Almost-Symbolic Synthesis via Δ_2-Normalisation for Linear Temporal Logic

Remco Abraham
r.abraham@student.utwente.nl
University of Twente
Enschede, The Netherlands

Tom van Dijk
t.vandijk@utwente.nl
University of Twente
Enschede, The Netherlands

Salomon Sickert
salomon.sickert@mail.huji.ac.il
The Hebrew University of Jerusalem
Jerusalem, Israel

Abstract

The classic approach to synthesis of reactive systems from linear temporal logic (LTL) specifications involves the translation of the specification to a deterministic ω-automaton and computing a winning strategy for the corresponding game with an ω-regular winning condition. Unfortunately, this procedure has an unavoidable double-exponential blow-up in the worst-case and suffers from the state-explosion problem. To address this state-explosion problem in practice we propose an almost-symbolic version of this classic idea that performs the following steps: (1) normalisation of the specification into a Boolean combination of "simple" fragment of LTL, (2) translation of each "simple" subformula into a deterministic automaton, (3) encoding of each automaton into a binary decision diagram (BDD), (4) construction of a parity automaton (and thus game) by operations on the BDD, (5) symbolic computation of a winning strategy, and finally (6) extraction of a symbolic controller. We prototype this approach in the tool `Otus`, compare it against `Strix`, the winner of SYNTCOMP 2018-2020, on the SYNTCOMP benchmarks, and identify several specifications where `Otus` outperforms `Strix`.

Keywords: Binary Decision Diagram, Linear Temporal Logic, Reactive Synthesis, Parity Game

1 Introduction

In reactive synthesis, as outlined by [18], we are tasked with constructing a controller such that all interactions with an unknown environment satisfy a linear temporal logic (LTL) specifications. It is shown by [18] that this is 2-EXPTIME-complete, and the upper bound is established by constructing a deterministic Rabin automaton (DRW).

Due to the inherent state-space explosion problem, various ideas were developed that make the task more manageable such as: *bounded synthesis* [9], which tries to find solutions smaller than a fixed bound, or restrictions to less expressive fragments of LTL, e.g., the GR1-fragment [4], which has been successfully been applied in practice [10]. We refer the reader to [2] for in-depth overview.

A few years ago reactive synthesis using deterministic parity automata (DPW, a subclass of DRW) and parity games was deemed infeasible in practice. One reason was the lack of an efficient translation from LTL to deterministic ω-automata, since at that time, translations obtaining deterministic ω-automata used non-deterministic BÃijchi automata (NBW) as an intermediate step and determinisation of NBW is a famously hard problem. Despite landmark results, such as [20], in practice the deterministic automata quickly became too large. With the rise of direct translations, LTL synthesis tools such as `ltlsynt` [16] using [19] and `Strix` [14] using a combination of [7, 8] showed that with clever engineering[1] explicit state-space techniques are capable of solving a wide range of specifications and performed better than some of the previous techniques avoiding DPW. Still, all these techniques need to construct a double-exponentially large state-space in the worst case. We propose a symbolic variant of these algorithms to tackle the state explosion problem. This has been attempted before in [17], where Morgenstern and Schneider propose to use the safety-progress hierarchy developed in [6, 15] to obtain a symbolic reactive synthesis algorithm. This hierarchy shows that every LTL formula is equivalent to a formula from one of six syntactic classes and using the notation from [21] these are denoted Σ_i, Π_i, and Δ_i for $i \in \{1, 2\}$. Furthermore, Δ_i is the Boolean closure of Σ_i and Π_i. Thus in order to obtain a symbolic algorithm one needs to define a symbolic translation from Σ_i and Π_i. At that time it was unclear how to efficiently compute for an arbitrary formula an equivalent one in the corresponding syntactic classes. Thus [17] also needs to resort to a determination construction. We reexamine this idea, and make use of the new normalisation result from [21] that translates every (future) LTL formula to a formula in Δ_2.

2 Construction

We now detail the six steps of the construction outlined in the beginning. For the rest of the section we fix an LTL formula φ over a set of inputs I and outputs O.

Step 1. The formula φ is translated into the Δ_2-normalform using the normalisation procedure by [21]. While it is already established in [6] that every formula can be decomposed into a Boolean combination of persistence (Σ_2) and recurrence (Π_2) properties, the constructive proof uses a sub-procedure with non-elementary complexity. [21] addresses this and presents a simple and syntactic translation

[1] `Strix` for example uses on-the-fly exploration of the parity game to avoid constructing the whole game.

to Δ_2 with exponential complexity. Further, this translation is well-behaved in the sense that one still obtains double-exponential deterministic Rabin automata (DRW), despite the exponential blow-up incurred by the normalisation in-between. Let now φ' be a formula in disjunctive normal form from Δ_2 that is equivalent to φ.

Step 2. Let $\psi_1, \psi_2, \ldots, \psi_n$ be subformulas of φ' from Π_2 and Σ_2. Each subformula ψ_i is now separately and directly translated to either a deterministic BÃijchi (DBW) or co-BÃijchi automaton (DCW) using the break-point construction specifically tailored for Π_2 and Σ_2 from [21]. The underlying assumption is that in practice the intermediate DBW and DCW are small, since specifications tend to be large Boolean combinations of small formulas.

Step 3. *We now switch to a symbolic representation* and encode each DBW and DCW in a binary decision diagram (BDD). For our prototype we use a simple encoding scheme: We assign each state of the automaton an integer and we use the binary representation in the BDD. Thus if an automaton has n states, we use $\lceil \log_2(n) \rceil$ variables to encode a state. Thus storing the transition relation of a single automaton requires $2 \cdot \lceil \log_2(n) \rceil + |I| + |O| + 1$ variables. The last variable is used to store if an edge is either accepting or rejecting. Observe that there might be encoding schemes that yield smaller BDDs and we leave an investigation of the effect of better encoding schemes as future work.

In this prototype we use a simple, fixed categorical variable ordering as follows: Atomic propositions are at the top of the BDD. Next come variables encoding the current state, then variables encoding the acceptance condition, and finally variables encoding the successor state. We leave evaluating the impact of other fixed orderings or dynamic variable reordering to future work.

Step 4. We obtain a symbolically represented DRW from Step 3 by union and intersection following the structure of φ'. We implement union and intersection of automata by "and"-operations in the underlying BDD to obtain the product automaton. In order to obtain a deterministic parity automaton (DPW) we apply a symbolic implementation of the "typeness"-construction from [5]. For this we implement the symbolic SCC-decomposition due to [3] as a sub-procedure. We rely on two results from [5]: (1) Given a DRW R one can effectively compute if there exists a parity acceptance condition γ on the structure of R such that the resulting parity automaton accepts the same language. (2) Let R and S be a DRW and deterministic Streett automaton (DSW) for the same language L. Then the algorithm in (1) always finds a parity acceptance condition γ on top of the product automaton $R \times S$ where we ignore the acceptance condition of S. We use this construction in the following way: We translate φ into a DRW R and apply (1). If this succeeds, we continue to Step 5 with the obtained DPW P. Otherwise, we construct a DSW S by translating $\neg\varphi$ to a DRW and then complementing the acceptance condition. We then build the symbolic product automaton $R \times S$ using an "and"-operation in the underlying BDD, apply (1) again and obtain a DPW P.

Step 5. The symbolic DPW is reinterpreted as a parity game and we apply the distraction fix-point iteration [23] to compute a winning strategy for either the environment or the system. This algorithm has been shown to be competitive and easy to implement symbolically [13].

Step 6. The BDD representing the winning strategy for the system player (if there is one) is then converted to an and-inverter graph (AIG) with Mealy semantics. We resolve potential non-determinism in the symbolic representation, i.e., "don't cares", by preferring to output 0 for don't cares. We leave a refinement of this simple heuristic as future work, e.g., one could try to find an assignment for the output values that yield the smallest BDD representing the winning strategy.

3 Experimental Evaluation

We implement the proposed approach in the tool `Otus` and base it on the LTL and ω-automata library Owl [12] and on the multi-core BDD library Sylvan [22]. We evaluate the construction using a subset of the specifications of the SYNTCOMP competition [11] and compare it against `Strix` (version 2020.06-Syntcomp) using the configuration[2] that was ranked at the first place in SYNTCOMP 2020. Since we only measure the time needed for synthesis and not the quality, i.e., the size, of the circuits, we disable post-processing using ABC's AIGER minimization tool [1] for `Strix`.

The experiments are run on a cluster of Dell PowerEdge M610 servers with two Xeon E5520 processors and each run gets 8 cores and 56 GB memory assigned.

In total we use 421 realizable and 157 unrealizable specifications from github.com/meyerphi/syntcomp-reference and filter the specifications as follows: for each specification a five minute time-budget is allocated and only specifications that are processed within five minutes by each tool are selected. We thus select 320 realizable and 85 unrealizable specifications. For selected formulas we repeat the experiment five times and collect the average execution times. The results are presented in Figure 1 and 2 for the realizable and unrealizable specifications, respectively.

We observe that the results are mixed. This is to be expected, since the goal of our prototype, which skips several possible optimisations, is to evaluate the feasibility of our approach and not develop a tool that outperforms `Strix`. However, we identify specifications that are challenging for `Strix`, but can be solved comparably fast by `Otus`. Indeed, for some specifications our construction is over 30× faster.

[2] We use `strix -f "$formula" --ins "$ins" --outs "$outs" --no-compress-circuit --auto -e pq -c`.

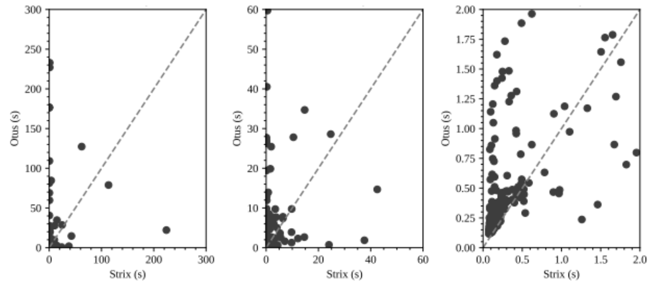

Figure 1. Execution time comparison of Otus (vertical) vs. Strix (horizontal) for realizable specifications; different magnification levels.

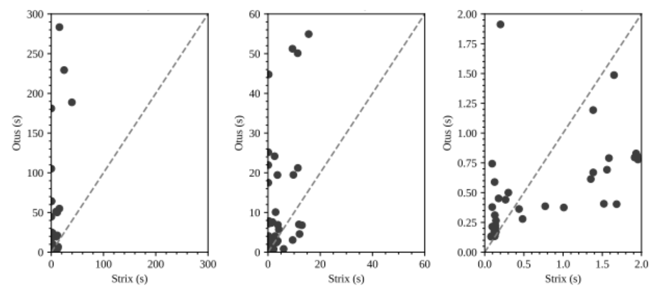

Figure 2. Execution time comparison of Otus (vertical) vs. Strix (horizontal) for unrealizable specifications; different magnification levels.

This indicates that the symbolic technique is promising to tackle larger specifications in the future. For many specifications, Strix is over 1000× faster than our construction, but these are often inputs that Strix can solve in just a few seconds. One reason for this is that Strix has several optimizations, including early termination that avoid computing the full parity game, which we currently do not do. Further, the symbolic technique we propose has a certain fixed overhead that potentially amortises with larger specifications. We list in Table 1 ten instances, with highest and lowest ratios in runtime. These are specifications that can be solved by both our construction and by Strix.

4 Conclusion

We outlined an "almost"-symbolic algorithm for LTL synthesis using parity games, implemented a prototype, and compared it against the explicit-state, on-the-fly LTL synthesis tool Strix. Although step (2) uses an explicit representation and in the subsequent steps we often choose the simple and naive approaches, e.g., state encoding with a simple fixed variable ordering, we observe promising results for a subset of the specifications. This motivates further research into the construction and refinement of each of the outlined steps. Finally, we expect that we can also reduce the execution time by better engineering of the tool, e.g., reducing the number of variables used in the BDDs.

Table 1. The ten specifications with the highest and lowest total execution time ratio of the construction over Strix. Execution times are presented in seconds and are rounded to two decimals. Ratios are computed using the exact execution times.

Specification	Otus (s)	Strix (s)	Ratio
collector_v1_5	0.72	24.02	33.19956
amba_decomposed_lock_10	1.86	37.59	20.15616
amba_decomposed_lock_6	0.33	4.24	12.77229
LedMatrix	22.15	224.09	10.11631
amba_decomposed_encode_10	1.28	9.74	7.62835
TwoCounters3	0.67	4.38	6.50238
ltl2dba_beta_5	2.66	14.62	5.48810
lilydemo22	0.24	1.26	5.32375
tictactoe	2.31	12.16	5.27006
amba_decomposed_lock_8	0.54	2.54	4.66618
...	...	...	...
ltl2dba_C1_6	27.72	0.11	0.00399
EscalatorSmart	226.97	0.91	0.00399
escalator_smart	232.94	0.90	0.00386
prioritized_arbiter_6	109.24	0.42	0.00385
ltl2dba_C2_5	26.29	0.10	0.00366
detector_5	26.27	0.09	0.00346
ltl2dpa03	81.31	0.16	0.00195
ltl2dba_C1_7	176.54	0.12	0.00066
ltl2dba_C2_6	176.26	0.11	0.00064
detector_6	176.77	0.10	0.00059

Acknowledgments

The authors want to thank Orna Kupferman for suggesting to use [5] for the symbolic translation of deterministic Rabin automata to deterministic parity automata. Tom van Dijk is supported by the European UnionâĂŹs Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 893732. Salomon Sickert is supported by the Deutsche Forschungsgemeinschaft (DFG) under project number (436811179).

References

[1] Berkeley Logic Synthesis and Verification Group. [n.d.]. ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/.

[2] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. 2018. Graph Games and Reactive Synthesis. In *Handbook of Model Checking*. Springer, 921–962.

[3] Roderick Bloem, Harold N. Gabow, and Fabio Somenzi. 2006. An Algorithm for Strongly Connected Component Analysis in $n \log n$ Symbolic Steps. *Formal Methods Syst. Des.* 28, 1 (2006), 37–56.

[4] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2012. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.* 78, 3 (2012), 911–938. https://doi.org/10.1016/j.jcss.2011.08.007

[5] Udi Boker, Orna Kupferman, and Avital Steinitz. 2010. Parityizing Rabin and Streett. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India (LIPIcs, Vol. 8)*, Kamal Lodaya and Meena Mahajan (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 412–423. https://doi.org/10.4230/LIPIcs.FSTTCS.2010.412

[6] Edward Y. Chang, Zohar Manna, and Amir Pnueli. 1992. Characterization of Temporal Property Classes. In *ICALP (Lecture Notes in Computer Science, Vol. 623)*. Springer, 474–486.

[7] Javier Esparza, Jan Kretínský, Jean-François Raskin, and Salomon Sickert. 2017. 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 (Lecture Notes in Computer Science, Vol. 10205)*, Axel Legay and Tiziana Margaria (Eds.). 426–442. https://doi.org/10.1007/978-3-662-54577-5_25

[8] Javier Esparza, Jan Kretínský, and Salomon Sickert. 2020. A Unified Translation of Linear Temporal Logic to ω-Automata. *J. ACM* 67, 6 (2020), 33:1–33:61. https://doi.org/10.1145/3417995

[9] Bernd Finkbeiner and Sven Schewe. 2013. Bounded synthesis. *Int. J. Softw. Tools Technol. Transf.* 15, 5-6 (2013), 519–539. https://doi.org/10.1007/s10009-012-0228-z

[10] Yashdeep Godhal, Krishnendu Chatterjee, and Thomas A. Henzinger. 2013. Synthesis of AMBA AHB from formal specification: a case study. *International Journal on Software Tools for Technology Transfer* 15, 5 (01 Oct 2013), 585–601. https://doi.org/10.1007/s10009-011-0207-9

[11] Swen Jacobs and Guillermo Alberto Perez. [n.d.]. The Reactive Synthesis Competition. http://www.syntcomp.org/. Accessed: 2021-01-17.

[12] Jan Kretínský, Tobias Meggendorfer, and Salomon Sickert. 2018. Owl: A Library for ω-Words, Automata, and LTL. In *ATVA 2018*. Springer, 543–550. https://doi.org/10.1007/978-3-030-01090-4_34

[13] Oebele Lijzenga and Tom van Dijk. 2020. Symbolic Parity Game Solvers that Yield Winning Strategies. In *Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2020, Brussels, Belgium, September 21-22, 2020 (EPTCS, Vol. 326)*, Jean-François Raskin and Davide Bresolin (Eds.). 18–32. https://doi.org/10.4204/EPTCS.326.2

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

[15] Zohar Manna and Amir Pnueli. 1987. A Hierarchy of Temporal Properties. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1987*, Fred B. Schneider (Ed.). ACM, 205. https://doi.org/10.1145/41840.41857

[16] Thibaud Michaud and Maximilien Colange. 2018. Reactive Synthesis from LTL Specification with Spot. In *Proceedings of the 7th Workshop on Synthesis, SYNT@CAV 2018 (Electronic Proceedings in Theoretical Computer Science)*.

[17] Andreas Morgenstern and Klaus Schneider. 2010. 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 (EPTCS, Vol. 25)*, Angelo Montanari, Margherita Napoli, and Mimmo Parente (Eds.). 89–102. https://doi.org/10.4204/EPTCS.25.11

[18] Amir Pnueli and Roni Rosner. 1989. On the Synthesis of a Reactive Module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*. ACM Press, 179–190. https://doi.org/10.1145/75277.75293

[19] Florian Renkin, Alexandre Duret-Lutz, and Adrien Pommellet. 2020. Practical "Paritizing" of Emerson-Lei Automata. In *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12302)*, Dang Van Hung and Oleg Sokolsky (Eds.). Springer, 127–143. https://doi.org/10.1007/978-3-030-59152-6_7

[20] Shmuel Safra. 1988. On the Complexity of omega-Automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*. IEEE Computer Society, 319–327. https://doi.org/10.1109/SFCS.1988.21948

[21] Salomon Sickert and Javier Esparza. 2020. An Efficient Normalisation Procedure for Linear Temporal Logic and Very Weak Alternating Automata. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller (Eds.). ACM, 831–844. https://doi.org/10.1145/3373718.3394743

[22] Tom van Dijk. 2016. *Sylvan: multi-core decision diagrams.* Ph.D. Dissertation. University of Twente. https://doi.org/10.3990/1.9789036541602 CTIT Ph.D. thesis series no. 16-398 IPA dissertation series no. 2016-09.

[23] Tom van Dijk and Bob Rubbens. 2019. Simple Fixpoint Iteration To Solve Parity Games. In *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2-3rd September 2019 (EPTCS, Vol. 305)*, Jérôme Leroux and Jean-François Raskin (Eds.). 123–139. https://doi.org/10.4204/EPTCS.305.9