Symbolic Approaches to LTL\(_f\) Best-Effort Synthesis

Giuseppe De Giacomo, Gianmarco Parretti and Shufang Zhu
Sapienza Univ. Roma, Rome, Italy
{degiacomo, zhu}@diag.uniroma1.it, parretti.165064@studenti.uniroma1.it

Abstract

We consider an agent acting to fulfil tasks in a nondeterministic environment. When a strategy that fulfills the task regardless of how the environment acts does not exist, the agent should at least avoid adopting strategies that prevent from fulfilling its task. Best-effort synthesis captures this intuition. In this paper, we devise and compare various symbolic approaches for best-effort synthesis in Linear Temporal Logic on finite traces (LTL\(_f\)). These approaches are based on the same basic components, however they change in how these components are combined, and this has a significant impact on the performance of the approaches as confirmed by our empirical evaluations.

1 Introduction

We consider an agent acting to fulfil tasks in a nondeterministic environment, as considered in Planning in nondeterministic (adversarial) domains [Cimatti et al., 2003; Ghallab et al., 2004], except that we specify both the environment and the task in Linear Time Logic (LTL) [Aminof et al., 2019], the formalism typically used for specifying complex dynamic properties in Formal Methods [Baier et al., 2008].

In fact, in this paper we consider Linear Time Logic on finite traces (LTL\(_f\)) [De Giacomo and Vardi, 2013; De Giacomo and Vardi, 2015], which maintains the syntax of LTL [Pnueli, 1977], but is interpreted on finite traces. In this setting, we study synthesis [Pnueli and Rosner, 1989; Finkbeiner, 2016; De Giacomo and Vardi, 2015; Aminof et al., 2019], a general form of planning. In particular, we look at how to synthesize a strategy that is guaranteed to satisfy the task against all environment behaviors that conform to the environment specification.

When a winning strategy that fulfills the agent task regardless of how the environment acts does not exist, the agent should at least avoid adopting strategies that prevent from fulfilling its task. Best-effort synthesis captures this intuition. More precisely, best-effort synthesis captures the game-theoretic rationality principle that a player would not use a strategy that is “dominated” by another of its strategies (i.e., if the other strategy would fulfill the task against more environment behaviors than the one chosen by the player). Aminof et al. [2021b] studied best effort strategies. Best-effort strategies have some notable properties: (i) they always exist, (ii) if a winning strategy exists then best-effort strategies are exactly the winning strategies, (iii) best-effort strategies can be computed in 2EXPTIME as computing winning strategies (best-effort synthesis is indeed 2EXPTIME-complete).

Aminof et al. [2021b] presented algorithms for best-effort synthesis in LTL and LTL\(_f\). These algorithms are based on creating, solving and combining the solutions of three distinct games, but of the same game arena. The arena is obtained from the automata corresponding to the formulas \(E\) and \(\varphi\) constituting the environment and the task specifications.

In particular, the algorithm for LTL\(_f\) best-effort synthesis appears to be quite promising in practice, since well-performing techniques for each component of the algorithm are available in the literature. These components are: (i) transformation of the LTL\(_f\) formulas \(E\) and \(\varphi\) into deterministic finite automata (DFA), which can be double-exponential in the worst case, but for which various good techniques have been developed [Henriksen et al., 1995; Zhu et al., 2017; Bansal et al., 2020; De Giacomo and Favorito, 2021b]; (ii) Cartesian product of DFAs, which is polynomial; (iii) minimization of DFAs, which is also polynomial; (iv) fixpoint computation over DFA to compute adversarial and cooperative winning strategies for reaching the final states, which is again polynomial.

In this paper, we refine the LTL\(_f\) best-effort synthesis presented by Aminof et al. [2021b] by using symbolic techniques [Bryant, 1992; Baier et al., 2008; Zhu et al., 2017]. In particular, we show three different symbolic approaches that combine the above operations in different ways (and in fact allow for different level of minimization). We then compare the three approaches through empirical evaluations. From this comparison, a clear winner emerges. Interestingly, the winner does not fully exploit DFA minimization to minimize the DFA whenever it is possible. Instead, this approach uses uniformly the same arena for all the three games (hence giving up on minimization at some level). Finally, it turns out that the winner performs better in computing best-effort solutions even than state-of-the-art tools that compute only adversarial and cooperative solutions. These findings confirm that LTL\(_f\) best-effort synthesis is indeed well suited for efficient and scalable implementations.

The rest of the paper is organized as follows. In Section 2,
we remind the main notions of \( \text{LTL}_f \) synthesis, and \( \text{LTL}_f \) synthesis under environment assumption. In Section 3, we discuss \( \text{LTL}_f \) best-effort synthesis, and the algorithm by Aminof et al. [2021b]. In Section 4, we introduce three distinct symbolic approaches for \( \text{LTL}_f \) best-effort synthesis: the first (c.f., Subsection 4.2) is a direct symbolic implementation of the algorithm by [Aminof et al., 2021b]; the second one (c.f., Subsection 4.3) favors maximally conducting DFA minimization, thus getting the smallest possible arenas for the three games; and the third one (c.f., Subsection 4.4) gives up DFA minimization at some level, and creates a single arena for the three games. In Section 5, we perform an empirical evaluation of the three algorithms. We conclude the paper in Section 6.

2 Preliminaries

\textit{LTL Basics.} Linear Temporal Logic on finite traces (\( \text{LTL}_f \)) is a specification language to express temporal properties on finite traces [De Giacomo and Vardi, 2013]. In particular, \( \text{LTL}_f \) has the same syntax as \( \text{LTL} \), which is instead interpreted over infinite traces [Pnueli, 1977]. Given a set of propositions \( \text{Prop} \), \( \text{LTL}_f \) formulas are generated as follows: 

\( \phi := a \mid (\phi \land \phi) \mid (\neg \phi) \mid (\phi \lor \phi) \mid (\phi \land \pi) \mid \text{true} \mid \text{false} \),

where \( a \in \text{Prop} \) is an atom, \( \land \) (Next), and \( \lor \) (Until) are temporal operators. We make use of standard Boolean abbreviations such as \( \lor \) (or) and \( \rightarrow \) (implies), \text{true} and \text{false}. In addition, we define the following abbreviations Weak Next

\( \diamond \phi \equiv \neg \Box \neg \phi \). Eventually \( \phi \equiv \text{true} \lor \phi \) and Always \( \Box \phi \equiv \text{false} \lor \phi \), where \( \Box \) is for Release.

A trace \( \pi = \pi_0 \pi_1 \ldots \) is a sequence of propositional interpretations (sets), where for every \( i \geq 0 \), \( \pi_i \in 2^{\text{Prop}} \) is the \( i \)-th interpretation of \( \pi \). Intuitively, \( \pi_i \) is interpreted as the set of propositions that are true at instant \( i \). We denote the last instant (i.e., index) in a trace \( \pi \) by \( \text{lst}(\pi) \). A trace \( \pi \) is an infinite trace if \( \text{lst}(\pi) = \infty \), which is formally denoted as \( \pi \in (2^{\text{Prop}})^\infty \); otherwise \( \pi \) is a finite trace, denoted as \( \pi \in (2^{\text{Prop}})^* \). Moreover, by \( \pi^k = \pi_0 \cdots \pi_k \) we denote the prefix of \( \pi \) up to the \( k \)-th iteration, and \( \pi^k = \epsilon \) denotes an empty trace if \( k < 0 \). \( \text{LTL}_f \) formulas are interpreted over finite, nonempty traces. Given \( \pi \), we define when an \( \text{LTL}_f \) formula \( \phi \) holds at instant \( i \), \( 0 \leq i \leq \text{lst}(\pi) \), written as \( \pi, i \models \phi \), inductively on the structure of \( \phi \), as:

\( \pi, i \models \phi \) if \( a \models \phi \) for \( a \in \text{Prop} \);

\( \pi, i \models \neg \phi \) iff \( \pi, i \not\models \phi \);

\( \pi, i \models \phi_1 \land \phi_2 \) iff \( \pi, i \models \phi_1 \) and \( \pi, i \models \phi_2 \);

\( \pi, i \models \phi_1 \lor \phi_2 \) iff \( \pi, i \models \phi_1 \) or \( \pi, i \models \phi_2 \); and \( \pi, i \models \phi_1 \land \phi_2 \) iff \( \exists j \geq i \) such that \( \pi, j \models \phi_1 \) and \( \pi, j \models \phi_2 \).

We say \( \pi \) satisfies \( \phi \), written as \( \pi \models \phi \), if \( \pi, 0 \models \phi \).

Reactive Synthesis Under Environment Assumptions. Reactive synthesis is the problem of finding a strategy that allows an agent to achieve a goal while continuously interacting with the external environment. In reactive synthesis under environment assumptions, we can intuitively think of the environment assumption as the knowledge the agent knows a-priori about how the environment can enact. Therefore, the environment behaviors are restricted to those that are consistent with the environment assumption. In this work, we specify environment assumptions and agent goals as \( \text{LTL}_f \) formulas.

An agent strategy is a function \( \sigma_{ag} : (2^X)^+ \rightarrow 2^X \) that maps non-empty sequences of environment choices to an agent choice. Similarly, an environment strategy is a function \( \sigma_{env} : (2^Y)^* \rightarrow 2^Y \) mapping (possibly empty) sequences of agent choices to an environment choice. A play \( \pi = (X_0 \cup Y_0)(X_1 \cup Y_1) \ldots \) is \( \sigma_{ag} \)-consistent if \( Y_k = \sigma_{ag}(X_0, X_1 \ldots, X_k) \) for all \( k \geq 0 \). Similarly, \( \pi \) is \( \sigma_{env} \)-consistent if \( X_0 = \sigma_{env}(\epsilon) \), where \( \epsilon \) denotes the empty trace, and \( X_k = \sigma_{env}(Y_0, Y_1 \ldots Y_{k-1}) \) for all \( k > 0 \). We denote by \( \pi(\sigma_{ag}, \sigma_{env}) \) the unique trace that is consistent with strategies \( \sigma_{ag} \) and \( \sigma_{env} \).

Consider \( \text{LTL}_f \) formula \( \phi \) over \( X \cup Y \), we say that agent strategy \( \sigma_{ag} \) realizes \( \phi \), denoted as \( \sigma_{ag} \models \phi \), if \( \pi(\sigma_{ag}, \sigma_{env}) \models \phi \) holds for every environment strategy \( \sigma_{env} \). \( \phi \) is agent realizable if there exists an agent strategy \( \sigma_{ag} \) that realizes \( \phi \). Environment realizability is defined similarly. Furthermore, we say that agent strategy \( \sigma_{ag} \) cooperatively realizes \( \phi \), denoted as \( \sigma_{ag} \sim \phi \), if there exists an environment strategy \( \sigma_{env} \) such that \( \pi(\sigma_{ag}, \sigma_{env}) \models \phi \). Hence, \( \phi \) is agent cooperatively realizable if such an agent strategy exists. Cooperative environment realizability is defined similarly.

We define the problem of \( \text{LTL}_f \) reactive synthesis under environment assumptions as follows.

**Definition 1.** The \( \text{LTL}_f \) Reactive synthesis under environment assumptions problem is defined as a pair \( \mathcal{P} = (\mathcal{E}, \phi) \), where \( \text{LTL}_f \) formula \( \mathcal{E} \) is an environment assumption, and \( \text{LTL}_f \) formula \( \phi \) is the agent goal. Realizability of \( \mathcal{P} \) checks whether there exists an agent strategy \( \sigma_{ag} \) that realizes \( \phi \) under \( \mathcal{E} \), i.e.,

\( \forall \sigma_{env} \models \mathcal{E}, \pi(\sigma_{ag}, \sigma_{env}) \models \phi \).

Synthesis of \( \mathcal{P} \) computes such a strategy if exists.

A naive approach to this problem is a reduction to standard synthesis of \( \text{LTL}_f \) formula \( \mathcal{E} \rightarrow \phi \) [Aminof et al., 2018]. Moreover, it has been shown that the problem of \( \text{LTL}_f \) reactive synthesis under environment assumptions is \( 2\text{EXPTIME} \)-complete [Aminof et al., 2018].

3 Best-effort Synthesis Under Environment Assumptions

Reactive synthesis, in general, considers that the environment exhibits an adversarial behavior, thus aims at computing an agent strategy such that no matter how the environment behaves, the agent will achieve the goal. In this setting, the agent, instead of keep trying, just give up when the synthesis procedure returns unrealizable, although the environment can be possibly "over-approximated". In this section, we synthesize a strategy ensuring that the agent will do nothing that would needlessly prevent it from achieving its task – which we call a best-effort strategy. Best-effort synthesis is the problem of finding such a strategy [Aminof et al., 2021b]. As for reactive synthesis under environment assumptions, in best-effort synthesis environment behaviors are restricted to those that are consistent with the environment assumption.

We start with showing how a strategy makes more effort with respect to another. Let \( \mathcal{E} \) and \( \phi \) be two \( \text{LTL}_f \) formulas.
denoting an environment assumption and an agent goal, respectively, and σ₁ and σ₂ be two agent strategies. We say that σ₁ dominates σ₂ for goal φ under E, written σ₁ ≥<sub>E</sub>σ₂, if for every σ<sub>env</sub> ∈ E, π(σ₂, σ<sub>env</sub>) |= φ implies π(σ₁, σ<sub>env</sub>) |= φ. Furthermore, we say that σ₁ strictly dominates σ₂, written σ₁ ><sub>E</sub>σ₂, if σ₁ ≥<sub>E</sub>σ₂ and σ₂ ∉ Z<sub>E</sub>σ₁. An agent strategy σ* is best-effort, or maximal, for the goal φ under the environment assumption E, if there is no agent strategy σ such that σ ><sub>E</sub>σ*. Intuitively, σ₁ ><sub>E</sub>σ₂ means σ₁ does at least as well as σ₂ against every environment strategy realizing E and strictly better against one such strategy. If σ₁ strictly dominates σ₂, then σ₁ makes more effort than σ₂ to satisfy the goal. In fact, if σ₂ is strictly dominated by σ₁, then an agent that uses σ₂ does not its best to achieve the goal: if it used σ₁ instead, it could achieve the goal against a strictly larger set of environment behaviors. Then, since best-effort strategies achieve the goal against the largest possible set of environment behaviors, an agent strategy is best-effort if it is not strictly dominated by another strategy.

We define the problem of LTLₜ best-effort synthesis under environment assumptions as follows:

**Definition 2.** The LTLₜ best-effort synthesis problem under environment assumptions is defined as a pair P = (E, φ), where LTLₜ formula E is an environment assumption, and LTLₜ formula φ is the agent goal. Best-effort synthesis of P computes an agent strategy that is best-effort.

In standard synthesis under assumptions, it is common to first check whether the problem is realizable, indicating that there exists an agent strategy that realizes the problem. However, it has been shown that in the best-effort synthesis, there always exists an agent strategy that is best-effort for the goal φ under environment assumption E.

**Theorem 1** ([Aminof et al., 2021b]). Let P = (E, φ) be an LTLₜ best-effort synthesis problem, there always exists an agent strategy that is best-effort.

Aminof et al. [2021b] presented that LTLₜ best-effort synthesis can be solved by a reduction to suitable DFA games, and it has been shown to be 2EXPTIME-complete [Aminof et al., 2021b].

**DFA Games.** A DFA game is a two-player game played on a DFA, described as a pair (D, F), where D is a deterministic transition system such that D = (2<sup>X∪Y</sup>, S, s₀, δ), where δ: S × 2<sup>X∪Y</sup> → S is the transition function and F ⊆ S is the set of the final states of the game. Given a play π = (X₀ ∪ Y₀)(X₁ ∪ Y₁)... ∈ (2<sup>Χ∪Y</sup>)<sup>ω</sup>, running π on D gives us an infinite sequence ρ = s₀s₁... ∈ S<sup>ω</sup> such that s₀ is the initial state and s₁+₁ = δ(sᵢ, Xᵢ ∪ Yᵢ) for all i ≥ 0. Since the transitions in D are all deterministic, we thus denote by ρ = Run(π, D) the unique sequence of running π on D. Analogously, we denote by ρ<sup>k</sup> = Run(π<sup>k</sup>, D) the unique finite sequence of running π<sup>k</sup> on D, and ρ<sup>ℓ</sup> = s₀s₁...s<sub>ℓ</sub> for k ≥ 0. The set of final states F indicates the states in which the agent wins the game. Then, a play π ∈ (2<sup>Χ∪Y</sup>)<sup>ω</sup> is a winning play for the agent if ρ = Run(π, D) and ∃ℓ ≥ 0: ρ<sub>ℓ</sub> ∈ F. Intuitively, DFA games require F to be visited at least once.

Given σ<sub>ag</sub> and σ<sub>env</sub> denoting an agent strategy and an environment strategy, respectively, we denote by PLAY(σ<sub>ag</sub>, σ<sub>env</sub>) the unique play that is consistent with both σ<sub>ag</sub> and σ<sub>env</sub>. An agent strategy σ<sub>ag</sub> is winning in the game (D, F) if ∀σ<sub>env</sub> it results that PLAY(σ<sub>ag</sub>, σ<sub>env</sub>) is winning. An environment strategy σ<sub>env</sub> is winning in the game (D, F) if ∀σ<sub>ag</sub> it results that PLAY(σ<sub>ag</sub>, σ<sub>env</sub>) is not winning for the agent.

In DFA games, s ∈ S is a winning state for the agent (resp. environment) if the agent (resp. the environment) has a winning strategy in the game (D', F), where D' = (2<sup>Χ∪Y</sup>, S, s₀, δ), i.e., the same area D but with the new initial state s. By W<sub>ag</sub>(D, F) (resp. W<sub>env</sub>(D, F)) we denote the set of all agent (resp. environment) winning states. Intuitively, W<sub>ag</sub> represents the "agent winning region", from which the agent is able to win the game, no matter how the environment behaves.

We also define cooperatively winning strategies for DFA games. An agent strategy σ<sub>ag</sub> is cooperatively winning in game (D, F) if ∃σ<sub>env</sub> such that PLAY(σ<sub>ag</sub>, σ<sub>env</sub>) is a winning play. An analogous definition holds for cooperatively winning environment strategies. Hence, s ∈ S is a cooperatively winning state for the agent (resp. environment) if the agent (resp. the environment) has a cooperatively winning strategy in the game (D', F), where D' = (2<sup>Χ∪Y</sup>, S, s₀, δ). By W<sub>coop</sub>(D, F) (resp. W<sub>coop</sub>(D, F)) we denote the set of all agent (resp. environment) cooperative winning states.

If the agent makes its choices based only on the current state of the game and on the choice of the environment, then we say that the agent uses a positional strategy. An agent positional strategy (a.k.a. agent memory-less strategy) is a function τ<sub>ag</sub>: S × 2<sup>X</sup> → 2<sup>X</sup>. Similarly, we can define an environment positional strategy as a function τ<sub>env</sub>: S → 2<sup>X</sup>. An agent positional strategy τ<sub>ag</sub> induces an agent strategy σ<sub>ag</sub> as follows:

σ<sub>ag</sub>(X₀) = τ<sub>ag</sub>(s₀, X₀) and for k ≥ 0

σ<sub>ag</sub>(X₀X₁...X<sub>k</sub>) = τ<sub>ag</sub>(s<sub>k</sub>, X<sub>k</sub>) such that s<sub>k</sub> is the last state in the finite sequence ρ<sup>k</sup> = Run(π<sup>k</sup>−1, D). Similarly, an environment positional strategy induces an environment strategy. An agent (resp. environment) positional strategy is winning if the agent (resp. environment) strategy it induces is winning. A positional strategy for a player that is winning from every state in its winning region is called uniform winning. Similarly, a positional strategy for a player that is cooperatively winning if the strategy it induces is cooperatively winning. Finally, a positional strategy for a player that is winning from every state in the cooperatively winning region is called uniform cooperatively winning.

The solution to LTLₜ best-effort synthesis presented by Aminof et al. [2021b] can be summarized as follows.

**Algorithm 0** [Aminof et al., 2021b]. Given LTLₜ best-effort synthesis problem P = (E, φ), proceed as follows:

1. For every ξ ∈ {¬E, E ∨ φ, E ∧ φ} compute the DFAs A<sub>ξ</sub> = (D<sub>ξ</sub>, F<sub>ξ</sub>).
2. Form the product D = D<sub>¬E</sub> × D<sub>E</sub>φ × D<sub>E∧F</sub>. Lift the final states of each component to the product i.e. if A<sub>ξ</sub> = (D<sub>ξ</sub>, F<sub>ξ</sub>) is the DFA for ξ, then the lifted condition G<sub>ξ</sub> consists of all states(s, s<sub>ξ</sub>, s<sub>ξφ</sub>, s<sub>ξ∧φ</sub>) s.t. s<sub>ξ</sub> ∈ F<sub>ξ</sub>
3. In the DFA game (D, G<sub>ξ</sub>φ) compute a uniform positional winning strategy f<sub>ag</sub>. Let W<sub>ag</sub> ⊆ S be the agent’s winning region.
4. In the DFA game \((D, G_{\neg \varphi})\) compute the environment’s winning region \(V \subseteq Q\).

5. Compute the environment restriction \(D'\) of \(D\) to \(V\).

6. In the DFA game \((D', G_{\varphi \land \psi})\) find a uniform cooperatively winning uniform positional strategy \(g_{ag}\).

7. Return the agent strategy induced by the positional strategy \(k_{ag}(s, X') = f_{ag}(s, X')\) if \(s \in W\), and \(k_{ag}(s, X') = g_{ag}(s, X')\) otherwise.

4 Symbolic Best-effort Synthesis

In this section, we present our symbolic approaches to \(\text{LTL}_f\) best-effort synthesis, consisting of 3 different versions, namely monolithic, explicit-compositional, and symbolic-compositional. In particular, we exploit the symbolic techniques presented by Zhu et al. [2017] to deal with DFA games

4.1 Symbolic DFA Games

We consider the DFA game representation described in 3 is an explicit-state representation. Instead, we are able to represent a DFA game compactly in a symbolic way. More specifically, the game arena \(D = (2^{X \cup Y \cup Z}, S, s_0, \delta)\) can be represented symbolically, by encoding the state space using a logarithmic number of propositions. The symbolic representation of \(D\) is a tuple \(D^s = (X, Y, Z, Z_0, \eta)\), where \(Z\) is a set of variables such that \(|Z| = \lceil \log |S| \rceil\), and every state \(s \in S\) corresponds to an interpretation \(Z \in 2^Z\) over \(Z\).

\(Z_0 \in 2^Z\) is the interpretation corresponding to the initial state \(s_0\); \(2^X \times 2^Y \times 2^Z \rightarrow 2^Z\) is a Boolean function mapping interpretations \(X, Y\) and \(Z\) of the propositions of \(X, Y\) and \(Z\) to a new interpretation \(Z'\) of the propositions of \(Z\), such that if \(Z\) corresponds to a state \(s \in S\) then \(Z'\) corresponds to the state \(\delta(s, X \cup Y)\). The set of goal states is represented by a Boolean formula \(f\) over \(Z\) that is only satisfied by the interpretations of states in \(F\). Accordingly, we denote the symbolic reachability game as \((D^s, f)\). Note that the transition function \(\eta\) can be represented by an indexed family consisting of a Boolean formula \(\eta_z\) for each state variable \(z \in Z\), which when evaluated over an assignment to \(X \cup Y \cup Z\) returns the next assignment to \(z\).

We also define the product of two symbolic transition systems \(D^s_1 = (X, Y, Z_1, Z_0_1, \eta_1)\) and \(D^s_2 = (X, Y, Z_2, Z_0_2, \eta_2)\) as \(D^s_1 \times D^s_2 = (X, Y, Z, Z_0, \eta)\), where \(X, Y\) and \(Z\) are as defined for \(D^s_1\) and \(D^s_2\), \(Z = Z_1 \cup Z_2, Z_0 = (Z_0_1, Z_0_2)\) and \(\eta : 2^X \times 2^Y \times 2^{Z_1} \times 2^{Z_2} \rightarrow 2^{Z_1} \times 2^{Z_2}\). Since the transition function can be represented by an indexed family, the product transition function \(\eta\) is, in fact, a union of the two indexed families of \(\eta_1\) and \(\eta_2\).

Given a symbolic DFA game \((D^s, f)\), we can compute a uniform winning positional agent strategy by a least fixpoint computation over two Boolean formulas \(w\) over \(Z\) and \(t\) over \(Z \cup X \cup Y\), which represent the agent winning region \(W_{ag}(D^s, f)\) and tuples of winning states, environment actions with suitable agent actions, respectively. \(w\) and \(t\) are initialized as \(w_0(Z) = f(Z)\) and \(t_0(Z, X, Y) = f(Z)\), since every goal state is an agent winning state. Note that \(t_0\) is independent of the propositions from \(X \cup Y\), since once the play reaches goal states, the agent can do whatever it wants. \(t_{i+1}\) and \(w_{i+1}\) are constructed as follows:

\[
t_{i+1}(Z, X, Y) = t_i(Z, X, Y) \lor (\neg w_i(Z) \land w_i(\eta(X, Y, Z)))
\]

\[
w_{i+1}(Z) = \forall X. \exists Y, t_{i+1}(Z, X, Y);
\]

The computation reaches a fixpoint when \(w_{i+1} \equiv w_i\). At this point, no more states will be added, and so all agent winning states have been collected. By evaluating \(Z_0\) on \(w_{i+1}\) we can know if there exists a winning strategy. If that is the case, \(t_{i+1}\) can be used to compute a uniform positional winning strategy through the mechanism of Boolean synthesis [Fried et al., 2016]. More specifically, passing \(t_i\) to a Boolean synthesis procedure, setting \(Z \cup X\) as input variables and \(Y\) as output variables, we obtain a uniform winning positional strategy \(\tau : 2^Z \times 2^X \rightarrow 2^Y\) that can be used to induce an agent winning strategy.

Also computing a uniform cooperatively winning positional strategy can be performed by a least fixpoint computation. To do this, we define again Boolean functions \(w\) over \(Z\) and \(t\) over \(Z \cup X \cup Y\), now representing the agent cooperatively winning region and tuples of cooperatively winning states and cooperative agent and environment actions, respectively. As before, we initialize \(w_0(Z) = f(Z)\) and \(t_0(Z, X, Y) = f(Z)\), since every goal state is a cooperatively winning state. Then, we construct \(t_{i+1}\) and \(w_{i+1}\) as follows:

\[
\hat{t}_{i+1}(Z, X, Y) = \hat{t}_i(Z, X, Y) \lor (\neg w_i(Z) \land \hat{w}_i(\eta(X, Y, Z)))
\]

\[
\hat{w}_{i+1}(Z) = \exists X. \exists Y, \hat{t}_{i+1}(Z, X, Y);
\]

Once the computation reaches a fixpoint, checking the existence and computing a uniform cooperatively winning positional strategy can be done as shown above.

Intuitively, when computing a uniform winning positional strategy we look for agent actions for which, no matter how the environment behaves, the agent is able to reach the winning region \(W_{ag}(D^s, f)\). Differently, when computing a uniform cooperatively winning positional strategy we’re looking for agent actions for which, if the environment acts cooperatively, the agent is able to reach the cooperatively winning region \(W_{ag}^c(D, f)\).

4.2 Monolithic Approach

The monolithic approach is a direct implementation of the best-effort synthesis by Aminof et al. [2011b] (i.e., of Algorithm 0), utilizing the symbolic synthesis framework introduced by Zhu et al. [2017]. Given a best-effort synthesis problem \(P = (\mathcal{E}, \varphi)\), we first construct the DFAs following the synthesis algorithm described in Section 3, and convert them into a symbolic representation. Finally, we solve suitable games on the symbolic DFAs and obtain a best-effort strategy. More specifically, this approach solves the best-effort synthesis problem \(P = (\mathcal{E}, \varphi)\) as follows.

Algorithm 1.

1. For \(\text{LTL}_f\) formulas \(\mathcal{E} \rightarrow \varphi, \neg \mathcal{E} \land \varphi\) compute the corresponding minimal explicit-state DFAs \(\mathcal{A}_{\mathcal{E} \rightarrow \varphi} = (D_{\mathcal{E} \rightarrow \varphi}, F_{\mathcal{E} \rightarrow \varphi}), \mathcal{A}_{\neg \mathcal{E}} = (D_{\neg \mathcal{E}}, F_{\neg \mathcal{E}})\) and \(\mathcal{A}_{\mathcal{E} \land \varphi} = (D_{\mathcal{E} \land \varphi}, F_{\mathcal{E} \land \varphi})\), and convert the DFAs to a symbolic representation and obtain \(\mathcal{A}_{\mathcal{E} \rightarrow \varphi}^s = (D_{\mathcal{E} \rightarrow \varphi}^s, F_{\mathcal{E} \rightarrow \varphi}^s), \mathcal{A}_{\neg \mathcal{E}}^s = (D_{\neg \mathcal{E}}^s, F_{\neg \mathcal{E}})\) and \(\mathcal{A}_{\mathcal{E} \land \varphi}^s = (D_{\mathcal{E} \land \varphi}^s, F_{\mathcal{E} \land \varphi})\).
Construct symbolic DFA games \((D^s, f_{E→φ})\) and \((D^φ, f_{E→φ})\), where \(D^s = D^s_{E→φ} \times D^s_{E→φ} \times D^s_{E∧φ}\) is the product of the symbolic transition systems \(D^s_{E→φ}\), \(D^s_{E→φ}\), and \(D^s_{E∧φ}\).

3. In the DFA game \((D^s, f_{E→φ})\) compute a uniform positional strategy \(τ_{ag}\), and the agent winning region, which is represented as Boolean formula \(W_{ag}(D^s, f_{E→φ})\) over state variables \(Z\).

4. In the DFA game \((D^s, f_{E→φ})\) compute the environment’s winning region, represented as Boolean formula \(W_{env}(D^s, f_{E→φ})\) over \(Z\).

5. Construct symbolic DFA game \((D^s, f_{E∧φ})\), where \(D^s = D^s \wedge W_{env}(D^s, f_{E→φ})\) such that restricting the state space of transition system \(D^s\) to considering \(W_{env}(D^s, f_{E→φ})\) only.

6. In the DFA game \((D^s, f_{E∧φ})\) find a positional cooperatively winning strategy \(γ_{ag}\).

7. Return the best-effort strategy \(σ_{ag}\) induced by the positional strategy constructed as follows: 
\[
κ_{ag}(Z, X) = \begin{cases}
τ_{ag}(Z, X) & \text{if } Z = W_{ag}(D^s, f_{E→φ}) \\
γ_{ag}(Z, X) & \text{otherwise}
\end{cases}
\]

The operations workflow of the monolithic approach is shown in Figure 1. First, to transform the formulas \(E → φ, ¬E \wedge φ\) and \(E \wedge φ\) into minimal explicit-state DFAs \(A_{E→φ}, A_{E∧φ}\), and \(A_{E∧φ}\). Then, we transform these explicit-state DFAs to symbolic representations \(A^s_{E→φ}, A^s_{E∧φ}\), and \(A^s_{E∧φ}\). At this point, the arena of the games played by the algorithm is constructed through product as \(D^s_{E→φ} \times D^s_{E→φ} \times D^s_{E∧φ}\). Then, the games of the algorithms are constructed as: \((D^s, f_{E→φ})\), \((D^φ, f_{E→φ})\), and \((D^φ, f_{E∧φ})\). These DFA games are left unminimized. Here, games \((D^s, f_{E→φ})\), \((D^φ, f_{E→φ})\), and \((D^φ, f_{E∧φ})\) are solved as shown in Section 4.1.

After computing the winning positional strategy \(τ_{ag}\) and the cooperatively winning positional strategy \(γ_{ag}\), we construct the positional best-effort strategy \(κ_{ag}(Z, X)\) as follows: 
\[
κ_{ag}(Z, X) = \begin{cases}
τ_{ag}(Z, X) & \text{if } Z = W_{ag}(D^s, f_{E→φ}) \\
γ_{ag}(Z, X) & \text{otherwise}
\end{cases}
\]

In case the winning positional strategy is not defined in a state \(Z\), then the output of the cooperatively winning positional strategy is used. Note that the positional best-effort strategy induces an \(LTL_f\) best-effort strategy.

The main challenge in the monolithic approach comes from the \(LTL_f\)-to-DFA conversion, which can take, in the worst case, doubly exponential time [De Giacomo and Vardi, 2013], and thus also considered as the bottleneck of \(LTL_f\) synthesis [Zhu et al., 2017]. To that end, we propose an explicit compositional approach to diminish this difficulty by decreasing the number of performing \(LTL_f\)-to-DFA conversions.

### 4.3 Explicit-Compositional Approach

As described in Section 4.2, the monolithic approach to a best-effort synthesis problem \(P = (E, φ)\) involves three rounds of \(LTL_f\)-to-DFA conversions with respect to \(LTL_f\) formulas \(E → φ, ¬E \wedge φ\) and \(E \wedge φ\). We observe that the constructed DFAs \(A_{E→φ}, A_{E∧φ}\), and \(A_{E∧φ}\) can, in fact, be constructed by manipulating the two DFAs \(A_{φ}\) and \(A_{E}\) of \(LTL_f\) formulas \(E\) and \(φ\), respectively. Specifically, given explicit-state DFAs \(A_{φ}\) and \(A_{E}\), we obtain \(A_{E→φ}, A_{E∧φ}\), and \(A_{E∧φ}\) as follows:

- \(A_{E→φ} = \text{Comp}(\text{Inter}(A_{φ}, \text{Comp}(A_{E})))\)
- \(A_{E∧φ} = \text{Comp}(A_{φ})\)
- \(A_{E∧φ} = \text{Inter}(A_{φ}, A_{E})\)

where \(\text{Comp}\) and \(\text{Inter}\) denote complement and intersection on explicit-state DFA, respectively. Note that, the \(LTL_f\)-to-DFA is 2EXPTIME-complete, while DFA complement and intersection only takes linear time in the size of the DFA. Thus, \(LTL_f\) synthesis is 2EXPTIME-complete, while DFA complement and intersection only takes linear time in the size of the DFA.

Thus, \textbf{Algorithm 2}, which follows the explicit-compositional approach to a best-effort synthesis problem \(P = (E, φ)\), proceeds as shown in Figure 2. Here, we first translate directly the formulas \(E\) and \(φ\) into minimal explicit-state DFAs \(A_{φ}\) and \(A_{E}\). Then, the DFA complement and intersection are performed to build DFAs \(A_{E→φ}, A_{E∧φ}\), and \(A_{E∧φ}\). Indeed, the constructed explicit-state DFAs can also be minimized. The remaining steps are the same as in the monolithic approach.

### 4.4 Symbolic-Compositional Approach

A few aspects, however, make the explicit-compositional approach unsatisfactory to us. Let us start with focusing on the two DFAs \(A_{E→φ}\) and \(A_{E∧φ}\). Suppose we put automata minimization aside, the initial states and transition functions of both DFAs are essentially the same, and they only differ in accepting conditions. This is because the initial states and transition functions of \(A_{E→φ}\) and \(A_{E∧φ}\) are obtained by performing cross product of \(A_{φ}\) and \(A_{E}\). Consequently, if no minimization takes place, symbolic DFAs \(A_{E→φ} = (D^s_{E→φ}, f_{E→φ})\) and \(A_{E∧φ} = (D^s_{E∧φ}, f_{E∧φ})\) share the same transition system such that \(D^s_{E→φ} = D^s_{E∧φ}\). Moreover, since
Lemma 1. Let $E$ and $\varphi$ be two LTL$_f$ formulas over $X \cup Y$. We have that $D^*_E \times D^*_{\varphi} \equiv D^*_E \times D^*_{\varphi}$.  

**Proof.** First, we show that $D^*_E \times D^*_{\varphi} = D^*_{E \land \varphi}$, where $D^*_{E \land \varphi} = (X, Y, Z_{E \land \varphi}, Z_{0, E \land \varphi}, \eta_{E \land \varphi})$. To see this, note that $D^*_{E \land \varphi} = D^*_{E \land \varphi}$. Now, since for every LTL$_f$ formula $\psi$, it holds that $D^*_E = D^*_{\psi}$, we have that $D^*_{E \land \varphi}$ holds. The transition system $D^*_{E \land \varphi}$ can indeed be obtained by performing symbolic product of two transition systems $D^*_E$ and $D^*_{\varphi}$, such that $D^*_{E \land \varphi} = D^*_E \times D^*_{\varphi}$. As a result, it holds that $D^*_{E \land \varphi} = D^*_{E \land \varphi} = D^*_{E \land \varphi} = D^*_{E \land \varphi} = D^*_{E \land \varphi}$. Thus, we can simplify $D^*_{E \land \varphi} = D^*_{E \land \varphi}$ as $D^*_{E \land \varphi} = D^*_{E \land \varphi}$.

Furthermore, since $D^*_E \times D^*_{\varphi} = D^*_E = (X, Y, Z_{0, E}, \eta_E)$ and $D^*_{E \land \varphi} = D^*_E \times D^*_{\varphi} = (X, Y, Z_{E \land \varphi}, \eta_{E \land \varphi})$, the state variables $Z_{E \land \varphi}$ and the transition function $\eta_{E \land \varphi}$, as represented by the indexed family, of $D^*_{E \land \varphi}$ are actually, the subsets of the state variables $Z_{E \land \varphi}$ and the transition function (and $\eta_{E \land \varphi}$) of $D^*_E \times D^*_{\varphi}$, respectively. Hence, $D^*_E \times D^*_{E \land \varphi}$ can be further simplified as $D^*_{E \land \varphi}$, which is equivalent to $D^*_E \times D^*_{\varphi}$. We thus conclude that $D^*_E \times D^*_{\varphi} \equiv D^*_{E \land \varphi}$.

Furthermore, since no DFA minimization takes place then the DFAs have the following form: $A^*_{E \land \varphi} = (D^*_E, f_{E \land \varphi})$, $A^*_{E \land \varphi} = (D^*_E, f_{E \land \varphi})$ and $A^*_{E \land \varphi} = (D^*_E, f_{E \land \varphi})$. This means that we can construct $D^* = D^*_E \times D^*_{\varphi}$ once and for all and then get the three automata by defining the final states functions (which are Boolean functions) from $f_E$ and $f_\varphi$ as follows:

- $f_{E \land \varphi} = \neg(f_E \land \neg(f_\varphi))$;
- $f_E = \neg f_E$;
- $f_{E \land \varphi} = f_E \land f_\varphi$.

Thus, given a best-effort synthesis problem $P = (E, \varphi)$, the symbolic-compositional approach solves it as follows.

**Algorithm 3.**

1. Compute the minimal explicit-state DFAs $A^*_E = (D^*_E, F_E)$ and $A^*_E = (D^*_E, F_E)$ and convert the DFAs to a symbolic representation and obtain $A^*_{E \land \varphi} = (D^*_E, F_{E \land \varphi})$ and $A^*_{E \land \varphi} = (D^*_E, F_{E \land \varphi})$.

2. Construct the symbolic DFA games $G^*_{E \land \varphi} = (D^* \land \neg f_E)$ and $G^*_{E \land \varphi} = (D^* \land f_E \land \neg(f_\varphi))$.

3. In the DFA game $G^*_{E \land \varphi} = (D^* \land \neg(f_E \land \neg(f_\varphi)))$ compute a uniform positional winning strategy $\tau_{ag}$, and the agent winning region, represented as Boolean formula $W_{ag}(D^* \land \neg f_E \land \neg(f_\varphi))$ over state variables $Z$.

4. In the DFA game $(D^* \land f_E)$ compute the environment’s winning region, represented as Boolean formula $W_{env}(D^* \land \neg f_E)$ over $Z$.

5. Construct symbolic DFA game $(D^* \land \neg f_E \land f_\varphi)$, where $D^* = D^* \land W_{env}(D^* \land \neg f_E)$ such that restricting the state space of transition system $D^*$ to considering $W_{env}(D^* \land \neg f_E)$ only.
6. In the DFA game \((D^s, f_E \land f_\varphi)\) find a positional cooperatively winning strategy \(\gamma_{ag}\).

7. \textbf{Return} the best-effort strategy \(\sigma_{ag}\) induced by the positional strategy constructed as follows: \(\kappa_{ag}(Z, X) = \tau_{ag}(Z, X)\) if \(Z \models \mathcal{W}_{ag}(D^s, \neg(f_E \land \neg(f_\varphi)))\) and \(\kappa_{ag}(Z, X) = \gamma_{ag}(Z, X)\) otherwise.

As shown in Figure 3, the symbolic-compositional approach first transforms formulas \(E\) and \(\varphi\) into minimal explicit-state DFAs \(A_E\) and \(A_\varphi\), and then directly into a symbolic representation \(A_E^\lor\) and \(A_\varphi^\lor\). Hence, we construct the arena of the games as \(D_E^\lor \times D_\varphi^\lor\). Note that the result of this product is unminimized. At this point the games played by the algorithm are constructed as \((D^s, \neg(f_E \land \neg(f_\varphi)))\), \((D^s, f_E)\) and \((D^s, f_\varphi)\). From now on, the remaining steps are the same as in the monolithic approach, indeed also the explicit-compositional approach.

5 Empirical Evaluations

In this section, we first describe how we implemented our symbolic \(\text{LTL}_f\) best-effort synthesis approaches described in Section 4. Then, by empirical evaluation, we show that Algorithm 3, i.e., the symbolic-compositional approach, shows an overall best-performance. In particular, we show that performing best-effort synthesis only brings a minimal overhead with respect to standard synthesis.

5.1 Implementation

We implemented the three different symbolic best-effort synthesis approaches described in Section 4 in a tool called BeSyft by extending the symbolic synthesis framework [Zhu et al., 2017; Tabajara and Vardi, 2019] integrated in state-of-the-art synthesis tools [Bansal et al., 2020; De Giacomo and Favorito, 2021a]. In particular, we based on LYDIA ¹, the overall best performing \(\text{LTL}_f\)-to-DFA conversion tool, to construct the explicit-state DFA from \(\text{LTL}_f\) formulas. It should be noted that all the explicit-state DFAs constructed by LYDIA are minimized. Moreover, BeSyft borrows the rich APIs from LYDIA to perform the relevant explicit-state DFA manipulations required by both of Algorithm 1, i.e., the monolithic approach (c.f., Subsection 4.2) and Algorithm 2, i.e., the explicit-compositional approach (c.f., Subsection 4.3), such as complement, intersection, minimization, and occurrence in accordance with the environment assumption. As in [Zhu et al., 2017; Tabajara and Vardi, 2019], the symbolic DFA games are represented in Binary Decision Diagrams (BDDs) [Bryant, 1992], utilizing CUDD-3.0.0 [Somenzi, 2016] as the BDD library. Thereby, BeSyft constructs and solves symbolic DFA games by making use of the Boolean operations provided by CUDD-3.0.0, such as negation, conjunction, qualification. The positional winning strategy \(\tau_{ag}\) and the positional cooperatively winning strategy \(\gamma_{ag}\) are abstracted utilizing Boolean synthesis [Fried et al., 2016]. The positional best-effort strategy is obtained by applying suitable Boolean operations on \(\tau_{ag}\) and \(\gamma_{ag}\). As a result, we have three derivations of BeSyft, namely BeSyft-Alg-1, BeSyft-Alg-2 and BeSyft-Alg-3, corresponding to the monolithic approach, the explicit-compositional approach, and the symbolic-compositional approach, respectively.

5.2 Experiment Methodology

\textbf{Experiment Setup.} All experiments were run on a laptop with operating system 64-bit Ubuntu 20.04, 3.6 GHz CPU, and 12 GB of memory. Time out was set to 1000 seconds.

\textbf{Benchmark.} We devised a counter game benchmark, based on the one proposed by Zhu et al. [2020]. More specifically, there is an \(n\)-bit binary counter. At each round, the environment chooses whether to issue an increment request for the counter or not. The agent can choose to grant the request or ignore it. The goal is to get the counter having all bits set to 1. The increment requests only come from the environment, and occur in accordance with the environment assumption. Note that the size of the corresponding minimal DFA grows exponentially as the size of the counter game \(n\) grows.

In the experiments, we considered two types of environment assumptions: \(E_\Box = \Box(\text{add})\) and \(E_\Diamond = \Diamond(\text{add})\). We name the counter game instances with the assumption \(E_\Box\) as \(\Box\)-instances, and the ones with the assumption \(E_\Diamond\) as \(\Diamond\)-instances. It is easy to see that \(\Box\)-instances are always agent realizable, i.e., the agent has a strategy to realize the goal under environment assumption \(E_\Box\). Indeed, such a strategy is to grant all increment request coming from the environment. Instead, \(\Diamond\)-instances are agent realizable for 1-bit counter games, and cooperatively realizable otherwise. Indeed, in the first case, a winning strategy for the agent is to grant the increment request that will eventually occur. Differently, in the second case, the agent can achieve the goal only if the environment behaves cooperatively such that issuing more increment requests than that specified by the environment assumption. That is to say, the agent needs a best-effort strategy.

5.3 Experimental Results and Analysis.

Figure 4 (note that the \(y\)-axis is in log scale) shows the running time of each derivation of BeSyft on every instance of the counter games. Figure 5 shows the same result as Figure 4, but having the \(y\)-axis in linear scale, thus showing a clearer view of the performance of three derivations of BeSyft. Across these instances, we see that all the derivations were able to manage instances with up to 8 bits, for both \(\Box\)- and \(\Diamond\)-counter instances. In particular, BeSyft-Alg-1 is beaten by the other two derivations, since it requires three rounds of \(\text{LTL}_f\)-to-DFA conversions, which in the worst case, can lead

¹https://github.com/whitemech/lydia

Figure 4: Comparison of the 3 best-effort synthesis alg in log scale.
to a double-exponential blowup. Note that the time cost of all derivations of BeSyft grows exponentially, instead of double-exponential. This is because LYDIA always returns the minimal explicit-state DFA of a given LTL$_f$ formula, and the powerful automata minimization can possibly disappear the exponential gap, as observed in [Tabajara and Vardi, 2019; Zhu et al., 2020]. This also enables the state-of-the-art LTL$_f$ synthesis tools to take the maximal advantage of automata minimization [Bansal et al., 2020; De Giacomo and Favorito, 2021a]. Nevertheless, it is not the case that automata minimization always leads to an improvement. Instead, there is a trade-off of performing automata minimization. As shown in Figures 4 and 5, BeSyft-Alg-3 shows better performance comparing to BeSyft-Alg-2, though the former cannot minimize the game arena after symbolic product, and the latter minimizes the game arena as much as possible.

On a closer inspection, we evaluated the time cost of each major operation of BeSyft-Alg-3, and present the results on ♦-counter instances in Figure 6. First, the results show that LTL$_f$-to-DFA conversion is the bottleneck of LTL$_f$ best-effort synthesis, the cost of which dominates the total running time. Moreover, we can see that the total time cost of cooperative DFA game and strategy merging of best-effort synthesis, only counts for 20% ~ 30% of the total time cost. As a result, we conclude that, in general, performing best-effort synthesis only brings a minimal overhead.

Furthermore, we also compared the time cost of BeSyft-Alg-3 with that of standard synthesis on counter games. More specifically, we consider two kinds of reactive synthesizer: adversarial synthesizer that computes an agent winning strategy for LTL$_f$ formula $E \rightarrow \varphi$; cooperative synthesizer that computes an agent cooperatively winning strategy for LTL$_f$ formula $E \land \varphi$. Figure 7 shows that for small instances (up to 7 bits), BeSyft-Alg-3 only takes minor extra time than those of adversarial synthesizer and cooperative synthesizer. Interestingly, for the 8-bit counter instances, BeSyft-Alg-3 even takes less time. Although BeSyft-Alg-3 performs LTL$_f$-to-DFA conversion of LTL$_f$ formulas $\varphi$ and $E$ separately and combines them to obtain the final game arena without having automata minimization, it can happen that the total time cost of such is even less than the LTL$_f$-to-DFA conversion of a larger formula $E \rightarrow \varphi$. This, again, confirms that performing best-effort synthesis only brings a minimal overhead with respect to standard synthesis.

6 Conclusion

We presented three different symbolic LTL$_f$ best-effort synthesis approaches: monolithic, explicit-compositional and symbolic-compositional. Empirical evaluations proved the outperformance of the symbolic-compositional approach. An interesting observation is that, although previous studies suggest taking the maximal advantage of automata minimization [Tabajara and Vardi, 2019; Zhu et al., 2020], in the case of LTL$_f$ best-effort synthesis, there can be a trade-off of doing so. Another significant finding is that the best-performing LTL$_f$ best-effort synthesis approach only brings a minimal overhead comparing to standard synthesis. Given this nice computational result, a natural future direction would be looking into LTL best-effort synthesis [Aminof et al., 2021b] and LTL$_f$ best-effort synthesis with multiple environment assumptions [Aminof et al., 2021a].

Acknowledgments

This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the PRIN project RIPER (No.
References


