SN Computer Science (2025) 6:147
https://doi.org/10.1007/s42979-024-03337-8

ORIGINAL RESEARCH

Symbolic LTL$_f$ Synthesis: A Unified Approach for Synthesizing Winning, Dominant, and Best-Effort Strategies

Giuseppe De Giacomo[1,2] · Gianmarco Parretti[2] · Shufang Zhu[3]

Received: 5 March 2024 / Accepted: 24 September 2024

Abstract
Synthesis typically focuses on finding strategies that win against all possible responses of the environment. When a winning strategy does not exist, the agent can either give up or do its best to achieve the goal. In this paper, we develop symbolic techniques to handle the latter case in the context of LTL$_f$. Specifically, we consider *winning*, *dominant*, and *best-effort* strategies, which achieve the goal against *all*, *the maximum* subset, and *a maximal* subset of environment responses, respectively. While a unified game-theoretic technique that simultaneously solves the three synthesis problems exists, we present several symbolic refinements of such technique. Depending on key choices, such refinements behave in a radically different way. We provide an effective implementation of our symbolic techniques and show, by empirical evaluation, how they compare in practice. In particular, we show that one of them brings only a minor overhead compared to existing standard synthesis techniques for winning strategies.

Keywords LTL$_f$ synthesis · Winning strategies · Dominant strategies · Best-effort strategies · Symbolic synthesis

Introduction

We consider an agent acting to achieve goals in a nondeterministic environment, as in Planning in nondeterministic (adversarial) domains [1–3]. However, we specify both the environment and the goal in Linear Temporal Logic (LTL) [4], the formalism typically used for specifying complex dynamic properties in Formal Methods [5].

Giuseppe De Giacomo, Gianmarco Parretti and Shufang Zhu have contributed equally to this work.

✉ Giuseppe De Giacomo
giuseppe.degiacomo@cs.ox.ac.uk; degiacomo@diag.uniroma1.it

✉ Gianmarco Parretti
parretti@diag.uniroma1.it

✉ Shufang Zhu
shufang.zhu@liverpool.ac.uk

[1] Department of Computer Science, University of Oxford, Parks Rd 7, Oxford OX1 3QG, UK

[2] Dipartimento Ingegneria Informatica, Automatica e Gestionale (DIAG), University of Rome "La Sapienza", Via Arioso 25, Rome 00185, Italy

[3] Department of Computer Science, University of Liverpool, Ashton Street, Liverpool L69 3BX, UK

In fact, we consider Linear Temporal Logic on finite traces (LTL$_f$) [6], which maintains the syntax of LTL [7], but is interpreted over finite traces. In this setting, we study synthesis [4, 8–10]. The standard solution concept in synthesis is that of a *winning* strategy, i.e., an agent strategy that achieves the goal regardless of how the environment responds. Unfortunately, not every synthesis problem admits a winning strategy, which leads to the introduction of other solution concepts, such as *dominant* and *best-effort* strategies [11–14]. When a winning strategy does not exist, dominant strategies capture the game-theoretic rationality principle that a player should use a strategy that "dominates" all its other strategies, i.e., one that achieves the goal against a *maximum* set of (vs. all) environment responses; when a dominant strategy does not exist, best-effort strategies capture the game-theoretic rationality principle that a player should *not* use a strategy that is "dominated" by another of its strategies, i.e., one that achieves the goal against a *maximal* (vs. maximum) set of environment responses.

Dominant and best-effort strategies have been investigated and proved to be related with winning strategies in [12, 14]. Specifically: (*i*) every winning strategy is a dominant strategy, and every dominant strategy is a best-effort strategy; (*ii*) if a winning strategy exists, the dominant strategies are exactly the winning strategies; (*iii*) if a dominant

Published online: 07 February 2025

strategy exists, the best-effort strategies are exactly the dominant strategies. Furthermore, best-effort strategies always exist [12], unlike winning and dominant strategies. Nonetheless, best-effort strategies can be computed in 2EXPTIME, as winning and dominant strategies (in fact, strategy synthesis is 2EXPTIME-complete for winning, dominant, and best-effort strategies [4, 10, 12, 14]).

Unified algorithms for synthesizing winning, dominant, and best-effort strategies for LTL and LTL_f have been presented in [14]. Such algorithms create, solve, and combine the solutions of three distinct games but of the same game arena. To compute the arena, these algorithms suitably combine the automata corresponding to the LTL_f formulas $\mathcal{E}$ and φ constituting the environment specification and the agent goal, respectively. Finally, these algorithms decide whether the synthesized strategy is winning, dominant, or best-effort only, by looking at properties of the solved games.

The algorithm for LTL_f appears to be promising in practice. In fact, well-performing techniques for each component of that 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 [15–18]; (*ii*) Cartesian product of DFAS, which is polynomial; (*iii*) minimization of DFAS, which is also polynomial; (*iv*) fixpoint computation over DFAS to compute adversarial and cooperative winning strategies for reaching the final states, which is again polynomial.

In this paper, we refine the unified LTL_f synthesis algorithm presented in [14] by using symbolic techniques [5, 16, 19, 20]. Specifically, we present three different symbolic approaches that combine the above operations in different ways (and, in fact, allow for different levels of DFA minimization). We then compare implementations of the three symbolic approaches through empirical evaluations. From the comparison, a clear winner emerges. Interestingly, the winner does not fully exploit DFA minimization to minimize the DFA whenever it is possible, but gives up on minimization at some level. Furthermore, the empirical results show that our techniques perform with a minor overhead compared to existing standard synthesis techniques that compute winning strategies. In fact, both computing best-effort strategies and checking the existence of dominant strategies can be performed very efficiently compared to transforming LTL_f formulas into DFAS, the bottleneck of the synthesis [16]. In particular, we observed that our symbolic technique for checking the existence of dominant strategies is very efficient, bringing virtually no overhead to the total time cost of the synthesis. These results confirm that unified techniques that synthesize winning, dominant, and best-effort strategies are indeed well suited for efficient and scalable implementations.

The rest of the paper is organized as follows. In Section “Preliminaries”, we recall the main notions of LTL_f reactive synthesis. In Section “Winning, Dominant, and Best-Effort Strategies ”, we formally introduce winning, dominant, and best-effort strategies, and, in Section “Local Characterization of Strategies”, we review their corresponding local characterizations. In Section “Synthesis Technique”, we review basic notions of DFA games and the unified LTL_f synthesis technique presented in [14]. Based on that algorithm, we present in Section “Checking Dominance” a simple technique to check the existence of dominant strategies that is well suited for symbolic implementation. In Section “Symbolic Synthesis Techniques”, we present three distinct symbolic techniques to synthesize winning, dominant, and best-effort strategies: the first (c.f., Subsection “Direct Technique”) is a direct symbolic implementation of the algorithm presented in [14]; the second (c.f., Subsection “Compositional-Minimal Technique”) favors maximally conducting DFA minimization, thus getting the smallest possible arena for the three games; the third (c.f. Subsection “Compositional Technique”) gives up DFA minimization at some level when creating arena for the three games. In Section “Checking Dominance Symbolically”, we refine our algorithm for checking the existence of dominant strategies presented in Section “Checking Dominance” by using symbolic techniques. In Section “Implementation and Empirical Evaluation”, we present an empirical evaluation of our techniques. We review related works in Section “Related Works” and conclude the paper in Section “Conclusion”.

Preliminaries

A *trace* over an alphabet of symbols Σ, denoted by π, is a finite or infinite sequence of elements from Σ. Traces are indexed starting at zero, and we write $\pi = \pi_0\pi_1 \cdots$. For a finite trace π, let $\mathsf{lst}(\pi)$ be the index of the last element of π, i.e., $\mathsf{lst}(\pi) = |\pi| - 1$. The empty trace is denoted by λ.

Linear Temporal Logic on Finite Traces (LTL_f) *Linear Temporal Logic on Finite Traces* (LTL_f) [6] is a formalism for expressing temporal properties over finite, non-empty traces. LTL_f shares the same syntax as Linear Temporal Logic (LTL) [7], but is instead interpreted over finite traces. Given an alphabet of propositional symbols AP, LTL_f formulas are generated by the following grammar:

$$\varphi := \top \mid \bot \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \supset \varphi_2 \\ \mid \circ\varphi \mid \bullet\,\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid \varphi_1 \mathcal{R} \varphi_2,$$

where $a \in AP$ is an atom. The temporal operators are: $\circ$ (*next*); $\bullet$ (*weak next*); $\Diamond$ (*eventually*); $\Box$ (*always*); $\mathcal{U}$ (*until*); and $\mathcal{R}$ (*release*). The size of φ, written $|\varphi|$, is the number of subformulas in its abstract syntax tree [6].

LTL$_f$ formulas are interpreted over finite traces π over the alphabet $\Sigma = 2^{AP}$, i.e., the alphabet consisting of propositional interpretations of AP. Thus, for $0 \leq i \leq \mathsf{lst}(\pi)$, $\pi_i \in 2^{AP}$ is the i-th interpretation of π. Given a finite trace π, we define when an LTL$_f$ formula φ *holds* at an instant i, written $\pi, i \models \varphi$, inductively on the structure of φ as:

- $\pi, i \models \top$;
- $\pi, i \not\models \bot$;
- $\pi, i \models a$ iff $a \in \pi_i$ (for $a \in AP$);
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \varphi_1 \vee \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$;
- $\pi, i \models \varphi_1 \supset \varphi_2$ iff $\pi, i \models \varphi_1$ implies $\pi, i \models \varphi_2$;
- $\pi, i \models \circ\varphi$ iff $i < \mathsf{lst}(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \bullet\varphi$ iff $i < \mathsf{lst}(\pi)$ implies $\pi, i+1 \models \varphi$;
- $\pi, i \models \Diamond\varphi$ iff $\exists j.\ i \leq j \leq \mathsf{lst}(\pi)$ such that $\pi, j \models \varphi$;
- $\pi, i \models \Box\varphi$ iff $\forall j.\ i \leq j \leq \mathsf{lst}(\pi)$ holds that $\pi, j \models \varphi$;
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists j.\ i \leq j \leq \mathsf{lst}(\pi)$ such that $\pi, j \models \varphi_2$, and $\forall k.\ i \leq k < j$ holds that $\pi, k \models \varphi_1$;
- $\pi, i \models \varphi_1 \mathcal{R} \varphi_2$ iff $\forall j.\ i \leq j \leq \mathsf{lst}(\pi)$ it either holds that $\pi, j \models \varphi_2$ or $\exists k.\ i \leq k < j$ such that $\pi, k \models \varphi_1$.

Observe that we have the usual Boolean equivalences such as $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ and $\varphi_1 \supset \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$. Also, note that the following equivalences hold: $\circ\varphi \equiv \neg \bullet \neg\varphi$ (i.e., $\circ$ and $\bullet$ are dual operators); $\Diamond\varphi \equiv \top\mathcal{U}\varphi$; $\Box\varphi \equiv \bot\mathcal{R}\varphi$; $\Diamond\varphi \equiv \neg\Box\neg\varphi$ (i.e., $\Diamond$ and $\Box$ are dual operators); and $\varphi_1\mathcal{U}\varphi_2 \equiv \neg(\neg\varphi_2\mathcal{R}\neg\varphi_1)$ (i.e., $\mathcal{U}$ and $\mathcal{R}$ are dual operators).

A finite trace π *satisfies* an LTL$_f$ formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

LTL$_f$ ***Reactive Synthesis*** Reactive synthesis [8] is the problem of computing a strategy that allows the agent to achieve its goal while continuously interacting with an uncontrollable environment. Intuitively, reactive synthesis can be considered as a two-player game between the agent and the environment. At each step, both players make their respective moves, hence generating an execution sequence. The goal of the agent is to find a strategy that forces the generated execution sequence to satisfy certain expected behaviors. These expected behaviors are often expressed in logical specifications, which we call *goal specifications*. In this paper, we focus on goal specifications expressed in LTL$_f$.

More specifically, an LTL$_f$ goal specification is defined over propositional symbols $\mathcal{Y} \cup \mathcal{X}$, where $\mathcal{Y}$ and $\mathcal{X}$ are disjoint sets of variables under the control of the *agent* and the *environment*, respectively. It is worth noting that, LTL$_f$ goal specifications can capture all LTL *guarantee* specifications [21]. Infinite traces over $\mathcal{Y} \cup \mathcal{X}$ are denoted as $\pi = (Y_0 \cup X_0)(Y_1 \cup X_1) \cdots \in (2^{\mathcal{Y} \cup \mathcal{X}})^\omega$, where $Y_i \subseteq \mathcal{Y}$ and $X_i \subseteq \mathcal{X}$ (for $i \geq 0$) denote the respective moves of the agent and the environment, and are also called *plays*. Finite traces $\pi \in (2^{\mathcal{Y} \cup \mathcal{X}})^*$ are denoted analogously and are also called *histories*.

An *agent strategy* is a function $\sigma_{ag} : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ that maps sequences of environment moves to an agent move. An *environment strategy* is a function $\sigma_{env} : (2^{\mathcal{Y}})^+ \to 2^{\mathcal{X}}$ that maps sequences of agent moves to an environment move. The domain of σ_{ag} includes the empty sequence of environment moves λ as we assume that the agent moves first. A trace $\pi = (Y_0 \cup X_0)(Y_1 \cup X_1) \cdots$ is σ_{ag}-*consistent* if $Y_0 = \sigma_{ag}(\lambda)$ and $Y_i = \sigma_{ag}(X_0 \cdots X_{i-1})$ for every $i > 0$. Analogously, π is σ_{env}-*consistent* if $X_i = \sigma_{env}(Y_0 \cdots Y_i)$ for every $i \geq 0$. We denote by $\pi(\sigma_{ag}, \sigma_{env})$ the unique trace that is consistent with both σ_{ag} and σ_{env}.

Given an LTL$_f$ goal φ, an agent strategy σ_{ag} is *winning* for (aka *enforces*) φ if, for every environment strategy σ_{env}, there exists a finite prefix of $\pi(\sigma_{ag}, \sigma_{env})$ that satisfies φ. An agent strategy σ_{ag} is *cooperatively winning* for φ if there exists an environment strategy σ_{env} such that there exists a finite prefix of $\pi(\sigma_{ag}, \sigma_{env})$ that satisfies φ. That is, a winning strategy allows the agent to satisfy the goal regardless all environment strategies, while a cooperatively winning strategy allows the agent to satisfy the goal for at least one environment strategy. LTL$_f$ reactive synthesis is the problem of computing a winning strategy for φ, if one exists.

Definition 1 [10] The problem of LTL$_f$ reactive synthesis is defined as a tuple $\mathcal{P} = (\mathcal{Y}, \mathcal{X}, \varphi)$, where: $\mathcal{Y}$ and $\mathcal{X}$ are disjoint sets of variables under the control of the agent and the environment, respectively; and φ is an LTL$_f$ agent goal over $\mathcal{Y} \cup \mathcal{X}$. Synthesis of $\mathcal{P}$ computes an agent winning strategy σ_{ag} for φ, if one exists; otherwise, it returns *unrealizable*.

LTL$_f$ ***Reactive Synthesis Under Environment Specifications*** Reactive synthesis under environment specifications [4] is the problem of computing an agent strategy that allows the agent to achieve its goal considering certain knowledge of how the environment behaves, which is captured as an *environment specification*. In this paper, we consider environment specifications expressed in LTL$_f$.

Let $\mathcal{E}$ be an LTL$_f$ formula over $\mathcal{Y} \cup \mathcal{X}$. An environment strategy σ_{env} is *winning* for (aka *enforces*) $\mathcal{E}$ if, for every agent strategy σ_{ag}, *every* finite prefix of $\pi(\sigma_{ag}, \sigma_{env})$ satisfies $\mathcal{E}$. An *environment specification* $\mathcal{E}$ is an LTL$_f$ formula such that there exists an environment strategy that is winning for $\mathcal{E}$. We denote by $\Sigma^{\mathcal{E}}_{env}$ the set of environment strategies σ_{env} that enforce the environment specification $\mathcal{E}$. Using LTL$_f$ for environment specifications allows to capture all LTL *safety* specifications [21], which notably include fully observable nondeterministic (FOND) planning domains [4, 22, 23], say written in PDDL [24], and extend them, e.g., with non-Markovian features [25].

Within this framework, an agent strategy is *winning for φ under $\mathcal{E}$* (aka *enforces* φ under $\mathcal{E}$) if, for every environment strategy σ_{env} that enforces $\mathcal{E}$, there exists a finite prefix of $\pi(\sigma_{ag}, \sigma_{env})$ that satisfies φ. Analogously, an agent strategy is *cooperatively winning for φ under $\mathcal{E}$* if there exists an environment strategy σ_{env} that enforces $\mathcal{E}$ and such that there exists a finite prefix of $\pi(\sigma_{ag}, \sigma_{env})$ that satisfies φ. That is, a winning strategy for φ under $\mathcal{E}$ guarantees that, regardless of all environment strategies enforcing $\mathcal{E}$, the agent will satisfy its goal φ, while a cooperatively winning strategy for φ under $\mathcal{E}$ guarantees that there exists at least an environment strategy enforcing $\mathcal{E}$ such that the agent will satisfy its goal. Reactive synthesis under environment specifications is the problem of finding a winning strategy for φ under $\mathcal{E}$, if one exists.

Definition 2 [4] The problem of LTL_f reactive synthesis under environment specifications is defined as a tuple $\mathcal{P} = (\mathcal{Y}, \mathcal{X}, \varphi, \mathcal{E})$, where: $\mathcal{Y}$ and $\mathcal{X}$ are disjoint sets of variables under the control of the agent and the environment, respectively; and φ and $\mathcal{E}$ are LTL_f formulas over $\mathcal{Y} \cup \mathcal{X}$ denoting an agent goal and an environment specification, respectively. Synthesis of $\mathcal{P}$ computes a winning strategy σ_{ag} for φ under $\mathcal{E}$, if one exists; otherwise, it returns *unrealizable*.

LTL_f reactive synthesis, both with and without environment specifications, is 2EXPTIME-complete [4, 10]. LTL_f reactive synthesis under environment specifications is a generalization of standard LTL_f reactive synthesis obtained by taking $\mathcal{E} = \top$.

Winning, Dominant, and Best-Effort Strategies

While winning strategies guarantee that the agent enforces its goal, not every synthesis problem admits a winning strategy. In cases where no winning strategy exists, the synthesis procedure usually terminates and declares that the problem is *unrealizable*. Such scenarios call for notions of strategy synthesis that are less strict than the usual ones used in Formal Methods [9] and Planning [3]. In this section, we review *dominant* and *best-effort* strategies, which are based on the game-theoretic notion of *dominance* [26]. These strategies have been investigated in the context of LTL_f goals and environment specifications in [11–14].

Definition 3 *(Dominance* [11, Sec. 3]*)* Let φ and $\mathcal{E}$ be LTL_f formulas over $\mathcal{Y} \cup \mathcal{X}$ denoting an agent goal and an environment specification, respectively. σ_1 and σ_2 are agent strategies. σ_1 *dominates* σ_2, written $\sigma_1 \geq_{\varphi|\mathcal{E}} \sigma_2$, if, and only if, for every $\sigma_{env} \in \Sigma^{\mathcal{E}}_{env}$, if some finite prefix of $\pi(\sigma_2, \sigma_{env})$ satisfies φ, then some finite prefix of $\pi(\sigma_1, \sigma_{env})$ satisfies φ. Furthermore, σ_1 *strictly dominates* σ_2, written $\sigma_1 >_{\varphi|\mathcal{E}} \sigma_2$, if, and only if, $\sigma_1 \geq_{\varphi|\mathcal{E}} \sigma_2$ and $\sigma_2 \not\geq_{\varphi|\mathcal{E}} \sigma_1$.

It should be noted that $\geq_{\varphi|\mathcal{E}}$ is a *preorder*, while $>_{\varphi|\mathcal{E}}$ is a *strict partial order*. An agent strategy that is *maximum* in the preorder $\geq_{\varphi|\mathcal{E}}$ is called *dominant*.

Definition 4 *(Dominant Strategy* [14, Sec. 3]*)* An agent strategy σ is dominant for φ under $\mathcal{E}$ if, and only if, for every agent strategy σ', we have $\sigma \geq_{\varphi|\mathcal{E}} \sigma'$. If there is no environment specification, i.e., $\mathcal{E} = \top$, we simply say that σ is dominant for φ.

When a winning strategy does not exist, the agent can search for a dominant strategy. In this case, an agent using a dominant strategy satisfies its goal against a maximum set of environment strategies (though not all) that conform to the environment specification. Doing so is the *best possible decision* for the agent when no winning strategy exists. However, while dominant strategies may exist when no winning strategy exists, it is not the case that a dominant strategy always exists [14].

A slightly weaker notion than that of dominant strategy is that of *best-effort* strategy. Best-effort strategies are *maximal* in the strict partial order $>_{\varphi|\mathcal{E}}$.

Definition 5 *(Best-Effort Strategy* [12, Sec. 3]*)* An agent strategy σ is best-effort for φ under $\mathcal{E}$ if, and only if, it does not exist an agent strategy σ' such that $\sigma' >_{\varphi|\mathcal{E}} \sigma$. If there is no environment specification, i.e., $\mathcal{E} = \top$, we simply say that σ is best-effort for φ.

The main idea behind the notion of best-effort strategy is the following. 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 at least one such strategy. As a result, an agent using σ_2 is not doing its best, since it could achieve its goal against a strictly larger set of environment strategies using σ_1. It should be noted that, differently from dominant strategies, distinct best-effort strategies may achieve the goal against distinct maximal sets of environment strategies enforcing $\mathcal{E}$. That is, an agent using a best-effort strategy makes the *best possible decision* given that such a decision is made *before* the exact environment behavior is known.

A notable property of best-effort strategies is that, unlike winning and dominant strategies, they always exist, regardless of what the agent goal φ and the environment specification $\mathcal{E}$ are.

Theorem 1 (Existence of a Best-Effort Strategy [12, Thm. 3]) *Let φ and $\mathcal{E}$ be LTL_f formulas over $\mathcal{Y} \cup \mathcal{X}$ denoting an* *agent goal and an environment specification, respectively. There always exists a best-effort strategy for φ under $\mathcal{E}$.*

When φ and $\mathcal{E}$ are clear from the context we may simply say that σ is winning, dominant, and best-effort, as an abbreviation for σ is winning, dominant, and best-effort for φ under $\mathcal{E}$, respectively.

We now review the relation between winning, dominant, and best-effort strategies [14]. Let σ_{ag} be an agent strategy. We denote by $\Sigma^{\mathcal{E}}_{env}(\varphi, \sigma_{ag}) \subseteq \Sigma^{\mathcal{E}}_{env}$ the set of environment strategies σ_{env} that enforce $\mathcal{E}$ and such that $\pi(\sigma_{ag}, \sigma_{env})$ has a finite prefix that satisfies φ. With this notation, we can characterize winning, dominant, and best-effort strategies as follows: (*i*) σ_{ag} is a winning strategy if, and only if, $\Sigma^{\mathcal{E}}_{env}(\varphi, \sigma_{ag}) = \Sigma^{\mathcal{E}}_{env}$, i.e., σ_{ag} enforces the goal φ against every environment strategy enforcing $\mathcal{E}$; (*ii*) σ_{ag} is a dominant strategy if, and only if, for every other agent strategy σ'_{ag}, it holds that $\Sigma^{\mathcal{E}}_{env}(\varphi, \sigma'_{ag}) \subseteq \Sigma^{\mathcal{E}}_{env}(\varphi, \sigma_{ag})$, i.e., σ_{ag} satisfies the goal against a *maximum* set of environment strategies enforcing $\mathcal{E}$; and (*iii*) σ_{ag} is a best-effort strategy if, and only if, for every other agent strategy σ'_{ag}, $\Sigma^{\mathcal{E}}_{env}(\varphi, \sigma_{ag}) \not\subset \Sigma^{\mathcal{E}}_{env}(\varphi, \sigma'_{ag})$, i.e., σ_{ag} satisfies the goal against a *maximal* set of environment strategies. As an immediate consequence, we have the following:

Theorem 2 (Relation Between Winning, Dominant, and Best-Effort Strategies [14, Thm. 1]) *Let φ and $\mathcal{E}$ be* LTL$_f$ *formulas over $\mathcal{Y} \cup \mathcal{X}$ denoting an agent goal and an environment specification, respectively. The following hold*:

- *Every winning strategy is a dominant strategy and every dominant strategy is a best-effort strategy;*
- *If a winning strategy exists, the dominant strategies are exactly the winning strategies;*
- *If a dominant strategy exists, the best-effort strategies are exactly the dominant strategies.*

Synthesis problems of best-effort and dominant strategies are 2EXPTIME-complete [12, 14], as the synthesis of winning strategies [4, 10]. An immediate consequence of such result and Theorems 1 and 2 is the following alternative approach to strategy synthesis: instead of searching for a winning strategy, which may not exist, one should directly search for a best-effort strategy which, on one hand, always exists, and, on the other hand, is a dominant (resp. winning) strategy if the problem admits a dominant (resp. winning) one.

With this result, we are now ready to define our synthesis task. Intuitively, we are interested in the synthesis of best-effort strategies for LTL $_f$ goals under environment specifications and deciding whether the synthesized strategies are winning and/or dominant as well. We formalize this problem below.

Definition 6 (LTL$_f$ *Best-Effort Synthesis Under Environment Specifications* [12, 14]) The problem of LTL$_f$ best-effort synthesis under environment specifications is defined as a tuple $\mathcal{P} = (\mathcal{Y}, \mathcal{X}, \varphi, \mathcal{E})$, where: $\mathcal{Y}$ and $\mathcal{X}$ are disjoint sets of variables under the control of the agent and the environment, respectively; and φ and $\mathcal{E}$ are LTL$_f$ formulas over $\mathcal{Y} \cup \mathcal{X}$ denoting an agent goal and an environment specification, respectively. Best-effort synthesis of $\mathcal{P}$ computes a best-effort strategy σ_{ag} for φ under $\mathcal{E}$, and decides whether σ is *winning*, *dominant*, or *best-effort* only.

Local Characterization of Strategies

We now review the alternative characterization of winning, dominant, and best-effort strategies based on their local behavior when executed after a history, i.e., finite sequences of agent and environment moves $\pi \in (2^{\mathcal{Y} \cup \mathcal{X}})^*$ [12, 14]. We first define the value of a history h depending on how the agent can extend h to satisfy the LTL$_f$ goal specification φ. Intuitively, the value of history h is: "winning", if the agent can enforce φ in $\mathcal{E}$ starting from h; otherwise, "pending", if the agent has a cooperatively winning strategy for φ in $\mathcal{E}$ starting from h; otherwise, "losing". With this notion, best-effort strategies are those that witness the maximum value of each history h consistent with them. Furthermore, dominant strategies are those that witness that, for pending histories h (that do not extend winning ones), exactly one agent move Y allows extending h so that the extended history $h \cdot Y$ is not losing. We formalize these notions below.

For a history h and an agent strategy σ_{ag}, we denote by $\Sigma^{\mathcal{E}}_{env}(h, \sigma_{ag})$ the set of environment strategies σ_{env} enforcing $\mathcal{E}$ such that h is consistent with σ_{ag} and σ_{env}. For an agent strategy σ_{ag}, we denote by $\mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ the set of all histories h such that $\Sigma^{\mathcal{E}}_{env}(h, \sigma_{ag})$ is non-empty, i.e., $\mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ is the set of all histories that are consistent with σ_{ag} and some environment strategy enforcing $\mathcal{E}$. For $h \in \mathcal{H}_{\mathcal{E}}(\sigma_{ag})$, we define:

1. $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h) = +1$ ("winning"), if for every $\sigma_{env} \in \Sigma^{\mathcal{E}}_{env}(h, \sigma_{ag})$, $\pi(\sigma_{ag}, \sigma_{env})$ has a finite prefix that satisfies φ; otherwise,
2. $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h) = 0$ ("pending"), if for some $\sigma_{env} \in \Sigma^{\mathcal{E}}_{env}(h, \sigma_{ag})$, $\pi(\sigma_{ag}, \sigma_{env})$ has a finite prefix that satisfies φ; otherwise,
3. $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h) = -1$ ("losing").

Finally, we denote by $val_{\varphi|\mathcal{E}}(h)$ the maximum of $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h)$ over all σ_{ag} such that $h \in \mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ (we define $val_{\varphi|\mathcal{E}}(h)$ only in case $h \in \mathcal{H}_{\mathcal{E}}(\sigma)$ for some σ).

Winning strategies are those that are winning in every history h that is consistent with them. Here is the local characterization of best-effort and dominant strategies:

Theorem 3 (Local Characterization of Best-Effort and Dominant Strategies [14, Thm. 8]) *Let φ and $\mathcal{E}$ be* LTL$_f$ *formulas over $\mathcal{Y} \cup \mathcal{X}$ denoting an agent goal and an environment specification, respectively.*

(a) *An agent strategy σ_{ag} is best-effort for φ under $\mathcal{E}$ if, and only if, for every $h \in \mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ ending in an environment move, $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h) = val_{\varphi|\mathcal{E}}(h)$;*
(b) *Furthermore, σ_{ag} is dominant for φ under $\mathcal{E}$ if, and only if, for every $h \in \mathcal{H}_{\mathcal{E}}(\sigma_{ag})$ ending in an environment move, $val_{\varphi|\mathcal{E}}(h) = 0$ implies that $val_{\varphi|\mathcal{E}}(h \cdot Y) = -1$ for every $Y \neq \sigma_{ag}(h_{\mathcal{X}})$, where $h_{\mathcal{X}}$ is the sequence of environment moves obtained by projecting h on the environment variables $\mathcal{X}$ only.*

A best-effort strategy σ_{ag} will intuitively behave as follows. Starting from every history h that is σ_{ag}-consistent, (i) if φ is satisfiable regardless of the strategy chosen by the environment, σ prescribes how to satisfy the goal; else, (ii) if φ is satisfiable only depending on a certain strategy chosen by the environment, σ prescribes a move which possibly allows satisfying the goal; else, (iii) if φ is unsatisfiable, σ prescribes some move. Furthermore, (iv) if σ_{ag} is a dominant strategy and h is a pending history, σ_{ag} prescribes the unique agent move that allows the agent to possibly satisfy the goal. As a result, all dominant strategies behaves exactly the same on pending histories (that do not extend winning ones): the difference between two dominant strategies is only in how they win from winning histories or lose from losing histories. This result is the basis of the following characterization of when dominant strategies exist:

Theorem 4 (Existence of Dominant Strategies [14, Thm. 9]) *Let φ and $\mathcal{E}$ be* LTL$_f$ *formulas denoting an agent goal and an environment specification, respectively. The agent has a dominant strategy if, and only if, for every history h that ends in an environment move:*

1. *Either $val_{\varphi|\mathcal{E}}(h') = +1$ for some prefix h' of h; or*
2. *$val_{\varphi|\mathcal{E}}(h \cdot Y) = 0$ for at most one agent move Y.*

Synthesis Technique

In this section, we review the synthesis technique for computing a best-effort strategy and decide whether it is winning, dominant, or best-effort only, presented in [14, Section Checking Dominance]. This technique is based on a reduction to solving and combining the solutions of suitable games played over deterministic finite automata (DFA games).

DFA Games

A *deterministic transition system* (aka *transition system*) is a tuple $\mathcal{D} = (\Sigma, S, s_0, \delta)$, where: Σ is a finite input alphabet; S is a finite set of states; $s_0 \in S$ is the initial state; and $\delta : S \times \Sigma \to S$ is the transition function. The cardinality of S is the *size* of $\mathcal{D}$. We extend δ to be a function $\delta : S \times \Sigma^* \to S$ as follows: $\delta(s, \lambda) = s$, and if $s_n = \delta(s, \alpha_0 \dots \alpha_{n-1})$ then $\delta(s, \alpha_0 \dots \alpha_n) = \delta(s_n, \alpha_n)$.

Definition 7 *(Product of Transition Systems)* The product of two transition systems $\mathcal{D}_i = (\Sigma, S_i, s_{(0,i)}, \delta_i)$ (for $i = 1, 2$) over the same alphabet is the transition system $\textsc{Product}(\mathcal{D}_1, \mathcal{D}_2) = \mathcal{D}_1 \times \mathcal{D}_2 = (\Sigma, S, s_0, \delta)$ 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 $\textsc{Product}(\mathcal{D}_1, \cdots, \mathcal{D}_n) = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ is defined analogously for any finite sequence $\mathcal{D}_1, \cdots, \mathcal{D}_n$ of transition systems.

A *deterministic finite automaton* (DFA) is a transition system with an acceptance condition on input words. Formally, we define a DFA as a pair $\mathcal{A} = (\mathcal{D}, F)$, where $\mathcal{D} = (\Sigma, S, s_0, \delta)$ is a deterministic transition system and $F \subseteq S$ is the set of *final states*. The *size* of $\mathcal{A}$, written $|\mathcal{A}|$, is the size of $\mathcal{D}$. An input word $\pi \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if $\delta(s_0, \pi) \in F$. The *language* recognized by $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. Two DFAS $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if they recognize the same language.

Definition 8 [27] A DFA $\mathcal{A}$ that recognizes a language $\mathcal{L}(\mathcal{A}) = \mathcal{L}$ is *minimal* if, and only if, it does not exist another DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}$ and $|\mathcal{A}'| < |\mathcal{A}|$.

For the kind of languages we consider in this paper, called *regular languages*, there always exists a minimal DFA and this is unique [27].

Proposition 5 (Complement and Intersection of DFAS [27]) *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be* DFAS *that recognize the languages $\mathcal{L}(\mathcal{A}_1)$ and $\mathcal{L}(\mathcal{A}_2)$, respectively.*

- *We can build in polynomial time the minimal* DFA $\textsc{Comp}(\mathcal{A}_1)$, *called the complement of $\mathcal{A}_1$, such that $\mathcal{L}(\textsc{Comp}(\mathcal{A}_1)) = \Sigma^* \backslash \mathcal{L}(\mathcal{A}_1)$.*
- *We can build in polynomial time the minimal* DFA $\textsc{Inter}(\mathcal{A}_1, \mathcal{A}_2)$, *called the intersection of $\mathcal{A}_1$ and $\mathcal{A}_2$, such that $\mathcal{L}(\textsc{Inter}(\mathcal{A}_1, \mathcal{A}_2)) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.*

From Proposition 5, any Boolean combination of languages recognized by DFAs, such as, e.g., implication, can be built by suitably combining complements and intersections. For instance, let $\mathcal{A}$ and $\mathcal{A}'$ be DFAs recognizing the languages $\mathcal{A}$ and $\mathcal{A}'$, respectively: the DFA recognizing the language $\mathcal{L}(\mathcal{A}) \supset \mathcal{L}(\mathcal{A}')$ can be built as $\textsc{Impl}(\mathcal{A}, \mathcal{A}') = \textsc{Comp}(\textsc{Inter}(\mathcal{A}, \textsc{Comp}(\mathcal{A}')))$ (recalling that $\alpha \supset \beta \equiv \neg(\alpha \wedge \neg\beta)$).

We now introduce an alternative construction technique to obtain the DFA recognizing any Boolean combination of languages recognized by DFAs. Unlike Proposition 5, this technique is based on *dropping* DFA minimization.

Proposition 6 [27] *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be DFAs that recognize the languages $\mathcal{L}(\mathcal{A}_1)$ and $\mathcal{L}(\mathcal{A}_2)$, respectively. The DFA recognizing an arbitrary Boolean combination of $\mathcal{L}(\mathcal{A}_1)$ and $\mathcal{L}(\mathcal{A}_2)$, i.e., $\mathcal{L}(\mathcal{A}_1)$ $[op]$ $\mathcal{L}(\mathcal{A}_2)$, where $[op] \in \{\cap, \cup, \supset, \equiv\}$, can be built as $\mathcal{A} = (\mathcal{D}_1 \times \mathcal{D}_2, F)$, where $F = \{(s_1, s_2) \in S_1 \times S_2 \mid s_1 \in F_1 \ [op] \ s_2 \in F_2\}$*

In this paper, we are interested in the property that every LTL$_f$ formula φ can be transformed into a DFA $\mathcal{A}_\varphi$ that accepts exactly the traces that satisfy the formula.

Theorem 7 (LTL$_f$ -to-DFA [10, Thm. 1]). *Given an LTL$_f$ formula φ defined over Σ, we can build in 2EXPTIME the minimal DFA $\textsc{ToDFA}(\varphi) = \mathcal{A}_\varphi = (\mathcal{D}_\varphi, F_\varphi)$ with input alphabet 2^Σ, size at most doubly-exponential size in $|\varphi|$, and such that*:

$\pi \models \varphi ifandonlyif \pi \in \mathcal{L}(\mathcal{A}_\varphi)$

A DFA game is a DFA $\mathcal{G} = (\mathcal{D}, F)$, where: $\mathcal{D} = (2^{\mathcal{Y} \cup \mathcal{X}}, S, s_0, \delta)$ is a deterministic transition system, also called the *game arena*, where $\mathcal{Y}$ and $\mathcal{X}$ are two disjoint sets of variables under the control of the agent and the environment, respectively; and $F \subseteq S$ is set of final states, also called the *goal states*. That is, a DFA game is a DFA whose input alphabet is partitioned into two disjoint sets under the control of the agent and the environment. The notions of play, history, agent strategy, and environment strategy defined in Sect. Preliminaries also apply to DFA games. A play $\pi \in (2^{\mathcal{Y} \cup \mathcal{X}})^\omega$ is *winning* if it contains a *finite* prefix that is accepted by the DFA. That is, DFA games require the set of final states to be visited at least once.

An agent strategy σ_{ag} is *winning* if, for every environment strategy σ_{env}, the play $\pi(\sigma_{ag}, \sigma_{env})$ is winning. An agent strategy σ_{ag} is *cooperatively winning* if there exists an environment strategy σ_{env} such that the play $\pi(\sigma_{ag}, \sigma_{env})$ is winning. Conversely, an environment strategy σ_{env} is *winning* if, for every agent strategy σ_{ag}, the play $\pi(\sigma_{ag}, \sigma_{env})$ is *not* winning. That is, winning strategies for the agent guarantee that, regardless of the strategy chosen by the environment, the set of final states is visited at least once; cooperatively winning strategies guarantee that there exists at least one environment strategy such that the set of final states is visited at least once; and winning strategies for the environment guarantee that, regardless of the strategy chosen by the agent, the set of final states is *never* visited.

The *winning* (resp. *cooperatively winning*) region is the set of states $s \in S$ for which the agent has a winning (resp. cooperatively winning) strategy in the game $\mathcal{G}' = (\mathcal{D}', F)$, where $\mathcal{D}' = (2^{\mathcal{Y} \cup \mathcal{X}}, S, s, \delta)$, i.e., as $\mathcal{D}$ but with initial state s. An agent strategy that is winning from every state in the agent winning region (resp. cooperatively winning region) is called *uniform winning* (resp. *uniform cooperatively winning*). The *environment winning region* is the set of states $s \in S$ for which the environment has a winning strategy in the game $\mathcal{G}' = (\mathcal{D}', F)$, where $\mathcal{D}' = (2^{\mathcal{Y} \cup \mathcal{X}}, S, s, \delta)$.

Of special interest is the case where the agent strategy can be derived from a function $\kappa : S \to 2^{\mathcal{Y}}$, called a *game strategy*, mapping game states to agent moves. While a game strategy is not formally an agent strategy, i.e., a function from sequences of environment moves to agent moves, it is *equivalent* to one such a strategy.

Definition 9 Let $\mathcal{D} = (2^{\mathcal{Y} \cup \mathcal{X}}, S, s_0, \delta)$ be a transition system. The game strategy $\kappa : S \to 2^{\mathcal{Y}}$ is *equivalent* to the agent strategy $\textsc{Strategy}(\mathcal{D}, \kappa) : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ defined as follows: for every $h \in (2^{\mathcal{Y} \cup \mathcal{X}})^*$, $\textsc{Strategy}(\mathcal{D}, \kappa)(h_{\mathcal{X}}) = \kappa(\delta(s_0, h))$, where $h_{\mathcal{X}}$ corresponds to h projected on environment variables $\mathcal{X}$ only. The pair $(\mathcal{D}, \kappa)$ is also called a *transducer*, i.e., a transition system with an output function.

Solving a DFA game is the problem of computing the winning (resp. cooperatively winning) region and a uniform winning (resp. cooperatively winning) game strategy. DFA games can be solved in linear time by a backward-induction algorithm that performs a fixpoint computation over the state space of the game.

Theorem 8 ([26]) *Let $\mathcal{G} = (\mathcal{D}, F)$ be a DFA game. Computing the winning (resp. cooperatively winning) region W (resp. $\hat{W}$) and a uniform winning (resp. cooperatively winning) game strategy κ (resp. γ), written $(W, \kappa) = \textsc{SolveAdv}(\mathcal{D}, F)$ (resp. $(\hat{W}, \gamma) = \textsc{SolveCoop}(\mathcal{D}, F)$), can be done in linear time in the size of $\mathcal{D}$.*

Games played over DFAs are determined, meaning that the environment winning region of a game $(\mathcal{D}, F)$, denoted $\textsc{EnvWin}(\mathcal{D}, F)$, is the complement of the agent winning region with respect to the state space of $\mathcal{D}$ [28].

We will also need to restrict the transitions of a transition system to those that do not leave a set of states. To do this, we use the notion of *restriction* defined below.

Definition 10 *(Restriction of Transition Systems)* Let $\mathcal{D} = (\Sigma, S, s_0, \delta)$ be a transition system and $S' \subseteq S$ a non-empty set of states. The restriction of $\mathcal{D}$ to S' is the transition system $\textsc{Restrict}(\mathcal{D}, S') = (\Sigma, S \cup \{sink\}, s_0, \delta')$ where, for $a \in \Sigma$, $\delta'(s, a) = sink$ if $s = sink$ or $\delta(s, a) \notin S'$, and $\delta'(s, a) = \delta(s, a)$ otherwise.

Basic Synthesis Algorithm

We review in Algorithm 0 the technique to synthesize a best-effort strategy and decide whether it is winning, dominant, or best-effort only, as described in [14, Sec. 6].

Algorithm 0 returns a pair consisting of the output strategy σ and a flag T describing whether σ is winning, dominant, or best-effort only, written `Win`, `Dom`, and `Be`, respectively. Algorithm 0 constructs the DFAs corresponding to three different LTL$_f$ formulas, i.e., $\mathcal{E} \supset \varphi$, $\neg\mathcal{E}$, and $\mathcal{E} \wedge \varphi$, where $\mathcal{E}$ and φ are the environment specification and the agent goal, respectively (Line **1**), solves three different games (Lines **4**, **5**, and **7**) constructed in various stages (Lines **2**, **3**, and **6**), and combines the solutions to compute the output best-effort strategy (Lines **8** and **9**). Specifically, the output best-effort strategy is returned in

Algorithm 0 $\textsc{BestEffortSynthesis}\ (\varphi, \mathcal{E})$

```
Input: LTL_f goal $\varphi$ and environment specification $\mathcal{E}$
Output: Pair $(\sigma, \mathtt{T})$, where $\sigma$ is a $\mathtt{T}$ strategy for $\varphi$ under $\mathcal{E}$ and $\mathtt{T} \in \{\mathtt{Win}, \mathtt{Dom}, \mathtt{Be}\}$
 1: $\mathcal{A}_{\mathcal{E}\supset\varphi} = \textsc{ToDFA}(\mathcal{E} \supset \varphi)$; $\mathcal{A}_{\neg\mathcal{E}} = \textsc{ToDFA}(\neg\mathcal{E})$; $\mathcal{A}_{\mathcal{E}\wedge\varphi} = \textsc{ToDFA}(\mathcal{E} \wedge \varphi)$
    Say $\mathcal{A}_{\mathcal{E}\supset\varphi} = (\mathcal{D}_{\mathcal{E}\supset\varphi}, F_{\mathcal{E}\supset\varphi})$, $\mathcal{A}_{\neg\mathcal{E}} = (\mathcal{D}_{\neg\mathcal{E}}, F_{\neg\mathcal{E}})$, and $\mathcal{A}_{\mathcal{E}\wedge\varphi} = (\mathcal{D}_{\mathcal{E}\wedge\varphi}, F_{\mathcal{E}\wedge\varphi})$
 2: $\mathcal{D} = \textsc{Product}(\mathcal{D}_{\mathcal{E}\supset\varphi}, \mathcal{D}_{\neg\mathcal{E}}, \mathcal{D}_{\mathcal{E}\wedge\varphi})$
    Say $\mathcal{D} = (2^{\mathcal{Y}\cup\mathcal{X}}, S, s_0, \delta)$ and $S = S_{\mathcal{E}\supset\varphi} \times S_{\neg\mathcal{E}} \times S_{\mathcal{E}\wedge\varphi}$
 3: Define:
      - $G_{\mathcal{E}\supset\varphi} = \{(s_{\mathcal{E}\supset\varphi}, s_{\neg\mathcal{E}}, s_{\mathcal{E}\wedge\varphi}) \in S \mid s_{\mathcal{E}\supset\varphi} \in F_{\mathcal{E}\supset\varphi}\}$
      - $G_{\neg\mathcal{E}} = \{(s_{\mathcal{E}\supset\varphi}, s_{\neg\mathcal{E}}, s_{\mathcal{E}\wedge\varphi}) \in S \mid s_{\neg\mathcal{E}} \in F_{\neg\mathcal{E}}\}$
      - $G_{\mathcal{E}\supset\varphi} = \{(s_{\mathcal{E}\wedge\varphi}, s_{\neg\mathcal{E}}, s_{\mathcal{E}\wedge\varphi}) \in S \mid s_{\mathcal{E}\wedge\varphi} \in F_{\mathcal{E}\wedge\varphi}\}$
 4: $(W, \kappa) = \textsc{SolveAdv}(\mathcal{D}, G_{\mathcal{E}\supset\varphi})$
 5: $V = \textsc{EnvWin}(\mathcal{D}, G_{\neg\mathcal{E}})$
 6: $\mathcal{D}_{\mathcal{E}} = \textsc{Restrict}(\mathcal{D}, V)$
 7: $(\hat{W}, \gamma) = \textsc{SolveCoop}(\mathcal{D}_{\mathcal{E}}, G_{\mathcal{E}\wedge\varphi})$
 8: Define the game strategy $\nu$. for every $s \in S$:
      - if $s \in W$ then $\nu(s) = \kappa(s)$
      - else if $s \in \hat{W} \setminus W$ then $\nu(s) = \gamma(s)$
      - else $\nu(s) = \mathcal{Y}$ (i.e., defined arbitrarily)
 9: $\sigma = \textsc{Strategy}(\mathcal{D}, \nu)$
10: if $s_0 \in W$ then return $(\sigma, \mathtt{Win})$
11: else:
      - if $(a)$ $\exists s \in \hat{W}$ reachable from $s_0$ with a path not traversing $W$ and
          $(b)$ $\exists Y', Y'' \in 2^{\mathcal{Y}}.\exists X', X'' \in 2^{\mathcal{X}}$ s.t. $\delta(s, Y' \cup X') \in \hat{W}$ and $\delta(Y'' \cup X'') \in \hat{W}$
        return $(\sigma, \mathtt{Be})$
      - else return $(\sigma, \mathtt{Dom})$
```

the form of a transducer over the transition system of the solved games with output function ν (Line **9**). The game strategy ν (constructed in Line **8**) is obtained by combining the game strategies κ and γ using the winning region W and the cooperatively winning region $\hat{W}$ (computed in Lines **4** and **7**) as follows. For every state $s \in S$ in the state space of the solved games: if (*i*) s is in the winning region W, then ν follows κ; else, if (*ii*) s in the cooperatively winning region $\hat{W}$, then ν follows γ; else (*iii*) ν is arbitrarily defined. Deciding whether the synthesized best-effort strategy is winning, dominant, or best-effort only, is done by checking suitable properties of the solved games (Lines **10** and **11**).

Specifically, Algorithm 0 uses the local characterization in Theorem 3 and the existence condition in Theorem 4 to compute a best-effort strategy and decide whether it is winning, dominant, or best-effort only. The intuition is as follows. For every history $h \in (2^{\mathcal{Y}\cup\mathcal{X}})^*$: (*i*) if the run induced by h in $\mathcal{D}$ reaches a state in the winning region W, i.e., $\delta(s_0, h) \in W$, where s_0 and δ are the initial state and the transition function of the game arena computed in Line 2, respectively, h is a winning history and the synthesized best-effort strategy wins starting from h by following the winning strategy κ; (*ii*) else, if the run induced by h in $\mathcal{D}$ reaches a state in the cooperatively winning region $\hat{W}$, i.e., $\delta(s_0, h) \in \hat{W}\backslash W$, h is a pending history and the synthesized best-effort strategy is pending starting from h as it follows the cooperatively winning strategy γ; (*iii*) else, h is a losing history and any strategy loses starting from h, hence the synthesized best-effort strategy can be arbitrarily defined. By Theorem 3, item (*a*), the synthesized strategy is best-effort. In case no winning strategy exists, Line 11 evaluates the conditions defined in Theorem 4 to decide the existence of a dominant strategy. Specifically, conditions (*a*) and (*b*) in Line 11 check whether items *1.* and *2.* in Theorem 4 are violated, respectively. If yes, the synthesized strategy is best-effort only; else, it is dominant as a consequence of Theorem 4 and Theorem 3, item (*b*).

Checking Dominance

We now present more details on determining whether the synthesized best-effort strategy is dominant or best-effort only when no winning strategy exists. This is done in Line 11 of Algorithm 0, which can be executed in *polynomial time* using standard graph search techniques such as, e.g., *breadth-first search*. However, in this section, we present an alternative technique that is more suited for symbolic implementation.

Algorithm 0.1 CHECKDOMINANT $(\mathcal{D}, W, \hat{W}, \sigma)$

```
Input: Game arena $\mathcal{D} = (2^{\mathcal{Y}\cup\mathcal{X}}, S, s_0, \delta)$, winning region $W$, cooperatively winning
    region $\hat{W}$, and strategy $\sigma$ computed in Lines 2, 4, 7, and 9 of Algorithm 0, resp.
Output: $(\sigma, \texttt{Dom})$ if $\sigma$ is a dominant strategy and $(\sigma, \texttt{Be})$ if $\sigma$ is best-effort only.
1: $\hat{W}'_{old} = \emptyset$
2: $\hat{W}' = \{s_0\}$
3: while $\hat{W}' \neq \hat{W}'_{old}$ do
      - $\hat{W}'_{old} = \hat{W}'$
      - $\hat{W}' = \hat{W}' \cup PostE(\hat{W}')$
          /* $PostE(\hat{W}') = \{d \in \hat{W} \setminus W \mid \exists s \in \hat{W}'.\exists Y \cup X.2^{\mathcal{Y}\cup\mathcal{X}}.d = \delta(s, Y \cup X)\}$ */
4: if $\exists s \in \hat{W}'.\exists Y', Y'' \in 2^{\mathcal{Y}}.\exists X', X''.\delta(s, Y' \cup X') \in \hat{W}$and $\delta(s, Y'' \cup X'') \in \hat{W}$
     then return $(\sigma, \texttt{Be})$
5: else return $(\sigma, \texttt{Dom})$
```

Our technique is detailed in Algorithm 0.1 and can be viewed as a subprocedure to be executed within Algorithm 0 when no winning strategy exists and the initial state is a cooperatively winning state. Else, the synthesized strategy is trivially dominant.

Algorithm 0.1 works as follows. Lines **1**-**3** perform a fixpoint computation over the state space of the game arena $\mathcal{D}$ (computed in Line 2 of Algorithm 0) to determine the states in the cooperatively winning region $\hat{W}$ (computed in Line 7 of Algorithm 0) that are reachable from the initial state s_0 without visiting any state in the winning region W (computed in Line 4 of Algorithm 0), i.e., states which satisfy condition (a), Line 11 of Algorithm 0. Lines **4** and **5** check, for every state $s \in \hat{W}'$, condition (b), Line 11 of Algorithm 0, and decide whether σ is a dominant strategy or best-effort only.

We now prove the correctness of Algorithm 0.1. First, observe that *PostE* is a *monotone* function. Since $\mathcal{D}$ is finite, it follows that the fixpoint computation in Line **3** terminates in at most a *linear* number of steps in the size of $\mathcal{D}$. The computation proceeds as follows. Initially, $\hat{W}'_0$ consists only of the initial state s_0. At every stage of the fixpoint computation, $\hat{W}'_{i+1}$ takes in those cooperatively winning states that are not winning and that can be reached from $\hat{W}'_i$ with some agent/environment move $(Y \cup X)$. The computation reaches fixpoint when $\hat{W}' = \hat{W}'_{i+1} = \hat{W}'_i$. When the fixpoint is reached, $\hat{W}$ consists exactly of the cooperatively winning states that can be reached from s_0 without visiting the winning region W.

Theorem 9 *Let $\mathcal{D} = (2^{\mathcal{Y}\cup\mathcal{X}}, S, s_0, \delta)$, W and $\hat{W}$ be the game arena, the winning region, and the cooperatively winning*

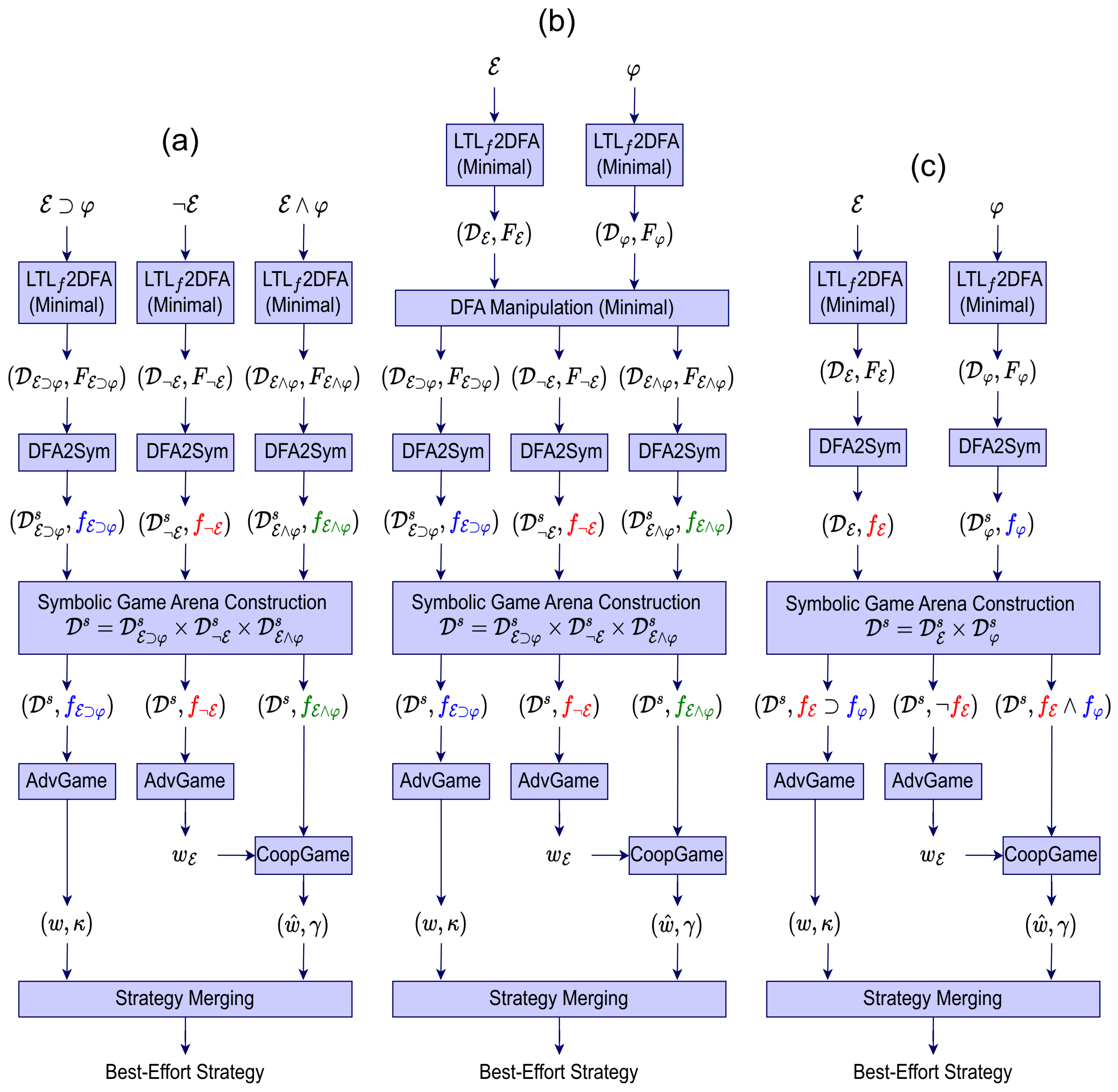

(a)
(b)
(c)
$\mathcal{E} \supset \varphi$
$\neg\mathcal{E}$
$\mathcal{E} \wedge \varphi$
$\mathcal{E}$
φ
LTL$_f$2DFA (Minimal)
$(\mathcal{D}_\mathcal{E}, F_\mathcal{E})$
$(\mathcal{D}_\varphi, F_\varphi)$
DFA Manipulation (Minimal)
$(\mathcal{D}_{\mathcal{E}\supset\varphi}, F_{\mathcal{E}\supset\varphi})$
$(\mathcal{D}_{\neg\mathcal{E}}, F_{\neg\mathcal{E}})$
$(\mathcal{D}_{\mathcal{E}\wedge\varphi}, F_{\mathcal{E}\wedge\varphi})$
DFA2Sym
$(\mathcal{D}^s_{\mathcal{E}\supset\varphi}, f_{\mathcal{E}\supset\varphi})$
$(\mathcal{D}^s_{\neg\mathcal{E}}, f_{\neg\mathcal{E}})$
$(\mathcal{D}^s_{\mathcal{E}\wedge\varphi}, f_{\mathcal{E}\wedge\varphi})$
$(\mathcal{D}_\mathcal{E}, f_\mathcal{E})$
$(\mathcal{D}^s_\varphi, f_\varphi)$
Symbolic Game Arena Construction
$\mathcal{D}^s = \mathcal{D}^s_{\mathcal{E}\supset\varphi} \times \mathcal{D}^s_{\neg\mathcal{E}} \times \mathcal{D}^s_{\mathcal{E}\wedge\varphi}$
$\mathcal{D}^s = \mathcal{D}^s_\mathcal{E} \times \mathcal{D}^s_\varphi$
$(\mathcal{D}^s, f_{\mathcal{E}\supset\varphi})$
$(\mathcal{D}^s, f_{\neg\mathcal{E}})$
$(\mathcal{D}^s, f_{\mathcal{E}\wedge\varphi})$
$(\mathcal{D}^s, f_\mathcal{E} \supset f_\varphi)$
$(\mathcal{D}^s, \neg f_\mathcal{E})$
$(\mathcal{D}^s, f_\mathcal{E} \wedge f_\varphi)$
AdvGame
CoopGame
$w_\mathcal{E}$
(w, κ)
$(\hat{w}, \gamma)$
Strategy Merging
Best-Effort Strategy

Fig. 1 From left to right, (**a**) direct, (**b**) compositional-minimal, and (**c**) compositional technique for LTL$_f$ best-effort synthesis

region computed in Lines 2, 4, *and* 7 *of Algorithm* 0, *respectively, and* $\hat{W}'$ *the region computed in Line* 3 *of Algorithm* 0.1. *For every state* $s \in S$, *it holds that* $s \in \hat{W}'$ *if, and only if,* s *is a cooperatively winning state that is reachable from* s_0 *without visiting any state in* W.

Proof ($\rightarrow$) Assume that $s \in \hat{W}'$. We prove that s is reachable from s_0 without visiting W by induction on the stages of the fixpoint computation for $\hat{W}'$, say $\hat{W}'_0, \cdots, \hat{W}'_n$ such that $\hat{W}' = \hat{W}'_{n-1} = \hat{W}'_n$. As base step, $\hat{W}'_0 = \{s_0\}$, and the claim immediately holds (recall that we assumed that s_0 is a cooperatively winning state that is not in the winning region).

Then, $s \in \hat{W}'_i$ implies that s is reachable from s_0 without visiting the winning region W is the inductive hypothesis.

We prove the inductive step. Assume $s \in \hat{W}'_{i+1}$ and $s \notin \hat{W}'_i$ (as otherwise the claim trivially follows by the inductive hypothesis), i.e., $s \in \hat{W}'_{i+1} \backslash \hat{W}'_i$. We prove that s is reachable from s_0 without visiting the winning region W. Since $s \in \hat{W}'_{i+1} \backslash \hat{W}'_i$ it follows by the fixpoint computation in Line 3 that $s \in PostE(\hat{W}'_i)$. That is, $s \in \hat{W} \backslash W$, and there exists a state $s' \in \hat{W}'_i$, for which there exists an agent/environment move $Y \cup X$ such that $\delta(s', Y \cup X) = s$. Since $s' \in \hat{W}'_i$, it follows by the inductive hypothesis that s' is reachable from s_0 without visiting the winning region W. Let $\rho = s_0 \cdots s'$ denote such a path. Observe that $\rho \cdot s = s_0 \cdots s' \cdot s$ is a path leading from s_0 to s that does not visit the winning region, since ρ does not visit the winning region and $\delta(s', Y \cup X) = s \in \hat{W} \setminus W$. This completes the inductive step and the claim holds.

($\leftarrow$) Assume that s is reachable from s_0 without visiting any state of W. We prove that $s \in \hat{W}'$. We do so by proving the contradiction.

That is, assume that $s \notin \hat{W}'$. If $s \notin \hat{W}'$, s is not added to $\hat{W}'$ in any stage of the fixpoint computation, i.e., $s \notin PostE(\hat{W}')$. By definition of $PostE(\hat{W}')$ (†) it is not the case that there exists a state $s' \in \hat{W}'$ for which there exists an agent/environment move $Y \cup X$ such that $\delta(s', Y \cup X) = s$, i.e., $\forall s' \in \hat{W}'. \forall Y \in 2^{\mathcal{Y}}. \forall X \in 2^{\mathcal{X}}. \delta(s', Y \cup X) \neq s$.

Let $\rho = s_0 \cdots \hat{s} \cdot s$ be the path from s_0 to s that does not traverse the winning region W. Observe that s_0 is a cooperatively winning state that is not winning, i.e., $s_0 \in \hat{W} \setminus W$, which in turn means $s_0 \in \hat{W}'$. By definition of $PostE$ and the fixpoint computation in Line 3, it holds that $\hat{s} \in PostE(\hat{W}'_i)$ for some i, and hence, $\hat{s} \in \hat{W}'$. By (†), it holds that $\hat{s}$ is such that, for every agent/environment move $Y \cup X$, we have $\delta(\hat{s}, Y \cup X) \neq s$, which contradicts that $\rho = s_0 \cdots \hat{s} \cdot s$ is a path from s_0 to s. □

By noting that Lines 1–3 of Algorithm 0.1 collect states satisfying condition (a), Line 11 of Algorithm 0, as stated in Theorem 9, that Line 4 of Algorithm 0.1 checks, for every such a state, condition (b), Line 11 of Algorithm 0, and the correctness of Algorithm 0, it immediately follows:

Theorem 10 (Correctness of Algorithm 0.1) *Algorithm* 0.1 *is correct, i.e.:*

- *If* σ *is a dominant strategy, Algorithm 0.1 returns* $(\sigma, \mathtt{Dom})$;
- *If* σ *is a best-effort and not dominant strategy, Algorithm 0.1 returns* $(\sigma, \mathtt{Be})$.

Symbolic Synthesis Techniques

We now present three symbolic techniques for the LTL$_f$ best-effort synthesis problem introduced in Definition 6. We base our techniques, namely *direct*, *compositional-minimal*, and *compositional* (presented in Sections "Direct Technique", "Compositional-Minimal Technique", and Compositional Technique, respectively) on Algorithm 0 in Section "Synthesis Technique". Figure 1 shows the workflows of these three symbolic synthesis techniques. In the next section, we will introduce Algorithm 4 as a symbolic technique to implement Line 11 in Algorithm 0 (which determines if the synthesized strategy is dominant or best-effort only when no winning strategy exists, see Section "Checking Dominance"). The three symbolic techniques that we will present in this section use Algorithm 4 to check the existence of a dominant strategy, for which we refer to Section "Checking Dominance Symbolically".

Symbolic DFA Games

We base our techniques on the symbolic framework for DFA games presented in [16, Sec. 4]. Consider the DFA representation described in Section DFA games 5.1 as an explicit-state representation, a symbolic DFA representation provides a more compact way by using a logarithmic number of propositions to encode the state space.

Definition 11 (Symbolic DFA) Let $\mathcal{A} = (\mathcal{D}, F)$, where $\mathcal{D} = (\Sigma, S, s_0, \delta)$ and $F \subseteq S$, be an explicit-state DFA (see Section DFA Games 5.1). The symbolic representation of $\mathcal{A}$ is defined as $\textsc{ToSym}(\mathcal{A}) = \mathcal{A}^s = (\mathcal{D}^s, f)$, with $\mathcal{D}^s = (\Sigma, \mathcal{Z}, Z_0, \eta)$, where: $\mathcal{Z}$ is a set of state variables such that $|\mathcal{Z}| = \lceil \log |S| \rceil$; $Z_0 \in 2^{\mathcal{Z}}$ is the interpretation corresponding to the initial state s_0; $\eta : 2^{\mathcal{Z}} \times \Sigma \rightarrow 2^{\mathcal{Z}}$ is a Boolean function representing the transition function; and f is a Boolean formula representing the final states.

That is, $\mathcal{Z}$ is a set of state variables such that every state $s \in S$ corresponds to an interpretation $Z \in 2^{\mathcal{Z}}$; η is a Boolean function such that, for every $a \in \Sigma$, $\eta(Z, a) = Z'$ if, and only if, Z is the interpretation of a state s and Z' is the interpretation of the successor state $s' = \delta(s, a)$; and f is a Boolean formula over $\mathcal{Z}$ that is satisfied exactly by the interpretations Z corresponding to states in F, i.e., $Z \models f$ if, and only if, Z is the interpretation corresponding to some $s \in F$. Given a DFA game $\mathcal{G} = (\mathcal{D}, F)$, we denote by $\mathcal{G}^s = (\mathcal{D}^s, f)$ its corresponding symbolic representation. To ease the terminology, we sometimes simply say that $Z \in 2^{\mathcal{Z}}$ is a state $s \in S$ as an abbreviation for Z is the interpretation corresponding to a state s.

All constructive operators defined for explicit-state DFAS can be analogously defined for their respective symbolic representations.

Definition 12 *(Symbolic Product)* The symbolic product of two symbolic transition systems $\mathcal{D}^s_i = (\Sigma, \mathcal{Z}_i, Z_{(0,i)}, \eta_i)$ (for $i = 1, 2$) over the same alphabet is the transition system PRODUCT$(\mathcal{D}^s_1, \mathcal{D}^s_2) = \mathcal{D}^s_1 \times \mathcal{D}^s_2 = (\Sigma, \mathcal{Z}, Z_0, \eta)$ with: $\mathcal{Z} = \mathcal{Z}_1 \cup \mathcal{Z}_2$; $Z_0 = Z_{(0,1)} \cup Z_{(0,2)}$; and $\eta((s_1, s_2), a) = \eta(s_1, a) \cup \eta(s_2, a)$. The product $\mathcal{D}^s_1 \times \cdots \times \mathcal{D}^s_n$ is defined analogously for any finite sequence $\mathcal{D}^s_1, \cdots, \mathcal{D}^s_n$ of symbolic transition systems.

Definition 13 *(Symbolic Restriction)* Let $\mathcal{D}^s = (\Sigma, \mathcal{Z}, Z_0, \eta)$ 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 RESTRICT$(\mathcal{D}^s, g) = (\Sigma, \mathcal{Z}, Z_0, \eta')$, where η' only agrees with η if $Z \models g$, i.e., $\eta' = \eta \wedge g$.

The symbolic synthesis framework introduced in [16] utilizes Boolean synthesis for the final step of strategy construction, which we also use in our synthesis context.

Definition 14 *(Boolean Synthesis [29])* Given two disjoint sets $\mathcal{Y}$ and $\mathcal{X}$ of output and input variables, respectively, and a Boolean formula ξ over $\mathcal{Y} \cup \mathcal{X}$, the Boolean synthesis constructs a function $\tau : 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ such that, for all $X \in 2^{\mathcal{X}}$, if there exists $Y \in 2^{\mathcal{Y}}$ such that $Y \cup X \models \xi$, then $\tau(X) \cup X \models \xi$, written $\tau = $ BOOLSYNTHESIS(ξ).

We treat Boolean synthesis as a black box, using the function τ to obtain the positional winning and cooperatively winning strategies of symbolic DFA games. For more details about techniques and algorithms for Boolean synthesis we refer to [29].

Given a symbolic DFA game $(\mathcal{D}^s, f)$, we can compute a positional uniform winning strategy for the agent through a fixpoint computation over two Boolean formulas w over $\mathcal{Z}$ and t over $\mathcal{Z} \cup \mathcal{Y}$, which represent the winning region as well as pairs of winning states with agent moves, respectively. Specifically, for each pair of winning state $Z \in 2^{\mathcal{Z}}$ and agent move $Y \in 2^{\mathcal{Y}}$, it holds that regardless of the countermove $X \in 2^{\mathcal{X}}$ chosen by the environment, the transition leads to a winning state. 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 a winning state for the agent. Note that t_0 is independent of the propositions from $\mathcal{Y}$, since once the play reaches the goal states, the agent can do whatever it wants. t_{i+1} and w_{i+1} are constructed as follows:

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

The computation reaches a fixpoint when $w_i \equiv w_{i+1} \equiv w$. To see why a fixpoint is eventually reached, note that function w_{i+1} is *monotone*. 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 $\neg w_i(\mathcal{Z})$ in function $t_{i+1}(\mathcal{Z}, \mathcal{Y})$ above, *and* there exists an agent choice Y such that, for every environment choice X, the transition leads to an already detected winning state in w_i, written $\forall \mathcal{X}.w_i(\eta(\mathcal{Z}, \mathcal{Y}, \mathcal{X}))$. We write $(w, t) = $ SOLVEADV$(\mathcal{D}^s, f)$ to denote the Boolean formulas w and t resulting from the fixpoint computation above applied to the symbolic DFA game $(\mathcal{D}^s, f)$.

When the fixpoint is reached, no more states will be added, and all agent winning states have been collected. By evaluating Z_0 on w, we can determine if there exists a winning strategy. If that is the case, t can be used to compute a uniform positional winning strategy through Boolean synthesis [29]. Passing t to Boolean synthesis, setting $\mathcal{Z}$ and $\mathcal{Y}$ as input and output variables, respectively, we obtain a uniform positional winning strategy $\tau : 2^{\mathcal{Z}} \rightarrow 2^{\mathcal{Y}}$ that is equivalent to a positional agent winning strategy, which is in turn equivalent to an agent strategy by Definition 9.

Computing a uniform positional cooperatively winning strategy can be performed through an analogous fixpoint computation. 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 moves, respectively. 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:

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

We write $(\hat{w}, \hat{t}) = $ SOLVECOOP$(\mathcal{D}^s, f)$ to denote the Boolean formulas $\hat{w}$ and $\hat{t}$ resulting from the fixpoint computation above applied to the symbolic DFA game $(\mathcal{D}^s, f)$.

Once the fixpoint is reached, checking the existence of a uniform positional cooperatively winning strategy and computing one can be done as for the uniform positional winning strategy. We evaluate $\hat{w}$ on Z_0 to determine if there exists a cooperatively winning strategy. If that is the case, we compute a uniform positional cooperatively winning strategy $\hat{\tau}$ by passing $\hat{t}$ to Boolean synthesis, setting $\mathcal{Z}$ and $\mathcal{Y}$ as input and output variables, respectively.

Direct Technique

We now present our direct technique for symbolic LTL$_f$ best-effort synthesis, shown in Fig. 1a, referred to as Algorithm 1. Please note that we do not detail the technique for checking whether the synthesized strategy is dominant or best-effort only (referred to as SYMBOLICCHECKDOMINANT in Line 12) here, which is detailed in Section Checking Dominance Symbolically.

Algorithm 1 SYMBOLICDIRECTBESTEFFORTSYNTHESIS $(\varphi, \mathcal{E})$

```
Input: LTL_f goal $\varphi$ and environment specification $\mathcal{E}$
Output: Pair $(\sigma, \mathtt{T})$, where $\sigma$ is a $\mathtt{T}$ strategy for $\varphi$ under $\mathcal{E}$ and $\mathtt{T} \in \{\mathtt{Win}, \mathtt{Dom}, \mathtt{Be}\}$
 1: $\mathcal{A}_{\mathcal{E}\supset\varphi} = \textsc{ToDFA}(\mathcal{E}\supset\varphi)$; $\mathcal{A}_{\neg\mathcal{E}} = \textsc{ToDFA}(\neg\mathcal{E})$; $\mathcal{A}_{\mathcal{E}\wedge\varphi} = \textsc{ToDFA}(\mathcal{E}\wedge\varphi)$
 2: $\mathcal{A}^s_{\mathcal{E}\supset\varphi} = \textsc{ToSym}(\mathcal{A}_{\mathcal{E}\supset\varphi})$; $\mathcal{A}^s_{\neg\mathcal{E}} = \textsc{ToSym}(\mathcal{A}_{\neg\mathcal{E}})$; $\mathcal{A}^s_{\mathcal{E}\wedge\varphi} = \textsc{ToSym}(\mathcal{A}_{\mathcal{E}\wedge\varphi})$
    Say $\mathcal{A}^s_{\mathcal{E}\supset\varphi} = (\mathcal{D}^s_{\mathcal{E}\supset\varphi}, f_{\mathcal{E}\supset\varphi})$, $\mathcal{A}^s_{\neg\mathcal{E}} = (\mathcal{D}^s_{\neg\mathcal{E}}, f_{\neg\mathcal{E}})$, and $\mathcal{A}^s_{\mathcal{E}\wedge\varphi} = (\mathcal{D}^s_{\mathcal{E}\wedge\varphi}, f_{\mathcal{E}\wedge\varphi})$
 3: $\mathcal{D}^s = \textsc{Product}(\mathcal{D}^s_{\mathcal{E}\supset\varphi}, \mathcal{D}^s_{\neg\mathcal{E}}, \mathcal{D}^s_{\mathcal{E}\wedge\varphi})$
    Say $\mathcal{D}^s = (2^{\mathcal{Y}\cup\mathcal{X}}, \mathcal{Z}, Z_0, \eta)$
 4: $(w, t) = \textsc{SolveAdv}(\mathcal{D}^s, f_{\mathcal{E}\supset\varphi})$
 5: $w_{\mathcal{E}} = \textsc{EnvWin}(\mathcal{D}^s, f_{\neg\mathcal{E}})$
 6: $\mathcal{D}^s_{\mathcal{E}} = \textsc{Restrict}(\mathcal{D}^s, w_{\mathcal{E}})$
 7: $(\hat{w}, \hat{t}) = \textsc{SolveCoop}(\mathcal{D}^s_{\mathcal{E}}, f_{\mathcal{E}\wedge\varphi})$
 8: $\nu(\mathcal{Z}, \mathcal{Y}) = (w(\mathcal{Z}) \wedge t(\mathcal{Z}, \mathcal{Y})) \vee (\hat{w}(\mathcal{Z}) \wedge \neg w(\mathcal{Z}) \wedge \hat{t}(\mathcal{Z}, \mathcal{Y}))$
 9: $\tau = \textsc{BoolSynthesis}(\nu)$
10: $\sigma = \textsc{Strategy}(\mathcal{D}^s, \tau)$
11: if $Z_0 \models w$ then return $(\sigma, \mathtt{Win})$
12: else return SYMBOLICCHECKDOMINANT$(\mathcal{D}^s, w, \hat{w}, t, \hat{t}, \sigma)$
```

The direct technique is a straightforward implementation of Algorithm 0 which uses the symbolic framework. This is achieved by simply replacing the computed explicit-state DFAS of the LTL$_f$ formulas $\mathcal{E} \supset \varphi$, $\neg\mathcal{E}$, and $\mathcal{E} \wedge \varphi$ (Line **1**) with their corresponding symbolic representations (Line **2**). We solve three different symbolic, rather than explicit, DFA games (Lines **4**, **5**, and **7**) over a symbolic game arena constructed in various stages with symbolic manipulations (Lines **3** and **6**). We combine the solutions from the solved games to obtain a best-effort strategy (Lines **8**–**10**). Deciding whether the synthesized strategy is dominant or best-effort only is done by checking suitable properties of the solved symbolic DFA games (Lines **11** and **12**).

Specifically, the output best-effort strategy is returned in the form of a transducer over the symbolic transition system of the solved games with output function τ (Line **10**). The game strategy τ is obtained by applying Boolean synthesis (Line **9**) to the Boolean formula $\nu(\mathcal{Z}, \mathcal{Y})$ (Line **8**) that is satisfied by pairs (Z, Y) such that either Z is a winning state and Y is an agent winning move, written $(w(\mathcal{Z}) \wedge t(\mathcal{Z}, \mathcal{Y}))$, *or* Z is a cooperatively winning state, but not winning, and Y is a cooperatively winning move, written $(\hat{w}(\mathcal{Z}) \wedge \neg w(\mathcal{Z}) \wedge \hat{t}(\mathcal{Z}, \mathcal{Y}))$. Applying Boolean synthesis to $\nu(\mathcal{Z}, \mathcal{Y})$ yields a game strategy as that in Line 8 of Algorithm 0.

Compositional-Minimal Technique

The main challenge in the direct technique comes from the three LTL$_f$-to-DFA conversions, one for each of the formulas $\mathcal{E} \supset \varphi$, $\neg\mathcal{E}$, and $\mathcal{E} \wedge \varphi$, which can take, in the worst case, double-exponential time [6]. To mitigate the difficulty of such LTL$_f$-to-DFA conversions, we propose a compositional-minimal technique based on constructing and manipulating the minimal explicit-state DFAS $\mathcal{A}_{\mathcal{E}}$ and $\mathcal{A}_{\varphi}$ of the LTL$_f$ formulas $\mathcal{E}$ and φ, respectively. Specifically, given $\mathcal{A}_{\varphi}$ and $\mathcal{A}_{\mathcal{E}}$, we obtain the minimal explicit-state DFAS $\mathcal{A}_{\mathcal{E}\supset\varphi}$, $\mathcal{A}_{\neg\mathcal{E}}$ and $\mathcal{A}_{\mathcal{E}\wedge\varphi}$ as follows:

- $\mathcal{A}_{\mathcal{E}\supset\varphi} = \textsc{Impl}(\mathcal{A}_{\mathcal{E}}, \mathcal{A}_{\varphi})$;
- $\mathcal{A}_{\neg\mathcal{E}} = \textsc{Comp}(\mathcal{A}_{\mathcal{E}})$;
- $\mathcal{A}_{\mathcal{E}\wedge\varphi} = \textsc{Inter}(\mathcal{A}_{\mathcal{E}}, \mathcal{A}_{\varphi})$;

where COMP and INTER, IMPL denote complement, intersection and implication of explicit-state DFAS (see Proposition 5), respectively. Transforming LTL$_f$ formulas into DFAS takes double-exponential time in the size of the formula; instead, complement, intersection, and implication of DFAS take *polynomial time* in the size of the DFA. That is, the compositional-minimal technique replaces a double-exponential time LTL_f-to-DFA conversion in the direct technique with several polynomial time DFA operations.

The workflow of the compositional-minimal technique for LTL_f best-effort synthesis, i.e., **Algorithm 2**, is shown in Fig. 1(*b*). Basically, we first translate the formulas $\mathcal{E}$ and φ into their corresponding minimal explicit-state DFAS $\mathcal{A}_\mathcal{E}$ and $\mathcal{A}_\varphi$; then, we construct the minimal explicit-state DFAS $\mathcal{A}_{\mathcal{E}\supset\varphi}$, $\mathcal{A}_{\neg\mathcal{E}}$ and $\mathcal{A}_{\mathcal{E}\wedge\varphi}$ by manipulating $\mathcal{A}_\mathcal{E}$ and $\mathcal{A}_\varphi$ with implication, complement, and intersection, respectively. The remaining steps are the same as in the direct technique.

Compositional Technique

The direct and compositional-minimal techniques are based on playing three games over the symbolic product of the (minimized) transition systems $\mathcal{D}^s_{\mathcal{E}\supset\varphi}$, $\mathcal{D}^s_{\neg\mathcal{E}}$, and $\mathcal{D}^s_{\mathcal{E}\wedge\varphi}$. However, by Proposition 6 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 distinct sets of final states over the resulting transition system. It follows that the DFAS recognizing $\mathcal{E}\supset\varphi, \neg\mathcal{E}$, and $\mathcal{E}\wedge\varphi$ can be constructed as $\mathcal{A}_{\mathcal{E}\supset\varphi} = (\mathcal{D}, F_{\mathcal{E}\supset\varphi})$, $\mathcal{A}_{\neg\mathcal{E}} = (\mathcal{D}, F_{\neg\mathcal{E}})$, 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:

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

The compositional technique for LTL_f best-effort synthesis, i.e., **Algorithm 3**, shown in Fig. 1(*c*), bases on this observation. We transform the LTL_f formulas $\mathcal{E}$ and φ, into their corresponding minimal explicit-state DFAS $\mathcal{A}_\mathcal{E}$ and $\mathcal{A}_\varphi$; we construct their corresponding symbolic representations $\mathcal{A}^s_\mathcal{E}$ and $\mathcal{A}^s_\varphi$; we build the symbolic product $\mathcal{D}^s = \mathcal{D}^s_\mathcal{E}\times\mathcal{D}^s_\varphi$ and construct the three DFA games $\mathcal{G}^s_{\mathcal{E}\supset\varphi} = (\mathcal{D}^s, f_{\mathcal{E}\supset\varphi})$, $\mathcal{G}^s_{\neg\mathcal{E}} = (\mathcal{D}^s, f_{\neg\mathcal{E}})$, and $\mathcal{G}^s = (\mathcal{D}^s, f_{\mathcal{E}\wedge\varphi})$ by defining the final states in symbolic representations (which we recall being Boolean functions) from $f_\mathcal{E}$ and f_φ as follows:

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

The remaining steps of game-solving and combining strategies are the same as in the direct and compositional-minimal techniques.

Algorithm 4 SymbolicCheckDominant $(\mathcal{D}^s, w, \hat{w}, t, \hat{t}, \sigma)$

Input: Symbolic game arena $\mathcal{D}^s$, winning region w, cooperatively winning region $\hat{w}$, winning moves t, cooperatively winning moves $\hat{t}$, and agent strategy σ, computed as in Lines 3, 4, 7, and 10 of Algorithm 1, respectively.
Output: $(\sigma, \texttt{Dom})$ if σ is a dominant strategy and $(\sigma, \texttt{Be})$ if σ is best-effort only.

```
1: $\hat{w}'_{old}(\mathcal{Z}) = \bot$
2: $\hat{w}'(\mathcal{Z}) = Z_0$
3: while $\hat{w}'(\mathcal{Z}) \neq \hat{w}'_{old}(\mathcal{Z})$ do
       - $\hat{w}'_{old}(\mathcal{Z}) = \hat{w}'(\mathcal{Z})$
       - $\hat{w}'(\mathcal{Z}) = \hat{w}'(\mathcal{Z}) \bigvee_{Z' \in PostE^s(\hat{w}'(\mathcal{Z}))} Z'$
           /* $PostE^s(\hat{w}'(\mathcal{Z})) = \{Z' \models (\hat{w}\wedge\neg w)(\mathcal{Z}) \mid \exists (Z, Y, X) \in 2^\mathcal{Z}\times 2^\mathcal{Y}\times 2^\mathcal{X}:$
           $Z \models \hat{w}'(\mathcal{Z}) \wedge Z' = \eta(Z, Y\cup X)\}$ */
4: $\hat{t}'(\mathcal{Z},\mathcal{Y}) = \hat{t}(\mathcal{Z},\mathcal{Y}) \vee (\hat{w}(\mathcal{Z}) \wedge \exists X.\hat{w}(\eta(\mathcal{Z},\mathcal{Y},\mathcal{X})))$
5: $\hat{t}''(\mathcal{Z},\mathcal{Y}) = \hat{w}'(\mathcal{Z}) \wedge \hat{t}'(\mathcal{Z},\mathcal{Y})$
6: $\hat{t}(\mathcal{Z},\mathcal{Y},\mathcal{Y}') = \hat{t}''(\mathcal{Z},\mathcal{Y}) \wedge \hat{t}''(\mathcal{Z},\mathcal{Y}') \wedge \mathcal{Y} \neq \mathcal{Y}'$.
7: if $\hat{t}(\mathcal{Z},\mathcal{Y},\mathcal{Y}')$ is satisfiable then return $(\sigma, \texttt{Be})$
8: else return $(\sigma, \texttt{Dom})$
```

Checking Dominance Symbolically

We present our symbolic technique for checking whether the synthesized strategy is dominant or best-effort only as Algorithm 4, which can be viewed as a subprocedure to be executed within Algorithms 1, 2, and 3 when no winning strategy exists and the initial state is a cooperatively winning state. Otherwise, the synthesized strategy is trivially a dominant strategy. Algorithm 4 is based on Algorithm 0.1 that we presented in Section Checking Dominance. However, differently from Algorithm 0.1, Algorithm 4 checks the existence of dominant strategies through manipulations of Boolean formulas. At the end of such manipulations, checking the existence of a dominant strategy amounts to checking the *satisfiability* of a Boolean formula. We now elaborate on the main steps.

Lines **1-3** compute the Boolean formula $\hat{w}'(\mathcal{Z})$ representing the set of cooperatively winning states in $\hat{w}(\mathcal{Z})$ that are reachable from the initial state of $\mathcal{D}^s$ with a path not visiting the winning region $w(\mathcal{Z})$, i.e., states that satisfy condition (a) in Line 11 of Algorithm 0. Such a formula is computed by implementing symbolically the fixpoint computation defined in Lines 1–3 of Algorithm 0.1 and discussed in Theorem 9.

Line **4** computes a Boolean formula $\hat{t}'(\mathcal{Z},\mathcal{Y})$ that represents the states in the cooperatively winning region and *every* of their corresponding cooperatively winning agent moves. Such a formula is computed as $\hat{t}'(\mathcal{Z},\mathcal{Y}) = \hat{t}(\mathcal{Z},\mathcal{Y}) \vee (\hat{w}(\mathcal{Z}) \wedge \exists X.\hat{w}(\eta(\mathcal{Z},\mathcal{X},\mathcal{Y})))$, i.e., Y is a cooperative agent move in state Z if, either there exists a cooperative environment move that allows the agent to move closer to the set of final states, denoted by $\hat{t}(\mathcal{Z},\mathcal{Y})$ and computed as shown in Section "Symbolic DFA Games", *or* there exists a cooperative environment move that allows the agent to stay in the cooperatively winning region $\hat{w}$, instead of moving closer to the set of final states, written $\hat{w}(\mathcal{Z}) \wedge \exists X.\hat{w}(\eta(\mathcal{Z},\mathcal{X},\mathcal{Y})))$.

Line **5** computes the Boolean formula $\hat{t}''(\mathcal{Z},\mathcal{Y}) = \hat{w}'(\mathcal{Z}) \wedge \hat{t}'(\mathcal{Z},\mathcal{Y})$ representing states in the cooperatively winning region that are reachable with a path not visiting the winning region $w(\mathcal{Z})$, written $\hat{w}'(\mathcal{Z})$, *and* every of their corresponding cooperatively winning agent moves, written $\hat{t}'(\mathcal{Z},\mathcal{Y})$.

Line **6** constructs the Boolean formula $\hat{t}(\mathcal{Z},\mathcal{Y},\mathcal{Y}') = \hat{t}''(\mathcal{Z},\mathcal{Y}) \wedge \hat{t}''(\mathcal{Z},\mathcal{Y}') \wedge \mathcal{Y} \neq \mathcal{Y}'$, where $\mathcal{Y}'$ is a set of "renamed" $\mathcal{Y}$ variables, representing cooperatively winning states Z that are reachable via a path not visiting the winning region and pairs Y and Y' corresponding to two distinct cooperatively winning agent moves. Such states satisfy the conditions (a) and (b) in Line 11 of Algorithm 0, and witness, by Theorem 4, that no dominant strategy exists. To determine if such a state exists, Line **7** checks whether $\hat{t}(\mathcal{Z},\mathcal{Y},\mathcal{Y}')$ is satisfiable: if that is the case, the synthesized strategy is best-effort only; else, the synthesized strategy is dominant.

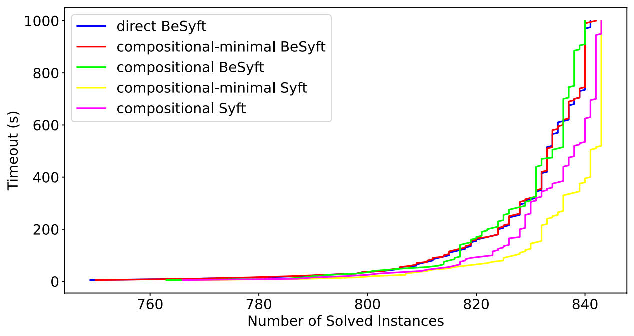


Fig. 2 Comparison of *BeSyft*- and *Syft*-implementations on random conjunction benchmarks

The correctness of Algorithm 4 is immediate by observing that it implements symbolically Algorithm 0.1, whose correctness is stated in Theorem 10.

Implementation and Empirical Evaluation

We no describe our implementations of the symbolic techniques presented in Sections "Symbolic Synthesis Techniques" and "Checking Dominance Symbolically" and present an empirical evaluation on standard LTL$_f$ synthesis benchmarks.

Implementation

We implemented our three symbolic techniques for LTL$_f$ best-effort synthesis in a tool called *BeSyft*[1] (i.e., **B**est-**E**ffort **Sy**nthesizer on **F**inite **T**races), by extending the symbolic synthesis framework in [16, Sec. 4] that has been integrated in state-of-the-art synthesis tools [17, 30]. We based *BeSyft* on LYDIA [18],[2] the overall best performing LTL$_f$-to-DFA conversion tool, to construct the minimal explicit-state DFAS of LTL$_f$ formulas. *BeSyft* also borrows from LYDIA's rich APIs to perform relevant explicit-state DFA manipulations required by both Algorithm 1, i.e., the direct technique (c.f., Subsection Direct Technique), and Algorithm 2, i.e., the compositional-minimal technique (c.f., Subsection Compositional-Minimal Technique), such as complement, intersection, and minimization. *BeSyft* represents Boolean formulas and symbolic DFA games using Binary Decision Diagrams (BDDs) [19], with CUDD 3.0.0 [31] as the BDD library. Uniform winning game strategies and uniform cooperatively winning game strategies are computed utilizing Boolean synthesis [29].

[1] https://github.com/GianmarcoDIAG/BeSyft.

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

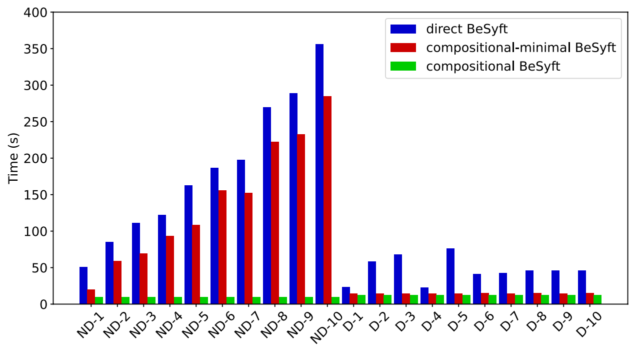


Fig. 3 Comparison of *BeSyft*-implementation on 8-bits counter games. We denote by ND-k (resp. D-k) 8-bits counter game instances where no dominant strategy exists (resp. where dominant strategies exist), where $k \in \{1, \cdots, 10\}$ is the number of increment requests

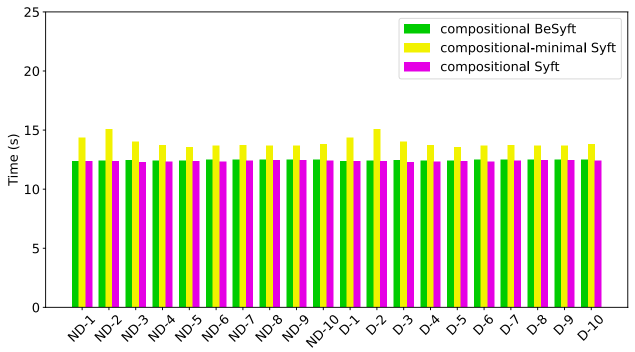


Fig. 4 Comparison of compositional-*BeSyft* and *Syft*-implementations on 8-bits counter games. We denote by ND-k (resp. D-k) 8-bits counter game instances where no dominant strategy exists (resp. where dominant strategies exist), where $k \in \{1, \cdots, 10\}$ is the number of increment requests

Best-effort game strategies are obtained by applying suitable Boolean operations to such game strategies. We distinguish three derivations of *BeSyft*, referred to as *BeSyft*-implementations. These include direct-*BeSyft*, compositional-minimal-*BeSyft*, and compositional-*BeSyft*, corresponding to the direct, compositional-minimal, and compositional techniques, respectively (c.f. Subsections Direct Technique, Compositional-Minimal Technique, and Compositional Technique, respectively). *BeSyft*-implementations utilize Algorithm 4 for checking the existence of dominant strategies (c.f. Section Checking Dominance Symbolically).

Baseline for Experiments

The goal of the empirical evaluation is to see the overhead of best-effort synthesis compared to standard reactive synthesis. We considered two reactive synthesizers based on *Syft*,[3] the tool that implements the symbolic framework in [16, Sec. 4], referred to as *Syft*-implementations. Such implementations synthesize a winning strategy for the formula $\mathcal{E} \supset \varphi$, where $\mathcal{E}$ is the environment specification and φ is the agent goal. Synthesis of $\mathcal{E} \supset \varphi$ allows to compute a winning strategy for φ under $\mathcal{E}$, if one exists [4]. The two implementations use the construction techniques in Propositions 5 and 6 and are denoted as compositional-minimal-*Syft* and compositional-*Syft*, respectively. We employ two distinct *Syft*-implementations for a fair comparison with compositional-minimal- and compositional-*BeSyft*, which use the construction techniques in Propositions 5 and 6, respectively.

Experimental Methodology

The comparison of *Syft*- and *BeSyft*-implementations was analyzed according to runtime and number of solved benchmarks. The runtime of each best-effort synthesis derivation is divided into five major phases: **1.** Conversion of the input LTL$_f$ formulas into their corresponding minimal explicit-state DFAS (LTL$_f$2DFA); **2.** Transformation of the minimal explicit-state DFAS into their corresponding symbolic representations (DFA2SYM); **3.** Fixpoint computation for obtaining the agent winning region and the agent winning moves, referred to as adversarial game (*AdvGame*); **4.** Fixpoint computation for obtaining the cooperatively winning region and the agent cooperatively winning moves, referred to as cooperative game; (*CoopGame*): and **5.** Checking the existence of a dominant strategy (*DomCheck*).

We performed our experiments on a benchmark set consisting of 1200 instances taken from existing works, which includes random conjunctions [16] (1000 instances) and variations of classical counter games [17, 32, 33] for best-effort synthesis, with and without dominant strategies (100 instances in each case).

Random Conjunction. Each instance is constructed as follows. For a length parameter ℓ, it selects ℓ base cases from a pool of LTL benchmarks interpreted with the LTL$_f$ semantics [34], takes their conjunction and renames propositions so that they are shared across conjuncts. A large value of ℓ does not guarantee a larger minimal explicit-state DFA. Not all random conjunction instances admit a winning strategy. In our experiments, ℓ ranges from 1 to 5. For each ℓ, we have 200 instances, for a total of 1000 instances. In each instance, we take $\mathcal{E} = \top$ as the environment specification.

Counter Games. Each instance is defined as follows. The agent maintains an n-bits counter such that: (i) at each round, the environment chooses whether to request an increment of the counter (*add*), and the agent chooses whether to grant such a request (*accept*) or not; (ii) the counter is initialized with all bits set to 0, and the agent goal is for the counter to have all bits set to 1; (iii)

[3] https://github.com/Shufang-Zhu/Syft

environment specifications define possible policies according to which the environment issues increment requests. We consider as environment specifications 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. Given an n-bit counter and an environment specification $\mathcal{E}_k$, the existence of a winning strategy depends on k and n: if $k \geq 2^n - 1$, a winning strategy exists. Regardless of the realizability of the agent goal, a best-effort (possibly winning) strategy for the agent is to accept all increment requests coming from the environment. In counter games, both n and k range from 1 to 10, generating a total number of 100 instances.

Counter Games with Dominant Strategy. In counter games defined as above, when no winning strategy exists, no dominant strategy exists either. The reason is that, when receiving an increment request, the agent can decide to accept it or not. We now formalize this argument. Let $h = \{add\}$ be the history where the environment issues an increment request in the first time instant (and all agent variables are set to false). Observe that $val_{\varphi|\mathcal{E}}(h) = 0$ and that the extensions $h_1 = h \cdot \{accept\}$ and $h_2 = h \cdot \{\emptyset\}$, where the agent accepts and rejects the environment increment request, respectively, are such that $val_{\varphi|\mathcal{E}}(h_1) = val_{\varphi|\mathcal{E}}(h_2) = 0$. In fact, the agent strategy σ_{ag} that accepts all but the first increment request achieves $val_{\varphi|\mathcal{E}}(\sigma_{ag}, h_1) = val_{\varphi|\mathcal{E}}(\sigma_{ag}, h_2) = 0$. To see this, it suffices to observe that, for the environment strategy σ_{env} that always issues increment requests, the induced play $\pi(\sigma_{ag}, \sigma_{env})$ eventually satisfies the agent goal specification. As a result, the history $h = \{add\}$ violate both conditions in Theorem 4, showing that no dominant strategy exists.

To obtain counter game instances where the agent has a dominant strategy, we modify the agent goal as follows: *(iv)* all counter bits set to 1 *and* accept every increment request coming from the environment. With such a goal specification, a dominant (possibly winning) strategy for the agent is to accept all increment requests coming from the environment. In such counter games, both n and k range from 1 to 10, hence also generating a total number of 100 instances.

Experimental Setup. All experiments were run on a laptop with an operating system 64-bit Ubuntu 22.04, 3.6 GHz CPU, and 12 GB of memory. The timeout was set to 1000s.

Empirical Results and Analysis

Comparison of *Syft*- and *BeSyft*-implementations. We compared the performance of *Syft*- and *BeSyft*-implementations on both random conjunction and counter-game benchmarks. Figure 2 shows that, on random conjunction benchmarks, *Syft*-implementations outperform *BeSyft*-implementations with less running time. This result follows because *Syft*-implementations involve neither solving cooperative games nor checking the existence of dominant strategies, which are required in *BeSyft*-implementations. In particular, if no winning strategy exists, *Syft*-implementations terminate returning no strategy. *BeSyft*-implementations, instead, continue with cooperative game solving and subsequent steps. Nonetheless, we also observed that *(i)* *Syft*- and *BeSyft*-implementations perform very similarly in terms of number of solved random conjunction instances, and *(ii)* *BeSyft*-implementations often show minor overhead compared to *Syft*-implementations, even when no winning strategy exists.

On counter-game benchmarks, all *Syft*- and *BeSyft*-implementations manage to solve at most 8-bits counter-game instances, both with and without dominant strategies, with the number of increment requests k ranging from 1 to 10. As a result, *Syft*- and *BeSyft*-implementations perform exactly the same in terms of the number of solved counter-game instances. We show in Fig. 3 the performance comparison of *BeSyft*-implementations in 8-bits counter games. The results show that compositional-*BeSyft* outperforms both direct- and compositional-minimal-*BeSyft* in all 8-bits counter games, both with and without dominant strategies, up to gaining several orders of magnitude better performance. In Fig. 4, we report the performance comparison of compositional-*BeSyft*, the best performing best-effort synthesis tool, and *Syft*-implementations on 8-bits counter games. The results show that compositional-*BeSyft* only performs slightly worse than compositional-*Syft*, and, surprisingly, slightly better than compositional-minimal-*Syft*.

Notably, the empirical results in both random conjunction and counter-game benchmarks show that performing best-effort synthesis and checking the existence of dominant strategies bring indeed a minor overhead compared to standard reactive synthesis. In fact, all *Syft*- and *BeSyft*-implementations perform very similarly both in terms of running times and number of solved instances.

Relative Time Costs of Major Operations. In order to have a deeper understanding of the obtained results, we evaluated the relative time costs of each of the major operations in *BeSyft*-implementations (mentioned in Sect. Experimental Methodology). In random conjunction benchmarks, we observed that transforming LTL$_f$ specification into DFAS counts, on average, for about $\sim 70\%$ of the total time cost. Differently, solving cooperative games and checking the existence of dominant strategies counts for about $\sim 10\%$ of the total time cost. Compared with *Syft*-implementations, *BeSyft*-implementations require solving cooperative games and checking the existence of dominant strategies. As mentioned, these operations involve a minor overhead, which explains the little performance difference between *Syft*- and *BeSyft*-implementations on random conjunction benchmarks observed in Fig. 2.

In Fig. 5, we report the relative time costs of the major operations performed by *BeSyft*-implementations in counter-game instances where no dominant strategy exists. We can see that the running time of compositional-*BeSyft*, the best performing best-effort synthesis tool in counter-game benchmarks, is dominated by transforming LTL$_f$ specifications into DFAS, counting for about $\sim 80\%$ of the total time cost. Differently, solving cooperative games and checking the existence of dominant strategies count for about $\sim 10\%$ of the total time cost of compositional-*BeSyft*. As discussed above for random conjunction benchmarks, this result explains the little performance difference between compositional-*BeSyft* and *Syft*-implementations observed in Fig. 4.

We observe that the total time cost of direct- and compositional-minimal-*BeSyft* is dominated by solving games, counting for about $\sim 45\%$ and $\sim 60\%$ of the total time cost, respectively. In fact, it should be noted that, differently from direct- and compositional-minimal-*Syft*, compositional-*BeSyft*, being based on Proposition 6, does not fully exploit the power of DFA minimization. To see this, observe that direct- and compositional-minimal-*BeSyft* constructs the game arena by taking the symbolic product of the minimized transition systems $\mathcal{D}^s_{\mathcal{E}\supset\varphi}$, $\mathcal{D}^s_{\neg\mathcal{E}}$, and $\mathcal{D}^s_{\mathcal{E}\wedge\varphi}$. That is, the resulting game arena is as minimized as possible for the solved games, i.e., $\mathcal{G}^s_{\mathcal{E}\supset\varphi}$, $\mathcal{G}^s_{\neg\mathcal{E}}$, and $\mathcal{G}^s_{\mathcal{E}\wedge\varphi}$. Differently, compositional-*BeSyft* constructs the game arena as the symbolic product of the transition systems $\mathcal{D}^s_{\mathcal{E}}$ and $\mathcal{D}^s_{\varphi}$. As a result, the constructed game arena is not as minimized as possible for the games $\mathcal{G}^s_{\mathcal{E}\supset\varphi}$, $\mathcal{G}^s_{\neg\mathcal{E}}$, and $\mathcal{G}^s_{\mathcal{E}\wedge\varphi}$. 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. 3, compositional-*BeSyft* performs better than both direct- and compositional-minimal-*BeSyft*, though the first does not minimize the game arena after the symbolic product, and the others minimize the game arena as much as possible. This result applies to *Syft*-implementations as well, which explains why compositional-*Syft* performs slightly better than compositional-minimal-*Syft* in Fig. 4.

We observe that, in all considered benchmarks, the total time cost of *BeSyft*-implementations is dominated by transforming LTL$_f$ specifications into DFAS. This results confirms that, as for standard reactive synthesis [16], the performance bottleneck of best-effort synthesis is transforming LTL$_f$ specifications into DFAS. Also, we observe that solving the cooperative game takes slightly longer than solving the adversarial game. Indeed, this is because the fixpoint computation in the cooperative game often requires more iterations than that in the adversarial game. Finally, we note that checking the existence of dominant strategies brings virtually no overhead to the running time of all *BeSyft*-implementations.

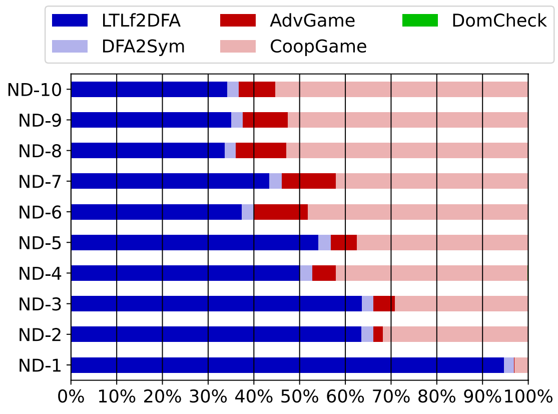
Related Works

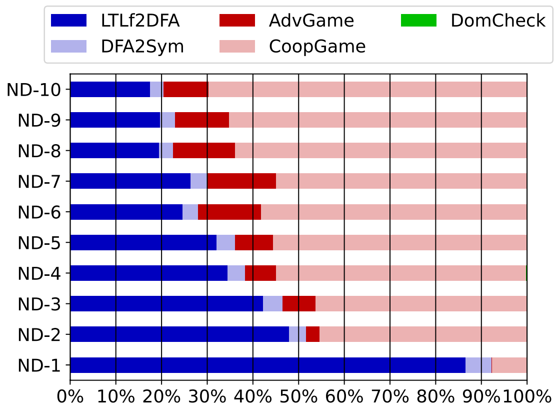
This work primarily contributes to strategy synthesis considering LTL$_f$ goals and environment specifications, specifically addressing winning, dominant, and best-effort strategies. In particular, we leverage symbolic techniques to compute solutions for DFA games and check the existence of dominant strategies. Furthermore, we employ compositional techniques to combine solutions from different games into the final strategy. In this section, we review related works in these areas.

Temporal Logics on Finite Traces. Linear Temporal Logic (LTL$_f$) and Linear Dynamic Logic (LDL$_f$) on finite traces, an extension of LTL$_f$ that captures *regular expressions*, have been introduced by De Giacomo and Vardi [6]. Both logics are extensively used in both Artificial Intelligence and Computer Science, in particular, LTL$_f$. For instance, they are used in planning in fully observable nondeterministic (FOND) domains with temporal specifications [35, 36], to express trajectory constrains in PDDL 3.0 [35, 37], in the theory of Markov Decision Processes with non-Markovian rewards [38–41] with applications in Reinforcement Learning [42–44], in Business Process Management [45, 46], and many others.

LTL$_f$ and LDL$_f$ are finite trace variants of Linear Temporal Logic (LTL) and Linear Dynamic Logic (LDL), which have been introduced in [7] and [47], respectively. Recently, Pure-past versions of LTL$_f$ and LDL$_f$, namely PLTL$_f$ and PLDL$_f$ [48], respectively, have attracted extensive research interests due to their exponential computational advantage in transforming to DFAS compared to LTL$_f$ and LDL$_f$, and are extensively used, e.g., in planning for temporally extended goals [49, 50].

LTL$_f$ **Reactive Synthesis.** De Giacomo and Vardi introduced LTL$_f$ reactive synthesis in [10], which was proven to be 2EXPTIME-complete. They provided an optimal (wrt computational complexity) game-theoretic approach to it. LTL$_f$ environment specifications have been later used to express formal knowledge about how the external environment works and have been investigated in several works, such as, e.g., [4, 22]. These works show that considering LTL$_f$ environment specifications enables capturing safety environment specifications [21, 51], which notably includes FOND domains used in planning [3, 36]. Building on these works, many variants of LTL$_f$ reactive synthesis have also been developed to consider environment specifications expressed in combinations of liveness and safety properties [52], as





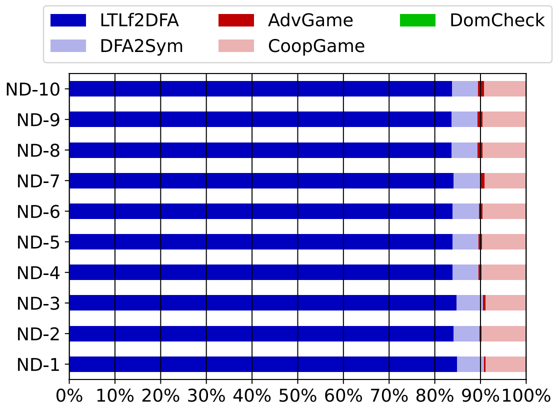


Fig. 5 Top left and right, relative time cost of each of the major operations performed by direct-*BeSyft*, compositional-minimal-*BeSyft*, respectively; bottom, the relative time cost of each of the major operations performed by compositional-*BeSyft*. We denote by ND-k 8-bits counter game instances where no dominant strategy exists, where $k \in \{1, \cdots, 10\}$ is the number of increment requests

LTL specifications [33], or as combinations of LTL$_f$ and LTL specifications [32].

Symbolic and Compositional Techniques to LTL$_f$ Reactive Synthesis. Symbolic techniques for LTL$_f$ reactive synthesis have been introduced by Zhu et al. in [16], which shows that converting LTL$_f$ specifications into DFAS is the bottleneck of LTL$_f$ synthesis and that symbolic techniques provide promising performance. Currently, symbolic techniques are integrated into all state-of-the-art tools for reactive synthesis with LTL$_f$ specifications.

Compositional techniques aiming to further tackle the double-exponential blowup resulting from LTL$_f$-to-DFA conversion have been investigated in [17, 18]. Intuitively, these techniques transform small components of the input LTL$_f$ specifications into DFAS, and combine the obtained DFAS utilzing common DFA manipulations, such as, e.g., negation, intersection, (explicit and symbolic) product, and minimization. Along these lines, compositional techniques have also been developed for LTL$_f$ reactive synthesis [53–55] and LTL$_f$ reactive synthesis under environment specifications [32, 56].

Best-Effort and Dominant Strategies. Best-effort and dominant strategies are maximal and maximum, respectively, in the *dominance* order. The notion of dominance originates from *game theory*, where it is also called *admissibility* [26]. In the context of synthesis for LTL$_f$ goals under environment specifications, Aminof et al. introduced best-effort strategies as solutions to the problem of synthesizing a strategy that wins against all expected environment strategies, and that, simultaneously, handles exceptional environment strategies resulting from, e.g., deteriorated working conditions due to mechanical failures, so that two environment models are considered [11]. Lately, Aminof et al. investigated best-effort synthesis for LTL$_f$ goals under environment specifications in the case of a single environment model. They presented an optimal (wrt computational complexity) game-theoretic approach to it [12]. Finally, dominant strategies for LTL$_f$ goals and environment specifications have been investigated by Aminof et al. in [14]. Specifically, they studied the relation between winning, dominant, and best-effort strategies, provided necessary and sufficient conditions for the existence of dominant strategies, and presented a unified approach for synthesizing strategies and deciding whether they are winning, dominant, or best-effort. We reported their technique as Algorithm 0 in Section Basic Synthesis Algorithm.

More generally, dominance has been investigated in games on graphs. Iterated avoidance of dominated strategies has been investigated in [57] and admissibility has been advocated as a suitable solution concept in environments that may not be adversarial [58]. Different variations of the concept of synthesizing dominant strategies have been explored in the context of *good-enough synthesis* for the multi-valued setting and LTL_f goals in [59, 60], respectively.

Relationship with Planning in Nondeterministic Domains. Synthesis for LTL_f goals and environment specifications can be seen as a general form of planning in FOND domains [4, 61]. In this context, De Giacomo and Rubin [62] investigated the synthesis of *strong* and *strong-cyclic plans* for LTL_f goals in FOND domains. In our terminology, strong plans are winning strategies under the environment specification of the planning domain, and strong-cyclic plans are winning strategies under both the environment specification of the planning domain and the *fairness* assumption on the effects of agent actions. Fairness assumes that if an agent performs the same action in the same state infinitely often, then all possible nondeterministic effects of such action in that state will eventually manifest. As a result, an agent using a strong-cyclic plan considers the environment as not strictly adversarial when choosing the nondeterministic effects of agent actions, i.e., environment responses might help the agent to achieve its goal. Strong and strong-cyclic plans have been introduced in [1, 2] and since then have been considered standard solution concepts in the literature on planning in nondeterministic planning domains. Building on [62] and [12], De Giacomo et al. [23] investigated *best-effort plans*, i.e., best-effort strategies in nondeterministic planning domain. As strong-cyclic plans, best-effort plans consider that environment responses might help the agent to achieve its goal. Unlike strong-cyclic plans, however, best-effort plans consider that the environment can help the agent even when fairness on the effects of agent actions can not be assumed, which has been motivated in many works [63, 64]. The relationship between strong-cyclic and best-effort strategies is not trivial and has been investigated separately in [65].

Conclusion

While many works in the literature often focus on the synthesis of winning strategies [10, 16], it is not guaranteed that such a strategy exists. In such cases, the agent could adopt a dominant or best-effort strategy, ensuring goal satisfaction against a maximum or a maximal set of environment strategies, respectively [11, 12, 14].

In this paper, we proposed unified symbolic techniques to the synthesis of winning, dominant and best-effort strategies for LTL_f goals and environment specifications, leveraging the synthesis approach in [14] and the symbolic framework in [16]. Our techniques construct and manipulate Boolean formulas corresponding to properties of interest in several DFA games obtained from the agent goal and the environment specification. An empirical analysis shows that our techniques bring indeed a minor overhead to standard reactive synthesis. This interesting result shows the suitability of our unified synthesis approaches as an alternative to standard synthesis techniques, which only compute a winning strategy if one exists. Given this nice computational result, an interesting future direction is exploring LTL_f best-effort synthesis with multiple specifications [11, 13, 22]. However, the details of such extensions are left for future work.

Acknowledgements This work has been partially supported by the ERC-ADG WhiteMech (No. 834228), the PRIN project RIPER (No. 20203FFYLK), and the PNRR MUR project FAIR (No. PE0000013). This work has been carried out while Gianmarco Parretti was enrolled in the Italian National Doctorate on Artificial Intelligence run by Sapienza University of Rome.

Author Contributions All authors contributed equally to this work.

Data Availability Statement All source codes, benchmarks, and instructions to reproduce the experiments, are publicly available at: https://github.com/GianmarcoDIAG/BeSyft.

Declaractions

Conflict of interest The authors declare they have no financial interests.

Research Involving Human And/Or Animals Not applicable.

Informed Consent Not applicable.

References

1. Cimatti A, Roveri M. Traverso P. Strong planning in non-deterministic domains via model checking. In: AIPS, 1998:36–43
2. Cimatti A, Pistore M, Roveri M, Traverso P. Weak, strong, and strong cyclic planning via symbolic model checking. AIJ. 2003;1–2:35–84.
3. Geffner H, Bonet B. A Concise Introduction to Models and Methods for Automated Planning. Synthesis Lectures on Artificial Intelligence and Machine Learning. 2013
4. Aminof B, De Giacomo G, Murano A, Rubin S. Planning under LTL environment specifications. In: ICAPS, 2019:31–39
5. Baier C, Katoen J-P. Principles of Model Checking. 2008
6. De Giacomo G, Vardi MY. Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI, 2013:854–860
7. Pnueli A. The temporal logic of programs. In: FOCS, 1977:46–57
8. Pnueli A, Rosner R. On the Synthesis of a Reactive Module. In: POPL 1989
9. Finkbeiner B. Synthesis of reactive systems. Dependable Software Systems Eng. 2016;45:72–98.
10. De Giacomo G, Vardi MY. Synthesis for LTL and LDL on finite traces. In: IJCAI, 2015:854–860
11. Aminof B, De Giacomo G, Lomuscio A, Murano A, Rubin,S. Synthesizing strategies under expected and exceptional environment behaviors. In: IJCAI, 2020:1674–1680
12. Aminof B, De Giacomo G, Rubin S. Best-effort synthesis: Doing your best is not harder than giving up. In: IJCAI, 2021:1766–1772
13. Aminof B, De Giacomo G, Lomuscio A, Murano A, Rubin S. Synthesizing best-effort strategies under multiple environment specifications. In: KR, 2021:42–51
14. Aminof B, De Giacomo G, Rubin S. Reactive synthesis of dominant strategies. In: AAAI, 2023:6228–6235
15. Henriksen JG, Jensen JL, Jørgensen ME, Klarlund N, Paige R, Rauhe T, Sandholm A. Mona: Monadic second-order logic in practice. In: TACAS
16. Zhu S, Tabajara LM, Li J, Pu G, Vardi MY. Symbolic LTL$_f$ synthesis. In: IJCAI, 2017:1362–1369
17. Bansal S, Li Y, Tabajara LM, Vardi MY. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In: AAAI, 2020:9766–9774
18. De Giacomo G, Favorito M. Compositional approach to translate LTL$_f$/LDL$_f$ into deterministic finite automata. In: ICAPS, 2021:122–130
19. Bryant RE. Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv. 1992;24:293–318.
20. De Giacomo G, Parretti G, Zhu S. Symbolic LTL$_f$ best-effort synthesis. In: EUMAS, 2023:228–243
21. Manna Z, Pnueli A. A hierarchy of temporal properties. In: PODC, 1990:377–410
22. Camacho A, Bienvenu M, McIlraith SA. Finite LTL synthesis with environment assumptions and quality measures. In: KR, 2018:454–463
23. De Giacomo G, Parretti G, Zhu S. LTL$_f$ best-effort synthesis in nondeterministic planning domains. In: ECAI, 2023:533–540
24. Haslum P, Lipovetzky N, Magazzeni D, Muise C. An Introduction to the Planning Domain Definition Language. Synthesis Lectures on Artificial Intelligence and Machine Learning, 2019
25. Gabaldon A. Non-Markovian control in the situation calculus. AIJ. 2011;175:25–48.
26. Apt KR, Grädel E (eds.): Lectures in Game Theory for Computer Scientists, 2011
27. Sipser M. Introduction to the Theory of Computation, 1997
28. Gale D, Stewart FM. Infinite games with perfect information. Contributions to the Theory of Games. 1953;2:2–16.
29. Fried D, Tabajara LM, Vardi MY. BDD-based Boolean functional synthesis. In: CAV, 2016:402–421
30. Favorito M, Zhu S. LydiaSyft: A compositional symbolic synthesizer for LTL$_f$ specifications
31. Somenzi F. CUDD: CU decision diagram package 3.0.0. Universiy of Colorado at Boulder 2016
32. De Giacomo G, Di Stasio A, Moshe V, Zhu S. Two-stage technique for LTL$_f$ synthesis under LTL assumptions. In: KR, 2020:304–314
33. Zhu S, De Giacomo G, Pu G, Vardi MY. LTL$_f$ synthesis with fairness and stability assumptions. In: AAAI, 2020:3088–3095
34. Jobstmann B, Bloem R. Optimizations for LTL synthesis. In: FMCAD, 2006:117–124
35. Bacchus F, Kabanza F. Planning for temporally extended goals. Ann Math Artif Intell. 1998;22:5–27.
36. Camacho A, McIlraith SA. Strong fully observable non-deterministic planning with LTL and LTL$_f$ goals. In: IJCAI, 2019:5523–5531
37. Gerevini AE, Haslum P, Long D, Saetti A, Dimopoulos Y. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. AIJ. 2009;173:619–68.
38. Bacchus F, Boutilier C, Grove AJ. Rewarding behaviors In: AAAI/IAAI. 1996;2:1160–7.
39. Brafman R, De Giacomo G, Patrizi F. LTL$_f$/LDL$_f$ non-Markovian rewards. In: AAAI, 2018:1771–1778
40. Brafman R, De Giacomo G, et al. Planning for LTL$_f$/LDL$_f$ goals in non-Markovian fully observable nondeterministic domains. In: IJCAI, 2019:1602–1608
41. Brafman R, De Giacomo G. Regular decision processes: A model for non-Markovian domains. In: IJCAI, 2019:5516–5522
42. Camacho A, Icarte RT, Klassen TQ, Valenzano RA, McIlraith SA. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In: IJCAI, 2019:6065–6073
43. De Giacomo G, Favorito M, Iocchi L, Patrizi F, Ronca A. Temporal logic monitoring rewards via transducers. In: KR, 2020:860–870
44. De Giacomo G, Iocchi L, Favorito M, Patrizi F. Foundations for restraining bolts: Reinforcement learning with LTL$_f$/LDL$_f$ restraining specifications. In: ICAPS, 2019:13659–13662
45. Pešić M, Bošnački D, Aalst WM. Enacting declarative languages using LTL: avoiding errors and improving performance. In: SPIN, 2010:146–161
46. Ciccio CD, Montali M. Declarative process specifications: Reasoning, discovery, monitoring. In: Process Mining Handbook. Lecture Notes in Business Information Processing, vol. 448, 2022:108–152
47. Harel D, Kozen D, Tiuryn J. Dynamic logic. SIGACT News. 2001;32:66–9.
48. De Giacomo G, Stasio AD, Fuggitti F, Rubin S. Pure-past linear temporal and dynamic logic on finite traces. In: IJCAI, 2020:4959–4965
49. Bonassi L, De Giacomo G, Favorito M, Fuggitti F, Gerevini A, Scala E. Planning for temporally extended goals in pure-past linear temporal logic. In: ICAPS, 2023:61–69
50. Bonassi L, De Giacomo G, Favorito M, Fuggitti F, Gerevini AE, Scala E. FOND planning for pure-past linear temporal logic goals. In: ECAI, 2023:279–286
51. De Giacomo G, Di Stasio A, Tabajara LM, Vardi MY, Zhu S. Finite-trace and generalized-reactivity specifications in temporal synthesis. In: IJCAI, 2021:1852–1858

52. Aminof B, De Giacomo G, Di Stasio A, Francon H, Rubin S, Zhu S. LTL_f synthesis under environment specifications for reachability and safety properties. In: EUMAS, 2023:263–279
53. Camacho A, Baier JA, Muise CJ, McIlraith SA. Finite LTL synthesis as planning. In: ICAPS, 2018:29–38
54. Tabajara LM, Vardi MY. Partitioning techniques in LTL_f synthesis. In: IJCAI, 2019:5599–5606
55. Bansal S, De Giacomo G, Di Stasio A, Li Y, Vardi MY, Zhu S. Compositional safety LTL synthesis. In: VSTTE, 2022:1–19
56. He K, Wells AM, Kavraki LE, Vardi MY. Efficient symbolic reactive synthesis for finite-horizon tasks. In: ICRA, 2019:8993–8999
57. Berwanger D. Admissibility in infinite games. In: STACS, 2007:188–199
58. Faella M. Admissible strategies in infinite games over graphs. In: MFCS, 2009:307–318
59. Almagor S, Kupferman O. Good-enough synthesis. In: CAV (2), 2020:541–563
60. Li Y, Turrini A, Vardi MY, Zhang L. Synthesizing good-enough strategies for LTL_f specifications. In: IJCAI, 2021:4144–4151
61. Camacho A, Bienvenu M, McIlraith SA. Towards a unified view of AI planning and reactive synthesis. In: ICAPS, 2019:58–67
62. De Giacomo G, Rubin S. Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals. In: IJCAI, 2018:4729–4735
63. Camacho A, McIlraith SA. Strong-cyclic planning when fairness is not a valid assumption. In: KnowProS@IJCAI 2016
64. Geffner T, Geffner H. Compact policies for fully observable nondeterministic planning as SAT. In: ICAPS, 2018:88–96
65. Aminof B, De Giacomo G, Rubin S, Zuleger F. Beyond strong-cyclic: Doing your best in stochastic environments. In: IJCAI, 2022:2525–2531