Strix: Explicit Reactive Synthesis Strikes Back! [⋆]

Philipp J. Meyer[0000−0003−1334−9079],
Salomon Sickert[0000−0002−0280−8981], and
Michael Luttenberger

Technical University of Munich, Germany
`{meyerphi,sickert,luttenbe}@in.tum.de`

Abstract. Strix is a new tool for reactive LTL synthesis combining a direct translation of LTL formulas into deterministic parity automata (DPA) and an efficient, multi-threaded explicit state solver for parity games. In brief, Strix (1) decomposes the given formula into simpler formulas, (2) translates these on-the-fly into DPAs based on the queries of the parity game solver, (3) composes the DPAs into a parity game, and at the same time already solves the intermediate games using strategy iteration, and (4) finally translates the wining strategy, if it exists, into a Mealy machine or an AIGER circuit with optional minimization using external tools. We experimentally demonstrate the applicability of our approach by a comparison with Party, BoSy, and ltlsynt using the syntcomp2017 benchmarks. In these experiments, our prototype can compete with BoSy and ltlsynt with only Party performing slightly better. In particular, our prototype successfully synthesizes the full and unmodified LTL specification of the AMBA protocol for $n = 2$ masters.

1 Introduction

Reactive synthesis refers to the problem of finding for a formal specification of an input-output relation, in our case a *linear temporal logic (LTL)*, a matching implementation [22], e.g. a *Mealy machine* or an *and-inverter-graph (AIG)*. Since the automata-theoretic approach to synthesis involves the construction of a potentially double exponentially sized automaton (in the length of the specification) [13], most existing tools focus on symbolic and bounded methods in order to combat the state-space explosion [11,5,18,9]. A beneficial side effect of these approaches is that they tend to yield succinct implementations.

In contrast to these approaches, we present a prototype implementation of an LTL synthesis tool which follows the automata theoretic approach using parity games as an intermediate step. Strix[1] uses the LTL-to-DPA translation presented in [23,10] and the multi-threaded explicit-state parity game solver presented in [20,14]: First, the given formula is decomposed into much simpler requirements,

⋆ This work was partially funded and supported by the German Research Foundation (DFG) projects "Game-based Synthesis for Industrial Automation" (253384115) and "Verified Model Checkers" (317422601).

1 `https://strix.model.in.tum.de/`

often resulting in a large number of safety and co-safety conditions and only a few requiring Büchi or parity acceptance conditions, which is comparable to the approach of [21,5]. These requirements are then translated on-the-fly into automata, keeping the invariant that the parity game solver can easily compose the actual parity game. Further, by querying only for states that are actually required for deciding the winner, the implementation avoids unnecessary work.

The parity game solver is based on the *strategy iteration* of [19] which iteratively improves non-deterministic strategies, i.e. strategies that can allow several actions for a given state as long as they all are guaranteed to lead to the specified system behaviour. When translating the winning strategy into a Mealy automaton or an AIG this non-determinism can be used similarly to "don't cares" when minimizing boolean circuits. Strategy iteration offers us two additional advantages, first, we can directly take advantage of multi-core systems; second, we can reuse the winning strategies which have been computed for the intermediate arenas.

Related Work and Experimental Evaluation. From the tools submitted to SYNTCOMP2017, LTLSYNT [15] is closest to our approach: it also combines an LTL-to-DPA-translation with an explicit-state parity game solver, but it does not intertwine the two steps, instead it uses a different approach for the translation leading to one monolithic DPA which is then turned in a parity game. In contrast, the two best performing tools from SYNTCOMP2017, BoSy and Party, use bounded synthesis, by reduction either to SAT, SMT, or safety games.

In order to give a realistic estimation of how our tool would have faired at SYNTCOMP2017 (TLSF/LTL track), we tried to re-create the benchmark environment of SYNTCOMP2017 as close as possible on our hardware: in its current state, our tool would have been ranked below Party, but before LTLSYNT and BoSy. Due to time and resource constraints, we could only do an in-depth comparison with the current version of LTLSYNT; in particular we used the TLSF specification of the complete[2] AMBA protocol for $n = 2$ as a benchmark. We refer to Section 3 for details on the benchmarking procedure.

2 Design and Implementation

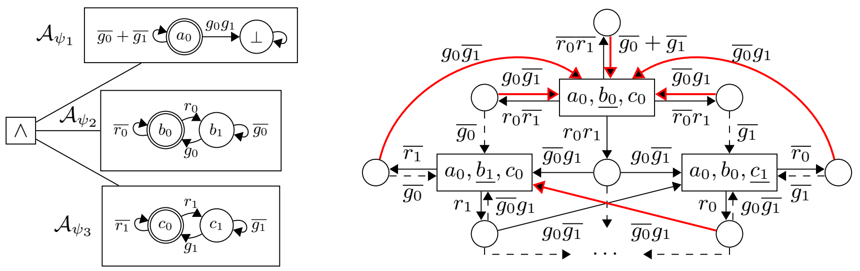
Strix is implemented in Java and C++. It supports `LTL` and `TLSF` [16] (only the reduced *basic* variant) as input languages, while the latter one is preferred, since it contains more information about the specification. We describe the main steps of the tool in the following paragraphs with examples given in Figure 1.

Splitting and Translation. As a preprocessing step the specification is split into syntactic (co)safety and (co)Büchi formulas, and one remaining general LTL formula. These are then translated into the simplest deterministic automaton class using the constructions of [23,10]. To speed up the process these automata are constructed on-the-fly, i.e., states are created only if requested by later

[2] i.e. no decomposition in masters and clients or structural properties were used

stages. Furthermore, since DPAs can be easily complemented, the implementation translates the formula and its negation and chooses the faster obtained one.

$$\varphi = \underbrace{\mathbf{G}(\neg g_0 \vee \neg g_1)}_{\psi_1} \wedge \underbrace{\mathbf{G}(r_0 \rightarrow \mathbf{F} g_0)}_{\psi_2} \wedge \underbrace{\mathbf{G}(r_1 \rightarrow \mathbf{F} g_1)}_{\psi_3} \quad I = \{r_0, r_1\} \quad O = \{g_0, g_1\}$$



Splitted specification with one safety and two Büchi automata.

Partial min-even parity arena. Red thick edges have parity 0 and thin black edges parity 1.

Fig. 1: Synthesis of a simple arbiter with two clients. Here, a winning strategy is already obtained on the partial arena: always take any of the non-dashed edges.

Arena Construction. Here we construct one product automaton and combine the various acceptance conditions into a single parity acceptance condition: for this, we use the idea underlying the last-appearance-record construction, known from the translation of Muller to parity games, to directly obtain a parity game again.

Parity Game Solving. The parity game solver runs in parallel to the arena construction on the partially constructed game in order to guide the translation process, with the possibility for early termination when a winning strategy for the system player is found. It uses strategy iteration that supports non-deterministic strategies [19] from which we can benefit in several ways: First, in the translation process, the current strategy stays valid when adding nodes to the arena and thus can be used as initial strategy when solving the extended arena. Second, the non-deterministic strategies allow us to later heuristically select actions of the strategy that minimize the generated controller and to identify irrelevant output signals (similar to "don't care"-cells in Karnaugh maps). Finally, the strategy iteration can easily take advantage of multi-core architectures [14,20].

Controller Generation and Minimization. From the non-deterministic strategy we obtain an incompletely specified Mealy machine and optionally pass it to the external SAT-based minimizer MeMin [1] for Mealy machines and extract a more compact description.

AIGER Circuit Generation and Minimization. We translate the minimized Mealy machine with the tool Speculoos[3] into an AIGER circuit. In parallel, we also construct an AIGER circuit out of the non-minimized Mealy machine, since this can sometimes result in smaller circuits. The two AIGER circuits are then further compressed using ABC [6], and the smaller one is returned.

3 Experimental Evaluation

We evaluate Strix on the TLFS/LTL-track benchmark of the syntcomp2017 competition, which consists of 177 realizable and 67 unrealizable temporal logic synthesis specifications [15]. The experiment was run on a server with an Intel E5-2630 v4 clocked at 2.2 GHz (boost disabled). To mimic syntcomp2017 we imposed a limit of 8 threads for parallelization, a memory limit of 32 GB and a timeout of one hour for each specification. Every specification for that a tool correctly decides realizability within these limits is counted as solved for the category **Realizability**, and every specification for that it can additionally produce an AIGER circuit that is successfully verified is counted as solved for the category **Synthesis**. For this we verified the circuits with an additional time limit of one hour using the nuXmv model checker [7] with the `check_ltlspec` and `check_ltlspec_klive` routines in parallel.

We compared Strix with ltlsynt in the latest available release (version 2.5) at time of writing. This version differs from the one used during syntcomp2017 as it contains several improvements, but also performs worse in a few cases and exhibits erroneous behaviour: for **Realizability**, it produced one wrong answer, and for **Synthesis**, it failed in 72 cases to produce AIGER circuits due to a program error.

Additionally, we compare our results with the best configuration of the top tools competing in syntcomp2017: Party (portfolio), ltlsynt and BoSy (spot). Due to the difficulty of recreating the syntcomp2017 hardware setup[4], we compiled the results for these tools in Table 1 from the syntcomp2017 webpage[5] combining them with our results.

The **Quality** rating compares the size of the solutions according to the syntcomp2017 formula, where a tool gets $2 - \log_{10} \frac{n+1}{r+1}$ quality points for each

[3] `https://github.com/romainbrenguier/Speculoos`

[4] syntcomp2017 was run on an Intel E3-1271 v3 (4 cores/8 threads) at 3.6 GHz with 32 GB of RAM available for the tools. As stated above, we imposed the same constraints regarding timeout, maximal number of threads, and memory limit; but the Intel E3-1271 v3 runs at 3.6 GHz (with boost 4.0 GHz), while the Intel E5-2630 v4 used by us runs at only 2.2 GHz (boost disabled) resulting in a lower per-thread-performance (potentially 30% slower); on the other hand our system has a larger cache and a theoretically much higher memory bandwidth from up to 68.3 GB/s compared to 25.6 GB/s (for random reads, as in the case of dynamically generated parity games, these numbers are much closer). It seems therefore likely that for some benchmark-tool combinations our system is faster while for others it is slower.

[5] `http://syntcomp.cs.uni-saarland.de/syntcomp2017/experiments/`

verified solution of size n for a specification with reference size r. We now move on to a detailed discussion of the results and their interpretation.

		Our system		SYNTCOMP2017		
		STRIX	LTLSYNT (2.5)	PARTY	LTLSYNT	BOSY
Solved	**Realizability**	214	204	224	195	181
	Synthesis	197	123	203	182	181
	Quality	330	136	308	180	298
	Avg. Quality	1.68	1.10	1.52	0.99	1.64
Time (s) Realizability	`full_arbiter_7`	11.34	MEM	8.77	MEM	TIME
	`prioritized_arbiter_7`	58.53	TIME	372.95	TIME	TIME
	`round_robin_arbiter_6`	8.45	158.33	TIME	733.92	TIME
	`ltl2dba_E_10`	6.79	324.84	TIME	TIME	TIME
	`ltl2dba_Q_8`	2.13	346.12	TIME	TIME	TIME
Size (AIG)	`amba_..._encode_12`	89	ERR	1040	3251	369
	`full_arbiter_5`	531	ERR	2257	7393	TIME
	`full_arbiter_6`	626	ERR	7603	26678	TIME
	`ltl2dba_E_4`	7	406	243	406	TIME
	`ltl2dba_E_6`	11	3952	1955	3952	TIME

Table 1: Results for STRIX compared with LTLSYNT and selected results from SYNTCOMP2017 on the TLSF/LTL-track benchmark and on noteable instances. We mark timeouts by TIME, memouts by MEM, and errors by ERR.

Realizability. We were able to correctly decide realizability for 163 and unrealizability for 51 specifications, resulting in 214 solved instances. We solve five instances that were previously unsolved in SYNTCOMP2017.

Synthesis. We produced AIGER circuits for 148 of the realizable specifications. In 15 cases, we only constructed a Mealy machine, but the subsequent steps (MEMIN for minimization or SPECULOOS for circuit generation) reached the time or memory limit. We were able to verify correctness for 146 of the circuits, reaching the model checking time limit in two case. Together with the 51 specifications for which we determined unrealizability, this results in 197 solved instances.

Quality. We produced 36 solutions that are smaller than any solution during SYNTCOMP2017. The most significant reductions are for the AMBA encoder and the full arbiter, with reductions of over 75%, and for `ltl2dba_E_4` and `ltl2dba_E_6`, where we produce indeed the smallest implementation there is.

3.1 Effects of minimization

We could reduce the size of the Mealy machine in 80 cases, and on average by 45%. However the data showed that this did not always reduce the size of the generated AIGER circuit: in 13 cases (most notably for several arbiter specifications) the size of the circuit generated from the Mealy machine actually increased when applying minimization (on average by 190%), while it decreased in 62 cases (on average by 55%).

We conjecture that the structure of the product-arena is sometimes amenable to compact representation in an AIGER circuit, while after the (SAT-based) minimization this is lost. In these cases the SAT/SMT-based bounded synthesis tools such as BoSy and Party also have difficulties producing a small solution, if any at all.

3.2 Synthesis of complete AMBA AHB arbiter

To test maturity and scalability of our tool, we synthesized the AMBA AHB arbiter [2], a common case study for reactive synthesis. We used the parameterized specification from [17] for $n = 2$ masters, which was also part of SYNTCOMP2016, but was left unsolved by any tool. With a memory limit of 128 GB, we could decide realizability within 26 minutes and produce a Mealy machine with 83 states after minimization. While specialised GR(1) solvers [2,4,12] or decompositional approaches [3] are able to synthesize the specification in a matter of minutes, to the best of our knowledge we are the first full LTL synthesis tool that can handle the complete non-decomposed specification in a reasonable amount of time. For comparison, ltlsynt (2.5) needs more than 2.5 days on our system and produces a Mealy machine with 340 states.

3.3 Discussion

The ltlsynt tool is part of Spot [8], which uses a Safra-style determinization procedure for NBAs. Conceptually, it also uses DPAs and a parity game solver as a decision procedure. However, as shown in [10] the produced automata tend to be larger compared to our translation, which probably results in the lower quality score. Our approach has similar performance and scales better on certain cases. The instances where ltlsynt performs better than Strix are specifications that we cannot split efficiently and the DPA construction becomes the bottleneck.

Bounded synthesis approaches (BoSy, Party) tend to produce smaller Mealy machines and to be able to handle larger alphabets. However, they fail when the minimal machine implementing the desired property is large, even if there is a compact implementation as a circuit. In our approach, we can often solve these cases and still regain compactness of the implementation through minimization afterwards. The strength of the Party portfolio is the combination of traditional bounded synthesis and a novel approach by reduction to safety games, which results in a large number of solved instances, but reduces the avg. quality score.

Future Work Strix combines Java (LTL simplification and automata translations) and C++ (parity game construction and solving). We believe that a pure C++ implementation will further improve the overall runtime and reduce the memory footprint. Next, there are several algorithmic questions we want to investigate going forward, especially expanding parallelization of the tool. Furthermore, we want to reduce the dependency on external tools for circuit generation in order to be able to fine-tune this step better. Especially replacing Speculoos is important, since it turned out that it was unable to handle complex transition systems.

References

1. Abel, A., Reineke, J.: Memin: Sat-based exact minimization of incompletely specified mealy machines. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015, Austin, TX, USA, November 2-6, 2015. pp. 94–101 (2015). https://doi.org/10.1109/ICCAD.2015.7372555
2. Bloem, R., Galler, S.J., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Specify, compile, run: Hardware from PSL. Electr. Notes Theor. Comput. Sci. **190**(4), 3–16 (2007). https://doi.org/10.1016/j.entcs.2007.09.004
3. Bloem, R., Jacobs, S., Khalimov, A.: Parameterized synthesis case study: AMBA AHB. In: Proceedings 3rd Workshop on Synthesis, SYNT 2014, Vienna, Austria, July 23-24, 2014. pp. 68–83 (2014). https://doi.org/10.4204/EPTCS.157.9
4. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. J. Comput. Syst. Sci. **78**(3), 911–938 (2012). https://doi.org/10.1016/j.jcss.2011.08.007
5. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.: Acacia+, a tool for LTL synthesis. In: Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings. pp. 652–657 (2012). https://doi.org/10.1007/978-3-642-31424-7_45
6. Brayton, R.K., Mishchenko, A.: ABC: an academic industrial-strength verification tool. In: Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings. pp. 24–40 (2010). https://doi.org/10.1007/978-3-642-14295-6_5
7. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. pp. 334–342 (2014). https://doi.org/10.1007/978-3-319-08867-9_22
8. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 - A framework for LTL and \omega -automata manipulation. In: Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings. pp. 122–129 (2016). https://doi.org/10.1007/978-3-319-46520-3_8
9. Ehlers, R.: Unbeast: Symbolic bounded synthesis. In: Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings. pp. 272–275 (2011). https://doi.org/10.1007/978-3-642-19835-9_25

10. Esparza, J., Kretínský, J., Raskin, J., Sickert, S.: From LTL and limit-deterministic büchi automata to deterministic parity automata. In: Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. pp. 426–442 (2017). https://doi.org/10.1007/978-3-662-54577-5_25
11. Faymonville, P., Finkbeiner, B., Tentrup, L.: Bosy: An experimentation framework for bounded synthesis. In: Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II. pp. 325–332 (2017). https://doi.org/10.1007/978-3-319-63390-9_17
12. Godhal, Y., Chatterjee, K., Henzinger, T.A.: Synthesis of AMBA AHB from formal specification: a case study. STTT **15**(5-6), 585–601 (2013). https://doi.org/10.1007/s10009-011-0207-9
13. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001], Lecture Notes in Computer Science, vol. 2500. Springer (2002). https://doi.org/10.1007/3-540-36387-4
14. Hoffmann, P., Luttenberger, M.: Solving parity games on the GPU. In: Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings. pp. 455–459 (2013). https://doi.org/10.1007/978-3-319-02444-8_34
15. Jacobs, S., Basset, N., Bloem, R., Brenguier, R., Colange, M., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Michaud, T., Pérez, G.A., Raskin, J., Sankur, O., Tentrup, L.: The 4th reactive synthesis competition (SYNTCOMP 2017): Benchmarks, participants & results. arXiv:1711.11439 [cs.LO] (2017), `http://arxiv.org/abs/1711.11439`
16. Jacobs, S., Klein, F., Schirmer, S.: A high-level LTL synthesis format: TLSF v1.1. In: Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016. pp. 112–132 (2016). https://doi.org/10.4204/EPTCS.229.10
17. Jobstmann, B.: Applications and Optimizations for LTL Synthesis. Ph.D. thesis, Graz University of Technology (2007)
18. Khalimov, A., Jacobs, S., Bloem, R.: PARTY parameterized synthesis of token rings. In: Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. pp. 928–933 (2013). https://doi.org/10.1007/978-3-642-39799-8_66
19. Luttenberger, M.: Strategy iteration using non-deterministic strategies for solving parity games. arXiv:0806.2923 [cs.GT] (2008), `http://arxiv.org/abs/0806.2923`
20. Meyer, P.J., Luttenberger, M.: Solving mean-payoff games on the GPU. In: Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings. pp. 262–267 (2016). https://doi.org/10.1007/978-3-319-46520-3_17
21. Morgenstern, A., Schneider, K.: Exploiting the temporal logic hierarchy and the non-confluence property for efficient LTL synthesis. In: Proceedings First Symposium on Games, Automata, Logic, and Formal Verification, GANDALF 2010, Minori (Amalfi Coast), Italy, 17-18th June 2010. pp. 89–102 (2010). https://doi.org/10.4204/EPTCS.25.11
22. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 179–190. POPL '89, ACM, New York, NY, USA (1989). https://doi.org/10.1145/75277.75293

23. Sickert, S., Esparza, J., Jaax, S., Kretínský, J.: Limit-deterministic büchi automata for linear temporal logic. In: Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. pp. 312–332 (2016). https://doi.org/10.1007/978-3-319-41540-6_17