LydiaSyft: A Compositional Symbolic Synthesizer for LTL$_f$ Specifications*

Marco Favorito
Banca d'Italia
marco.favorito@bancaditalia.it

Shufang Zhu
University of Oxford
shufang.zhu@cs.ox.ac.uk

We present LydiaSyft, a compositional and symbolic synthesizer for specifications expressed in Linear Temporal Logic on finite traces (LTL$_f$). LydiaSyft exploits the classical backward LTL$_f$ synthesis approach by first employing a fully compositional approach to construct the corresponding Deterministic Finite Automaton (DFA) of LTL$_f$ specification, represented symbolically in Binary Decision Diagrams (BDDs), then solves an adversarial reachability game on the symbolic DFA to abstract a winning strategy, if exists.

1 Introduction

Reactive synthesis promises to automatically generate a verifiably correct program from a high-level specification [9]. Recently, there have been more and more interests on synthesis for specifications expressed in Linear Temporal Logic on finite traces (LTL$_f$), a finite-trace variant of LTL [4]. Roughly speaking, we consider an alphabet of propositions partitioned into those controlled by the agent (one may think of these as a binary encoding of agent actions) and those controlled by the environment (one may think of these as fluents), and then we use LTL$_f$ to specify which finite traces are desirable. The outcome of the synthesis procedure is a program (a finite-state controller) that at every time step, given the values of the environment propositions in the history so far, sets the next value of the agent propositions so that the traces generated satisfy the LTL$_f$ specification [5].

The classical approach to LTL$_f$ synthesis is based on first constructing a Deterministic Finite Automaton (DFA) corresponding to the LTL$_f$ specification, and then considering it as a game arena where the agent tries to get to an accepting state in spite that the environment tries to avoid it. A winning strategy, which is a finite-state controller returned by the synthesis procedure, can be obtained through a backward fixpoint computation for *adversarial reachability* of the DFA accepting states [5].

In this paper, we present a compositional and symbolic solver LydiaSyft for LTL$_f$ synthesis. First of all, LydiaSyft is integrated with a fully compositional approach to handle LTL$_f$ formulae. That is, regardless of the structure of the LTL$_f$ specification, LydiaSyft processes all the subformulae recursively up to the leaves of the syntax tree and then compose the partial DFAs of the subformulae using common operations over automata (e.g. union, intersection, concatenation), according to the LTL$_f$ operator being processed [3]. After obtaining the DFA of the LTL$_f$ specification, LydiaSyft represents the DFA in a symbolic form using Binary Decision Diagrams (BDDs) and solves a symbolic adversarial reachability game on the DFA through a backward fixpoint computation. If the synthesis problem is realizable, a winning strategy is abstracted using Boolean functional synthesis [12].

*Both authors are corresponding author.

Submitted to:
SYNTCOMP 2023

2 The Tool LydiaSyft

LydiaSyft is an open-source tool implemented in C++11. More specifically, LydiaSyft uses Syfco to parse the synthesis problems described in TLSF format [7] to obtain the LTL_f specification and the partition of agent/environment propositions. LydiaSyft integrates the preprocessing techniques presented in [11] to perform one-step realizability/unrealizability checks, which is implemented using Z3 [8], at the beginning of the synthesis procedure. If neither one-step check succeeds, Lydia [3] is adopted as the backend to construct the explicit-state DFA, which is later synthesized with respect to the partitioned propositions adopting Syft [12]. We use the BDD library CUDD-3.0.0 [10] for the symbolic DFA representation. In this section, we introduce the usage and the architecture of LydiaSyft.

2.1 Usage

The command to execute LydiaSyft in Linux system is `./LydiaSyft [options]`.

-f,--spec-file The synthesis specification file in TLSF format.

-p,--print-strategy Print out the synthesized strategy in .dot form (default: False).

-t,--print-times Print out running times of each step (default: false).

2.2 Compositional DFA Construction (Lydia) [3]

Given a LTL_f formula φ, the compositional DFA construction works by inductively applying a transformation procedure to each subformula. The approach is "bottom-up": it computes the DFA of the deepest subformulas, and combines the partial results depending on the LTL_f operator under transformation. This is in contrast with the previous techniques known in the literature that are "top-down": they proceed from the root operator of the formula in order to compute the next states (see e.g. LDL_f2NFA in [5, 2]).

Since the transformation rules are defined over Linear Dynamic Logic on finite traces (LDL_f) [4], the input LTL_f formula φ is first translated into an equivalent (linear-size) LDL_f formula. The elementary formula *tt* (resp. *ff*) is translated into a DFA with only one accepting (resp. rejecting) state with a self-loop. Boolean operators are processed with the analogous automata-theoretic operations: e.g. conjunction is implemented as automata intersection, disjunction as union, and negation as complementation. The temporal operator $\langle\rho\rangle\psi$ is handled according to the regular expression ρ. Due to lack of space, we cannot describe it in full detail, and the interested reader should refer to [3].

On the implementation side, Lydia uses the semi-symbolic DFA representation provided by Mona [6]. In Mona, the transitions of a DFA are symbolically represented as a shared multi-terminal binary decision diagram (shMBDD), where the transition relation of a DFA is encoded as a binary decision diagram (BDD) with multiple terminal nodes. The alphabets of these DFAs are the sets of bit vectors of length k, i.e. $\mathbb{B}^k$, for some k. In our case, each bit is associated to an atomic proposition appearing in the LDL_f formula. In addition to a compact representation on transitions of DFAs, the Mona DFA library provides efficient implementations of standard automata operations. These operations include product, (existential) projection, determinization, and minimization. We extended the library so to include the Kleene closure, the concatenation, and the universal projection. Intuitively, these operations are needed in the modeling of nondeterminism of the $\mathscr{U}$-operator semantics.

2.3 Symbolic Adversarial Reachability Game (Syft) [12]

We start by defining the concept of *symbolic automaton*:

Definition 2.1 (Symbolic Automaton)**.** Given a DFA $\mathscr{G} = (2^{\mathscr{X}\cup\mathscr{Y}}, S, s_0, \delta, F)$, the corresponding symbolic automaton $\mathscr{F} = (\mathscr{X}, \mathscr{Y}, \mathscr{Z}, Z_0, \eta, f)$ is defined as follows:

- $\mathscr{X}$ and $\mathscr{Y}$ are as defined for $\mathscr{G}$;
- $\mathscr{Z}$ is a set of $\lceil \log_2 |S| \rceil$ new propositions such that every state $s \in S$ corresponds to an interpretation $Z \in 2^{\mathscr{Z}}$;
- $Z_0 \in 2^{\mathscr{Z}}$ is an interpretation of the propositions in $\mathscr{Z}$ corresponding to the initial state s_0;
- $\eta : 2^{\mathscr{X}} \times 2^{\mathscr{Y}} \times 2^{\mathscr{Z}} \to 2^{\mathscr{Z}}$ is a boolean function mapping interpretations X, Y and Z of the propositions of $\mathscr{X}$, $\mathscr{Y}$ and $\mathscr{Z}$ to a new interpretation Z' of the propositions of $\mathscr{Z}$, such that if Z corresponds to a state $s \in S$ then Z' corresponds to the state $\delta(s, X \cup Y)$;
- f is a boolean formula over the propositions in $\mathscr{Z}$, such that f is satisfied by an interpretation Z iff Z corresponds to a final state $s \in F$.

Intuitively, the *symbolic automaton* represents states by propositional interpretations, the transition function by a boolean function and the set of final states by a boolean formula.

To solve the realizability problem over a symbolic automaton we compute a boolean formula w over $\mathscr{Z}$ that is satisfied exactly by those interpretations that correspond to winning states. The specification is realizable if and only if Z_0 satisfies w. To solve the synthesis problem, we compute a boolean function $\tau : 2^{\mathscr{Z}} \to 2^{\mathscr{Y}}$ such that for any sequence $(X_0, Y_0, Z_0), (X_1, Y_1, Z_1), \ldots$ that satisfies: (1) Z_0 is the initial state; (2) For every $i \geq 0$, $Y_i = \tau(Z_i)$; (3) For every $i \geq 0$, $Z_{i+1} = \eta(X_i, Y_i, Z_i)$ there exists an i such that Z_i satisfies f. In other words, starting from the initial state, for any sequence of uncontrollable variables, if the controllable variables are computed by τ and the next state is computed by η, the play eventually reaches an accepting state.

2.3.1 Realizability and Synthesis over Symbolic Automata

We can compute w and τ through a fixpoint computation over two boolean formulas: w_i, over the set of propositions $\mathscr{Z}$, and t_i, over $\mathscr{Z} \cup \mathscr{Y}$. These formulas encode winning states and winning outputs in the following way: every interpretation $Z \in 2^{\mathscr{Z}}$ such that $Z \models w_i$ corresponds to a winning state, and every interpretation $(Z, Y) \in 2^{\mathscr{Z}} \times 2^{\mathscr{Y}}$ such that $(Z, Y) \models t_i$ corresponds to a winning state together with a winning output of that state. When we reach a fixpoint, w_i should encode all winning states and t_i all pairs of winning states and winning outputs.

In the procedure below, we compute the fixpoints of w_i and t_i starting from w_0 and t_0. We assume that we are able to perform basic Boolean operations over the formulas, as well as substitution, quantification and testing for logical equivalence of two formulas. In the first step of the computation, we initialize $t_0(Z, Y) = f(Z)$ and $w_0(Z) = f(Z)$, since every accepting state is a winning state. Note that t_0 is independent of the propositions from $\mathscr{Y}$, since once the play reaches an accepting state the game is over and we don't care about the outputs anymore. Then we construct t_{i+1} and w_{i+1} as follows:

$$t_{i+1}(Z, Y) = t_i(Z, Y) \vee (\neg w_i(Z) \wedge \forall X. w_i(\eta(X, Y, Z)))$$
$$w_{i+1}(Z) = \exists Y. t_{i+1}(Z, Y)$$

An interpretation $(Z, Y) \in 2^{\mathscr{Z}} \times 2^{\mathscr{Y}}$ satisfies t_{i+1} if either: (Z, Y) satisfies t_i; or Z was not yet identified as a winning state, and for every input X we can move from Z to an already-identified winning state by setting the output to Y. Note that it is important in the second case that Z has not yet been identified as a winning state, because it guarantees that the next transition will move closer to the accepting states. Otherwise, it would be possible, for example, for t_{i+1} to accept an assignment to Y that moves from Z back to itself, making the play stuck in a self loop. From t_{i+1}, we can construct w_{i+1} by existentially

quantifying the output variables. This means that w_{i+1} is satisfied by all interpretations $Z \in \mathscr{Z}$ that satisfy t_{i+1} for some output, ignoring what the output is. The computation reaches a fixpoint when $w_{i+1} \equiv w_i$ ($\equiv$ denoting logical equivalence). At this point, no more states will be added, and so all winning states have been found. By evaluating w_i on Z_0 we can know if there exists a winning strategy. If that is the case, t_i can be used to compute this strategy. This can be done through the mechanism of boolean synthesis.

By giving t_i as the input formula to a Boolean synthesis procedure, and setting $\mathscr{Z}$ as the input variables and $\mathscr{Y}$ as the output variables, we get back a function $\tau : 2^{\mathscr{Z}} \to 2^{\mathscr{Y}}$ such that $(Z, \tau(Z)) \models t_i$ if and only if there exists $Y \in 2^{\mathscr{Y}}$ such that $(Z,Y) \models t_i$. Using τ, we can define a *symbolic finite-state controller* $\mathscr{H}$ corresponding to the winning strategy of the DFA game. For more details of the BDD-based implementation of the synthesis technique, we refer to [12].

3 Conclusion

We presented in this paper the tool LydiaSyft, a compositional symbolic synthesizer for LTL_f specifications. We believe that there are a lot of interesting directions to improve the performance of LydiaSyft. A promising direction is to extend the composition methodology from the DFA construction level to the synthesis level, as in [1] for Safety LTL synthesis, a safety fragment of LTL.

Acknowledgments

We thank the contributions of all the co-authors: Giuseppe De Giacomo, Jianwen Li, Geguang Pu, Lucas M. Tabajara and Moshe Y. Vardi. This work is supported by the ERC Advanced Grant WhiteMech (No. 834228).

References

[1] Suguman Bansal, Giuseppe De Giacomo, Antonio Di Stasio, Yong Li, Moshe Y. Vardi & Shufang Zhu (2022): *Compositional Safety LTL Synthesis*. In: *VSTTE*, Springer, pp. 1–19.

[2] R. I. Brafman, G. De Giacomo & F. Patrizi (2018): *LTL_f/LDL_f Non-Markovian Rewards*. In: *AAAI*.

[3] Giuseppe De Giacomo & Marco Favorito (2021): *Compositional Approach to Translate LTL_f/LDL_f into Deterministic Finite Automata*. In: *ICAPS*.

[4] Giuseppe De Giacomo & Moshe Y. Vardi (2013): *Linear Temporal Logic and Linear Dynamic Logic on Finite Traces*. In: *IJCAI*.

[5] Giuseppe De Giacomo & Moshe Y. Vardi (2015): *Synthesis for LTL and LDL on Finite Traces*. In: *IJCAI*.

[6] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe & A. Sandholm (1995): *Mona: Monadic Second-order Logic in Practice*. In: *TACAS*.

[7] Swen Jacobs, Guillermo A. Perez & Philipp Schlehuber-Caissier (2023): *The Temporal Logic Synthesis Format TLSF v1.2*. arXiv:2303.03839.

[8] Leonardo de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver*. In: *TACAS*, pp. 337–340.

[9] A. Pnueli & R. Rosner (1989): *On the Synthesis of a Reactive Module*. In: *POPL*.

[10] Fabio Somenzi (2016): *CUDD: CU Decision Diagram Package 3.0.0. Universiy of Colorado at Boulder*.

[11] Shengpin Xiao, Jianwen Li, Shufang Zhu, Yingying Shi, Geguang Pu & Moshe Y. Vardi (2021): *On-the-fly Synthesis for LTL over Finite Traces*. In: *AAAI*.

[12] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu & Moshe Y. Vardi (2017): *Symbolic LTL$_f$ Synthesis*. In: *IJCAI*.