Modernising Strix

Philipp J. Meyer
meyerphi@in.tum.de
Technical University of Munich
Munich, Germany

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

Abstract

We describe the architectural changes applied to `Strix`, a tool for LTL reactive synthesis, that were made in preparation for SYNTCOMP 2021. We replace the specialised translation from linear temporal logic (LTL) to deterministic parity automata (DPW) (as described in [8]) by a simpler and more general translation based on the recent Δ_2-normalisation for LTL by [13] and Zielonka split trees. Further, we make use of a new parity game solving algorithm by [14]. These changes simplify overall design, put the tool on a cleaner theoretical foundation, and improve the performance.

Keywords: Linear Temporal Logic, Reactive Synthesis, Parity Game, Deterministic Automaton

Design Principles

`Strix` is a tool for synthesis of reactive systems from linear temporal logic (LTL) specifications. At its core it translates specifications to deterministic parity automata (DPW), transforms them into parity games (PG), and solves them. `Strix` uses since its inception in [9] the following assumptions on LTL specifications occurring in practice and bases all design decisions on it:

1. Specifications are Boolean combinations of small LTL formulas which often belong to "simple" syntactic fragments.
2. Constructing the complete and explicitly represented deterministic automaton for an LTL specification is often infeasible.

From this `Strix` derives the following constraints: The core synthesis algorithm must use a LTL translation that is compositional for Boolean operations and that is on-the-fly. Thus all states of the constructed deterministic automaton and the extracted game are computed on-the-fly and in parallel to the parity game solver. Thus whenever a winner of the parity game can already be determined on the partial arena the construction is stopped.

Since an on-the-fly construction cannot incorporate any techniques that require an exploration of a complete SCC, which a priori requires the exploration of the complete automaton, techniques such as simulation-based minimisation are left aside. In practice, direct LTL to deterministic automata translations [3, 4] construct small automata without using additional minimisation techniques, can be implemented on-the-fly and are (partly) compositional.

Changes

We organise the detailed description of the changes using the four phases[1] implemented in `Strix`, which are: 1) Formula Rewriting, 2) Automaton Construction, 3) Winning Strategy Computation (runs in parallel with 2.), and 4) Controller Extraction (if the specification is realizable.).

The fundamental change that effects every phase is that we replace the specialised deterministic parity automaton construction from [8] that relies on [3, 4] by a construction based on the [13] and Zielonka trees.

Phase 1: Formula Rewriting

`Strix` as described in [8] applies a set of "folklore" LTL rewriting rules, e.g., $\mathbf{F}\,\mathbf{F}\,a \to \mathbf{F}\,a$, $\mathbf{F}\,a \wedge a \to a$, or $\mathbf{G}(a\,\mathbf{U}\,b) \to \mathbf{G}(a \vee b) \wedge \mathbf{G}\,\mathbf{F}\,b$, and furthermore replaces atomic propositions that have only a single polarity, i.e., appear only positive or negative (assuming the formula is in negation normal-form), by an appropriate constant, e.g., $a_i \vee \psi \to \psi$ if a_i is an input and does not occur in ψ and $a_o \vee \psi \to \mathbf{tt}$ if a_o is an output, respectively.

In addition to these two steps we now always rewrite the formula into an equivalent formula in Δ_2-normal-form using the procedures from [13, Theorem 23 and 27]. The class Δ_2 is part of the *syntactic future hierarchy* ([13, Figure 1b]) that classifies each LTL formula into Σ_i, Π_i, or Δ_i for some $i \geq 1$ according to the number of alternations of least-fixed- ($\mathbf{F}$, $\mathbf{U}$, $\mathbf{M}$) and greatest-fixed-point ($\mathbf{G}$, $\mathbf{W}$, $\mathbf{R}$) operators. Furthermore, note that Δ_i is the Boolean closure of Σ_i and Π_i. The class Σ_1 is commonly known as syntactic co-safe formulas, and the class Π_1 as syntactic safe formulas. Moreover, common formulas such as $\mathbf{G}\,\mathbf{F}\,\psi$ with $\psi \in \Sigma_1$ belong to Π_2 and $\mathbf{F}\,\mathbf{G}\,\psi$ with $\psi \in \Pi_1$ belong to Σ_2.

An important point for our implementation is that we we do not apply the Δ_2-normalisation from [13] on the whole formula, but only substitute temporal operators that do not belong to Δ_2.

Phase 2: Automaton Construction

`Strix` as described in [8] transforms the input formula φ into a tree where nodes are labelled by Boolean operations and leaves are labelled by LTL formulas. This done according a set of rules that ensures that there exists an "easy" product construction yielding a deterministic parity automaton (DPW). Each leaf is then translated to a deterministic

[1] We refer the reader to [8] for a detailed description of each phase.

automaton using a portfolio of different LTL translations ([3, 4]) from which a suitable one is picked depending on syntactic criteria. This tree is then used to obtain a single DPW using Boolean operations.

We replace this by a two-step procedure: First, we translate φ into a deterministic Emerson-Lei automaton (DELW) $\mathcal{A}$, and second, convert the automaton $\mathcal{A}$ to a deterministic parity automaton (DPW) using Zielonka trees.

Due to phase 1 the formula φ is in Δ_2 and thus a Boolean combination of formulas $\psi_1, \psi_2, \ldots, \psi_n$ from Π_2 and Σ_2. Each subformula ψ_i is now separately and directly translated to either a deterministic Büchi (DBW) or co-Büchi automaton (DCW) using the break-point construction specifically tailored for Π_2 and Σ_2 from [13]. We then use a (modified) product-construction[2] inheriting ideas of the product construction appearing in [8, 11], e.g., state-formulas with short-circuiting in order to remove components from the product automaton. We then obtain a deterministic Emerson-Lei automaton (DELW) whose states are propositional formulas over states of DBWs.

In the second step, we implement the alternating-cycle-decomposition construction (ACD) [2] and a yet unpublished adaption of Zielonka trees, called *conditional Zielonka trees* (ZLK)[3]. While ACD has strong optimality guarantees, it requires the complete exploration of a strongly-connected-component (SCC) and thus requires a full exploration of the DELW, which we want to avoid at all costs. Thus we use several syntactic checks derived from the state-labels that over-approximate SCCs and add a lookahead parameter limiting the number of states that are explored for ACD. If we cannot identify an SCC within the given state-budget we fallback to ZLK, but extract from the state-formula and state-labels information in order to simplify the acceptance condition to reduce the size of the Zielonka tree. Thus the user can choose how much effort should be into obtaining an optimal DPW for the constructed DELW. The second step bears some inspiration from the LTL synthesis tool `ltlsynt` [10] that uses [12] to translate DELWs to DPWs.

In summary, we simplified the fine-grained classification and intricate construction present in [8] by a uniform LTL→ DELW-translation in combination with a Zielonka-tree based transformation to DPW. Further, these new translations are implemented as part of `Owl` [5] and use a semi-symbolic representation of the transition relation where each edge is stored as a leaf in a multi-terminal binary decision diagram (MTBDD), speeding up computation on large alphabets.

Phase 3: Winning Strategy Computation

We replace the strategy iteration (SI) [7] with the distraction fixpoint iteration algorithm (DFI) [14] and modify it such that yields non-deterministic strategies similarly as in [6]. Further, we update the scoring-based exploration mechanism from [8] that guides the on-the-fly construction to support the new construction.

Phase 4: Controller Extraction

For the controller extraction we enhance the minimisation of the intermediate Mealy machines and improve the circuit encoding compared to the implementation described by [8].

The SAT-based minimisation consists now of two phases: First, we refine the non-deterministic winning strategy to a deterministic winning strategy that minimises the number of reachable states and that still has "don't cares" for outputs. Second, we pass this deterministic strategy represented as a Mealy machine to `MeMin` [1] to further compress it.

While "unstructured" encoding assigns each state of the Mealy machine a non-negative integer and uses the binary representation to encode a state into circuit, the "structured" [8] encoding uses knowledge about the state structure to obtain a more succinct encoding. As the states of the constructed DPW (and thus parity game and Mealy machine) are obtained by composing several smaller automata (DBW, DCW) and adding path information for the Zielonka trees, we can encode each component into a separate range of variables. Furthermore, each state of the underlying DBW is labelled by an LTL formula for the language recognised by the state. We map each formula to the set of temporal operators occurring in the formula, which we call "profile", and use a this to encode states into vector of variables. In the case that this yields an ambiguous mapping we add extra bits to distinguish the states that are mapped to the same profile. Using this technique, we can obtain for some of the SYNTCOMP benchmarks significant size reductions, e.g., the circuit realising `ltl2dba_Q_8.tlsf` we obtain with the "unstructured" encoding has around 34000 latches and gates[4], while using "structured" encoding we obtain a circuit with around 300 latches and gates.

Acknowledgments

The authors want to thank Michael Luttenberger for his support in development of `Strix`. Salomon Sickert is supported by the Deutsche Forschungsgemeinschaft (DFG) under project number (436811179) and partly funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement PaVeS (No 787367).

References

[1] Andreas Abel and Jan Reineke. 2015. MeMin: SAT-based Exact Minimization of Incompletely Specified Mealy Machines. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design,*

[2] A paper describing the construction will be published soon.

[3] A paper describing the construction will be published soon.

[4] The exact value depends on the numbering of states, which is nondeterministic.

ICCAD 2015, Austin, TX, USA, November 2-6, 2015. 94–101. https://doi.org/10.1109/ICCAD.2015.7372555

[2] Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. 2020. Optimal transformations of Muller conditions. *CoRR* abs/2011.13041 (2020). arXiv:2011.13041 https://arxiv.org/abs/2011.13041

[3] 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

[4] 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

[5] 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

[6] 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

[7] Michael Luttenberger. 2008. Strategy Iteration using Non-Deterministic Strategies for Solving Parity Games. *CoRR* abs/0806.2923 (2008). arXiv:0806.2923 http://arxiv.org/abs/0806.2923

[8] 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

[9] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. 2018. Strix: Explicit Reactive Synthesis Strikes Back!. In *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)*, Hana Chockler and Georg Weissenbacher (Eds.). Springer, 578–586. https://doi.org/10.1007/978-3-319-96145-3_31

[10] 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)*.

[11] David Müller and Salomon Sickert. 2017. LTL to Deterministic Emerson-Lei Automata. In *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September 2017 (EPTCS, Vol. 256)*, Patricia Bouyer, Andrea Orlandini, and Pierluigi San Pietro (Eds.). 180–194. https://doi.org/10.4204/EPTCS.256.13

[12] 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

[13] 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

[14] 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