Symbolic LTL$_f$ Best-Effort Synthesis

Giuseppe De Giacomo[1,2], Gianmarco Parretti[1(✉)], and Shufang Zhu[2(✉)]

[1] University of Rome "La Sapienza", Rome, Italy
{degiacomo,parretti}@diag.uniroma1.com
[2] University of Oxford, Oxford, UK
{giuseppe.degiacomo,shufang.zhu}@cs.ox.ac.uk

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 fulfill tasks in a nondeterministic environment, as considered in Planning in nondeterministic (adversarial) domains [8,15], except that we specify both the environment and the task in Linear Temporal Logic (LTL) [3], the formalism typically used for specifying complex dynamic properties in Formal Methods [5].

In fact, we consider Linear Temporal Logic on finite traces (LTL$_f$) [11,12], which maintains the syntax of LTL [18] but is interpreted on finite traces. In this setting, we study synthesis [3,12,13,17]. 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's task, regardless of how the environment acts, does not exist, the agent should at least avoid adopting strategies that prevent it 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). Best-effort strategies have been studied in [4] and proven to 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).

V. Malvone and A. Murano (Eds.): EUMAS 2023, LNAI 14282, pp. 228–243, 2023.
https://doi.org/10.1007/978-3-031-43264-4_15

The algorithms for best-effort synthesis in LTL and LTL$_f$ have been presented in [4]. 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 $\mathcal{E}$ and φ constituting the environment and the task specifications, respectively.

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 $\mathcal{E}$ and φ into deterministic finite automata (DFA), which can be double-exponential in the worst case, but for which various good techniques have been developed [6,10,16,22]; (*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 techniques presented in [4] by using symbolic techniques [5,7,22]. In particular, we show three different symbolic approaches that combine the above operations in different ways (and in fact allow for different levels 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 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 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 Sect. 2, we recall the main notions of LTL$_f$ synthesis. In Sect. 3, we discuss LTL$_f$ best-effort synthesis, and the algorithm presented in [4]. In Sect. 4, we introduce three distinct symbolic approaches for LTL$_f$ best-effort synthesis: the first (c.f., Subsect. 4.2) is a direct symbolic implementation of the algorithm presented in [4]; the second one (c.f., Subsect. 4.3) favors maximally conducting DFA minimization, thus getting the smallest possible arenas for the three games; and the third one (c.f., Subsect. 4.4) gives up DFA minimization at some level, and creates a single arena for the three games. In Sect. 5, we perform an empirical evaluation of the three algorithms. We conclude the paper in Sect. 6.

2 Preliminaries

LTL$_f$ *Basics. Linear Temporal Logic on finite traces* (LTL$_f$) is a specification language to express temporal properties on finite traces [11]. In particular, LTL$_f$ has the same syntax as LTL, which is instead interpreted over infinite traces [18]. Given a set of propositions Σ, LTL$_f$ formulas are generated as follows:

$$\varphi ::= a \mid (\varphi_1 \wedge \varphi_2) \mid (\neg\varphi) \mid (\bigcirc\varphi) \mid (\varphi_1 \,\mathcal{U}\, \varphi_2)$$

where $a \in \Sigma$ is an *atom*, $\bigcirc$ (*Next*), and $\mathcal{U}$ (*Until*) are temporal operators. We make use of standard Boolean abbreviations such as $\vee$ (or) and $\rightarrow$ (implies), *true* and *false*. In addition, we define the following abbreviations *Weak Next* $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$, *Eventually* $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$ and *Always* $\Box\varphi \equiv \neg\Diamond\neg\varphi$. The length/size of φ, written $|\varphi|$, is the number of operators in φ.

A *finite* (resp. *infinite*) *trace* is a sequence of propositional interpretations $\pi \in (2^{\Sigma})^*$ (resp. $\pi \in (2^{\Sigma})^{\omega}$). For every $i \geq 0$, $\pi_i \in 2^{\Sigma}$ is the i-th interpretation of π. Given a finite trace π, we denote its last instant (i.e., index) by $\mathsf{lst}(\pi)$. LTL$_f$ formulas are interpreted over finite, nonempty traces. Given a finite, nonempty trace $\pi \in (2^{\Sigma})^+$, we define when an LTL$_f$ formula φ *holds* at instant i, $0 \leq i \leq \mathsf{lst}(\pi)$, written $\pi, i \models \varphi$, inductively on the structure of φ, as:

- $\pi, i \models a$ iff $a \in \pi_i$ (for $a \in \Sigma$);
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \bigcirc\varphi$ iff $i < \mathsf{lst}(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1\,\mathcal{U}\,\varphi_2$ iff $\exists j$ such that $i \leq j \leq \mathsf{lst}(\pi)$ and $\pi, j \models \varphi_2$, and $\forall k, i \leq k < j$ we have that $\pi, k \models \varphi_1$.

We say π *satisfies* φ, written as $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

Reactive Synthesis Under Environment Specifications. Reactive synthesis concerns computing a strategy that allows the agent to achieve its goal in an adversarial environment. In many AI applications, the agent has a model describing possible environment behaviors, which we call here an *environment specification* [2,3]. In this work, we specify both environment specifications and agent goals as LTL$_f$ formulas defined over $\Sigma = \mathcal{X} \cup \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets of variables under the control of the environment and the agent, respectively.

An *agent strategy* is a function $\sigma_{ag} : (2^{\mathcal{X}})^* \rightarrow 2^{\mathcal{Y}}$ that maps a sequence of environment choices to an agent choice. Similarly, an *environment strategy* is a function $\sigma_{env} : (2^{\mathcal{Y}})^+ \rightarrow 2^{\mathcal{X}}$ mapping non-empty sequences of agent choices to an environment choice. A trace $\pi = (X_0 \cup Y_0)(X_1 \cup Y_1)\ldots \in (2^{\mathcal{X}\cup\mathcal{Y}})^{\omega}$ is σ_{ag}-consistent if $Y_0 = \sigma_{ag}(\epsilon)$, where ϵ denotes empty sequence, and $Y_i = \sigma_{ag}(X_0, \ldots, X_{i-1})$ for every $i > 0$. Analogously, π is σ_{env}-consistent if $X_i = \sigma_{env}(Y_0, \ldots, Y_i)$ for every $i \geq 0$. We define $\pi(\sigma_{ag}, \sigma_{env})$ to be the unique infinite trace that is consistent with both σ_{ag} and σ_{env}.

Let ψ be an LTL$_f$ formula over $\mathcal{X}\cup\mathcal{Y}$. We say that agent strategy σ_{ag} *enforces* ψ, written $\sigma_{ag} \triangleright \psi$, if for every environment strategy σ_{env}, there exists a *finite* prefix of $\pi(\sigma_{ag}, \sigma_{env})$ that satisfies ψ. Conversely, we say that an environment strategy σ_{env} *enforces* ψ, written $\sigma_{env} \triangleright \psi$, if for every agent strategy σ_{ag}, every finite prefix of $\pi(\sigma_{ag}, \sigma_{env})$ satisfies ψ. ψ is *agent enforceable* (resp. *environment enforceable*) if there exists an agent (resp. environment) strategy that enforces it. An *environment specification* is an LTL$_f$ formula $\mathcal{E}$ that is environment enforceable.

The problem of LTL$_f$ reactive synthesis under environment specifications is defined as follows.

Definition 1. *The* LTL$_f$ *reactive synthesis under environment specifications problem is defined as a pair* $\mathcal{P} = (\mathcal{E}, \varphi)$*, where* LTL$_f$ *formulas* $\mathcal{E}$ *and* φ *correspond to an environment specification and an agent goal, respectively. Realizability of* $\mathcal{P}$ *checks whether there exists an agent strategy* σ_{ag} *that enforces* φ *under* $\mathcal{E}$*, i.e.,*

$$\forall \sigma_{env} \triangleright \mathcal{E}, \pi(\sigma_{ag}, \sigma_{env}) \models \varphi$$

Synthesis of $\mathcal{P}$ *computes such a strategy if it exists.*

A naive approach to this problem is a reduction to standard reactive synthesis of LTL$_f$ formula $\mathcal{E} \rightarrow \varphi$ [3]. Moreover, it has been shown that the problem of LTL$_f$ reactive synthesis under environment specifications is 2EXPTIME-complete [3].

3 Best-Effort Synthesis Under Environment Specifications

In reactive synthesis, the agent aims at computing a strategy that enforces the goal regardless of environment behaviors. If such a strategy does not exist, the agent just gives up when the synthesis procedure declares the problem *unrealizable*, although the environment can be possibly "over-approximated". In this work, we synthesize a strategy ensuring that the agent will do nothing that would needlessly prevent it from achieving its goal – which we call a *best-effort strategy. Best-effort synthesis* is the problem of finding such a strategy [4]. We start by reviewing what it means for an agent strategy to make more effort with respect to another.

Definition 2. *Let* $\mathcal{E}$ *and* φ *be* LTL$_f$ *formulas denoting an environment specification and an agent goal, respectively, and* σ_1 *and* σ_2 *be two agent strategies.* σ_1 dominates σ_2 *for* φ *under* $\mathcal{E}$*, written* $\sigma_1 \geq_{\varphi|\mathcal{E}} \sigma_2$*, if for every* $\sigma_{env} \triangleright \mathcal{E}, \pi(\sigma_2, \sigma_{env}) \models \varphi$ *implies* $\pi(\sigma_1, \sigma_{env}) \models \varphi$*.*

Furthermore, we say that σ_1 *strictly dominates* σ_2, written $\sigma_1 >_{\varphi|\mathcal{E}} \sigma_2$, if $\sigma_1 \geq_{\varphi|\mathcal{E}} \sigma_2$ and $\sigma_2 \not\geq_{\varphi|\mathcal{E}} \sigma_1$. Intuitively, $\sigma_1 >_{\varphi|\mathcal{E}} \sigma_2$ means that σ_1 does at least as well as σ_2 against every environment strategy enforcing $\mathcal{E}$ and strictly better against one such strategy. If σ_1 strictly dominates σ_2, then σ_1 makes more effort than σ_2 to satisfy the goal. In other words, if σ_2 is strictly dominated by σ_1, then an agent that uses σ_2 does not do its best to achieve the goal: if it used σ_1 instead, it could achieve the goal against a strictly larger set of environment behaviors. Within this framework, a best-effort strategy is one that is not strictly dominated by any other strategy.

Definition 3. *An agent strategy* σ *is best-effort for* φ *under* $\mathcal{E}$*, if there is no agent strategy* σ' *such that* $\sigma' >_{\varphi|\mathcal{E}} \sigma$*.*

It follows immediately from Definition 3 that if a goal φ is agent enforceable under $\mathcal{E}$, then best-effort strategies enforce φ under $\mathcal{E}$. Best-effort synthesis concerns computing a best-effort strategy.

Definition 4 ([4]). *The* LTL_f *best-effort synthesis problem is defined as a pair* $\mathcal{P} = (\mathcal{E}, \varphi)$*, where* LTL_f *formulas* $\mathcal{E}$ *and* φ *are the environment specification and the agent goal, respectively. Best-effort synthesis of* $\mathcal{P}$ *computes an agent strategy that is best-effort for* φ *under* $\mathcal{E}$*.*

While classical synthesis settings first require checking the realizability of the problem, i.e., the existence of a strategy that enforces the agent goal under environment specification [12,17], deciding whether a best-effort strategy exists is trivial, as they always exist.

Theorem 1 ([4]). *Let* $\mathcal{P} = (\mathcal{E}, \varphi)$ *be an* LTL_f *best-effort synthesis problem. There exists a best-effort strategy for* φ *under* $\mathcal{E}$*.*

LTL_f best-effort synthesis can be solved by a reduction to suitable DFA games and is 2EXPTIME-complete [4].

DFA *Game.* A DFA *game* is a two-player game played on a deterministic finite automaton (DFA). Formally, a DFA is defined as a pair $\mathcal{A} = (\mathcal{D}, F)$, where $\mathcal{D}$ is a deterministic transition system such that $\mathcal{D} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \delta)$, where $2^{\mathcal{X}\cup\mathcal{Y}}$ is the alphabet, S is the state set, $s_0 \in S$ is the initial state and $\delta\colon S \times 2^{\mathcal{X}\cup\mathcal{Y}} \to S$ is the deterministic *transition function*, and $F \subseteq S$ is a set of final states. We call $|S|$ the *size* of $\mathcal{D}$. Given a finite word $\pi = (X_0 \cup Y_0)\dots(X_n \cup Y_n) \in (2^{\mathcal{X}\cup\mathcal{Y}})^+$, running π in $\mathcal{D}$ yields the sequence $\rho = s_0 \dots s_{n+1}$ such that s_0 is the initial state of $\mathcal{D}$ and $s_{i+1} = \delta(s_i, X_i \cup Y_i)$ for all i. Since the transitions in $\mathcal{D}$ are all deterministic, we denote by $\rho = \mathsf{Run}(\pi, \mathcal{D})$ the unique sequence induced by running π on $\mathcal{D}$. We define the *product* of transition systems as follows.

Definition 5. *The product of transition systems* $\mathcal{D}_i = (\Sigma, S_i, s_{(0,i)}, \delta_i)$ *(with* $i = 1, 2$*) over the same alphabet is the transition system* $\mathcal{D}_1 \times \mathcal{D}_2 = (\Sigma, S, s_0, \delta)$ *with:* $S = S_1 \times S_2$*;* $s_0 = (s_{(0,1)}, s_{(0,2)})$*; and* $\delta((s_1, s_2), x) = (\delta(s_1, x), \delta(s_2, x))$*. The product* $\mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ *is defined analogously for any finite sequence* $\mathcal{D}_1, \ldots, \mathcal{D}_n$ *of transition systems over the same alphabet.*

A finite word π is *accepted* by $\mathcal{A} = (\mathcal{D}, F)$ if the last state of the run it induces is a final state, i.e., $\mathsf{lst}(\rho) \in F$, where $\rho = \mathsf{Run}(\pi, \mathcal{D})$. The language of $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, consists of all words accepted by the automaton. Every LTL_f formula φ can be transformed into a DFA $\mathcal{A}_\varphi$ that accepts exactly the traces that satisfy the formula, in other words, $\mathcal{A}_\varphi$ *recognizes* φ.

Theorem 2 ([11]). *Given an* LTL_f *formula over* Σ*, we can build a* DFA $\mathcal{A}_\varphi = (\mathcal{D}_\varphi, F_\varphi)$ *whose size is at most double-exponential in* $|\varphi|$ *such that* $\pi \models \varphi$ *iff* $\pi \in \mathcal{L}(\mathcal{A}_\varphi)$*.*

In a DFA game $(\mathcal{D}, F)$, the transition system $\mathcal{D}$ is also called the *game arena*. Given σ_{ag} and σ_{env} denoting an agent strategy and an environment strategy, respectively, the trace $\pi(\sigma_{ag}, \sigma_{env})$ is called a *play*. Specifically, a play is *winning* if it contains a finite prefix that is accepted by the DFA. Intuitively, DFA games require F to be visited at least once. An agent strategy σ_{ag} is *winning* in

$(\mathcal{D}, F)$ if, for every environment strategy σ_{env}, it results that $\pi(\sigma_{ag}, \sigma_{env})$ is winning. Conversely, an environment strategy σ_{env} is *winning* in the game $(\mathcal{D}, F)$ if, for every agent strategy σ_{ag}, it results that $\pi(\sigma_{ag}, \sigma_{env})$ is not winning. In DFA *games*, $s \in S$ is a *winning* state for the agent (resp. environment) if the agent (resp. the environment) has a winning strategy in the game $(\mathcal{D}', F)$, where $\mathcal{D}' = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s, \delta)$, i.e., the same arena $\mathcal{D}$ but with the new initial state s. By $\mathsf{W}_{ag}(\mathcal{D}, F)$ (resp. $\mathsf{W}_{env}(\mathcal{D}, F)$) we denote the set of all agent (resp. environment) winning states. Intuitively, W_{ag} 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_{ag} is *cooperatively winning* in game $(\mathcal{D}, F)$ if there exists an environment strategy σ_{env} such that $\pi(\sigma_{ag}, \sigma_{env})$ is winning. Hence, $s \in S$ is a *cooperatively winning state* if the agent has a cooperatively winning strategy in the game $(\mathcal{D}', F)$, where $\mathcal{D}' = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \delta)$. By $\mathsf{W}'_{ag}(\mathcal{D}, F)$ we denote the set of all agent cooperative winning states.

When the agent makes its choices based only on the current state of the game, we say that it uses a *positional strategy*. Formally, we define an *agent positional strategy* (a.k.a. *memory-less strategy*) as a function $\tau_{ag} : S \to 2^{\mathcal{X}}$. An agent positional strategy τ_{ag} *induces* an agent strategy $\sigma_{ag} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ as follows: $\sigma_{ag}(\epsilon) = \tau(s_0)$ and, for $i \geq 0$, $\sigma_{ag}(X_0 \dots X_i) = \tau_{ag}(s_{i+1})$, where s_{i+1} is the last state in the sequence $\rho = \mathsf{Run}(\pi, \mathcal{D})$, with π being the finite sequence played until now, i.e., $\pi = (\sigma_{ag}(\epsilon)\cup X_0)(\sigma_{ag}(X_0)\cup X_1) \dots (\sigma(X_0 \dots X_{k-1})\cup X_k)$. Similarly, we can define an *environment positional* strategy as a function $\tau_{env} \colon S \times 2^{\mathcal{Y}} \to 2^{\mathcal{X}}$. A positional strategy for a player that is winning (resp. cooperatively winning) from every state in its winning region is called *uniform winning* (resp. *uniform cooperatively winning*).

The solution to LTL$_f$ best-effort synthesis presented in [4] can be summarized as follows.

Algorithm 0 [4]**.** Given an LTL$_f$ best-effort synthesis problem $\mathcal{P} = (\mathcal{E}, \varphi)$, proceed as follows:

1. For every $\xi \in \{\neg\mathcal{E}, \mathcal{E} \to \varphi, \mathcal{E} \wedge \varphi\}$ compute the DFAs $\mathcal{A}_\xi = (\mathcal{D}_\xi, F_\xi)$.
2. Form the product $\mathcal{D} = \mathcal{D}_{\neg\mathcal{E}} \times \mathcal{D}_{\mathcal{E}\to\varphi} \times \mathcal{D}_{\mathcal{E}\wedge\varphi}$. Lift the final states of each component to the product, i.e. if $\mathcal{A}_\xi = (D_\xi, F_\xi)$ is the DFA for ξ, then the lifted condition G_ξ consists of all states $(s_{\neg\mathcal{E}}, s_{\mathcal{E}\to\varphi}, s_{\mathcal{E}\wedge\varphi})$ s.t. $s_\xi \in F_\xi$.
3. In DFA game $(\mathcal{D}, G_{\mathcal{E}\to\varphi})$ compute a uniform positional winning strategy f_{ag}. Let $W_{ag} \subseteq S$ be the agent's winning region.
4. In DFA game $(\mathcal{D}, G_{\neg\mathcal{E}})$ compute the environment's winning region $V \subseteq Q$.
5. Compute the environment restriction $\mathcal{D}'$ of $\mathcal{D}$ to V.
6. In DFA game $(\mathcal{D}', G_{\mathcal{E}\wedge\varphi})$ find a uniform positional cooperatively winning strategy g_{ag}.
7. **Return** the agent strategy σ_{ag} induced by the positional strategy k_{ag}, which is defined as follows: $k_{ag}(s) = \begin{cases} f_{ag}(s) & \text{if } s \in W_{ag}, \\ g_{ag}(s) & \text{otherwise.} \end{cases}$

4 Symbolic LTL$_f$ Best-Effort Synthesis

We present in this section three different symbolic approaches to LTL$_f$ best-effort synthesis, namely monolithic, explicit-compositional, and symbolic-compositional, as depicted in Fig. 1. In particular, we base on the symbolic techniques of DFA games presented in [22], which we briefly review below.

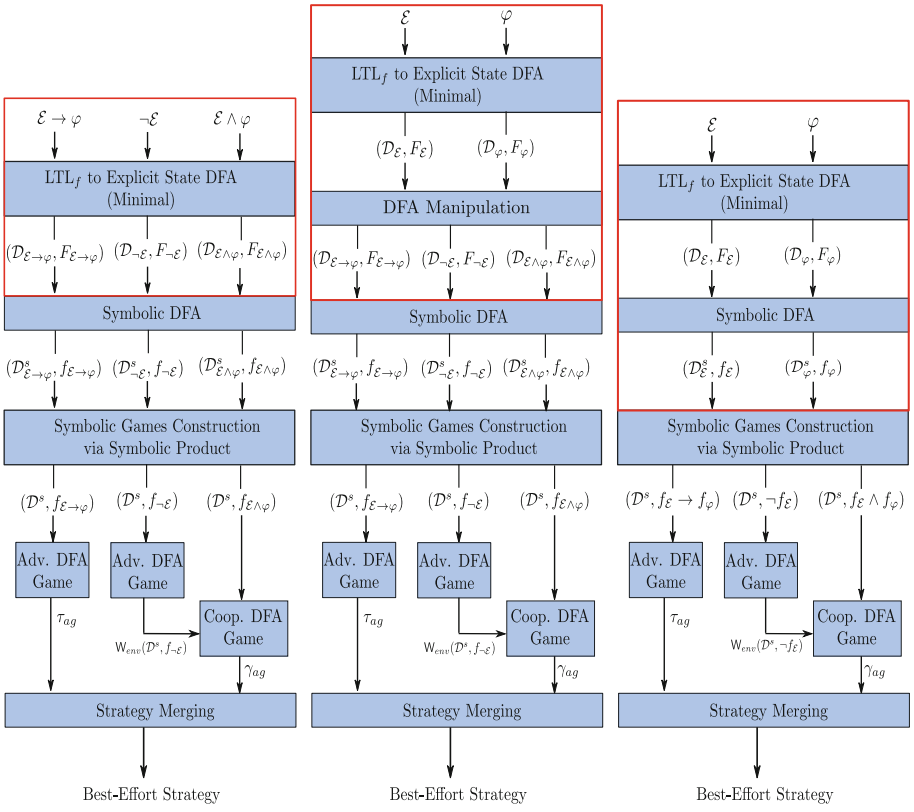

Fig. 1. From left to right, (a) monolithic, (b) explicit-compositional, and (c) symbolic-compositional techniques to LTL$_f$ best-effort synthesis. In particular, $\mathcal{D}^s = \mathcal{D}^s_{\mathcal{E}\to\varphi} \times \mathcal{D}^s_{\neg\mathcal{E}} \times \mathcal{D}^s_{\mathcal{E}\wedge\varphi}$ in (a) and (b). $\mathcal{D}^s = \mathcal{D}^s_{\mathcal{E}} \times \mathcal{D}^s_{\varphi}$ in (c). The specific operations of the three techniques are enclosed in red boxes.

4.1 Symbolic DFA Games

We consider the DFA representation described in Sect. 3 as an explicit-state representation. Instead, we are able to represent a DFA more compactly in a symbolic way by using a logarithmic number of propositions to encode the state

space. More specifically, the *symbolic* representation of $\mathcal{D}$ is a tuple $\mathcal{D}^s = (\mathcal{X}, \mathcal{Y}, \mathcal{Z}, Z_0, \eta)$, where $\mathcal{Z}$ is a set of state variables such that $|\mathcal{Z}| = \lceil \log |S| \rceil$, and every state $s \in S$ corresponds to an interpretation $Z \in 2^{\mathcal{Z}}$ over $\mathcal{Z}$; $Z_0 \in 2^{\mathcal{Z}}$ is the interpretation corresponding to the initial state s_0; $\eta \colon 2^{\mathcal{X}} \times 2^{\mathcal{Y}} \times 2^{\mathcal{Z}} \to 2^{\mathcal{Z}}$ is a Boolean function such that $\eta(Z, X, Y) = Z'$ if and only if Z is the interpretation of a state s and Z' is the interpretation of the state $\delta(s, X \cup Y)$. The set of goal states is represented by a Boolean function f over $\mathcal{Z}$ that is satisfied exactly by the interpretations of states in F. In the following, we denote symbolic DFAs as pairs $(\mathcal{D}^s, f)$.

Given a symbolic DFA game $(\mathcal{D}^s, f)$, we can compute a positional uniform winning agent strategy through a least fixpoint computation over two Boolean formulas w over $\mathcal{Z}$ and t over $\mathcal{Z} \cup \mathcal{Y}$, which represent the agent winning region and winning states with agent actions such that, regardless of how the environment behaves, the agent reaches the final states, respectively. Specifically, w and t are initialized as $w_0(\mathcal{Z}) = f(\mathcal{Z})$ and $t_0(\mathcal{Z}, \mathcal{Y}) = f(\mathcal{Z})$, since every goal state is an agent winning state. Note that t_0 is independent of the propositions from $\mathcal{Y}$, 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:

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

The computation reaches a fixpoint when $w_{i+1} \equiv w_i$. To see why a fixpoint is eventually reached, note that function w_{i+1} is *monotonic*. That is, at each step, a state Z is added to the winning region w_{i+1} only if it has not been already detected as a winning state, written $\neg w_i(Z)$ in function $t_{i+1}(Z, Y)$ above, *and* there exists an agent choice Y such that, for every environment choice X, the agent moves in w_i, written $\forall X. w_i(\eta(X, Y, Z))$.

When the fixpoint is reached, 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 determine 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 [14]. More specifically, by passing t_i to a Boolean synthesis procedure, setting $\mathcal{Z}$ as input variables and $\mathcal{Y}$ as output variables, we obtain a uniform positional winning strategy $\tau : 2^{\mathcal{Z}} \to 2^{\mathcal{Y}}$ that can be used to induce an agent winning strategy.

Computing a uniform positional cooperatively winning strategy can be performed through an analogous least-fixpoint computation. To do this, we define again Boolean functions $\hat{w}$ over $\mathcal{Z}$ and $\hat{t}$ over $\mathcal{Z} \cup \mathcal{Y}$, now representing the agent cooperatively winning region and cooperatively winning states with agent actions such that, if the environment behaves cooperatively, the agent reaches the final states. Analogously, we initialize $\hat{w}_0(\mathcal{Z}) = f(\mathcal{Z})$ and $\hat{t}_0(\mathcal{Z}, \mathcal{Y}) = f(\mathcal{Z})$. Then, we construct $\hat{t}_{i+1}$ and $\hat{w}_{i+1}$ as follows:

$$\hat{t}_{i+1}(Z, Y) = \hat{t}_i(Z, Y) \vee (\neg \hat{w}_i(Z) \wedge \exists X. \hat{w}_i(\eta(X, Y, Z)))$$
$$\hat{w}_{i+1}(Z) = \exists Y. \hat{t}_{i+1}(Z, Y);$$

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

Sometimes, the state space of a symbolic transition system must be restricted to not reach a given set of invalid states represented as a Boolean function. To do so, we redirect all transitions from states in the set to a *sink* state. Formally:

Definition 6. *Let $\mathcal{D}^s = (\mathcal{Z}, \mathcal{X}, \mathcal{Y}, Z_0, \eta)$ be a symbolic transition system and g a Boolean formula over $\mathcal{Z}$ representing a set of states. The restriction of $\mathcal{D}^s$ to g is a new symbolic transition system $\mathcal{D}'^s = (\mathcal{Z}, \mathcal{X}, \mathcal{Y}, Z_0, \eta')$, where η' only agrees with η if $Z \models g$, i.e., $\eta' = \eta \wedge g$.*

4.2 Monolithic Approach

The monolithic approach is a direct implementation of the best-effort synthesis approach presented in [4] (i.e., of Algorithm 0), utilizing the symbolic synthesis framework introduced in [22]. Given a best-effort synthesis problem $\mathcal{P} = (\mathcal{E}, \varphi)$, we first construct the DFAs following the synthesis algorithm described in Sect. 3, and convert them into a symbolic representation. Then, we solve suitable games on the symbolic DFAs and obtain a best-effort strategy. The workflow of the monolithic approach, i.e., **Algorithm** 1, is shown in Fig. 1(*a*). We elaborate on the algorithm as follows.

Algorithm 1. Given an LTL$_f$ best-effort synthesis problem $\mathcal{P} = (\mathcal{E}, \varphi)$, proceed as follows:

1. For LTL$_f$ formulas $\mathcal{E} \rightarrow \varphi$, $\neg\mathcal{E}$ and $\mathcal{E} \wedge \varphi$ compute the corresponding minimal explicit-state DFAs $\mathcal{A}_{\mathcal{E}\rightarrow\varphi} = (\mathcal{D}_{\mathcal{E}\rightarrow\varphi}, F_{\mathcal{E}\rightarrow\varphi})$, $\mathcal{A}_{\neg\mathcal{E}} = (\mathcal{D}_{\neg\mathcal{E}}, F_{\neg\mathcal{E}})$ and $\mathcal{A}_{\mathcal{E}\wedge\varphi} = (\mathcal{D}_{\mathcal{E}\wedge\varphi}, F_{\mathcal{E}\wedge\varphi})$.
2. Convert the DFAs to a symbolic representation to obtain $\mathcal{A}^s_{\mathcal{E}\rightarrow\varphi} = (\mathcal{D}^s_{\mathcal{E}\rightarrow\varphi}, f_{\mathcal{E}\rightarrow\varphi})$, $\mathcal{A}^s_{\neg\mathcal{E}} = (\mathcal{D}^s_{\neg\mathcal{E}}, f_{\neg\mathcal{E}})$ and $\mathcal{A}^s_{\mathcal{E}\wedge\varphi} = (\mathcal{D}^s_{\mathcal{E}\wedge\varphi}, f_{\mathcal{E}\wedge\varphi})$.
3. Construct the product $\mathcal{D}^s = \mathcal{D}^s_{\mathcal{E}\rightarrow\varphi} \times \mathcal{D}^s_{\neg\mathcal{E}} \times \mathcal{D}^s_{\mathcal{E}\wedge\varphi}$.
4. In DFA game $(\mathcal{D}^s, f_{\mathcal{E}\rightarrow\varphi})$, compute a uniform positional winning strategy τ_{ag} and the agent's winning region $\mathsf{W}_{ag}(\mathcal{D}^s, f_{\mathcal{E}\rightarrow\varphi})$.
5. In DFA game $(\mathcal{D}^s, f_{\neg\mathcal{E}})$, compute the environment's winning region $\mathsf{W}_{env}(\mathcal{D}^s, f_{\neg\mathcal{E}})$.
6. Compute the symbolic restriction $\mathcal{D}'^s$ of $\mathcal{D}^s$ to $\mathsf{W}_{env}(\mathcal{D}^s, f_{\neg\mathcal{E}})$ to restrict the state space of $\mathcal{D}^s$ to considering $\mathsf{W}_{env}(\mathcal{D}^s, f_{\neg\mathcal{E}})$ only.
7. In DFA game $(\mathcal{D}'^s, f_{\mathcal{E}\wedge\varphi})$, compute a uniform positional cooperatively winning strategy γ_{ag}.
8. **Return** the best-effort strategy σ_{ag} *induced* by the positional strategy κ_{ag} constructed as follows: $\kappa_{ag}(Z) = \begin{cases} \tau_{ag}(Z) & \text{if } Z \models \mathsf{W}_{ag}(\mathcal{D}^s, f_{\mathcal{E}\rightarrow\varphi}) \\ \gamma_{ag}(Z) & \text{otherwise.} \end{cases}$

The main challenge in the monolithic approach comes from the LTL$_f$-to-DFA conversion, which can take, in the worst case, double-exponential time [11], and thus is also considered the bottleneck of LTL$_f$ synthesis [22]. To that end, we propose an explicit-compositional approach to diminish this difficulty by decreasing the number of LTL$_f$-to-DFA conversions.

4.3 Explicit-Compositional Approach

As described in Sect. 4.2, the monolithic approach to a best-effort synthesis problem $\mathcal{P} = (\mathcal{E}, \varphi)$ involves three rounds of LTL$_f$-to-DFA conversions corresponding to LTL$_f$ formulas $\mathcal{E} \rightarrow \varphi$, $\neg\mathcal{E}$ and $\mathcal{E} \wedge \varphi$. However, observe that DFAs $\mathcal{A}_{\mathcal{E}\rightarrow\varphi}$, $\mathcal{A}_{\neg\mathcal{E}}$ and $\mathcal{A}_{\mathcal{E}\wedge\varphi}$ can, in fact, be constructed by manipulating the two DFAs $\mathcal{A}_{\mathcal{E}}$ and $\mathcal{A}_{\varphi}$ of LTL$_f$ formulas $\mathcal{E}$ and φ, respectively. Specifically, given the explicit-state DFAs $\mathcal{A}_{\varphi}$ and $\mathcal{A}_{\mathcal{E}}$, we obtain $\mathcal{A}_{\mathcal{E}\rightarrow\varphi}$, $\mathcal{A}_{\neg\mathcal{E}}$ and $\mathcal{A}_{\mathcal{E}\wedge\varphi}$ as follows:

- $\mathcal{A}_{\mathcal{E}\rightarrow\varphi} = \mathsf{Comp}(\mathsf{Inter}(\mathcal{A}_{\mathcal{E}}, \mathsf{Comp}(\mathcal{A}_{\varphi}))$;
- $\mathcal{A}_{\neg\mathcal{E}} = \mathsf{Comp}(\mathcal{A}_{\mathcal{E}})$;
- $\mathcal{A}_{\mathcal{E}\wedge\varphi} = \mathsf{Inter}(\mathcal{A}_{\mathcal{E}}, \mathcal{A}_{\varphi})$;

where Comp and Inter denote complement and intersection on explicit-state DFAs, respectively. Note that transforming LTL$_f$ formulas into DFAs takes double-exponential time in the size of the formula, while the complement and intersection of DFAs take polynomial time in the size of the DFA.

The workflow of the explicit-compositional approach, i.e., **Algorithm 2**, is shown in Fig. 1(*b*). As the monolithic approach, we first translate the formulas $\mathcal{E}$ and φ into minimal explicit-state DFAs $\mathcal{A}_{\mathcal{E}}$ and $\mathcal{A}_{\varphi}$, respectively. Then, DFAs $\mathcal{A}_{\mathcal{E}\rightarrow\varphi}$, $\mathcal{A}_{\neg\mathcal{E}}$ and $\mathcal{A}_{\mathcal{E}\wedge\varphi}$ are constructed by manipulating $\mathcal{A}_{\mathcal{E}}$ and $\mathcal{A}_{\varphi}$ through complement and intersection. Indeed, the constructed explicit-state DFAs are also minimized. The remaining steps of computing suitable DFA games are the same as in the monolithic approach.

4.4 Symbolic-Compositional Approach

The monolithic and explicit-compositional approaches are based on playing three games over the symbolic product of transition systems $\mathcal{D}_{\mathcal{E}\rightarrow\varphi}$, $\mathcal{D}_{\neg\mathcal{E}}$, and $\mathcal{D}_{\mathcal{E}\wedge\varphi}$. We observe that given DFAs $\mathcal{A}_{\mathcal{E}} = (\mathcal{D}_{\mathcal{E}}, F_{\mathcal{E}})$ and $\mathcal{A}_{\varphi} = (\mathcal{D}_{\varphi}, F_{\varphi})$ recognizing $\mathcal{E}$ and φ, respectively, the DFA recognizing any Boolean combination of $\mathcal{E}$ and φ can be constructed by taking the product of $\mathcal{D}_{\mathcal{E}}$ and $\mathcal{D}_{\varphi}$ and properly defining the set of final states over the resulting transition system.

Lemma 1. *Let $\mathcal{A}_{\psi_1} = (\mathcal{D}_{\psi_1}, F_{\psi_1})$ and $\mathcal{A}_{\psi_2} = (\mathcal{D}_{\psi_2}, F_{\psi_2})$ be the automata recognizing* LTL$_f$ *formulas ψ_1 and ψ_2, respectively, and $\psi = \psi_1$ op ψ_2 denoting an arbitrary Boolean combination of ψ_1 and ψ_2, i.e., op $\in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$. The* DFA *$\hat{\mathcal{A}}_{\psi} = (\hat{\mathcal{D}}_{\psi}, \hat{F}_{\psi})$ with $\hat{\mathcal{D}}_{\psi} = \mathcal{D}_{\psi_1} \times \mathcal{D}_{\psi_2}$ and $\hat{F}_{\psi} = \{(s_{\psi_1}, s_{\psi_2}) \mid s_{\psi_1} \in F_{\psi_1}$ op $s_{\psi_2} \in F_{\psi_2}\}$ recognizes ψ.*

Proof. ($\rightarrow$) Assume $\pi \models \psi$. We will prove that $\pi \in \mathcal{L}(\hat{\mathcal{A}}_{\varphi})$. To see this, observe that $\pi \models \psi$ implies $\pi \models \psi_1$ *op* $\pi \models \psi_2$. It follows by [11] that $\pi \in \mathcal{L}(\mathcal{A}_{\psi_1})$ *op* $\pi \in \mathcal{L}(\mathcal{A}_{\psi_2})$, meaning that running π in $\mathcal{D}_{\psi_1}$ and $\mathcal{D}_{\psi_2}$ yields the sequences of states $(s_0^{\psi_1}, \ldots, s_n^{\psi_1})$ and $(s_0^{\psi_2}, \ldots, s_n^{\psi_2})$ such that $s_n^{\psi_1} \in F_{\psi_1}$ *op* $s_n^{\psi_2} \in F_{\psi_2}$. Since $\hat{\mathcal{D}}_{\psi}$ is obtained through synchronous product of $\mathcal{D}_{\psi_1}$ and $\mathcal{D}_{\psi_2}$, running π in $\hat{\mathcal{A}}_{\psi}$ yields the sequence of states $((s_0^{\psi_1}, s_0^{\psi_2}), \ldots, (s_n^{\psi_1}, s_n^{\psi_2}))$, such that $(s_n^{\psi_1}, s_n^{\psi_2}) \in \hat{F}_{\psi}$. Hence, we have that $\pi \in \mathcal{L}(\hat{\mathcal{A}}_{\psi})$.

($\leftarrow$) Assume $\pi \in \mathcal{L}(\hat{\mathcal{A}}_\varphi)$. We prove that $\pi \models \psi$. To see this, observe that $\pi \in \mathcal{L}(\hat{\mathcal{A}}_\varphi)$ means that the run $\rho = (s_0^{\psi_1}, s_0^{\psi_2}) \dots (s_n^{\psi_1}, s_n^{\psi_2})$ induced by π on $\hat{\mathcal{D}}_\psi$ is such that $(s_n^{\psi_1}, s_n^{\psi_2}) \in \hat{F}_\psi$. This means, by construction of $\hat{F}_\psi$, that $(s_n^{\psi_1}, s_n^{\psi_2})$ s.t. $s_n^{\psi_1} \in F_{\psi_1}$ *op* $s_n^{\psi_2} \in F_{\psi_2}$. Since $\hat{\mathcal{D}}_\psi$ is obtained through synchronous product of $\mathcal{D}_{\psi_1}$ and $\mathcal{D}_{\psi_2}$, it follows that $\pi \in \mathcal{L}(\mathcal{A}_{\psi_1})$ *op* $\pi \in \mathcal{L}(\mathcal{A}_{\psi_2})$. By [11] we have that $\pi \models \psi_1$ *op* $\pi \models \psi_2$, and hence $\pi \models \psi$. □

Notably, Lemma 1 tells that the DFAs $\mathcal{A}_{\mathcal{E}\to\varphi}$, $\mathcal{A}_{\neg\mathcal{E}}$, and $\mathcal{A}_{\mathcal{E}\wedge\varphi}$ can be constructed from the same transition system by defining proper sets of final states. Specifically, given the DFAs $\mathcal{A}_\mathcal{E} = (\mathcal{D}_\mathcal{E}, F_\mathcal{E})$ and $\mathcal{A}_\varphi = (\mathcal{D}_\varphi, F_\varphi)$ recognizing $\mathcal{E}$ and φ, respectively, the DFAs recognizing $\mathcal{E} \to \varphi$, $\neg\mathcal{E}$, and $\mathcal{E} \wedge \varphi$ can be constructed as $\mathcal{A}_{\mathcal{E}\to\varphi} = (\mathcal{D}, F_{\mathcal{E}\to\varphi})$, $\mathcal{A}_{\neg\mathcal{E}} = (\mathcal{D}, F_{\neg\mathcal{E}})$, and $\mathcal{A}_{\mathcal{E}\wedge\varphi} = (\mathcal{D}, F_{\mathcal{E}\wedge\varphi})$, respectively, where $\mathcal{D} = \mathcal{D}_\mathcal{E} \times \mathcal{D}_\varphi$ and:

- $F_{\mathcal{E}\to\varphi} = \{(s_\mathcal{E}, s_\varphi) \mid s_\mathcal{E} \in F_\mathcal{E} \to s_\varphi \in F_\varphi\}$.
- $F_{\neg\mathcal{E}} = \{(s_\mathcal{E}, s_\varphi) \mid s_\mathcal{E} \notin F_\mathcal{E}\}$.
- $F_{\mathcal{E}\wedge\varphi} = \{(s_\mathcal{E}, s_\varphi) \mid s_\mathcal{E} \in F_\mathcal{E} \wedge s_\varphi \in F_\varphi\}$.

The symbolic-compositional approach precisely bases on this observation. As shown in Fig. 1(*c*), we first transform the LTL$_f$ formulas $\mathcal{E}$ and φ into minimal explicit-state DFAs $\mathcal{A}_\mathcal{E}$ and $\mathcal{A}_\varphi$, respectively, and then construct the symbolic representations $\mathcal{A}^s_\mathcal{E}$ and $\mathcal{A}^s_\varphi$ of them. Subsequently, we construct the symbolic product $\mathcal{D}^s = \mathcal{D}^s_\mathcal{E} \times \mathcal{D}^s_\varphi$, once and for all, and get the three DFA games by defining the final states (which are Boolean functions) from $f_\mathcal{E}$ and f_φ as follows:

- $f_{\mathcal{E}\to\varphi} = f_\mathcal{E} \to f_\varphi$.
- $f_{\neg\mathcal{E}} = \neg f_\mathcal{E}$.
- $f_{\mathcal{E}\wedge\varphi} = f_\mathcal{E} \wedge f_\varphi$.

From now on, the remaining steps are the same as in the monolithic and explicit-compositional approaches.

Algorithm 3. Given a best-effort synthesis problem $\mathcal{P} = (\mathcal{E}, \varphi)$, proceed as follows:

1. Compute the minimal explicit-state DFAs $\mathcal{A}_\mathcal{E} = (\mathcal{D}_\mathcal{E}, F_\mathcal{E})$ and $\mathcal{A}_\varphi = (\mathcal{D}_\varphi, F_\varphi)$.
2. Convert the DFAs to a symbolic representation to obtain $\mathcal{A}^s_\mathcal{E} = (\mathcal{D}^s_\mathcal{E}, f_\mathcal{E})$ and $\mathcal{A}^s_\varphi = (\mathcal{D}^s_\varphi, f_\varphi)$.
3. Construct the symbolic product $\mathcal{D}^s = \mathcal{D}^s_\mathcal{E} \times \mathcal{D}^s_\varphi$.
4. In DFA game $\mathcal{G}^s_{\mathcal{E}\to\varphi} = (\mathcal{D}^s, f_\mathcal{E} \to f_\varphi)$ compute a positional uniform winning strategy τ_{ag} and the agent winning region $\mathsf{W}_{ag}(\mathcal{D}^s, f_\mathcal{E} \to f_\varphi)$.
5. In the DFA game $(\mathcal{D}^s, \neg f_\mathcal{E})$ compute the environment's winning region $\mathsf{W}_{env}(\mathcal{D}^s, \neg f_\mathcal{E})$.
6. Compute the symbolic restriction $\mathcal{D}'^s$ of $\mathcal{D}^s$ to $\mathsf{W}_{env}(\mathcal{D}^s, f_{\neg\mathcal{E}})$ so as to restrict the state space of $\mathcal{D}^s$ to considering $\mathsf{W}_{env}(\mathcal{D}^s, f_{\neg\mathcal{E}})$ only.
7. In the DFA game $(\mathcal{D}'^s, f_\mathcal{E} \wedge f_\varphi)$ find a positional cooperatively winning strategy γ_{ag}.
8. **Return** the best-effort strategy σ_{ag} *induced* by the positional strategy κ_{ag} constructed as follows: $\kappa_{ag}(Z) = \begin{cases} \tau_{ag}(Z) & \text{if } Z \models \mathsf{W}_{ag}(\mathcal{D}^s, f_{\mathcal{E}\to\varphi}) \\ \gamma_{ag}(Z) & \text{otherwise.} \end{cases}$

5 Empirical Evaluations

In this section, we first describe how we implemented our symbolic LTL$_f$ best-effort synthesis approaches described in Sect. 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 and may even show better performance on certain instances.

5.1 Implementation

We implemented the three symbolic approaches to LTL$_f$ best-effort synthesis described in Sect. 4 in a tool called *BeSyft*, by extending the symbolic synthesis framework [20,22] integrated in state-of-the-art synthesis tools [6,9]. In particular, we based on Lydia[1], the overall best performing LTL$_f$-to-DFA conversion tool, to construct the minimal explicit-state DFAs of LTL$_f$ formulas. Moreover, *BeSyft* borrows the rich APIs from Lydia to perform relevant explicit-state DFA manipulations required by both Algorithm 1, i.e., the monolithic approach (c.f., Subsect. 4.2), and Algorithm 2, i.e., the explicit-compositional approach (c.f., Subsect. 4.3), such as complement, intersection, minimization. As in [20,22], the symbolic DFA games are represented in Binary Decision Diagrams (BDDs) [7], utilizing CUDD-3.0.0 [19] as the BDD library. Thereby, *BeSyft* constructs and solves symbolic DFA games using Boolean operations provided by CUDD-3.0.0, such as negation, conjunction, and quantification. The uniform positional winning strategy τ_{ag} and the uniform positional cooperatively winning strategy γ_{ag} are computed utilizing Boolean synthesis [14]. The positional best-effort strategy is obtained by applying suitable Boolean operations on τ_{ag} and γ_{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, explicit-compositional, and symbolic-compositional approach, respectively.

5.2 Experiment Methodology

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

Benchmarks. We devised a *counter-game* benchmark, based on the one proposed in [21]. More specifically, there is an n-bit binary counter and, 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 and its goal is to get the counter to have all bits set to 1. The increment requests only come from the environment, and occur in accordance with the environment specification.

[1] https://github.com/whitemech/lydia.

The size of the minimal DFA of a counter-game specification grows exponentially as n increases.

In the experiments, environment specifications ensure that the environment eventually issues a minimum number K of increment requests in sequence, which can be represented as LTL_f formulas $\mathcal{E}_K = \Diamond(add \wedge \bullet(add) \ldots \wedge \bullet(\ldots(\bullet(add))\ldots))$, where K is the number of conjuncts. Counter-game instances may be realizable depending on the parameter K and the number of bits n. In the case of a realizable instance, a strategy for the agent to enforce the goal is to grant all increment requests coming from the environment. Else, the agent can achieve the goal only if the environment behaves cooperatively, such as issuing more increment requests than that specified in the environment specification. That is, the agent needs a best-effort strategy. In our experiments, we considered counter-game instances with at most $n = 10$ bits and $K = 10$ sequential increment requests. As a result, our benchmarks consist of a total of 100 instances.

5.3 Experimental Results and Analysis

In our experiments, all *BeSyft* implementations are only able to solve counter-game instances with up to $n = 8$ bits. Figure 2 shows the comparison (in log scale) of the three symbolic implementations of best-effort synthesis on counter-game instances with $n = 8$ and $1 \leq K \leq 10$. First, we observe that *BeSyft*-Alg-1 (monolithic) and *BeSyft*-Alg-2 (explicit-compositional) reach timeout when $K \geq 8$, whereas *BeSyft*-Alg-3 (symbolic-compositional) is able to solve all 8-bit counter-game instances. We can also see that *BeSyft*-Alg-1 performs worse than the other two derivations since it requires three rounds of LTL_f-to-DFA conversions, which in the worst case, can lead to a double-exponential blowup. Finally, we note that *BeSyft*-Alg-3, which implements the symbolic-compositional approach, achieves orders of magnitude better performance than the other two implementations, although it does not fully exploit the power of DFA minimization. Nevertheless, it is not the case that automata minimization always leads to improvement. Instead, there is a tread-off of performing automata minimization. As shown in Fig. 2, *BeSyft*-Alg-3, performs better than *BeSyft*-Alg-2, though the former does not minimize the game arena after the 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-game instances with $n = 8$ and $1 \leq K \leq 10$ in Fig. 3. 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. Furthermore, we can see that the total time cost of solving the cooperative DFA game counts for less than 10% of the total time cost. As a result, we conclude that performing best-effort synthesis only brings a minimal overhead with respect to standard reactive synthesis, which consists of constructing the DFA of the input LTL_f formula and solving its corresponding adversarial game. Also, we observe that solving the cooperative game takes longer than solving the adversarial game. Indeed, this is because the fixpoint computation in the

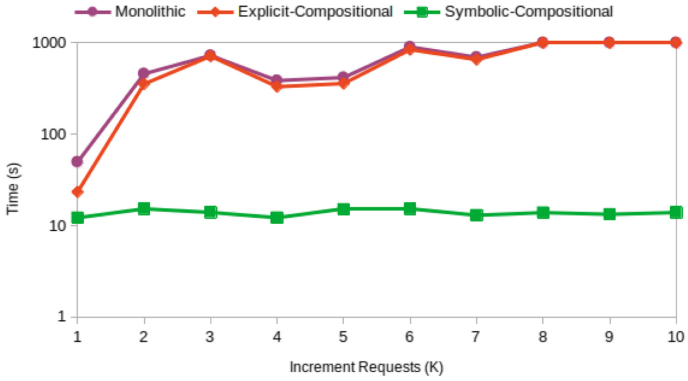

Fig. 2. Comparison (in log scale) of *BeSyft* implementations on counter game instances with $n = 8$ and $1 \leq K \leq 10$.

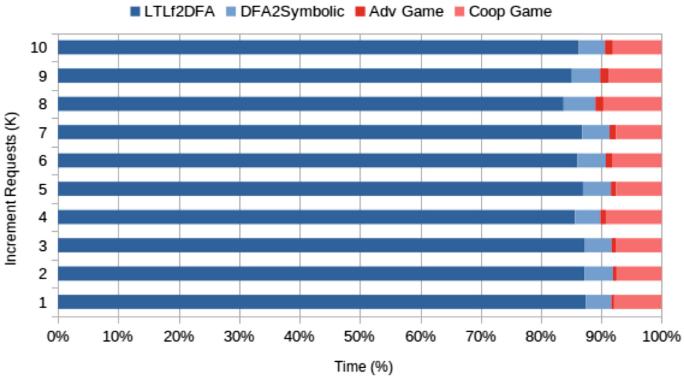

Fig. 3. Relative time cost of *BeSyft*-Alg-3 major operations on counter game instances with $n = 8$ and $1 \leq K \leq 10$.

cooperative game often requires more iterations than that in the adversarial game.

Finally, we also compared the time cost of symbolic-compositional best-effort synthesis with that of standard reactive synthesis on counter-game instances. More specifically, we considered a symbolic implementation of reactive synthesis that computes an agent strategy that enforces the LTL$_f$ formula $\mathcal{E} \rightarrow \varphi$ [10, 22], which can be used to find an agent strategy enforcing φ under $\mathcal{E}$, if it exists [3]. Interestingly, Fig. 4 shows that for certain counter-game instances, symbolic-compositional best-effort synthesis takes even less time than standard reactive synthesis. It should be noted that symbolic-compositional best-effort synthesis performs LTL$_f$-to-DFA conversions of LTL$_f$ formulas φ and $\mathcal{E}$ separately and combines them to obtain the final game arena without having automata minimization, whereas reactive synthesis performs the LTL$_f$-to-DFA conversion of formula $\mathcal{E} \rightarrow \varphi$ and minimizes its corresponding DFA. These results confirm

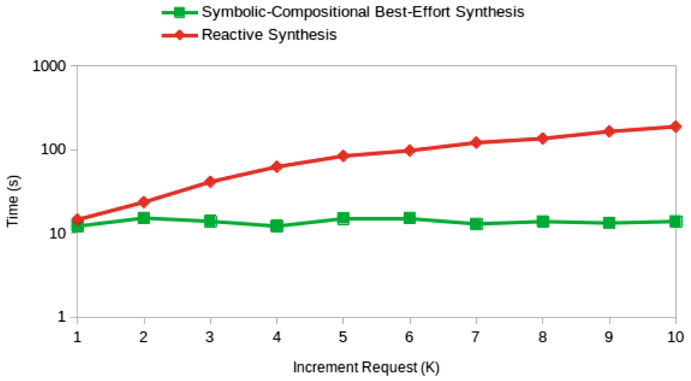

Fig. 4. Comparison (in log scale) of *BeSyft*-Alg-3 and implementations of symbolic LTL_f reactive synthesis on counter-game instances with $n = 8$ and $1 \leq K \leq 10$.

the practical feasibility of best-effort synthesis and that automata minimization does not always guarantee performance improvement.

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 [20,21], in the case of LTL_f best-effort synthesis, there can be a trade-off in doing so. Another significant finding is that the best-performing LTL_f best-effort synthesis approach only brings a minimal overhead compared to standard synthesis. Given this nice computational result, a natural future direction would be looking into LTL_f best-effort synthesis with multiple environment assumptions [1].

Acknowledgments. This work has been partially supported by the ERC-ADG White- Mech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the PRIN project RIPER (No. 20203FFYLK), and the PNRR MUR project FAIR (No. PE0000013).

References

1. Aminof, B., De Giacomo, G., Lomuscio, A., Murano, A., Rubin, S.: Synthesizing best-effort strategies under multiple environment specifications. In: KR, pp. 42–51 (2021)
2. Aminof, B., De Giacomo, G., Murano, A., Rubin, S.: Planning and synthesis under assumptions. arXiv (2018)
3. Aminof, B., De Giacomo, G., Murano, A., Rubin, S.: Planning under LTL environment specifications. In: ICAPS, pp. 31–39 (2019)

4. Aminof, B., De Giacomo, G., Rubin, S.: Best-effort synthesis: doing your best is not harder than giving up. In: IJCAI, pp. 1766–1772 (2021)
5. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
6. Bansal, S., Li, Y., Tabajara, L.M., Vardi, M.Y.: Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In: AAAI, pp. 9766–9774 (2020)
7. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv. **24**(3), 293–318 (1992)
8. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. AIJ **1**–**2**(147), 35–84 (2003)
9. De Giacomo, G., Favorito, M.: Compositional approach to translate LTL$_f$/LDL$_f$ into deterministic finite automata. In: ICAPS, pp. 122–130 (2021)
10. De Giacomo, G., Favorito, M.: Lydia: a tool for compositional LTL$_f$ /LDL$_f$ synthesis. In: ICAPS, pp. 122–130 (2021)
11. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI, pp. 854–860 (2013)
12. De Giacomo, G., Vardi, M.Y.: Synthesis for LTL and LDL on Finite Traces. In: IJCAI, pp. 1558–1564 (2015)
13. Finkbeiner, B.: Synthesis of reactive systems. Dependable Softw. Syst. Eng. **45**, 72–98 (2016)
14. Fried, D., Tabajara, L.M., Vardi, M.Y.: BDD-based Boolean functional synthesis. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 402–421. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_22
15. Ghallab, M., Nau, D.S., Traverso, P.: Automated Planning - Theory and Practice. Elsevier, Amsterdam (2004)
16. Henriksen, J.G., et al.: Mona: monadic second-order logic in practice. In: Brinksma, E., Cleaveland, W.R., Larsen, K.G., Margaria, T., Steffen, B. (eds.) TACAS 1995. LNCS, vol. 1019, pp. 89–110. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60630-0_5
17. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL, pp. 179–190 (1989)
18. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)
19. Somenzi, F.: CUDD: CU Decision Diagram Package 3.0.0. Universiy of Colorado at Boulder (2016)
20. Tabajara, L.M., Vardi, M.Y.: Partitioning techniques in LTL$_f$ synthesis. In: IJCAI, pp. 5599–5606 (2019)
21. Zhu, S., De Giacomo, G., Pu, G., Vardi, M.Y.: LTL$_f$ synthesis with fairness and stability assumptions. In: AAAI, pp. 3088–3095 (2020)
22. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: Symbolic LTL$_f$ synthesis. In: IJCAI, pp. 1362–1369 (2017)