepted by Alonzo Church himself [2]. Instead of taking a program and a specification as input and deciding whether the former adheres to the latter, in synthesis the input is only given by the specification (i.e. a logical formula and a partition of the variables into environment and system). The objective is to derive (or *synthesize*) a system that adheres to the specification *by construction*, or decide that no such system can exist. Employing LTL for the specification yields the problem of LTL synthesis [11], which to this day is an active field of research.

Automata Theoretic Approach Classically, (reactive) LTL synthesis is solved via automata theory. The intuition is that we first want a formalism that can distinguish between "good" and "bad" runs w.r.t. our LTL specification, for which ω-automata as incepted by Büchi [1] are prime candidates. Using this automaton, we can analyze how to obtain a spec-adhering (i.e. an *accepting*) path from every possible situation. Thus, in a first step we convert the LTL formula into a suitable deterministic ω-automaton which is known to be of doubly exponential size in the worst case [7]. In a second step, we address the partition of variables and check wether we can find an appropriate response to every possible valuation of the environment variables. This *antagonistic* view of the environment, motivates the usage of game theory, specifically *graph games* (where the graph is given by the automaton). A solution to the graph game then directly corresponds to a solution for the overall LTL Synthesis problem.

Partial Exploration Recall that the ω-automaton can be of double exponential size. Thus, constructing the entire automaton is often infeasible. Thankfully, constructing the entire automaton is also often not necessary. Through decisions in the graph game, it can happen that entire regions of the automaton become unreachable and thus unnecessary to construct. In other words, we would like to know a priori what part of the automaton will be relevant to solving the subsequent graph game, in order to only construct exactly that part. Unfortunately, we cannot *know* that, as it depends on the solution of the overall graph game which is what we are searching for in the first place. However, we can guess. Whenever, we construct a state of the graph game that has a decision, we can ask an oracle what might be the correct choice and proceed exploring only the suggested direction. If these guesses are correct, we indeed can spare a lot of irrelevant automaton from being constructed. However, we are not relying on the correctness of these guesses, as we can always backtrack and explore another option. This duality of huge potential and non-required reliability motivates the usage of machine learning for implementing this oracle. However, this poses the question, on what basis this model decide which choice looks better. The answer is *semantic labelling*.

Semantic Labelling In the earlier days of LTL synthesis, the automata construction would involve a procedure called "Safra's determinization" [12], which

yielded states whose meaning was utterly cryptic. However, over the last decade, automata translations [3, 6, 13, 4] emerged, which identifies automaton states by annotating them with semantic information. This so called *semantic labelling* is essentially a structured collection of LTL formulae. In particular, there is one formula denoting the overall objective of a state (the *master* formula). Intuitively, this denotes what there is left to be satisfied from this state onwards. Further a state hosts multiple formulae, tracking the progress of subgoals (so called *monitors*). For example, in our arbiter example, the first state of the automaton would have a master formula of $\mathbf{G}(\neg(g_1 \wedge g_2)) \wedge \mathbf{G}(r_1 \rightarrow \mathbf{F}(g_1)) \wedge \mathbf{G}(r_2 \rightarrow \mathbf{F}(g_2))$ (as that is the overall goal to be satisfied) and no monitors just yet. If we then see a request r_1, we move to a state that has the very same master formula (as the overall objective has not changed) but that now also has a monitor $\mathbf{F}\, g_1$, indicating that we still owe a grant. If on the other hand, we issue both grants at the same time, we violate the $\mathbf{G}(\neg(g_1 \wedge g_2))$ subformula and move to a new state where the master formula is *false*. An overall goal of *false* is impossible to be satisfied and thus we lose the game if we reach this state. In particular, we are thereby able to detect that g_1 and g_2 may never be true at the same time just by looking at the semantic labelling. And this is the exact intuition that machine learning can pick up on and exploit when guiding the exploration.

Current Contributions In a previous paper [5], we developed methods to obtain ML-based exploration heuristics based on supervised learning. In particular we investigated suitable architectures, how to obtain a meaningful ground truth labeling, as well as how to extract numerical features from the semantic labelling (i.e. a structured collection of LTL formulae). Further, we evaluated prototypes of such heuristics and demonstrated the huge potential of such heuristics.

Naturally, in a subsequent (unpublished) tool paper, we implemented a competitive tool for LTL Synthesis which makes use of such heuristics. This came with several challenges. First, the methods of [5] needed to be adapted to the state of the art automata construction, which hosts a vastly different semantic labelling. Further, we needed to develop a novel exploration algorithm capable of efficiently querying the ML heuristic and incorporating the responses. In particular, despite following a similar partial exploration approach, the algorithm of the previous state of the art tool Strix [8, 9] is not capable of handling such heuristics. Ultimately, we were able to present our tool SemML which outperforms Strix significantly on multiple benchmark sets. Most importantly we achieve a major improvement on the benchmarks of Syntcomp, the annual synthesis competition held as a satellite event to the CAV conference. Here SemML is able to solve 22 more instances than Strix, which given the difficulty curve of Syntcomp's benchmarks, is a very notable result. Further, on the non-trivial instances both tools could solve, SemML solves significantly faster. Interestingly, this speedup correlates with the complexity of the benchmarks, indicating great scalability of our tool.

References

[1] Büchi, J.: On a decision method in restricted second-order arithmetic. In: Nagel, E., Suppes, P., Tarski, A. (eds.) Proceedings of the First International Congress on Logic, Methodology, and Philosophy of Science 1960 (1962)

[2] Church, A.: Application of recursive arithmetic to the problem of circuit synthesis. Journal of Symbolic Logic (1963)

[3] Esparza, J., Kretínský, J., Raskin, J., Sickert, S.: From linear temporal logic and limit-deterministic büchi automata to deterministic parity automata. Int. J. Softw. Tools Technol. Transf. **24**(4), 635–659 (2022). https://doi.org/10.1007/s10009-022-00663-1

[4] Esparza, J., Kretínský, J., Sickert, S.: A unified translation of linear temporal logic to ω-automata. J. ACM **67**(6), 33:1–33:61 (2020). https://doi.org/10.1145/3417995

[5] Kretínský, J., Meggendorfer, T., Prokop, M., Rieder, S.: Guessing winning policies in LTL synthesis by semantic learning. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13964, pp. 390–414. Springer (2023). https://doi.org/10.1007/978-3-031-37706-8_20

[6] Kretínský, J., Meggendorfer, T., Waldmann, C., Weininger, M.: Index appearance record with preorders. Acta Informatica **59**(5), 585–618 (2022). https://doi.org/10.1007/s00236-021-00412-y, `https://doi.org/10.1007/s00236-021-00412-y`

[7] Kupferman, O., Rosenberg, A.: The blowup in translating LTL to deterministic automata. In: van der Meyden, R., Smaus, J. (eds.) Model Checking and Artificial Intelligence - 6th International Workshop, MoChArt 2010, Atlanta, GA, USA, July 11, 2010, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 6572, pp. 85–94. Springer (2010). https://doi.org/10.1007/978-3-642-20674-0_6

[8] Luttenberger, M., Meyer, P.J., Sickert, S.: Practical synthesis of reactive systems from LTL specifications via parity games. Acta Informatica **57**(1-2), 3–36 (2020). https://doi.org/10.1007/s00236-019-00349-3

[9] Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 578–586. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_31

[10] Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32

[11] Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Ausiello, G., Dezani-Ciancaglini, M., Rocca, S.R.D. (eds.) Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings. Lecture Notes in Computer Science, vol. 372, pp. 652–671. Springer (1989). https://doi.org/10.1007/BFb0035790

[12] Safra, S.: On the complexity of omega-automata. In: 29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988. pp. 319–327. IEEE Computer Society (1988). https://doi.org/10.1109/SFCS.1988.21948

[13] Sickert, S., Esparza, J., Jaax, S., Křetínský, J.: Limit-deterministic büchi automata for linear temporal logic. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9780, pp. 312–332. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_17