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 $\Lambda_2$-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., $\forall x \phi \rightarrow \exists x \phi$, $\phi \land \psi \rightarrow \phi$, or $G(a \lor b) \rightarrow G(a \lor b) \land G \phi b$, 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., $a_i \lor \psi \rightarrow \psi$ if $a_i$ is an input and does not occur in $\phi$ and $a_0 \lor \psi \rightarrow \top$ if $a_0$ is an output, respectively.

In addition to these two steps we now always rewrite the formula into an equivalent formula in $\Lambda_2$-normal-form using the procedures from [13, Theorem 23 and 27]. The class $\Lambda_2$ is part of the syntactic future hierarchy ([13, Figure 1b]) that classifies each LTL formula into $\Sigma_i$, $\Pi_i$, or $\Delta_i$ for some $i \geq 1$ according to the number of alternations of least-fixed (F, U, M) and greatest-fixed-point (G, W, R) operators. Furthermore, note that $\Delta_1$ is the Boolean closure of $\Sigma_i$ and $\Pi_i$. The class $\Sigma_1$ is commonly known as syntactic co-safe formulas, and the class $\Pi_1$ as syntactic safe formulas. Moreover, common formulas such as $G \phi \psi$ with $\psi \in \Sigma_1$ belong to $\Pi_2$ and $F \phi \psi$ with $\psi \in \Pi_1$ belong to $\Sigma_2$.

An important point for our implementation is that we do not apply the $\Lambda_2$-normalisation from [13] on the whole formula, but only substitute temporal operators that do not belong to $\Lambda_2$.

Phase 2: Automaton Construction
Strix as described in [8] transforms the input formula $\phi$ 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 \( \varphi \) 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 \( \varphi \) is in \( \Delta_2 \) and thus a Boolean combination of formulas \( \psi_1, \psi_2, \ldots, \psi_n \) from \( \Pi_2 \) and \( \Sigma_2 \). Each subformula \( \psi_i \) 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 \( \Pi_2 \) and \( \Sigma_2 \) from [13]. We then use a (modified) product-construction\(^a\) 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)\(^b\). 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 \texttt{ltlsynth} [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\( \rightarrow \) DELW-translation in combination with a Zielonka-tree based transformation to DPW. Further, these new translations are implemented as part of \texttt{ow1} [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.

\textbf{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.

\textbf{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 \texttt{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 \texttt{1t12dbaq_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.

\textbf{Acknowledgments}

The authors want to thank Michael Luttenberger for his support in development of \texttt{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).

\textbf{References}


\(^a\)A paper describing the construction will be published soon.

\(^b\)A paper describing the construction will be published soon.

\(^4\)The exact value depends on the numbering of states, which is nondeterministic.